CRÉATIONS NUMÉRIQUES

RESSOURCES TÉLÉCHARGEABLES

PYTHON POUR LE LYCÉE

Promenons-nous dans les maths !

Vincent Maille
Fatima Estevens
Guillaume Miannay

ellipses

CRÉATIONS NUMÉRIQUES

PYTHON POUR LE LYCÉE
Promenons-nous dans les maths !

Vincent Maille
Fatima Estevens
Guillaume Miannay

ellipses

Collection CRÉATIONS NUMÉRIQUES
dirigée par Vincent Maille
http://creations-numeriques.fr/

Retrouvez tous les titres de la collection et des extraits
sur www.editions-ellipses.fr

ISBN 9782340-089594

AVANT-PROPOS

Ce livre est une actualisation profonde du livre « *Python - Les bases de l'algorithmique et de la programmation* » paru chez Ellipses en 2015. Souhaitant insister sur les liens entre les mathématiques et la programmation, je me suis entouré de deux collègues et amis de longue date Fatima ESTEVENS et Guillaume MIANNAY tous deux enseignants-formateurs aux qualités pédagogiques reconnues.

Ensemble, nous avons entrepris de remettre ce livre en adéquation avec les nouveaux programmes en nous recentrant sur l'activité mathématique. Ainsi, certains exercices qui ne nous semblaient plus pertinents ont été mis de côté, d'autres ont été remodelés et nous en avons imaginé de nouveaux qui ont été testés avec nos élèves et étudiants.

Comme le titre le laisse penser, nous vous invitons à prendre ce livre comme une promenade mathématique. En effet, que vous soyez enseignant débutant ou plus expérimenté, lycéen, étudiant ou simplement intéressé par les sciences, portant un intérêt marqué pour les activités liant l'algorithmique aux mathématiques, vous aurez plaisir à découvrir un langage informatique simple et puissant au service de la résolution de problèmes mathématiques.

Le langage Python s'étant beaucoup popularisé ces dernières années, la partie cours sur le langage a été pensée de manière bien plus synthétique que dans le précédent ouvrage pour privilégier de nombreux exercices concrets ayant une réelle plus value avec ce langage. Lors de votre balade, n'hésitez pas à aller "chiner" dans toutes les pages, le livre n'a pas été pensé de manière linéaire mais les exercices ont plutôt été regroupés par thèmes. À l'instar d'une boussole, le lecteur pourra retrouver dans les dernières pages, différents tableaux résumant les notions traitées dans les exercices ou le niveau des classes dans lesquelles ils peuvent être envisagés. D'ailleurs un certain nombre d'entre eux peuvent être abordés dès le collège.

Sur votre chemin, vous découvrirez de nombreuses activités faisant la part belle aux mathématiques avec une utilisation pertinente de Python

tout en développant l'esprit critique. Parfois la programmation permet de découvrir une notion mathématique, parfois elle permet de réaliser des essais, des conjectures ou même des preuves partielles ou complètes. À l'inverse, vous pourrez être amené à rencontrer un problème informatique qui nécessitera une analyse mathématique ou une réflexion aux limites d'un langage de programmation... bref vous l'aurez compris, ces deux domaines sont étroitement liés !

Des prolongements sont proposés tout au long du manuel pour permettre aux lycéens de se préparer aux attentes de l'enseignement supérieur. N'hésitez donc pas à regarder **les corrections** des exercices dans les pages dédiées. En effet, elles proposent souvent plusieurs solutions à une même question, ce qui permet de comparer les avantages et inconvénients de chacune d'elles. Les notions traitées dans **les thèmes** abordent l'aspect culturel et historique des mathématiques. Par souci de gain de place, les thèmes ne sont pas corrigés dans ce livre, des CODES sont présents sur votre chemin pour vous permettre d'avoir accès à toutes les corrections des thèmes, des exercices ainsi qu'aux fichiers ressources le cas échéant.

Certaines questions sont plus ardues, elles sont identifiées par le pictogramme ★. Ne rebroussez pas chemin pour autant, dans un souci de vous accompagner au mieux dans la résolution des différents problèmes mathématiques proposés, vous trouverez pour la plupart des exercices un paragraphe "**coup de pouce**" à chaque fin de chapitre avec des pistes de résolution. Cela vous permettra ainsi de poursuivre votre démarche et d'appréhender plus aisément la résolution de l'exercice.

Dans tous les cas, l'intégralité des codes Python est disponible en téléchargement en tapant le numéro de l'exercice sur le site de la collection *Créations numériques* : **`http://creations-numeriques.fr`**.

Après avoir terminé votre exercice, prenez le temps de jouer, de flâner et d'imaginer différents essais pour tester la robustesse du code produit. Nous proposons parfois dans les exercices quelques tests pour révéler les erreurs classiques mais pour développer la prise d'initiative et également l'autonomie, nous invitons le lecteur à créer sa propre banque de tests afin de vérifier la cohérence des fonctions codées.

Pour finir sur une note plus technique, nous avons volontairement limité au maximum l'utilisation des **`input`** dans nos exercices en mettant en avant l'utilisation des fonctions. D'une part, cela est cohérent avec l'évolution des programmes officiels et d'autre part cela évite d'avoir à gérer les changements de types assez lourds lorsqu'on découvre le langage. D'ailleurs c'est une démarche à privilégier en informatique :

Programmer, c'est très souvent, écrire des fonctions !

Vincent MAILLE

TABLE DES MATIÈRES

PARTIE 1
EXERCICES

PARTIE 3
FICHES DE COURS

ANNEXES
CONNAISSANCES ET COMPÉTENCES TRAVAILLÉES

Partie 1

EXERCICES

Chapitre A

CALCUL NUMÉRIQUE

1. Pour s'échauffer

A1 *Voici les résultats de 6 participantes de la Parisienne 2023 sur le parcours 10 km ou 7 km .*

Participantes	*Temps*	*Distance (km)*
Carole	0 : 41 : 23	*10*
Camille	0 : 47 : 58	*10*
Fabienne	0 : 33 : 17	*7*
Estelle	0 : 44 : 30	*7*
Claire	1 : 04 : 15	*10*
Iris	0 : 45 : 25	*7*

On souhaite calculer la vitesse moyenne en km/h et le temps en minutes au kilomètre de chacune des participantes. Écrire deux fonctions en Python qui permettent d'automatiser chacun de ces calculs.

A2 *Une* **année-lumière** *est une unité de distance couramment utilisée en astronomie, définie comme la distance parcourue par la lumière en une année, soit 365,25 jours.*

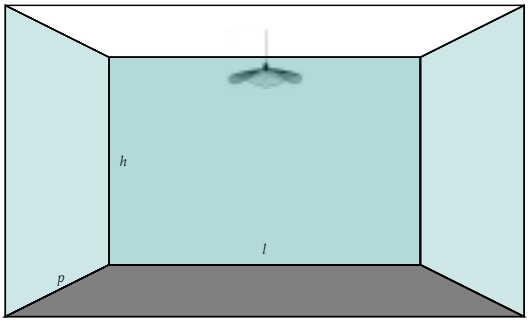
1. *La vitesse de la lumière étant environ égale à 299 792 km par seconde, écrire une fonction qui reçoit un nombre d'années-lumière et renvoie le nombre de kilomètres que cela représente.*

2. *L'étoile la plus proche de la Terre est « Proxima du Centaure ». Elle est située à une distance de la Terre de 4,3 années-lumière. En déduire la distance en km.*

3. *La planète Mars est située en moyenne à* 56×10^6 *km de la Terre. Sachant que la fusée SpaceX peut se déplacer à une vitesse de* 5400 *km/h, calculer le nombre de jours nécessaire pour atteindre Mars.*

4. *Une* ***unité astronomique*** *est la distance moyenne de la Terre au Soleil, soit* $1\ U.A \approx 1,496 \times 10^8$ *km. La distance entre le soleil et Jupiter est environ 778 millions de kilomètres. Déterminer cette distance en U.A. Vérifier la réponse à l'aide d'une fonction Python.*

A3 *Un magasin de bricolage propose une application pour calculer le nombre de pots de peinture nécessaires pour repeindre les 4 murs d'une pièce dont on connaît les dimensions* l, p *et* h *exprimées en mètres et sachant que l'un des murs comporte une fenêtre à 2 vantaux de largeur 1270 mm et de hauteur 1370 mm ainsi qu'une porte de 90 cm de large et 2 mètres de haut.*



1. *Exprimer la surface* S *des 4 murs latéraux de la pièce qui seront peints en tenant compte de la fenêtre.*

2. *Écrire une fonction qui reçoit les dimensions de la pièce en mètres, calcule et renvoie le nombre de pots de peinture nécessaires, sachant que sur le pot de peinture, on peut lire :*

Aspect	*Satin*
Conditionnement (en L)	2.5
Type de peinture	*Peinture acrylique*
Nettoyage des outils	*Eau*
Rendement	$10 m^2$/L
Temps séchage complet (en h)	*24*
Séchage entre 2 couches (en h)	*6*
Classe COV qualité de l'air	*A+*
Nombre de COV (en g/L)	*5*
COV (mg/m^3 d'air) Norme EN717-1	*Inférieur à 1000*

3. *Cette application propose d'adapter le calcul si on souhaite peindre le plafond. Modifier la fonction précédente pour obtenir cela.*

A4 *Éléonore et Fabienne se préparent pour le semi-marathon de Lisbonne. Pour cela, elles souhaitent calculer leur VMA. La VMA (Vitesse Maximale Aérobie) calculée en km/h, est un indicateur qui correspond à la vitesse de course à partir de laquelle la consommation d'oxygène maximale (VO2max) est atteinte. Le test demi-Cooper est le test de calcul VMA le plus utilisé, car il est réalisé sur un temps court et sans avoir besoin de matériel. Il consiste à parcourir la plus grande distance possible sur un temps donné de 6 minutes.*

1. *Créer une fonction en Python qui renvoie la VMA correspondant à une distance d entrée et donnée en mètres pour le test demi-Cooper.*

2. *En 6 minutes, Éléonore a parcouru 1120 mètres et Fabienne 1260 mètres. Donner leur VMA respective.*

3. *À l'aide de la fonction créée, compléter le tableau indiquant dans la première ligne la VMA et dans la deuxième ligne la distance parcourue en mètres pendant les 6 minutes.*

VMA												
distance d	*1330*	*1260*	*1190*	*1120*	*1050*	*980*	*910*	*840*	*770*	*700*	*630*	*560*

4. *Pour un entraînement à un semi marathon, la VMA moyenne conseillée est 85% de la VMA. À quelle vitesse doivent courir Éléonore et Fabienne pour suivre ce conseil ?*

5. *Créer une fonction ayant pour paramètres le % de VMA souhaité, la VMA et la durée t en minutes, qui renvoie la distance parcourue en km.*

6. *Lucie a une VMA de 12 km/h. Elle souhaite courir pendant 10 minutes à 70% VMA, quelle distance doit-elle parcourir ? Sachant que le stade a une piste ayant une longueur de 250 m, combien de tours doit-elle faire environ ?*

7. *Jeanne a une VMA de 14 km/h. Elle souhaite courir 5 minutes à 70% VMA puis 3 minutes à 90% VMA et enfin finir avec 3 minutes à 60% VMA. Quelle distance aura-t-elle parcourue ?*

A5 *Un loueur de véhicules propose la location d'une camionnette au tarif de 46€ la journée pour un forfait de 200 km. Les kilomètres supplémentaires sont facturés 14 centimes du kilomètre. Réaliser une fonction qui reçoit le nombre de kilomètres parcourus et qui renvoie le prix T.T.C. en euros à payer par le client, sachant que les prix affichés sont H.T. et que la T.V.A. s'élève à 20%.*

2. Jouons avec les entiers

A6

1. *Écrire une fonction mettant en jeu un test qui reçoit un entier naturel et renvoie 1 si celui-ci est multiple de 3 et 0 sinon.*

2. *Par un programme de calcul, créer une fonction répondant à la question précédente, sans utiliser l'instruction* `if`.

A7 *Pour effectuer une division euclidienne avec la Pascaline (l'ancêtre de la calculatrice), il fallait procéder par soustractions successives. Ainsi pour faire la division de* 2023 *par* 120, *on cherchait combien de fois successives on pouvait mettre* 120 *dans* 2023 *et on effectuait donc :* 2023 – 120 – 120... *jusqu'à arriver à un nombre inférieur à* 120.

1. *Créer une fonction qui reçoit deux entiers naturels* a *et* b ($b \neq 0$) *et qui effectue les soustractions successives de* b *à* a *tant que le résultat est supérieur à* b *et renvoie le nombre de fois que l'on a soustrait* b.

2. *En déduire le nombre de soustractions possibles de* 120 *à* 2023.

3. *À quelle opération correspond cette démarche ? Justifier.*

4. *En s'inspirant de la fonction précédente, créer une nouvelle fonction qui affiche si un entier* a *est un multiple d'un entier* b.

5. *De la même manière, créer une fonction qui renvoie le plus grand multiple de* b *inférieur ou égal à* a.

La Pascaline au musée des arts et métiers - Source Wikipédia

A8 *On considère le programme de calcul suivant :*

- *Choisir un nombre entier.*
- *Lui retirer 6*
- *Multiplier le résultat par le nombre de départ.*
- *Ajouter 9*

1. *Écrire une fonction qui reçoit l'entier choisi en paramètre, réalise ce programme de calcul et renvoie le résultat.*
2. *Des élèves ont testé cette fonction et proposent quelques conjectures :*
   a. *Le résultat ne se termine jamais par 7.*
   b. *Si n est pair, le résultat est impair et inversement.*
   c. *Le résultat est toujours positif ou nul.*
   d. *Quand le nombre est multiple de 6, le résultat est multiple de 9.*
   e. *Le résultat est un carré parfait.*
   f. *Quand le nombre se termine par un 9, le résultat se termine par un 6.*

   L'une de ces 6 conjectures est fausse, la trouver et démontrer les autres.

A9 *[La numération dans différentes bases]*

1. *Pour convertir un nombre entier n en écriture binaire, on peut utiliser la fonction* **`bin`** *: elle reçoit un entier et renvoie une chaîne de caractères commençant par `'0b'` représentant ce nombre en base 2. Créer une fonction qui reçoit un nombre entier et qui renvoie sous forme d'une chaîne l'écriture* ***en base 2*** *de cet entier. Par exemple :*

```
>>> binaire(6)
'110'
```

2. *Que permet d'afficher la ligne suivante ?*

```
print(int('110010',2))
```

3. *Écrire dans le système décimal, les nombres exprimés dans les bases suivantes, puis vérifier les réponses à l'aide d'un programme sur Python :*
   a. *Le nombre qui s'écrit* ***4444*** *en base 5*
   b. *Le nombre qui s'écrit* ***210210*** *en base 3*
   c. *Le nombre qui s'écrit* ***3450*** *en base 7*
   d. *Le nombre qui s'écrit* ***110011001*** *en base 2*
4. *Que dire des nombres qui se terminent par 0 en base n ?*

3. Programmes de calculs

A10 *On considère l'algorithme suivant :*

```
FONCTION Jeu(x) :
    a ⟵ x + 2
    b ⟵ a²
    c ⟵ (a + 2) × (a − 2)
    SI b < c ALORS :
        RENVOYER "Gagné"
    SINON :
        RENVOYER "Perdu"
FIN FONCTION
```

1. *Programmer cet algorithme en Python.*
2. *Prendre différentes valeurs pour x. Que peut-on conjecturer ?*
3. *Prouver cette conjecture.*

A11 *Que renvoie la fonction suivante lorsqu'on lui donne en arguments des valeurs entières pour n et d ?*

```
def trouve(n, d):
    pas = 1
    x = 0
    for i in range(d+1):
        while x ** 2 <= n:
            x = x + pas
        x = x - pas
        print("Entre", round(x, i), "et", round(x+pas, i))
        pas = pas / 10
    return x
```

A12 *Voici ci-dessous la déclaration d'une fonction* `iota` *en Python définie sur l'ensemble des nombres entiers :*

```
def iota(x) :
    s = x
    for i in range(2*x, 5*x) :
        s = s + i//x
        print(s)
    return s
```

1. *Existe-il une ou plusieurs valeurs de* `x` *pour lesquelles la fonction* `iota(x)` *ne peut être calculée ?*

2. *Recopier cette fonction et écrire un programme qui affiche les valeurs de* `iota(n)` *pour* `n` *entier de l'intervalle* $[-5;5]$. *Que peut-on conjecturer ?*

3. *Démontrer cette conjecture.*

A13 *Résolution d'une équation de type* $f(x) = 0$.

1. *Écrire une fonction qui reçoit un nombre* x *et renvoie un booléen indiquant si ce nombre est solution ou non de l'équation* $x^3 - 4x = 0$. *En résolvant cette équation dans* $\mathbb{R}$, *tester la fonction créée (il y a 3 solutions !).*

Plus généralement, on souhaite résoudre des équations de la forme $f(x) = 0$ *pour différentes fonctions* f.

2. *Compléter la fonction Python* `tableau`. *Elle reçoit en paramètres une fonction, la première et la dernière valeur du tableau et affiche 11 valeurs du tableau réparties avec un pas constant en arrondissant les images au millième :*

```
>>> tableau(f1, 3, 4)
x          f(x)
------------
3.0        9.0
3.1        9.61
3.2        10.24
3.3        10.89
3.4        11.56
3.5        12.25
3.6        12.96
3.7        13.69
3.8        14.44
3.9        15.21
4.0        16.0
```

```
def f1(x):
    return x**2

def f2(x):
    return 6*x + 3

def tableau(f, debut, fin):
    pas = (fin-debut) / ......
    print("x \t f(x)")
    print("------------")
    for i in range( ..... ):
        x = ..... + ..... * i
        print(round(x, ..... ), "\t", round(f(x), ..... ))
```

Quelques remarques sur le code précédent :

- *Vous aurez remarqué qu'une fonction peut à son tour être paramètre d'une autre fonction. C'est le principe de* ***pleine fonctionnalité*** *d'un langage.*
- *Le caractère spécial* `"\t"` *permet de réaliser un "saut de colonne" dans l'affichage et ainsi d'aligner les résultats.*

3. *Déduire de la fonction précédente les solutions des équations suivantes arrondies au centième près dans l'intervalle* $[-3;3]$ *:*

   a. $2x^2 - 6x + 3 = 0$ *b.* $8x^3 + 14x^2 - 2x = 0$

   Pour vérifier, retrouver alors les valeurs exactes des résultats par le calcul.

A14 *Voici un algorithme un peu surprenant qui crée une liste ininterrompue de nombres en partant d'une liste de 3 nombres strictement positifs et en respectant toujours la règle suivante :*

- *Noter les trois derniers nombres*
- *Effectuer le produit des deux derniers éléments*
- *Ajouter 1*
- *Diviser ensuite par le premier des trois nombres*
- *Mettre le résultat à la fin de la liste précédente*
- *Puis recommencer ...*

1. *On suppose que la liste contient au départ les nombres : 3, 4 et 5.*

   a. Vérifier que le nombre suivant obtenu avec cet algorithme est 7.

   b. Programmer une fonction `suivant` *qui reçoit une liste d'au moins 3 nombres et renvoie le nombre suivant obtenu par l'algorithme.*

2. *a. Créer une fonction* `suite(L, n)` *qui reçoit une liste de départ et renvoie une liste de longueur* n *selon l'algorithme présenté.*

   b. Donner les 10 premiers nombres obtenus en partant de la liste initiale (3,4,5).

3. *On choisit maintenant comme liste initiale (1,2,3).*

   a. Donner les 10 premiers nombres obtenus.

   b. Quelle conjecture peut-on réaliser sur l'ensemble des valeurs obtenues ?

4. Aaaah! Des racines !

A15 *Pour* x *et* y *deux nombres positifs, on peut définir différentes moyennes :*

Arithmétique	*Géométrique*	*Quadratique*
$\frac{x+y}{2}$	$\sqrt{xy}$	$\sqrt{\frac{x^2+y^2}{2}}$

1. *Programmer ces 3 fonctions.*

2. *Prendre 20 valeurs aléatoires de x et y et afficher ces trois moyennes. Quelle conjecture peut-on émettre ? La démontrer.*

3. ★ *Créer un programme à l'aide des fonctions précédentes pour trouver tous les nombres distincts à deux chiffres non nuls tels que la moyenne arithmétique de ces deux nombres soit la moyenne géométrique de ces deux nombres lus de droite à gauche.*

4. ★ *Même question avec les moyennes arithmétique et quadratique.*

A16 *[Fractions et racines carrées]*

1. *À l'aide d'un programme, déterminer le plus petit entier naturel n tel que :*

$$\frac{1}{1+\sqrt{2}}+\frac{1}{\sqrt{2}+\sqrt{3}}+\ldots.+\frac{1}{\sqrt{n}+\sqrt{n+1}}\geqslant 100$$

2. *On cherche à retrouver ce résultat sans ordinateur*

   a. *On appelle inverse d'un nombre réel $a \neq 0$, l'unique nombre x tel que $ax = 1$. Si $a = 3-\sqrt{2}$, calculer son inverse et donner une écriture sans radical au dénominateur.*

   b. *Écrire sans radical au dénominateur les inverses des nombres : $\sqrt{5}+\sqrt{4}$, $\sqrt{6}+\sqrt{5}$, $\sqrt{8}+\sqrt{7}$ et $\sqrt{9}+\sqrt{8}$*

   c. *Déterminer l'inverse de $\sqrt{k+1}+\sqrt{k}$ pour tout entier k.*

   d. *En déduire une technique pour trouver la solution à la première question sans l'aide de Python.*

Si vous souhaitez poursuivre votre chemin avec des racines carrées, un thème est dédié spécifiquement au nombre $\sqrt{2}$ dans les pages centrales du livre.

5. Encore quelques petits efforts, avec ces petits défis

A17 *Vincent écrit un livre et souhaite le paginer. Il note que pour l'ensemble du livre, il a fallu 276 chiffres.*

1. a. *Écrire une fonction* `nbChiffres` *qui reçoit un entier naturel* `n` *non nul et renvoie le nombre de chiffres nécessaires à son écriture décimale.*

   b. *En déduire une fonction sur Python qui reçoit un entier naturel* `n` *non nul et renvoie le nombre total de chiffres utilisés pour écrire les nombres de 1 à* `n`.

   c. *En testant différentes valeurs pour* `n`, *déterminer le nombre de pages que possède ce livre.*

2. a. *Écrire une fonction* `nb(chiffre,n)` *qui reçoit un chiffre et un entier naturel non nul* `n` *et qui renvoie le nombre de fois où le chiffre* `chiffre` *est présent dans l'écriture de* `n`.

   b. *Combien de fois le chiffre 8 a-t-il été utilisé pour paginer ce livre ?*

   c. *Combien de fois le chiffre 0 a-t-il été utilisé pour paginer ce livre ?*

3. *Retrouver ces résultats par le calcul.*

A18 *[Conversions de temps].*

1. *Écrire une fonction* `str2min` *qui transforme une chaîne de caractères type "h :mm" ou "hh :mm" en nombre de minutes que cela représente. Par exemple* `str2min('2:12')` *renvoie 132.*

2. *Écrire une fonction* `min2str` *ayant l'effet inverse.*

3. *Utiliser ces deux fonctions pour réaliser un programme qui demande les heures de départ et d'arrivée d'un train sous forme d'une chaîne de caractères et qui affiche le temps du voyage (maximum 24 heures) sous forme "hh :mm".*

4. *À la gare TGV de Roissy Charles de Gaulle, Lucie doit prendre le train pour Strasbourg. Elle monte dans le train à 08 h 09. Elle a prévu d'arriver à 13 h 12. Quelle sera la durée du voyage en train sachant que le contrôleur du train vient d'annoncer un retard de 1 h 10 ?*

A19 *[Fractions diaboliques] - Sur une copie d'élève on peut lire :*

$$A = \frac{1\not{6}}{\not{6}4} = \frac{1}{4}$$

1. *Que penser de cette égalité ?*
2. *Donner toutes les fractions dont le numérateur et le dénominateur comportent 2 chiffres et pour lesquelles ce type de simplification est mathématiquement correct.*

A20

D'après "Les maths au carré" (ed. Ellipses)

Résoudre ce cryptogramme sachant que tous les chiffres sont impairs à l'exception de celui représenté par la lettre O.

$$RIEN + RIEN = ZERO$$

A21

D'après le concours Kangourou junior 2012

On choisit 2 nombres entiers distincts a et b dans l'ensemble $\{1, 2, ..., 26\}$ tels que leur produit est égal à la somme des 24 autres valeurs restantes. Que peuvent valoir ces nombres ? Deux méthodes sont possibles : en résolvant mathématiquement ou à l'aide d'un programme Python.

6. Par ici la monnaie

A22 *Une somme de 910 euros est constituée de billets de 50 euros et de 20 euros. On cherche à déterminer le nombre de billets de chaque sorte sachant qu'on souhaite constituer la somme avec 23 billets au total.*

1. *Première méthode*
   a. *Créer une fonction Python qui admet en paramètres le nombre de billets de 20 euros et de 50 euros et renvoie la somme obtenue.*
   b. *En déduire la réponse à la question.*
2. *Deuxième méthode*
   a. *Traduire l'énoncé par un système de deux équations à deux inconnues.*
   b. *Résoudre le système*
3. *À présent on sait que la somme de 910 euros est constituée toujours de 23 billets au total : des billets de 20 et 50 euros, mais aussi de 10€. Peut-on connaître la répartition de ces billets avec chacune des deux méthodes ?*

A23 *Dans un pays imaginaire, on paie en @, mais il n'existe que des pièces de 5@ et 7@.*

1. *Peut-on payer une somme de 22@ ? de 23@ ? de 24@ ?*
2. *Écrire une fonction qui reçoit une somme entière en paramètre et renvoie un booléen indiquant si cette somme peut être payée avec des pièces de 5@ et 7@.*
3. *Tester toutes les sommes de 1@ à 200@. Que peut-on conjecturer ?*
4. *Démontrer cette conjecture.*

A24 *Dans le système de monnaie britannique d'avant 1971, on trouve des pièces de 1 penny et des pièces de 3, 6, 12, 24 et 30 pence (3 pence = 3×1 penny). Un client ne dispose que de pièces de 3, 6, 24 et 30 pence (en grande quantité).*

1. *Trouver une condition nécessaire et suffisante sur l'entier s pour que le client puisse payer exactement la somme s avec les 4 types de pièces dont il dispose.*
2. *À la main, trouver une décomposition possible de la somme de 51 pence. Trouver ensuite la décomposition qui utilise le moins de pièces possibles.*
3. *Écrire une fonction qui reçoit une somme s et renvoie la liste minimale de pièces réalisant cette somme, la liste vide sera renvoyée si ce n'est pas réalisable.*

7. Aide pour les exercices

Aide pour l'exercice A1 On pourra par exemple créer une fonction qui prend pour paramètres le temps en secondes et la distance parcourue en km et qui renvoie la vitesse en km/h. On peut aussi faire le choix d'entrer la durée en heures, minutes et secondes.

Aide pour l'exercice A6 Pour la seconde question, on peut déjà transformer le nombre de départ en 0, 1 ou 2 en calculant le reste de la division par 3. Reste ensuite à trouver une fonction mathématique f telle que $f(0) = 1$ et $f(1) = f(2) = 0$...

Aide pour l'exercice A8 Pensez qu'un entier n'est pas nécessairement positif !

Aide pour l'exercice A10 Pour la preuve, pensez au calcul littéral ! Exprimez les quantités a, b et c en fonction de x.

Aide pour l'exercice A11 Testez le programme pour $n = 4$ ou 81 ou 2 et des valeurs de d allant de 0 à 5.

Aide pour l'exercice A12

Pour la première question, on peut rechercher les problèmes parmi :

- La fonction ne renvoie rien ?
- La fonction déclenche une erreur ?
- La fonction réalise une boucle infinie ?

Aide pour l'exercice A15 Pour les deux dernières questions, penser que si n est un entier à deux chiffres, les quantités `n//10` et `n%10` représentent respectivement le chiffre des dizaines et des unités de n.

Aide pour l'exercice A17 Pour la fonction `nbChiffres`, pensez qu'à chaque fois que vous divisez un entier par 10, il perd un chiffre.

Aide pour l'exercice A19 Pour le programme, pensez qu'une fraction de la forme $\frac{a|b}{b|c}$ peut s'écrire mathématiquement $\frac{10a+b}{10b+c}$.

Aide pour l'exercice A20 Pour cet exercice vous pouvez choisir la piste mathématique ou la piste informatique selon vos préférences.

Aide pour l'exercice A21 On commence déjà par calculer la somme de tous les nombres $S = 1 + 2 + 3 + ... + 25 + 26$. Ainsi, si on choisit a et b, le produit vaut $a \times b$ et la somme de tous les autres $S - a - b$.

Aide pour l'exercice A24 Pour la dernière question un algorithme "force brute" qui teste toutes les possibilités avec 4 boucles fera l'affaire.

8. Solutions des exercices

Retour sur l'exercice A1 Si on décide d'entrer le temps en secondes et la distance en kilomètres, on peut créer cette fonction :

```
def vitesse(t, d):
   return d / t * 3600

def temps(t, d):
    return 60 / vitesse(t, d)
```

Si on ne souhaite pas effectuer soi-même la conversion, on peut donner le temps en heures, minutes, secondes et traiter la conversion avec Python :

```
def vitesse(h, m, s, d):
    t = h*3600 + m*60 + s
    return d / t * 3600

def temps(h, m, s, d):
    return 60 / vitesse(h, m, s, d)
```

On obtient ces résultats :

Participantes	vitesse (km/h)	temps (min/km)
Carole	14,5	4,14
Camille	12,51	4,80
Fabienne	12,62	4,75
Estelle	9,44	6,36
Claire	9,34	6,43
Iris	9,25	6,49

Retour sur l'exercice A2

1.
La lumière parcourt	299 792 km / s
soit	17 987 520 km / min
soit	1 079 251 200 km / h
soit	25 902 028 800 km / jour
soit	9 460 716 019 200 km / an

Voici donc une solution :

```
def conversion(a) :
    """
    En entrée : a est un nombre d'années lumières
    En sortie : l'équivalent en kilomètres
    """
    return 9460716019200 * a
```

2. Avec la fonction de la question 1, on obtient 40681078882560.0 km.
3. La fusée mettra environ 10370,37 heures pour atteindre Mars, soit environ 432 jours.
4. La distance entre Jupiter et le soleil est d'environ 5,24 U.A.

```
def conversion_UA(a) :
    """
    En entrée : a est un nombre de km
    En sortie : l'équivalent en unité astronomique
    """
    return a/(1.496*10**8)
```

Retour sur l'exercice A3

1. Deux murs latéraux mesurent $l \times h$ et les deux autres $p \times h$. Ainsi $S = 2(l \times h + p \times h) = 2(l + p)h$. Or, on doit tenir compte de la fenêtre dont l'aire est $1,7399$ m^2 et de la porte de $2 \times 0,9 = 1,8$ m^2.
   Donc $S = 2(l + p)h - 3,8299$.

2. Comme on peut peindre 10 m^2 avec un litre de peinture, il faut un pot par tranche de 25 m^2. Il suffit donc de diviser la surface par 25 et d'ajouter 1 pour obtenir le nombre de pots utiles.

```
def pots(h, l, p):
    S = 2 * (l + p) * h - 3.8299
    if S % 25 == 0 : # Exactement un nombre entier de pots
        n = S // 25
    else : # Sinon, il faut en prévoir un de plus
        n = S // 25 + 1
    return n
```

Une alternative avec la fonction **`ceil`** du module `math` qui arrondit un réel à l'entier supérieur le plus proche.

```
from math import ceil
def pots(h, l, p):
    S = 2 * (l + p) * h - 3.8299
    return ceil(S/25)
```

On a fait le choix ici de proposer des dimensions de fenêtre et de porte fixes, il est tout à fait envisageable d'entrer en paramètres ces dimensions et d'adapter les lignes de codes.
Si vous souhaitez ajouter une interface de saisie :

```
h = float(input("Entrer la hauteur"))
l = float(input("Entrer la longueur"))
p = float(input("Entrer la profondeur"))
nb = pots(h, l, p)
print("Il faut prévoir", nb, "pots pour cette surface.")
```

3. Pour cette dernière question, seul le calcul de la surface change :

```
def pots(h, l, p):
    S = 2 * (l + p) * h - 3.8299 + l * p
    if S % 25 == 0 : # Exactement un nombre entier de pots
        n = S // 25
    else : # Sinon, il faut en prévoir un de plus
        n = S // 25 + 1
    return n
```

Retour sur l'exercice A4

1.
```
def VMA(d):
        return (d*60) / (6*1000)
```

   Si on est perspicace, on remarquera que cela revient à diviser par 100, ce qui est bien pratique pour faire les calculs de tête !

2. La VMA d'Éléonore est 11,2 km/h et celle de Fabienne est 12,6 km/h.

3. Avec la remarque précédente, on trouve :

VMA	13.3	12.6	11.9	11.2	10.5	9.8	9.1	8.4	7.77	7	6.3	5.6
distance d	1330	1260	1190	1120	1050	980	910	840	770	700	630	560

4. Éléonore doit courir à $0,85 \times 11,2$ soit 9,52 km/h et Fabienne doit courir à 10,71 km/h.

5.
```
def distance(p, v, t):
        return p * v * t / 60
```

6. En utilisant la fonction précédente, Lucie doit parcourir environ 1,4 km soit 5,6 tours de stade.

7. On applique la fonction précédente, puis on effectue le calcul : $0,82+0,63+0,42=1,87$. Elle aura parcouru une distance de $1,87$ km environ.

Retour sur l'exercice A5 On peut procéder ainsi : fixer le montant total t à 46 € et l'augmenter en cas de dépassement :

```
def prix(n) :
    """
    En entrée : n est le nombre de km parcourus
    En sortie : le prix à payer
    """
    t = 46          # Prix de départ H.T
    if n > 200:     # Si on dépasse les 200 km
        t = t + (n - 200) * 0.14
    t = t * 1.2     # Prix TTC
    return t
```

Avec la fonction `max`, on peut raccourcir le code en calculant directement le prix H.T. à payer :

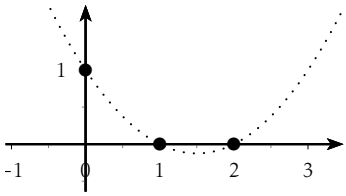
```
def prix(n) :
    t = 46 + max(0, n - 200) * 0.14
    return t * 1.2        # Prix TTC
```

Retour sur l'exercice A6

1. La première question peut être traitée avec un `if` assez facilement en testant la valeur du reste de la division par 3 :

```
def multiple(a):
    if a % 3 == 0:
        return 1
    else:    # Ce else est facultatif
        return 0
```

2. Pour cette question, si a est la variable de départ, `a % 3` donne le reste de la division de a par 3 qui peut valoir 0, 1 ou 2 (le reste d'une division dans $\mathbb{N}$ est toujours supérieur ou égal à 0 et strictement inférieur au diviseur). Il s'agit donc de trouver une fonction f qui vérifie : $\begin{cases} 0 \overset{f}{\longmapsto} 1 \\ 1 \overset{f}{\longmapsto} 0 \\ 2 \overset{f}{\longmapsto} 0 \end{cases}$



Chercher une fonction f sous la forme d'un polynôme du second degré semble assez pertinent. Les deux dernières conditions nous informent que 1 et 2 sont des racines de f. Donc $f(x) = a(x-1)(x-2)$. Enfin $f(0) = 1 \iff a \times (-1) \times (-2) = 1 \iff 2a = 1 \iff a = \dfrac{1}{2}$. Ainsi

$$f(x) = \frac{(x-1)(x-2)}{2}.$$

Finalement, voici un programme répondant à la question :

```
def multiple(a):
    a = a % 3
    a = (a - 1) * (a - 2) // 2
    return a
```

Bien d'autres solutions sont possibles. En voici une autre proposée par une élève (en notant a le nombre de départ) :

$a\%3$	$(a\%3)^2$	$(a\%3)^2\%3$	$1-\left[(a\%3)^2\%3\right]$
0	0	0	1
1	1	1	0
2	4	1	0

Encore une autre solution possible,

```
(1 - a ** 2) % 3
```

à démontrer en exercice si vous le souhaitez.

Retour sur l'exercice A7

1. Il suffit de traduire la procédure indiquée :

```
def soustraction(a, b):
    n = 0
    reste = a
    while reste >= b:
        n = n + 1
        reste = reste - b
    return n
```

2. La fonction renvoie 16.

3. Vous aurez sûrement reconnu le principe de la division euclidienne ! La fonction renvoie le quotient entier de la division de a par b.

4. On adapte le code précédent en regardant le reste après avoir retiré b un maximum de fois :

```
def multiple(a, b):
    reste = a - b
    while reste > 0:  # on travaille ici avec des entiers naturels
        reste = reste - b
    if reste == 0:
        print(a, " est un multiple de", b)
    else:
        print(a, " n'est pas un multiple de", b)
```

On pourrait aussi utiliser directement le reste de la division euclidienne sans utiliser alors le principe de la Pascaline :

```
def multiple(a, b):
    if  a %b == 0:
        print(a, "est un multiple de", b)
    else:
        print(a, "n'est pas un multiple de", b)
```

AU DÉTOUR DE LA BALADE...

Avec la Pascaline, une question se pose : combien de soustractions ont été effectuées et donc que vaut le quotient ?

Pour ne pas oublier le nombre de soustractions effectuées , Pascal a pensé à des rondelles de mémoire qui équipent certaines Pascalines : ce sont de petits cadrans à aiguilles placés auprès des lucarnes.

5. En utilisant le même principe, tant que l'on peut retirer b de a on effectue le calcul et en même temps on accumule les valeurs de b :

```
def multiple_inf(a, b):
    reponse = 0
    while a >= b :
        a = a - b
        reponse = reponse + b
    return reponse
```

Retour sur l'exercice A8

1. Cette fonction se programme facilement :

```
def programme(n) :
    return (n-6) * n + 9
```

2.

AU DÉTOUR DE LA BALADE...

Cet exercice peut être résolu avec des démarches différentes. Les élèves peuvent étudier chaque conjecture séparément sans remarquer de lien entre elles. Les plus à l'aise avec le calcul littéral, reconnaîtront une identité remarquable et pourront démontrer plus rapidement certaines conjectures en réfléchissant à l'ordre des preuves en amont.

b. Si n est pair, le résultat est impair et inversement. **La conjecture est vraie**.

Si n est pair , alors $n - 6$ est pair et le produit de deux nombres pairs est pair. Comme 9 est impair, la somme d'un nombre pair et d'un nombre impair est impair. On procède de même pour l'autre cas.

c. Le résultat est toujours positif ou nul. **La conjecture est vraie**. On remarquera que $(n-6)n+9 = n^2-6n+9 = (n-3)^2$.

a. Le résultat est un carré, or le chiffre des unités du carré d'un entier ne se déduit que du chiffre des unités de celui-ci. Par disjonction de cas :

Chiffre des unités de x	0	1	2	3	4	5	6	7	8	9
Chiffre des unités de x^2	0	1	4	9	6	5	6	9	4	1

Le résultat ne se termine jamais par 7. **La conjecture est vraie**.

d. Quand le nombre est multiple de 6, le résultat est multiple de 9. **La conjecture est vraie**.
Si n est un multiple de 6, alors il existe un entier relatif k tel que $n = 6k$. On en déduit que $(n-3)^2 = (6k-3)^2 = 9(2k-1)^2$ qui est bien un multiple de 9.

e. Le résultat est un carré parfait. **La conjecture est vraie**.
On reconnaît $(n-3)^2$.

f. Quand le nombre se termine par un 9, le résultat se termine par un 6. **La conjecture est fausse**. On peut prendre un contre-exemple. Si $n = -9$ alors le programme renvoie 144 qui ne se termine pas par 6. (Elle est cependant vraie pour les entiers naturels, vous pouvez le démontrer en exercice).

Retour sur l'exercice A9

1. Il suffit juste de retirer les 2 premiers caractères du résultat donné par la fonction `bin`... Ce qui n'est pas si simple au final. On peut par exemple récupérer un à un les caractères de la chaîne en question et les concaténer :

```
def binaire(n):
    b = bin(n)
    reponse = ""
    for i in range(2, len(b)) :
        reponse = reponse + b[i]
    return reponse
```

Pour information, on peut gagner du temps en utilisant le **slicing** qui consiste, comme le nom le laisse penser, à extraire un morceau de la chaîne de caractères : ch`[`i`:`j`]` renvoie la sous-chaîne de caractères de ch contenant seulement les caractères d'indices i (inclus) à j (exclu).

Ainsi on peut simplifier la fonction :

```
def binaire(n):
    b = bin(n)
    return b[2:len(b)]
```

Si on souhaite extraire une chaîne jusqu'à la fin, on peut même ne pas mettre le second indice :

```
def binaire(n):
    b = bin(n)
    return b[2:]
```

2. Cet appel produit l'effet inverse de la fonction précédente : on entre une chaîne de caractères représentant l'écriture binaire d'un nombre et cela affiche son écriture décimale.

3. Par le calcul :

   - $\overline{4444}^{5} = 4 + 4 \times 5 + 4 \times 5^2 + 4 \times 5^3 = 624$
   - $\overline{210210}^{3} = 588$
   - $\overline{3450}^{7} = 1260$
   - $\overline{110011001}^{2} = 409$

   Que l'on peut vérifier avec Python directement dans la console :

```
>>> int('4444',5)
624
>>> int('210210', 3)
588
```

```
>>> int('3450', 7)
1260
>>> int('110011001', 2)
409
```

4. Après quelques essais, on conjecture qu'un nombre qui se termine par 0 en base n est toujours multiple de n. Pour $n = 10$, vous le savez depuis longtemps, dans le cas général,

   si un nombre A s'écrit en base n : $A = \overline{a_p a_{p-1} \ldots a_1 a_0}^{n}$ alors

$$A = \sum_{j=0}^{p} a_j \times n^j = a_0 + \sum_{j=1}^{p} a_j \times n^j = a_0 + \sum_{j=0}^{p-1} a_{j+1} \times n^{j+1} = a_0 + n \sum_{j=0}^{p-1} a_{j+1} \times n^j$$

   Ainsi, si a_0 vaut 0, $A = n \times \underbrace{\sum_{j=0}^{p-1} a_{j+1} \times n^j}_{\in \mathbb{N}}$ est un multiple de n.

AU DÉTOUR DE LA BALADE...

Pour tester si le dernier argument a bien été compris, on pourrait prolonger par la question : « Que dire des nombres se terminant par 00 dans leur écriture en base n ? ».

Retour sur l'exercice A10

1. Voici une fonction Python répondant à la question :

```
def jeu(x):
    a = x + 2
    b = a ** 2
    c = (a+2) * (a-2)
    if b < c:
        return "Gagné"
    else:
        return "Perdu"
```

2. C'est un jeu où l'on perd tout le temps !

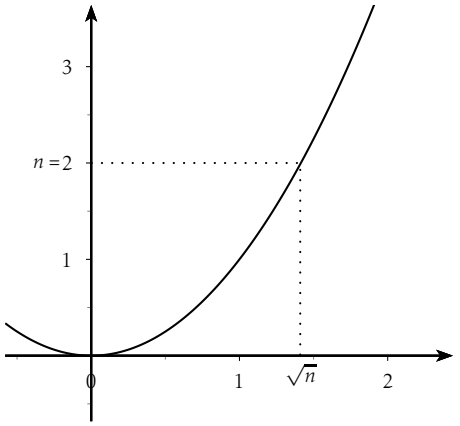
3. En effet, on sait que $\begin{cases} a = x+2 \\ b = a^2 = (x+2)^2 = x^2+4x+4 \\ c = (a+2)\times(a-2) = (x+4)\times x = x^2+4x \end{cases}$.

   Étudions la condition pour gagner :

$$b < c \iff x^2+4x+4 < x^2+4x \iff 4 < 0$$

   ce qui ne se produit jamais !

Retour sur l'exercice A11 Ce programme donne un encadrement du nombre $\sqrt{n}$ avec une amplitude de 10^{-d}. Regardons pas à pas ce qui se passe pour $n = 2$ et $d = 3$:



- $i = 0$:

pas	x	x^2	$x^2 \leqslant 2$?
1	0	0	OUI
1	1	1	OUI
1	2	4	NON

- $i = 1$:

pas	x	x^2	$x^2 \leqslant 2$?
0,1	1	1	OUI
0,1	1,1	1,21	OUI
0,1	1,2	1,44	OUI
0,1	1,3	1,69	OUI
0,1	1,4	1,96	OUI
0,1	1,5	2,25	NON

- ...

AU DÉTOUR DE LA BALADE...

On peut penser que le `round` *est inutile dans le* `print`*, mais si on remplace*

```
print("Entre", round(x, i), "et", round(x+pas, i))
```

par

```
print("Entre", x, "et", x+pas)
```

on constate que l'affichage ne donne pas des valeurs exactes, à cause des erreurs d'arrondi.

Retour sur l'exercice A12

1. La fonction réalise une boucle `for` et ne peut donc pas entrer dans une boucle infinie. D'autre part, la fonction se termine bien par un `return` donc renverra la valeur de `s` qui est bien initialisée à la première ligne. Le seul problème qui peut être rencontré est une erreur lors des calculs... or seule la division peut poser problème. Que se passe-t-il si on appelle `iota(0)` ?

   La variable i va parcourir tous les entiers de $[\![2\times 0;5\times 0[\![=[\![0;0[\![$ qui est vide, donc on n'entre pas dans la boucle `for`, il n'y a donc pas de problème de division dans ce cas non plus et $iota(0)$ renvoie 0.

2. En réalisant une boucle, on obtient le tableau de valeurs ci-dessous :

x	-5	-4	-3	-2	-1	0	1	2	3	4	5
`iota`(x)	-5	-4	-3	-2	-1	0	10	20	30	40	50

   On peut donc conjecturer que : `iota`$(x)=\begin{cases} x \text{ si } x\leqslant 0 \\ 10x \text{ sinon.}\end{cases}$

3. Si $x\leqslant 0$, alors $2x\leqslant 5x$ et donc on n'entre pas dans la boucle, la fonction renvoie x.

   Si $x>0$, notons $\mathrm{E}(u)$ la partie entière d'un réel u, à la fin de la fonction on a :

$$s=x+\mathrm{E}\left(\frac{2x}{x}\right)+\mathrm{E}\left(\frac{2x+1}{x}\right)+\ldots+\mathrm{E}\left(\frac{5x-1}{x}\right)$$

$$=x+\sum_{k=2x}^{5x-1}\mathrm{E}\left(\frac{k}{x}\right) \text{ que l'on sépare en 3 sommes}$$

$$=x+\sum_{k=2x}^{3x-1}\mathrm{E}\left(\frac{k}{x}\right)+\sum_{k=3x}^{4x-1}\mathrm{E}\left(\frac{k}{x}\right)+\sum_{k=4x}^{5x-1}\mathrm{E}\left(\frac{k}{x}\right)$$

Or,

- si $2x \leqslant k \leqslant 3x-1$ alors $\mathrm{E}\left(\frac{k}{x}\right) = 2$ et il y a x valeurs de k possibles ;
- si $3x \leqslant k \leqslant 4x-1$ alors $\mathrm{E}\left(\frac{k}{x}\right) = 3$ et il y a x valeurs de k possibles ;
- si $4x \leqslant k \leqslant 5x-1$ alors $\mathrm{E}\left(\frac{k}{x}\right) = 4$ et il y a x valeurs de k possibles ;

Finalement $s = x + 2 \times x + 3 \times x + 4 \times x = 10x$.

Retour sur l'exercice A13

1. Pour l'instant, on ne demande pas à la machine de résoudre mais juste de tester, la fonction pourrait ressembler à celle-ci (la fonction renvoie un booléen, il est donc possible de voir la fiche sur les types si nécessaire) :

```
def solution(x) :
    if x **3 - 4 * x == 0:
        return True
    else :
        return False
```

ou en version plus compacte :

```
def solution(x) :
    return x ** 3 - 4 * x == 0
```

En effet, l'expression `x ** 3 - 4 * x == 0` est déjà un booléen répondant à la question. Vous pouvez alors tester votre code avec les solutions que vous avez trouvées (qui devraient être 0, -2 et 2). En effet :

$$x^3 - 4x = 0 \iff x(x^2-4) = 0 \iff x = 0 \text{ ou } x = -2 \text{ ou } x = 2$$

2. On peut compléter ainsi :

```
def tableau(f, debut, fin):
   pas = (fin-debut) / 10    # 11 valeurs soit 10 intervalles
   print("x \t f(x)")
   print("------------")
   for i in range(11):
      x = debut + pas * i
      print(round(x, 3), "\t", round(f(x), 3))
```

3. On peut tester et par encadrements successifs trouver les valeurs approchées des solutions. Au niveau de la résolution mathématique, on trouve :

   a. $2x^2 - 6x + 3 = 0 \iff x = \dfrac{3 \pm \sqrt{3}}{2}$

   b. $8x^3 + 14x^2 - 2x = 0 \iff x \in \left\{\dfrac{-7-\sqrt{65}}{8}; 0; \dfrac{-7+\sqrt{65}}{8}\right\}$

On peut créer une **fonction anonyme** en Python, c'est à dire que l'on crée un objet de type fonction sans pour autant le définir par un nom avec `def`. Pour cela on utilise l'instruction **`lambda`**. Par exemple pour créer le tableau de variation sur $[-3, 3]$ de la fonction $x \mapsto 3x - 2$, on peut saisir dans la console :

```
tableau(lambda x : 3*x-2, -3, 3)
```

Retour sur l'exercice A14

1. a. $\dfrac{4 \times 5 + 1}{3} = 7$

   b. Si n est le nombre d'éléments de la liste `L`, le dernier élément se trouve en position $n - 1$, ainsi voici une solution :

```
def suivant(L) :
    n = len(L)
    return (L[n-1] * L[n-2] + 1) / L[n-3]
```

À noter que l'on peut aussi atteindre le dernier élément d'une liste avec l'appel à `L[-1]` (retenez que si l'indice est négatif, on part de la fin). Idem pour l'avant dernier avec `L[-2]`.... Ainsi, on peut raccourcir le code et écrire :

```
def suivant(L) :
    return (L[-1] * L[-2] + 1) / L[-3]
```

2. a. Il suffit d'appeler la fonction `suivant` jusqu'à obtenir une liste de longueur `n` :

```
def suite(L, n) :
    while len(L) < n :
        L.append(suivant(L))
    return L
```

b. Avec notre fonction, on trouve :

```
>>> suite([3, 4, 5], 10)
[3, 4, 5, 7.0, 9.0, 12.8, 16.6, 23.720000000000002, ↙
    ↳ 30.840000000000003, 44.12800000000001]
```

Soit sous forme exacte : $3,\ 4,\ 5,\ 7,\ 9,\ \frac{64}{5},\ \frac{83}{5},\ \frac{593}{25},\ \frac{771}{25},\ \frac{5516}{125}$

3. On choisit (1,2,3)

a. On obtient les nombres suivants :

```
>>> suite([1,2,3], 10)
[1, 2, 3, 7.0, 11.0, 26.0, 41.0, 97.0, 153.0, 362.0]
```

b. Les nombres obtenus semblent tous entiers. La preuve est cependant loin d'être triviale, on a :

n	u_{n+2}	u_n	u_{n-2}
2	11	3	1
3	26	7	2
4	41	11	3
5	97	26	7
6	153	41	11
7	362	97	26

Il semblerait que $\forall n \geqslant 2,\ u_{n+2} = 4u_n - u_{n-2}$, ce qui permettrait de conclure que tous les termes sont bien entiers. Malheureusement cette nouvelle conjecture n'est pas simple non plus à démontrer !

Retour sur l'exercice A15

1.

```
from math import *

def arithmetique(x, y):
    return (x + y) / 2

def geometrique(x, y):
    return sqrt(x*y)

def quadratique(x, y):
    return sqrt((x**2 + y**2) / 2)
```

2. En ajoutant ces quelques lignes, on peut tester rapidement beaucoup de valeurs pour x et y :

```
from random import *
for i in range(20):
    x = uniform(0, 100)
    y = uniform(0, 100)
    a = arithmetique(x, y)
    g = geometrique(x, y)
    q = quadratique(x, y)
    print(x, y, a, g, q)
```

Voici quelques exemples de valeurs que l'on peut obtenir :

x	y	arithmétique	géométrique	quadratique
8.371	57.511	32.941	21.942	41.095
9.78	12.618	11.199	11.109	11.289
85.561	13.145	49.353	33.536	61.211
60.888	34.166	47.527	45.61	49.369

On peut conjecturer que la moyenne géométrique est inférieure à la moyenne arithmétique, elle-même inférieure à la moyenne quadratique...

En effet, quels que soient les réels positifs x et y :

- $\sqrt{xy} \leqslant \dfrac{x+y}{2}$
  $\iff 2\sqrt{xy} \leqslant x+y$
  $\iff 2\sqrt{xy} \leqslant x+y$, comme $x \geqslant 0$ et $y \geqslant 0$, en élevant au carré :
  $\iff 4xy \leqslant x^2+2xy+y^2$
  $\iff 0 \leqslant x^2-2xy+y^2$
  $\iff 0 \leqslant (x-y)^2$. Ce qui est toujours vrai.
- $\dfrac{x+y}{2} \leqslant \sqrt{\dfrac{x^2+y^2}{2}}$
  $\iff x+y \leqslant 2\sqrt{\dfrac{x^2+y^2}{2}}$
  $\iff x^2+2xy+y^2 \leqslant 4\dfrac{x^2+y^2}{2}$
  $\iff 0 \leqslant x^2-2xy+y^2$
  $\iff 0 \leqslant (x-y)^2$. Ce qui est aussi toujours vrai.

3. Voici le principe que nous allons utiliser :

   - Prendre $10 \leqslant x < y \leqslant 99$
   - Calculer la moyenne arithmétique, regarder si elle est entière.
   - Si oui, extraire le chiffre des dizaines et celui des unités, puis comparer le nombre "miroir" avec la moyenne géométrique :

```
for x in range(10, 100):
    for y in range(x+1, 101):
        a = arithmetique(x, y)
        if round(a) == a:
            d, u = a // 10, a % 10
            if u * 10 + d == geometrique(x, y):
                print(x, y, a, geometrique(x, y))
```

4. On peut recommencer le même programme en remplaçant la fonction `geometrique` par `quadratique`. En résumé, on trouve :

x	y	arithmétique	géométrique	quadratique
32	98	65	56	-
23	89	56	-	65

Admirez les symétries dans ces résultats !

Retour sur l'exercice A16

1. À l'aide d'une boucle et d'un accumulateur :

```
from math import sqrt

somme = 0
n = 0
while somme < 100:
    n = n + 1
    somme = somme + 1/(sqrt(n)+sqrt(n+1))
print(n, somme)
```

2. a. L'inverse de a est $\dfrac{1}{a} = \dfrac{1}{3-\sqrt{2}} = \dfrac{3+\sqrt{2}}{7}$

   b. $\dfrac{1}{\sqrt{5}+\sqrt{4}} = \sqrt{5}-\sqrt{4}$, $\dfrac{1}{\sqrt{6}+\sqrt{5}} = \sqrt{6}-\sqrt{5}$
   $\dfrac{1}{\sqrt{8}+\sqrt{7}} = \sqrt{8}-\sqrt{7}$, $\dfrac{1}{\sqrt{9}+\sqrt{8}} = \sqrt{9}-\sqrt{8}$

   c. Pour tout entier k, $\dfrac{1}{\sqrt{k+1}+\sqrt{k}} = \sqrt{k+1}-\sqrt{k}$

d. D'après les questions précédentes, on remarque que

$$\frac{1}{1+\sqrt{2}}+\frac{1}{\sqrt{2}+\sqrt{3}}+....+\frac{1}{\sqrt{n}+\sqrt{n+1}}=-1+\sqrt{n+1}$$

Finalement, $\dfrac{1}{1+\sqrt{2}}+\dfrac{1}{\sqrt{2}+\sqrt{3}}+....+\dfrac{1}{\sqrt{n}+\sqrt{n+1}}\geqslant 100$

$\iff -1+\sqrt{n+1}\geqslant 100$

$\iff \sqrt{n+1}\geqslant 101$

$\iff n\geqslant 101^2-1=10200$

AU DÉTOUR DE LA BALADE...

Si on cherche informatiquement le rang à partir duquel cette somme dépasse 7000, une différence existe entre la valeur théorique et celle trouvée avec Python, cela est dû aux erreurs d'arrondis (représentation des nombres réels sur machine).

Retour sur l'exercice A17

1. a. Beaucoup de possibilités pour cette question assez ouverte :

- Idée 1 : Solution itérative car en divisant par 10, on retire les chiffres de n un à un, et parallèlement on compte le nombre de chiffres :

```
def nbChiffres(n) :
    nb = 0
    while n != 0 :
        n = n // 10      # On retire un chiffre
        nb = nb + 1      # On compte un chiffre en plus
    return nb
```

- Idée 2 : La même solution mais écrite de manière récursive :

```
def nbChiffres(n) :
    if n <= 9 :
        return 1
    else :
        return 1 + nbChiffres(n//10)
```

- Idée 3 : En transformant le nombre en chaîne de caractères et en regardant sa longueur (oui pour le matheux, c'est une triche "honteuse", et pourtant très efficace du point de vue informatique !) :

```
def nbChiffres(n) :
    return len(str(n))
```

AU DÉTOUR DE LA BALADE...

Si vous connaissez le logarithme en base 10, vous pouvez aussi l'utiliser :

```
from math import log10
def nbChiffres(n) :
    return 1 + int(log10(n))
```

Notez enfin que si l'on donne 0 en argument toutes ces solutions ne donnent pas le même résultat (ce qui n'est pas gênant ici).

b. À l'aide de la fonction précédente, on crée facilement la fonction demandée :

```
def pages(n) :
    total = 0
    for i in range(1, n+1) :
        total = total + nbChiffres(i)
    return total
```

c. Une boucle TANT QUE permet de répondre à la question :

```
p = 1
while pages(p) != 276 :
    p = p +1
print(p)
```

On trouve alors la réponse :

```
128
```

2. a. En s'inspirant de la première solution par exemple, on compte le nombre de fois où un chiffre apparaît dans un nombre :

```
def nb(chiffre, n):
    nbc = 0
    while n > 0:
        if n % 10 == chiffre:
            nbc = nbc + 1
        n = n // 10
    return nbc
```

b. On réalise alors une fonction qui accumule les résultats :

```
def nbp(chiffre, n):
    nbc = 0
    for i in range(1, n+1):
        nbc = nbc + nb(chiffre, i)
    return nbc
```

On réalise le bon appel :

```
>>> nbp(8, 128)
23
```

c. De la même manière :

```
>>> nbp(0, 128)
22
```

3. Pour le nombre de pages, un encadrement rapide du nombre de pages, nous permet de travailler entre 100 et 999 pages (inclus). Le nombre de chiffres est

$$\mathcal{N} = \underbrace{1\times 9}_{\text{de 1 à 9}} + \underbrace{2\times 90}_{\text{de 10 à 99}} + \underbrace{3\times(n-99)}_{\text{de 100 à } n}$$

Finalement,

$$\mathcal{N} = 276 \Longleftrightarrow 1\times 9 + 2\times 90 + 3\times(n-99) = 276 \Longleftrightarrow n = 128$$

AU DÉTOUR DE LA BALADE...

Pour ceux qui débutent en Python, on peut d'ailleurs créer une fonction qui renvoie le nombre de chiffres utilisés en fonction du nombre de pages (< 1000) :

```
def pages(n):
    if n < 10:
        return n
    elif n < 100:
        return 9 + (n-9)*2
    if n < 1000:
        return 9 + 2*90 + 3*(n-99)
```

Pour le nombre de 8, on trouve :

$$\mathcal{N}_8 = \underbrace{1}_{\text{de 1 à 9}} + \underbrace{9}_{\text{unités de 10 à 99}} + \underbrace{10}_{\text{dizaines de 80 à 89}} + \underbrace{3}_{\text{108,118,128}} = 23$$

Et pour 0 :

$$\mathcal{N}_0 = \underbrace{0}_{\text{de 1 à 9}} + \underbrace{12}_{\text{unités de 10 à 123}} + \underbrace{10}_{\text{dizaines de 100 à 109}} = 22$$

Retour sur l'exercice A18

1. Pour cette fonction, il n'y a pas de difficulté, il faut néanmoins distinguer les cas où l'heure comporte 1 ou 2 chiffres.

```
def str2min(txt):
    if txt[1] == ':':    # Si c'est au format h:mm
        return int(txt[0])*60 + int(txt[2]+txt[3])
    else:
        return int(txt[0]+txt[1])*60 + int(txt[3]+txt[4])
```

2. Pour le problème inverse, pas de difficulté non plus, il faut simplement penser à ajouter un "0" si le nombre de minutes ne comporte qu'un chiffre.

```
def min2str(minutes):
    h = minutes // 60
    m = minutes % 60
    if m < 10:
        return str(h) + ":0" + str(m)
    else:
        return str(h) + ":" + str(m)
```


Remarque : Au lieu d'écrire en deux lignes

```
h = minutes // 60
m = minutes % 60
```

on peut écrire en une fois à l'aide de la fonction **divmod** de cette manière :

```
h, m = divmod(minutes, 60)
```

3. Pour le programme principal, on utilise les deux fonctions précédemment créées. Si l'heure d'arrivée est inférieure à l'heure de départ, il faut ajouter une journée, ou plus exactement l'équivalent en minutes de 24 heures.

```
t1 = input("Heure de départ : ")
t2 = input("Heure d'arrivée : ")
temps = str2min(t2) - str2min(t1)
if temps <= 0:
    temps = temps + 24*60
print("Durée du trajet :", min2str(temps))
```

4. La durée du voyage est de 6 h 13 min.

Retour sur l'exercice A19

1. L'égalité est vraie... même si le raisonnement est faux.

2. Pour trouver toutes les solutions on peut avoir recours à un petit algorithme cherchant toutes les fractions $\frac{a|b}{b|c}$ pouvant se simplifier en $\frac{a}{c}$. Pour éviter les divisions qui risquent de donner des valeurs approchées à cause des nombres flottants, on peut remarquer que :

$$\frac{a|b}{b|c} = \frac{a}{c} \iff \frac{10a+b}{10b+c} = \frac{a}{c} \iff (10a+b)\times c = (10b+c)\times a$$

Ainsi, on peut réaliser le programme suivant :

```
for a in range(1, 10):
    for b in range(1, 10):
        for c in range(1, 10):
            if (10*a+b) * c == (10*b+c) * a:
                print(a, b, "/", b, c, "=", a, "/", c)
```

On trouve 13 solutions dont 9 sont triviales comme $\frac{33}{33} = \frac{3}{3}$. On peut améliorer le programme en imposant que $a \neq b$:

```
for a in range(1, 10):
    for b in range(1, 10):
        for c in range(1, 10):
            if (10*a+b) * c == (10*b+c) * a and a != b:
                print(a, b, "/", b, c, "=", a, "/", c, sep="")
```

Cette fois-ci on trouve 4 cas possibles :

$$\frac{16}{64} = \frac{1}{4}, \quad \frac{19}{95} = \frac{1}{5}, \quad \frac{26}{65} = \frac{2}{5} \quad \text{et} \quad \frac{49}{98} = \frac{4}{8}.$$

Retour sur l'exercice A20 Voici un cheminement permettant d'orienter les recherches pour aboutir à la solution :

- Comme Z, E et R sont impairs, les chiffres représentés par I, E et N doivent être supérieurs à 5 pour entraîner une retenue. Donc $\{I;E;N\} \subset \{5;7;9\}$.
- Inversement, R ne doit pas être trop grand pour que le résultat soit à 4 chiffres, donc $R \in \{1;3\}$.

- Des valeurs de R, on déduit que Z vaut soit 3 soit 7.
- Or $\{I;E;N\} \subset \{5;7;9\}$ sont des valeurs distinctes (3 inconnues pour 3 valeurs). Donc $\{I;E;N\} = \{5;7;9\}$. On déduit que Z ne peut valoir 7. D'où **Z=3**.
- Il suit **R=1**.
- Si I valait 5, alors E vaudrait 1 ce qui est impossible. De même, si I valait 9, on aurait E qui vaudrait aussi 9. Il reste **I=7**. Par suite **E=5** et **N=9**.
- Finalement : **1759+1759=3518**.

Après cette récréation mathématique, on peut s'intéresser à un programme répondant à la question en tenant compte du fait que :

$$\texttt{RIEN} + \texttt{RIEN} = 2 \times \texttt{RIEN}.$$

```
for R in range(1, 10, 2):
   for I in range(1, 10, 2):
      for E in range(1, 10, 2):
         for N in range(1, 10, 2):
            for Z in range(1, 10, 2):
               for O in range(0, 10, 2):
                  if 2*(R*1000+I*100+E*10+N) == Z*1000+E*100+R*10+O:
                     print(R, I, E, N, " x 2 = ", Z, E, R, O, sep="")
```

Ce programme donne 3 solutions, dont une seule ne comporte que des valeurs distinctes pour chaque lettre.

Retour sur l'exercice A21 Comme proposé dans l'aide, on calcule déjà la somme $S = 1 + 2 + 3 + ... + 25 + 26$ avec une boucle POUR.

```
s = 0
for i in range(1, 27):
    s = s + i
print(s)
```

On peut aussi réaliser le calcul à la main, si on a le courage, ou utiliser la formule vue en classe de 1ère : $S = \dfrac{26 \times (1 + 26)}{2}$. Quelle que soit la méthode, on trouve $S = 351$.

Il suffit alors de parcourir les valeurs possibles pour a et b. En remarquant qu'ils jouent tous deux des rôles symétriques, on peut supposer $a < b$. Avec deux boucles POUR, on teste toutes les possibilités.

```
S = 351
for a in range(1, 27):
    for b in range(a+1, 27):
        if a * b == S - a - b:
            print(a, b)
```

La seule possibilité est d'avoir choisi les nombres 15 et 21.

Deuxième méthode en utilisant un raisonnement mathématique : la somme vaut 351, donc on cherche deux entiers naturels a et b vérifiant : $351-a-b=ab$. Or $351-a-b=ab \iff 351=a+b+ab \iff 352=(a+1)(b+1)$.

On procède ensuite par décomposition en produit de facteurs : $352=2^5\times 11$ et on sait que $1\leqslant a<b\leqslant 26$. En testant les différentes possibilités on obtient les couples $(15,21)$ ou $(21,15)$.

Retour sur l'exercice A22

1. a. Première méthode :

```
def somme(x, y):
    return 50*x + 20*y
```

b. Si on a x billets de 50 euros, alors il y en a $23-x$ de 20 euros. On peut donc obtenir l'affichage des 24 possibilités :

```
for x in range(24):
        print(x, "billets de 50 euros et", 23-x, "billets de ↙
            ↳ 20 euros donnent la somme", somme(x, 23-x))
```

On obtient 15 billets de 50 euros et 8 billets de 20 euros. S'il y avait beaucoup de possibilités, on aurait ajouté un test pour n'afficher que les solutions valables :

```
for x in range(24):
    if somme(x, 23-x) == 910 :
        print(x, "billets de 50 euros et", 23-x, "billets de ↙
            ↳ 20 euros donnent la somme", somme(x, 23-x))
```

2. a. Soit x le nombre de billets de 50 euros et y le nombre de billets de 20 euros. L'énoncé se traduit ainsi : $(S):\begin{cases}50x+20y=910\\x+y=23\end{cases}$

b. $(S)\iff\begin{cases}50x+20(23-x)=910\\y=23-x\end{cases}\iff\begin{cases}30x=910-460\\y=23-x\end{cases}$

$$(S)\iff\begin{cases}x=15\\y=8\end{cases}$$

3. L'énoncé cette fois-ci se traduit par un système de deux équations à trois inconnues, qui n'a donc pas de solution unique dans $\mathbb{R}$. On ne peut pas utiliser la méthode classique de résolution. Par contre, Python nous permet de trouver une solution à l'aide de deux boucles en testant toutes les combinaisons de billets possibles : si on note x le nombre de billets de 50€, y le nombre de billets de 20€ et z le nombre de billets de 10€, on a $x+y+z=23$ d'où $z=23-x-y$. Ainsi, en s'inspirant de la première question :

```
for x in range(24):
    for y in range(24):
        z = 23 - x - y
        if x*50 + y*20 + z*10 == 910 :
            print(x, "de 50€;", y, "de 20€ et", z, "de 10€")
```

On trouve 6 solutions :

```
12 de 50€; 20 de 20€ et -9 de 10€
13 de 50€; 16 de 20€ et -6 de 10€
14 de 50€; 12 de 20€ et -3 de 10€
15 de 50€; 8 de 20€ et 0 de 10€
16 de 50€; 4 de 20€ et 3 de 10€
17 de 50€; 0 de 20€ et 6 de 10€
```

Les trois premières sont clairement impossibles, pour les 3 dernières, on sait qu'il y avait des billets des trois sortes, ainsi une seule solution est possible : 16 billets de 50€; 4 de 20€ et 3 de 10€. Dans $\mathbb{N}^* \times \mathbb{N}^* \times \mathbb{N}^*$, il y a donc une unique solution.

Lorsqu'on doit alterner beaucoup de texte dans un `print`, on peut utiliser des **f-strings** ainsi :

- On précède les chaînes du caractère `f`
- On place les variables à afficher entre accolades.

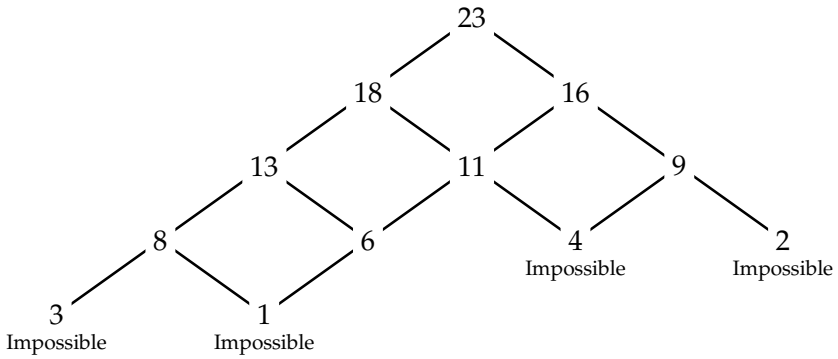
```
for x in range(24):
    for y in range(24):
        z = 23 - x - y
        if x*50 + y*20 + z*10 == 910 :
            print(f"{x} de 50€; {y} de 20€ et {z} de 10€")
```

Il existe de nombreuses autres possibilités avec les f-strings que vous pourrez facilement découvrir sur internet, nous n'en parlerons pas plus dans ce livre.

Retour sur l'exercice A23 Après quelques essais :

1. - $22 = 5+5+5+7$
   - 23 Impossible.
   - $24 = 5+5+7+7$

   Pour s'assurer que la somme de 23@ est bien impossible à réaliser, on peut lister tous les cas dans un arbre (branche de gauche, on retire 5@, branche de droite, on retire 7@).



2. En notant `a` le nombre de pièces de 5@ et `b` celui de 7@, on peut tester toutes les possibilités. Dès qu'une solution est trouvée, on renvoie `True` quittant ainsi les boucles et la fonction. Si en fin de boucle aucune solution n'a été trouvée, on renvoie `False` :

```
def somme(s):
    for a in range(s//5+1) :      # a ne peut dépasser s/5
        for b in range(s//7+1):   # de même pour b
            if a*5 + b*7 == s :
                return True
    return False
```

3. En réalisant une boucle `for`, on peut tester toutes les sommes de 1@ à 200@.

```
for s in range(200) :
    print(s, somme(s))
```

Il semblerait qu'à partir de 24@, il soit toujours possible de réaliser cette somme avec seulement ces deux types de pièces. En effet, si on note $s \geqslant 24$ la somme désirée. Le reste dans la division euclidienne de s par 5 ne pouvant être que $0, 1, 2, 3$ ou 4, et en notant q le quotient entier (comme $s \geqslant 24$, $q \geqslant 4$), seuls les cas suivants peuvent se produire :

- Si $s = 5q$: La décomposition est évidente.
- Si $s = 5q + 1$, on peut écrire $s = 5(q - 4) + 20 + 1 = 5(q - 4) + 3 \times 7$, il suffit donc de prendre $q - 4$ pièces de 5@ et trois pièces de 7@.
- Si $s = 5q + 2$, on peut écrire $s = 5(q - 1) + 5 + 2 = 5(q - 1) + 7$, il suffit donc de prendre $q - 1$ pièces de 5@ et une pièce de 7@.
- Si $s = 5q + 3$, on peut écrire $s = 5(q - 5) + 25 + 3 = 5(q - 5) + 4 \times 7$. Il faut néanmoins remarquer que comme $s \geqslant 24$, le premier nombre de cette forme est 28 et donc ici $q \geqslant 5$, ce qui n'était pas le cas pour 23 justement...
- Si $s = 5q + 4$, de la même manière $s = 5(q - 2) + 10 + 4 = 5(q - 2) + 2 \times 7$, il suffit donc de prendre $q - 2$ pièces de 5@ et deux pièces de 7@.

Par disjonction de cas, on a bien prouvé qu'à partir de 24@, on pouvait toujours fabriquer cette somme avec des pièces de 5@ et 7@.

Retour sur l'exercice A24

1. 3, 6, 24 et 30 étant tous des multiples de 3, il est nécessaire que s le soit aussi. Réciproquement si s est multiple de 3, il s'écrit sous la forme $s = 3p$ où $p \in \mathbb{N}$, ainsi en prenant p pièces de 3 pence, on peut payer la somme s. Finalement, on a prouvé que s peut être atteint avec les pièces données si et seulement si s est un multiple de 3.

2. $51 = 30 + 6 + 6 + 6 + 3$ soit 5 pièces. Cette solution qui consiste à distribuer un maximum de grosses pièces fonctionne bien avec l'euro pour minimiser le nombre de pièces, mais pas ici ! En effet, la décomposition $51 = 24 + 24 + 3$ ne nécessite que 3 pièces !

3. Comme indiqué à la question précédente, on ne peut pas trouver si facilement la meilleure décomposition. Comme il n'y a que 4 types de pièces, on peut réaliser une boucle testant toutes les solutions.

```
def decompose(s) :
  nb_max = s + 1
  reponse = []
  for n3 in range(s//3+1):
     for n6 in range(s//6+1):
        for n24 in range(s//24+1):
           for n30 in range(s//30+1):
              if n3*3 + n6*6+n24*24 + n30*30 == s :
                nb = n3+n6+n24+n30
                if nb < nb_max :
                    print('Trouvé', n3, n6, n24, n30, '->', nb)
                    reponse = [3]*n3 + [6]*n6+ [24]*n24 + [30]*n30
                    nb_max = nb
  return reponse
```

AU DÉTOUR DE LA BALADE...

*Le principe qui consiste à commencer par distribuer, jusqu'à ne plus pouvoir, les plus grosses pièces est appelé **algorithme glouton**. Il fonctionne bien avec le système européen car la suite des valeurs (1, 2, 5, 10, 20, 50, 100, 200) est une suite **super croissante** : chaque terme est strictement supérieur à la somme de tous les termes précédents, ce qui n'est pas le cas avec notre exemple.*

*Le problème proposé est un grand classique des cours d'informatique appelé **problème du rendu de monnaie**. Il n'est pas simple d'avoir une solution rapide, la **programmation dynamique** étudiée en enseignement de spécialité NSI peut permettre de résoudre ce problème en temps raisonnable.*

Chapitre B

ARITHMÉTIQUE

1. Pour s'échauffer

B1 *Si n est un entier positif, relier les expressions Python aux expressions en français correspondantes :*

- `n + 1`
- `n // 10 % 10`
- `n // 10`
- `n // 2 * 2 + 1`
- `n ** 2`
- `n % 10`
- `n * 2`

- *Le chiffre des unités de n*
- *Le premier nombre impair qui suit n*
- *Le chiffre des dizaines de n*
- *Le nombre de dizaines de n*
- *Le carré de n*
- *Le double de n*
- *L'entier qui suit n*

B2 ***Olympiades académiques Amiens 2021***

Voici un processus que l'on répète en prenant au départ un entier $n > 0$:

- *Si n est un multiple de 3, on divise n par 3.*
- *Si $n - 1$ est un multiple de 3, on calcule $2n$.*
- *Si $n - 2$ est un multiple de 3, on calcule $2n - 1$.*
- *Si $n = 1$ le processus s'arrête, sinon on recommence.*

Par exemple : si $n = 5$, on obtient en 4 étapes les nombres 5 , 9, 3, 1.

1. *Créer une fonction qui a pour paramètre un entier n non nul et qui renvoie le nombre suivant obtenu avec cet algorithme.*
2. *Quels nombres obtient-on si on prend $n = 7$? Le programme s'arrête-t-il ?*

3. *Créer une fonction qui renvoie le nombre d'étapes réalisées lorsque le processus s'arrête et la tester pour toutes les valeurs entières de n comprises entre 2 et 12.*

4. *Quel est le nombre à moins de 7 chiffres qui nécessite le plus d'étape ?*

5. *Si $n = 3^p$ où p est un entier naturel, que peut-on dire du nombre d'étapes de ce processus ?*

2. À la découverte des nombres

B3 *Deux nombres entiers n et m sont dits **amicaux** si la somme des diviseurs de l'un est égale à la somme des diviseurs de l'autre et si ces sommes valent la somme des deux nombres.*

Dit autrement, si l'on appelle σ la fonction qui, à un entier associe la somme de ses diviseurs :

$$m \text{ et } n \text{ sont amicaux si et seulement si } \sigma(n) = \sigma(m) = n + m.$$

1. *Justifier (mathématiquement) que 220 et 284 sont amicaux.*

2. *Écrire une fonction `somme(n)` en Python qui renvoie la somme des diviseurs d'un entier naturel non nul n.*

3. *En déduire une fonction `amicaux(a, b)` qui reçoit 2 entiers et renvoie un booléen indiquant si ces deux nombres sont amicaux ou non.*

4. *Écrire un programme qui affiche tous les couples de nombres amicaux distincts et possédant strictement moins de 5 chiffres.*

B4 *On appelle **nombre chanceux d'Euler**, un nombre entier c supérieur ou égal à 2 tel que pour tout entier n compris entre 0 et $c - 2$, le nombre $n^2 + n + c$ est encore un nombre premier.*

1. *Compléter la fonction suivante prenant pour paramètres un entier n et un entier c, avec $c \geqslant 2$ et renvoyant le nombre $n^2 + n + c$.*

```
def nombre(n, c):
    return .....
```

2. *Un élève a proposé cette fonction pour compter le nombre de diviseurs positifs d'un entier n non nul :*

```
def nbdiviseurs(n):
    nb = 0
    for d in range(1, n):
        if n % d == 0 :
            nb = nb + 1
        return nb
```

a. *Quels sont les diviseurs de l'entier* 6 *?*

b. *Tester la fonction et corriger les éventuels problèmes.*

c. *En déduire une fonction qui renvoie* `True` *si le nombre est premier et* `False` *sinon.*

3. a. *Écrire une fonction* `chanceux(c)` *qui reçoit un entier supérieur ou égal à* 2 *et renvoie un booléen indiquant si ce nombre est chanceux.*

b. *Déterminer les nombres chanceux d'Euler inférieurs à* 11*.*

c. *Il a été prouvé en 1967 qu'il existe exactement six nombres chanceux d'Euler. Quels sont ces six nombres ?*

AU DÉTOUR DE LA BALADE...

La formule d'Euler $n^2 + n + 41$ *fournit* 40 *nombres premiers pour* n *entier allant de* 0 *à* 39*. Cette formule célèbre a été source de beaucoup d'études et même en* 2006 *d'un concours international.*

B5 *[Nombres palindromes]* ***Inspiré d'une idée de Sylvain ALBISSER.***

Les deux questions peuvent être traitées de manière indépendante.

1. a. *Écrire une fonction qui reçoit un entier à deux chiffres non nuls et renvoie ce nombre écrit "à l'envers". Par exemple :*

```
>>> retourne(12)
21
```

b. *Même question avec un nombre à (exactement) 3 chiffres.*

2. *On dit qu'un nombre est un palindrome s'il peut se lire de gauche à droite ou de droite à gauche dans son écriture décimale, voici un exemple* 1289821*. Réaliser un programme pour répondre à la question suivante : "quels sont les palindromes à 5 chiffres qui sont des carrés de palindromes à 3 chiffres ?"*

B6 *On considère la fonction suivante qui reçoit un entier naturel, ne renvoie rien mais affiche la liste des diviseurs de cet entier.*

```
def diviseurs(n):
    for i in range(n+1):
        if n % i == 0:
            print(i)
```

1. a. *En exécutant l'appel* `diviseurs(10)` *dans la console, une erreur apparaît, quelle en est la raison ? Proposer une modification.*

   b. *Utiliser cette fonction pour trouver un diviseur commun aux nombres 222, 555, 777, 222555 et 555777. Que peut-on conjecturer pour les nombres de la forme* `aaabbb` *où* `a` *et* `b` *sont des chiffres de 0 à 9 ?*

   c. *Démontrer la conjecture.*

2. a. *En s'inspirant de la fonction précédente, proposer une nouvelle fonction* `diviseurs_communs` *qui affiche les diviseurs communs à deux nombres entiers.*

   b. *Tester la fonction en trouvant les diviseurs communs aux nombres suivants :*

   - *80 et 120*
   - *426 et 96*
   - *37 et 63*

3. *Dans la suite de l'exercice, on utilisera les listes.*

   a. *Compléter la fonction suivante pour qu'elle renvoie la liste des diviseurs d'un entier.*

```
def liste_diviseurs(n):
    div = ......
    for i in range( ...... , cadre):
        if n % i == 0:
            div.append( ...... )
    return div
```

   b. *Un nombre est parfait lorsqu'il est égal à la somme de tous ses diviseurs positifs autres que lui-même.La fonction* `sum(liste)` *permet de calculer la somme des éléments d'une liste. À l'aide de cette fonction, créer une fonction* `NBparfait` *qui renvoie* `True` *si le nombre est parfait et* `False` *sinon.*

   c. *À l'aide de cette fonction, indiquer quels sont les nombres parfaits parmi les nombres suivants : 6, 14, 28, 105, 130, 496, 8128. Vérifier la cohérence des résultats proposés par le programme.*

B7 *On s'intéresse aux nombres de 6 chiffres n'utilisant dans la même proportion que 2 chiffres distincts. Ces nombres peuvent être représentés de cette manière :*

où les symboles ○ et ★ représentent deux chiffres distincts. Parmi ces nombres, lesquels sont divisibles par 7 quels que soient les chiffres choisis pour ces deux symboles ?

1. *On cherche à répondre à cette question à l'aide d'un algorithme. Pour cela on va représenter les motifs par une chaîne avec comme convention, par exemple, 'C' pour cercle et 'E' pour étoile.*

   a. *Écrire une fonction* `valeur(motif,c,e)` *qui, recevant le motif sous forme de chaîne de caractères et les valeurs correspondant aux symboles ○ et ★, renvoie la valeur numérique du nombre représenté par le motif. Par exemple :*
   `valeur('CCEECE',3,2)` ⟼ 332232

   b. *Écrire un programme qui demande un motif et qui affiche la liste des nombres ayant ce motif et non divisibles par 7.*

   c. *Répondre à la question de l'énoncé.*

2. *Comment répondre à la question sans utiliser un programme ?*

3. Besoin d'une carte ?

B8 *[Autour des cartes bancaires]*

1. *On souhaite modéliser le fonctionnement d'un terminal de paiement fort sympathique dont voici le fonctionnement :*

   - *Comme tous les terminaux, il autorise 3 tentatives infructueuses avant de bloquer la carte bancaire.*
   - *Dans son extrême bonté, à chaque tentative, il affiche un message indiquant quels chiffres sont corrects et quels chiffres ne le sont pas !*

a. *Écrire une fonction* `retour(code,prop)` *qui reçoit deux entiers* `code` *(code à trouver) et* `prop` *(la proposition) et renvoie une chaine de caractères réponse formée de :*

   - `'x'` *: le chiffre est faux.*
   - `'o'` *: le chiffre est correct.*

   Par exemple :

```
>>> retour(1234, 5214)
'xoxo'
```

b. *Utiliser cette fonction pour modéliser un terminal qui tire au sort un code et propose 3 essais pour trouver le code en indiquant les chiffres trouvés.*

2. *Sur internet, lorsque l'on entre son numéro de carte bancaire, de nombreux tests sont réalisés pour éviter les erreurs de saisie et ne pas interroger les banques sans raison. Une des vérifications est donnée ici, elle est appelée* ***algorithme de Luhn*** *:*

   - *Entrer les 16 chiffres*
   - *Remplacer les chiffres de rang pair (on commence à 0) par leur double. Si le résultat dépasse 9, soustraire 9.*
   - *Additionner tous les chiffres de ce nouveau numéro de carte.*
   - *Si le nombre obtenu est un multiple de 10 alors le numéro de la carte est considéré comme valide.*

   a. *Créer une fonction* `num2liste` *qui reçoit un nombre et renvoie la liste des chiffres de ce nombre. Par exemple :*

```
>>> num2liste(123)
[1, 2, 3]
```

   b. *Créer une fonction qui admet en paramètre un nombre de 16 chiffres et qui renvoie si le test est valide.*

   c. *Tester avec les numéros de carte bleu suivants : 4624 7482 3324 9080 puis 4624 7482 3324 9780.*

B9 *[La carte vitale].*

En France, le numéro de sécurité sociale présent sur la carte vitale est un numéro "signifiant" (c'est-à-dire non aléatoire) composé de 13 chiffres permettant d'identifier une personne, il est suivi d'une clé de contrôle de 2 chiffres pour déceler d'éventuelles erreurs de saisie.

Image par Giesesamvitale sur Wikipédia.

Par exemple, le premier chiffre indique le sexe (1 pour un homme, 2 pour une femme), les deux suivants l'année de naissance, suivis de deux chiffres pour indiquer le mois de naissance et encore deux autres pour le département de naissance, et ainsi de suite. Pour la carte fictive ci-dessus, le code est donc 2690549588157 *et la clé* 80*. Cette clé correspond à la différence entre* 97 *et le reste de la division euclidienne du code dans la division par* 97*.*

1. *Écrire une fonction en Python qui calcule et renvoie la clé de sécurité d'un code* N *à 13 chiffres.*

2. *En utilisant cette fonction, déterminer le nombre de codes possibles pour la carte d'une femme née en Janvier* 72 *dans l'Oise (département* 60*) et dont la clé vaut* 42*.*

4. Euclide sur notre chemin

B10 *[Le PGCD dans tous ses états !]*

Plusieurs méthodes sont possibles pour déterminer le PGCD de deux nombres. Il peut être intéressant de tester la rapidité de ces démarches en les codant avec Python.

1. a. *Créer une fonction qui a pour paramètres deux nombres entiers non simultanément nuls et qui renvoie leur PGCD en testant tous les diviseurs.*

   b. *Créer une fonction qui renvoie le PGCD de deux nombres en utilisant l'algorithme d'Euclide dont on rappelle le principe sur un exemple :*

a	b	reste de la division de a par b
222	66	24
66	24	18
24	18	6
18	6	0
6	0	–

Le PGCD est le dernier reste non nul : ici, $\text{PGCD}(222,66) = 6$

c. *Créer une fonction qui renvoie le PGCD en programmant l'algorithme d'Euclide de manière récursive, basé sur le fait que si* a *et* b *sont deux entiers non simultanément nuls,*

$$\text{PGCD}(a,b) = \begin{cases} a \text{ si } b = 0 \text{ sinon,} \\ \text{PGCD}(b,r) \text{ où } r \text{ est le reste de la division de } a \text{ par } b \end{cases}$$

d. *La fonction* **`gcd`** *du module* `math` *existe déjà et permet de calculer le PGCD de deux entiers. L'utiliser pour vérifier les différentes fonctions créées et comparer les temps d'exécution des différentes méthodes.*

2. *Dans cette partie, on va utiliser à bon escient le PGCD pour calculer avec des fractions. On fait le choix ici de représenter la fraction* $\frac{a}{b}$ *par le couple* `(a, b)` *où* `a` $\in \mathbb{Z}$ *et* `b` $\in \mathbb{N}^*$.

a. *En utilisant une des fonctions précédentes, créer une fonction ayant pour paramètre une fraction et renvoyant la fraction irréductible correspondante.*

b. *En utilisant cette fonction, en créer deux autres* `ajouteFrac` *et* `multiplieFrac` *qui reçoivent 4 entiers* n_1, d_1, n_2 *et* d_2 *et renvoient respectivement le résultat de* $\frac{n_1}{d_1} + \frac{n_2}{d_2}$ *et* $\frac{n_1}{d_1} \times \frac{n_2}{d_2}$ *sous forme d'une fraction irréductible.*

3. *On peut utiliser le PGCD pour résoudre des problèmes : un chocolatier a fabriqué 186 pralines et 155 chocolats. Les colis sont constitués ainsi :*

- *le nombre de pralines est le même dans chaque colis ;*
- *le nombre de chocolats est le même dans chaque colis ;*
- *tous les chocolats et toutes les pralines sont utilisés.*

Quel nombre maximal de colis pourra-t-il réaliser et combien y aura-t-il de chocolats et de pralines dans chaque colis ?

5. Quelques belles équations

B11 *Laurence décide de randonner dans les Pyrénées en partant de Saint Jean Pied de Port. Elle réserve plusieurs nuitées dans deux hébergements différents. Une nuit dans le premier hébergement coûte 24 euros et dans le deuxième 45 euros. Elle se rappelle que le coût total de sa réservation est de 438 euros. L'objectif est de retrouver la répartition des nuitées dans les hébergements.*

1. *Une solution informatique :*

   On souhaite retrouver les nombres x et y de nuitées passées respectivement dans chaque hébergement avec un coût total de t euros.

   a. *Quelles contraintes doivent vérifier les nombres x et y (nature des nombres, encadrement) ?*

   b. *Recopier et compléter la fonction Python ci-dessous afin qu'elle affiche le ou les couples (x, y) possibles :*

```
def nuitees(t):
    for x in range( ..... ):
        for y in range( ..... ):
            if ..... :
                print(x, y)
```

   c. *Quels sont les couples solutions pour notre problème ?*

2. *Une solution purement mathématique :*

   a. *Justifier que le coût total de la réservation est toujours multiple de 3.*

   b. *Justifier l'existence d'un couple d'entiers relatifs tel que $8x + 15y = 1$. Donner un tel couple.*

   c. *En déduire les solutions de l'équation $8x + 15y = 146$ dans $\mathbb{Z}^2$.*

   d. *Répondre au problème posé.*

B12 *Le chiffrement affine est une méthode de cryptographie avec un chiffrement par substitution mono alphabétique. On choisit deux entiers naturels a et b comme clés et on procède par étapes :*

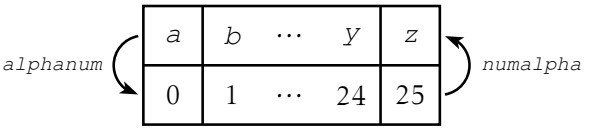
- *On commence par remplacer chaque lettre par son équivalent numérique noté x. C'est à dire la lettre A devient 0, B devient 1, etc.*
- *Pour une valeur de x donnée, on effectue la division euclidienne de $ax + b$ par 26. On note y le reste de cette division.*
- *On transforme alors la valeur de y avec la lettre correspondante.*

Dans cet exercice, on ne considère que des textes écrits en ***minuscules****, sans accent, les caractères autres que les lettres (espaces, ponctuation, ...) seront ici conservés à l'identique.*

AU DÉTOUR DE LA BALADE...

On pourra exploiter en classe de terminale dans l'option maths expertes les congruences en définissant la fonction affine f d'expression $f(x) = ax+b$. Chaque lettre du texte sera chiffrée par le calcul du reste de la division euclidienne par 26 de l'image $f(x)$, c'est à dire l'entier $y \in [\![0;25]\!]$ tel que : $y \equiv ax+b[26]$.

1. *Créer 2 fonctions* `alphanum` *et* `numalpha` *réciproques l'une de l'autre. La première associe un code à une lettre, la seconde associe une lettre à un code, comme illustré ci dessous :*



2. a. *Créer une fonction* `code_lettre` *qui reçoit en paramètres un caractère et deux entiers a et b et renvoie le caractère codé par la méthode précédente. On rappelle que si ce n'est pas une lettre, on renvoie le caractère d'origine.*

   b. *En déduire une fonction* `code_chaine` *qui reçoit en paramètres une chaîne de caractères et deux entiers a et b et renvoie la chaîne de caractères codée par la méthode précédente. Vérifier alors que :*

```
>>> code_chaine("les maths avec python c'est super !",10,4)
"ksc uemwc egsy ykmwoe y'scm cwyss !"
```

3. *On souhaite à présent déchiffrer le message. Il faut donc "inverser" l'équation $y = ax+b$. Évidemment, hors de question d'écrire $x = \dfrac{y-b}{a}$, car on travaille sur des entiers et surtout il ne faut pas oublier la division par 26. La fonction de codage étant affine, on cherche alors une fonction affine.*

   a. *Par essai-erreur, écrire une fonction* `trouve_cle` *qui reçoit 2 entiers* `a` *et* `b` *et qui renvoie un couple* `(c, d)` *tel que pour toute lettre ℓ,*

```
code_lettre( code_lettre(ℓ, a, b), c, d) = ℓ
```

   Si aucune clé n'est trouvée, la fonction renvoie `False`

   b. *Toutes les clés possèdent-elles un inverse ? Si non, que dire de celles qui n'en possèdent pas ?*

B13 *[Crible de Matiassevitch]*

N *est un entier naturel non nul fixé et supérieur à 2. On complètera le programme au fur et à mesure afin de réaliser les constructions suivantes.*

1. *Tracer en noir la parabole* $\mathcal{P}$ *d'équation* $y = x^2$ *sur l'intervalle* $[-\mathrm{N};\mathrm{N}]$.

2. *Tracer en bleu l'ensemble des points de l'axe des ordonnées dont l'ordonnée est un entier de l'intervalle* $[2;\mathrm{N}^2]$.

3. a. *Écrire une fonction* `segment(a, b)` *qui reçoit 2 entiers naturels non nuls* `a` *et* `b` *et qui trace en pointillés verts le segment dont les extrémités sont les points de* $\mathcal{P}$ *d'abscisses respectives* `-a` *et* `b`.

   b. *Compléter la fonction précédente pour qu'elle place en plus un point rouge à l'intersection du segment précédent avec l'axe des ordonnées.*

   c. *Tracer tous les segments qu'il est possible de réaliser avec la fonction* `segment(a,b)` *pour* `a` *et* `b` *des entiers de l'intervalle* $[2;\mathrm{N}^2/2]$.

4. a. *Que dire des ordonnées des points figurant encore en bleu ?*

   b. *Démontrer cette conjecture.*

B14 *En mémoire du mathématicien grec du III*$^{\text{ème}}$ *siècle Diophante d'Alexandrie, on appelle "équation diophantienne" une équation dont les inconnues sont des nombres entiers.*

1. *On cherche à résoudre l'équation diophantienne :* (E_1) : $x^2 - 4y^2 = 8633$.

   a. *Expliquer pourquoi on peut ne chercher que les solutions positives de cette équation.*

   b. *À l'aide d'un programme, lister les possibilités de décomposer l'entier* 8633 *en produit de deux entiers positifs.*

   c. *En factorisant le membre de gauche, déduire l'ensemble des solutions de l'équation* (E_1).

2. *On cherche à présent à résoudre l'équation diophantienne :*

$$(\mathrm{E}_2) : x^2 + 4y^2 = 8633.$$

   a. *Pourquoi la méthode précédente ne fonctionne-t-elle plus ? Justifier que le nombre de solutions est nécessairement fini.*

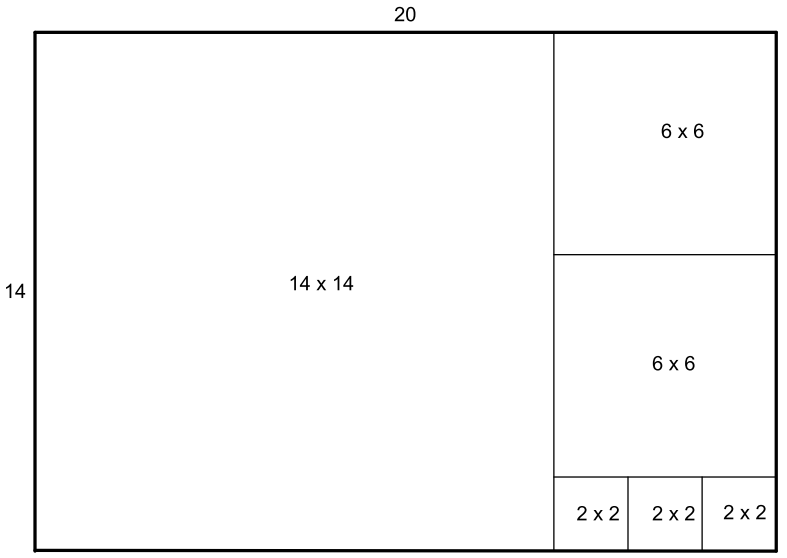
   b. *Écrire alors un programme qui les donne toutes.*

6. Si vous n'êtes pas encore fatigué, petits défis

B15 *D'après un exemple de la documentation AmiensPython*

Créer une fonction qui transforme un produit en somme de carrés d'entiers grâce à l'algorithme de Babylone présenté sous forme graphique ci-dessous. Par exemple, ici :

$$14 \times 20 = 14^2 + 6^2 + 6^2 + 2^2 + 2^2 + 2^2$$



B16 *On se propose de démontrer que les entiers égaux à la somme des cubes de leurs chiffres en écriture décimale sont* $0, 1, 153, 370, 371$ *et* 407

1. *Montrer que si x est un tel entier alors $x \leq 10^4$.*

2. *Écrire une fonction,* `somme_cubes`*, qui prend en paramètre un entier naturel et renvoie la somme des cubes des chiffres de ce nombre (en écriture décimale).*

3. *Écrire un script qui permet de répondre au problème.*

B17 *Sur la chaîne youtube Hedacademy...*

Dans cet exercice, on cherche tous les entiers a, b et c tels que

$$a < b < c \text{ et } (\mathrm{E}) : 2^a + 2^b + 2^c = 1120.$$

1. *Une première solution "100% informatique"*
   a. *Trouver un majorant pour la valeur c (qui en sera également un pour b et pour a).*
   b. *En déduire les solutions à l'aide d'un programme "force brute", c'est à dire en testant toutes les possibilités.*
2. *Une seconde solution "100% mathématique"*
   a. *Décomposer 1120 en produit de facteurs premiers.*
   b. *Dans l'équation (E) donnée, mettre 2^a en facteur et en déduire la valeur de a.*
   c. *Déterminer ensuite les valeurs de b et de c.*
3. *Une dernière solution "50% informatique - 50% mathématique"*
   a. *La fonction `bin` de Python donne l'écriture binaire d'un entier donné. Donner l'écriture binaire de 1120.*
   b. *Conclure.*

7. Aide pour les exercices

Aide pour l'exercice B1 Petit rappel sur un exemple : pour le nombre 123, le chiffre des dizaines est 2 alors que le nombre de dizaines est 12.

Aide pour l'exercice B2

1. Comment traduire qu'un nombre est multiple de 3 ?

Aide pour l'exercice B4 2b. Il y a 2 modifications à faire.

Aide pour l'exercice B5

1. Dans l'exercice B1, on a vu comment calculer le nombre de dizaines et le chiffre des unités d'un nombre...
2. Pour cette deuxième question au moins deux solutions sont envisageables :

- Générer des palindromes à 3 chiffres, les mettre au carré et regarder si le nombre obtenu est un palindrome.
- Générer des palindromes à 5 chiffres, prendre la racine carrée et regarder si le résultat obtenu est un entier et un palindrome.

La première solution est plus rapide d'exécution puisqu'il y a moins de valeurs possibles. Néanmoins ce n'est pas si simple de tester si un nombre à 5 chiffres est un palindrome. Il sera plus simple de programmer la seconde méthode. Vous devez trouver 5 solutions.

Aide pour l'exercice B7

On peut par exemple travailler par étapes en parcourant les symboles du motif. Par exemple pour `valeur('CCEECE',3,2)`, voici ce que cela donnerait. Il suffit alors d'évaluer la chaîne nombre afin de la retourner.

On peut aussi, à condition de réfléchir à l'opération mathématique permettant de passer d'une ligne à l'autre, ne pas utiliser la variable `nombre` sous forme d'une chaîne de caractères mais directement sous forme d'un entier.

symbole	nombre
	″
'C'	'3'
'C'	'33'
'E'	'332'
'E'	'3322'
'C'	'33223'
'E'	'332232'

Aide pour l'exercice B10

1a. Dans un des exercices précédents, une fonction a déjà été créée pour rechercher des diviseurs d'un entier.

2a. Pour rendre irréductible une fraction, on utilise le PGCD.

2b. Il suffit de remarquer que $\frac{n_1}{d_1} + \frac{n_2}{d_2} = \frac{n_1 d_2 + n_2 d_1}{d_1 d_2}$.

Aide pour l'exercice B12 Pour la première question, on peut par exemple utiliser la table ASCII.

Aide pour l'exercice B14 2b. Montrer qu'il existe un nombre $A > 0$ que ni x, ni y ne peuvent dépasser si (x, y) est solution de (E_2).

Aide pour l'exercice B15 Le schéma nous inspire l'algorithme suivant :

```
TANT QUE on n'a pas un carré FAIRE :
    Retirer le plus grand carré possible
    AFFICHER l'aire de ce carré
FIN TANT QUE
AFFICHER l'aire du carré restant
```

8. Solutions des exercices

Retour sur l'exercice B1

- Le chiffre des unités de n : `n % 10`
- Le premier nombre impair qui suit n : `n // 2 * 2 + 1`
- Le chiffre des dizaines de n : `n // 10 % 10`
- Le nombre de dizaines de n : `n // 10`
- Le carré de n : `n ** 2`
- Le double de n : `n * 2`
- L'entier qui suit n : `n + 1`

Retour sur l'exercice B2

1. Il suffit d'écrire tour à tour les conditions du problème. Ici on a fait le choix de renvoyer 1 si on entre 1, ce qui n'a pas d'incidence car dès que l'on obtient 1 on arrête.

```
def suivant(n) :
    if n == 1 :
        return 1
    elif n % 3 == 0:
        return n // 3
    elif (n-1) % 3 == 0 :
        return 2*n
    else:
        return 2*n-1
```

2. Avec une boucle TANT QUE :

```
n = 7
while n != 1 :
    print(n)
    n = suivant(n)
print(n)
```

On obtient en 6 étapes : 7, 14, 27, 9, 3, 1.

3. Pour cette question, on peut s'inspirer de la précédente, en ajoutant un compteur :

```
def temps(n) :
    t = 0
    while n != 1 :
        n = suivant(n)
        t = t + 1
    return t

for i in range(2,13) :
    print(i, temps(i))
```

n	nombre d'étapes
2	3
3	2
4	7
5	4
6	4
7	6
8	6
9	3
10	16
11	8
12	8

4. En utilisant la fonction précédente, on crée une variable `maxi` qui conserve en mémoire le plus grand nombre d'étapes trouvées que l'on met à jour au fur et à mesure de notre recherche :

```
maxi = 0
for i in range(2, 10**6):
    m = temps(i)
    if m > maxi :
        maxi = m
        print(i, maxi)
```

Finalement, c'est 988768 avec 175 étapes qui remporte le concours !

5. On démontre, par récurrence par exemple, que si $n = 3^p$ alors il y a $p + 1$ étapes.

Retour sur l'exercice B3

1. Les diviseurs de 220 sont 1, 2, 4, 5, 10, 11, 20, 22, 44, 55, 110 et 220. De plus, les diviseurs de 284 sont 1, 2, 4, 71, 142 et 284. On vérifie bien que les diviseurs de 220 additionnés font 284 et ceux de 284 font 220, alors 220 et 284 sont des nombres amicaux.
2.
```
def somme(n) :
    s = 0
    for d in range(1, n+1) :
        if n % d == 0 :
            s = s + d
    return s
```

3. Une première solution :

```
def amicaux(n, m):
    return somme(n) == somme(m) and somme(m) == n+m
```

Qui n'est pas optimale car la fonction `somme` demande du temps. Dans cette nouvelle solution, elle n'est appelée qu'au maximum 2 fois :

```
def amicaux(n, m):
    return somme(n) == n+m and somme(m) == n+m
```

4. Voici une solution répondant au problème :

```
for a in range(1, 10**5):
    for b in range(a+1, 10**5):
        if amicaux(a, b):
            print(a, b)
```

Plus efficace encore :

```
for a in range(1, 10**5):
    b = somme(a) - a
    if b > a and somme(b) == a+b
        print(a, b)
```

À l'heure actuelle,

- On connait plus de 2 millions de paires de nombres amicaux.
- On ne sait pas s'il y en a une infinité.
- On ne sait pas s'il y a des paires de nombres premiers entre eux.

Retour sur l'exercice B4

1. Pas de difficulté pour cette question :

```
def nombre(n, c):
        return n*n + n + c
```

2. a. Les diviseurs de 6 sont 1, 2, 3 et 6.

   b. Pourtant en testant :

```
>>> nbdiviseurs(6)
1
```

Deux raisons à cela :

- Le `return` est mal indenté : dès le premier tour de boucle, la fonction s'arrête.
- Si on veut tester `d` de 1 à `n`, on doit écrire `range(1, n+1)` (ou initialiser la variable `nb` à 1 pour compter le nombre `n` lui-même comme diviseur)

Voici une version corrigée :

```
def nbdiviseurs(n):
    nb = 0
    for d in range(1, n+1):
        if n % d == 0 :
            nb = nb + 1
    return nb
```

c. Un nombre est premier s'il possède exactement 2 diviseurs :

```
def premier(n) :
    return nbdiviseurs(n) == 2
```

3. a. Voici une première solution pour cette fonction : on initialise un booléen `chance` à VRAI - on parle de drapeau (ou flag) en informatique - et dès que l'on trouve un n tel que n^2+n+c n'est pas premier on remplace sa valeur par FAUX.

```
def chanceux(c):
    chance = True
    for n in range(c-1) :
        if premier(nombre(n, c)) == False:
            chance = False
    return chance
```

Ce code peut être raccourci :

- D'une part, comme `premier` renvoie déjà un booléen, on peut remplacer la ligne

```
premier(nombre(n, c)) == False
```

par cette ligne qui a le même effet :

```
not premier(nombre(n, c))
```

- D'autre part, on peut se passer du drapeau `chance` en utilisant le fait que dès que Python rencontre un `return`, on quitte la fonction.

Ainsi on peut écrire de manière plus concise :

```
def chanceux(c):
    for n in range(c-1) :
        if not premier(nombre(n, c)):
            return False
    return True
```

b. Pour trouver tous les nombres chanceux plus petits ou égaux à 11, une boucle `for` fera l'affaire :

```
for c in range(2, 12):
    if chanceux(c):
        print(c)
```

On trouve les nombres 2, 3, 5 et 11.

c. Pour cette dernière question, il faut plutôt une boucle `while`, car on ne sait pas jusqu'où il faudra chercher (espérons que cela ne sera pas trop loin !) :

```
nbc, c = 0, 2
while nbc < 6:
    if chanceux(c):
        nbc = nbc + 1
        print(c)
    c = c + 1
```

On trouve alors les six nombres cherchés : 2, 3, 5, 11, 17 et 41.

Retour sur l'exercice B5

1. a. On a vu dans l'exercice précédent que l'on pouvait récupérer le nombre de dizaines avec `n//10` et le chiffre des unités par `n%10`. Ici, n a deux chiffres donc le nombre de dizaines correspond au chiffre des dizaines. On peut alors fabriquer le nombre p en 3 lignes :

```
def retourne(n) :
    d = n // 10      # Chiffre des dizaines
    u = n % 10       # Chiffre des unités
    p = u * 10 + d   # Nouveau nombre
    return p
```

Ou directement,

```
def retourne(n) :
    return n // 10 + n % 10 * 10
```

b. Avec le même principe, on peut recommencer avec un nombre à 3 chiffres, voici la solution en une ligne :

```
def retourne(n) :
    return n // 100 + n % 100 // 10 * 10 +  n % 10 * 100
```

2. Nous allons générer des palindromes à 5 chiffres sous la forme $n =$ | u | d | c | d | u | où $c \in [\![0,9]\!]$, $d \in [\![0,9]\!]$ et $u \in [\![1,9]\!]$ (pour avoir 5 chiffres). On calcule alors $r = \sqrt{n}$ et si ce résultat est entier, on regarde si r est un palindrome. Ce qui est assez facile en utilisant la fonction de la question 1b ou juste ici en vérifiant si le chiffre des centaines (`r//100`) est égal au chiffre des unités (`r % 10`).

```
from math import sqrt

for u in range(1, 10):
    for d in range(10):
        for c in range(10):
            n = u * 10000 + d * 1000 + c * 100 + d * 10 + u
            r = round(sqrt(n))
            if r ** 2 == n and r // 100 == r % 10:
                print(r, "²=", n)
```

On trouve 5 solutions : $101^2 = 10201$; $111^2 = 12321$; $121^2 = 14641$; $202^2 = 40804$ et enfin $212^2 = 44944$.

AU DÉTOUR DE LA BALADE...

La plaque d'immatriculation de la voiture de Donald Duck est 313. Ce nombre possède quelques propriétés : c'est un nombre premier et aussi un palindrome (en base 10 et en base 2).

Retour sur l'exercice B6

1. a. L'appel `diviseurs(10)` déclenche une erreur, car le premier diviseur testé dans la boucle est 0. Voici une correction :

```
def diviseurs(n):
    for i in range(1, n+1):
        if n % i == 0:
            print(i)
```

b. On conjecture que les nombres du type bbb et du type $aaabbb$ sont tous divisibles par 1, 3, 37 et 111.

En effet | a | a | a | $= a \times 100 + a \times 10 + a$
$= a \times (100 + 10 + 1)$
$= a \times 111$
$= a \times 37 \times 3$.

De même | a | a | a | b | b | b | $= a \times 111000 + b \times 111$
$= 111(a \times 1000 + b)$

2. a. Cette nouvelle fonction possède 2 paramètres et il suffit dans le test de vérifier que le diviseur divise bien ces derniers :

```
def diviseurs_communs(n, m):
    for i in range(1, n+1):
        if n % i == 0 and m % i == 0:
            print(i)
```

Même si elle est correcte, dans cette solution, si `n` est plus grand que `m`, la boucle continue (sans rien trouver), on pourrait donc s'arrêter au plus petit des deux :

```
def diviseurs_communs(n, m):
    fin = min(n, m)
    for i in range(1, fin+1):
        if n % i == 0 and m % i == 0:
            print(i, sep=";")
```

b. On vérifie :

```
>>> diviseurs_communs(80,120)
1; 2; 4; 5; 8; 10; 20; 40;
>>> diviseurs_communs(426, 96)
1; 2; 3; 6;
>>> diviseurs_communs(37, 63)
1;
```

3. a. Cette fonction ressemble à la précédente, sauf qu'au lieu d'afficher, on stocke les diviseurs dans une liste :

```
def liste_diviseurs(n):
    div = []
    for i in range(1, n+1):
        if n % i == 0:
            div.append(i)
    return div
```

b. Attention, si n est un entier, la fonction précédemment réalisée renvoie aussi n dans la liste des diviseurs, ainsi lorsqu'on va calculer la somme, le total fera n de trop. On devra donc vérifier si :

somme $-n = n$, c'est à dire si somme $= 2n$:

```
def NBparfait(n):
    liste = liste_diviseurs(n)
    somme = sum(liste)
    if somme == 2*n :
        return True
    else :
        return False
```

On peut comme souvent écrire en une ligne :

```
def NBparfait(n):
    return sum(liste_diviseurs(n)) == 2*n
```

Au moment de l'écriture de ce livre,

- On connaît 51 nombres parfaits.
- À partir du 48$^{\text{ième}}$, qui comporte 77 chiffres, on ne sait pas s'il en manque entre 2 connus.
- On ne sait pas s'il existe des nombres parfaits impairs.
- On ne sait pas s'il existe une infinité de nombres parfaits.

Retour sur l'exercice B7

1. a. Voici deux solutions, l'une utilisant la fonction `int`, l'autre non.

```
def valeur(motif,c,e) :
    nombre = ""
    for symbole in motif :
        if symbole == 'C' :
            nombre = nombre + str(c)
        else :
            nombre = nombre + str(e)
    return int(nombre)
```

Si on souhaite se passer de la fonction `int`, il suffit de se souvenir que pour décaler les chiffres d'un nombre vers la gauche, il suffit de multiplier ce nombre par 10.

```
def valeur(motif, c, e):
    nombre = 0
    for symbole in motif:
        if symbole == 'C':
            nombre = 10*nombre + c
        else:
            nombre = 10*nombre + e
    return nombre
```

b. Une fois la fonction `valeur` créée, il n'y a plus qu'à parcourir tous les chiffres de 0 à 9 pour c et pour e. À chaque fois que le nombre obtenu n'est pas un multiple de 7, on l'affichera.

```
motif = input('Entrer un motif composé de C et de E : ')
for c in range(10):
    for e in range(10):
        v = valeur(motif, c, e)
        if v % 7 != 0:
            print(v, "n'est pas divisible par 7.")
```

c. Il apparaît que seul ○★○★○★ est possible.

2. En effet, nul besoin d'un appareil électronique pour répondre à ce problème. Le motif ○★○★○★ répond à la question car il peut s'écrire $100000c+10000e+1000c+100e+10c+e$ qui peut se factoriser en $c\times 101010+e\times 10101=7\times 1443\times(10c+e)$. Pour les autres motifs, il suffit de prendre des contre-exemples, ils sont nombreux.

Retour sur l'exercice B8

1. a. Voici une proposition de fonction, dont l'explication est donnée en commentaire :

```
def retour(code, prop):
    reponse = ""
    for _ in range(4) :
        if code % 10 == prop % 10 : # dernier chiffre correct?
            reponse = 'o' + reponse
        else:
            reponse = 'x' + reponse
        code = code // 10    # On supprime le dernier chiffre
        prop = prop // 10    # Idem
    return reponse
```

b. Voici l'interface du jeu :

```
from random import randint
code_a_trouver = randint(0, 9999)
#print(code_a_trouver)    # Vous pouvez décommenter cette ligne
proposition = -1
essai = 0
while code_a_trouver != proposition and essai < 3 :
    proposition = int(input("Entrez un code"))
    rep = retour(code_a_trouver, proposition)
    print('Réponse de terminal :', rep)
    essai = essai + 1
if code_a_trouver != proposition:
    print("Désolé, votre carte est bloquée à présent !")
else:
    print("Bravo, vous avez trouvé le code !")
```

2. a. Une solution qui consiste à récupérer les chiffres un à un. On utilise la fonction `insert` plutôt que `append` pour éviter d'obtenir la liste "à l'envers".

```
def num2liste(n):
    liste = []
    while n != 0 :
        liste.insert(0, n % 10)
        n = n // 10
    return liste
```

b. On applique alors l'algorithme :

```
def teste(n) :
    L = num2liste(n)
    for i in range(0, len(L), 2):
        L[i] = 2 * L[i]
        if L[i] > 9 :
            L[i] = L[i] - 9
    somme = 0
    for chiffres in L :
        somme = somme + chiffres
    return somme % 10 == 0
```

c. Le premier code est faux et le deuxième est correct. Vous pouvez aussi tester avec votre propre carte bleue !

Retour sur l'exercice B9

1. Il suffit de reprendre les opérations indiquées :

```
def cle(N):
    return 97 - N % 97
```

```
>>> cle(2690549588157)
80
```

2. Pour cette question, on va réaliser une boucle : les informations sur la personne nous indiquent que le numéro de la carte commence par 2720160. Reste à trouver les 6 derniers chiffres en vérifiant la clé à chaque fois :

```
nb = 0
for N in range(2720160000000, 2720161000000):
    if cle(N) == 42:
        nb = nb + 1
print(nb)
```

On trouve 10 309 possibilités.

Retour sur l'exercice B10

1. a. Voici quelques idées, plus ou moins efficaces... On peut déjà réutiliser la fonction de l'exercice 6 de ce chapitre, qui pour un entier non nul, renvoie la liste de ses diviseurs positifs.

```
def liste_diviseurs(n):
    div = []
    for i in range(1, n+1):
        if n % i == 0:
            div.append(i)
    return div
```

Que l'on peut d'ailleurs écrire de manière plus élégante en compréhension :

```
def liste_diviseurs(n):
    return [i for i in range(1, n+1) if n % i == 0]
```

- Première idée :
  - ◇ On construit la liste `L1` des diviseurs du premier nombre.
  - ◇ On construit la liste `L2` des diviseurs du second nombre.
  - ◇ On construit la liste `L3` des éléments communs à `L1` et `L2`.

```
def pgcd1(a, b):
    L1 = liste_diviseurs(a)
    L2 = liste_diviseurs(b)
    L = []
    for d in L1:
        if d in L2:
            L.append(d)
    return max(L)
```

Ou encore en utilisant les listes en compréhension :

```
def pgcd1(a, b):
    L1 = liste_diviseurs(a)
    L2 = liste_diviseurs(b)
    L = [d for d in L1 if d in L2]
    return max(L)
```

Cette première idée fonctionne mais est assez gourmande en temps : il faut générer 3 listes pour ensuite chercher l'élément maximal.

- Deuxième idée :
  - ◇ On construit la liste `L1` des diviseurs du premier nombre.
  - ◇ On regarde parmi les éléments de `L1` ceux qui divisent le second nombre, cela réduit le nombre de tests.
  - ◇ Dans le point précédent, comme la liste `L1` est classée par ordre croissant, on commence le parcours par la fin , ainsi le premier diviseur trouvé est nécessairement le plus grand.

Remarquons que le dernier nombre testé sera toujours 1, donc on trouvera forcément un diviseur commun.

```
def pgcd2(a, b):
    L1 = liste_diviseurs(a)
    for i in range(len(L1)-1, -1, -1): #du dernier au premier
        if b % L1[i] == 0:             #Un diviseur commun ?
            return L1[i]
```

- Dernière idée : sans utiliser de liste, on cherche tous les diviseurs communs :
  - ◇ Comme dans le précédent algorithme, on commence par tester le plus grand diviseur possible.
  - ◇ On sait que le diviseur commun à `a` et `b` est inférieur à `a` et à `b` et donc au min(`a`, `b`).

```
def pgcd3(a, b):
    for d in range(min(a, b), 0, -1): # de min(a,b) à 1
        if a % d == 0 and b % d == 0:
            return d
```

b. En traduisant l'algorithme :

```
def pgcd4(a, b):
    while b != 0:
        a, b = b, a % b
    return a
```

c. En traduisant encore cette fois la définition :

```
def pgcd5(a, b):
    if b == 0:
        return a
    else:
        return pgcd5(b, a%b)
```

d. Si on souhaite comparer les temps d'exécution, voici une fonction qui mesure le temps :

- Elle enregistre l'heure (en seconde)
- Elle calcule tous les appels à la fonction `f(a, b)` pour tous les couples $(a, b) \in [\![1; n[\![^2$
- Elle regarde l'heure à nouveau et par soustraction renvoie le temps écoulé depuis le début d'appel.

```
def temps(f, nb) :
    debut = time()
    for a in range(1, nb) :
        for b in range(1, nb) :
            x = f(a, b)
    return round(time() - debut, 3)
```

Pour information, on obtient ce temps (en secondes)

n	pgcd1	pgcd2	pgcd3	pgcd4	pgcd5	gcd
100	0.063	0.038	0.016	0.003	0.004	0.001
1000	49.866	25.377	15.84	0.377	0.643	0.126
10000				53.908	92.082	15.051

AU DÉTOUR DE LA BALADE...

- *La fonction `gcd` du module `math` est bien plus rapide car le module `math` a été écrit en C et compilé ce qui donne des résultats bien meilleurs qu'une exécution d'un script ligne à ligne.*
- *Les 5 premières fonctions illustrent que le choix de l'algorithme est primordial.*
- *Enfin pour une même idée (4 et 5), on constate ici que la fonction récursive est moins efficace que la version itérative... Cela n'est pas une vérité universelle et dépend du langage utilisé.*

2. a. On va simplifier la fraction $\frac{a}{b}$ avec le PGCD(a,b).

```
def simplifieFrac(num, den):
    d = gcd(num, den)
    return num // d, den // d
```

AU DÉTOUR DE LA BALADE...

Attention, si vous utilisez la fonction `gcd` vous n'aurez pas de souci avec les numérateurs négatifs, mais selon la fonction utilisée, vous pourriez avoir des surprises avec les entiers négatifs ! Par exemple avec la fonction `pgcd4` précédente :

```
>>> pgcd4(3,-5)
-1
```

b. Une solution :

```
def ajouteFrac(n1, d1, n2, d2):
    return simplifieFrac(n1*d2 + d1*n2, d1*d2)

def multiplieFrac(n1, d1, n2, d2):
    return simplifieFrac(n1*n2, d1*d2)
```

On pourra vérifier que $\frac{1}{2}+\frac{1}{6}=\frac{2}{3}$ et $\frac{2}{3}\times\frac{5}{4}=\frac{5}{6}$:

```
>>> ajouteFrac(1,2,1,6)
(2, 3)
>>> multiplieFrac(2,3,5,4)
(5, 6)
```

3. On reconnaît une situation où il est nécessaire de calculer le PGCD de deux nombres. En utilisant une fonction précédente, on obtient $\text{PGCD}(186;155) = 31$. Le chocolatier pourra réaliser au maximum 31 colis contenant chacun 6 pralines et 5 chocolats.

Retour sur l'exercice B11

1. a. x et y sont des entiers naturels. Si on paie n euros, x et y sont dans l'intervalle $[\![0;n]\!]$. On peut affiner en tenant compte des prix : $x \in [\![0;\text{E}(n/24)]\!]$ et $y \in [\![0;\text{E}(n/45)]\!]$ où E est la partie entière d'un nombre.

   b. En traduisant les contraintes et les tarifs, on obtient ce code :

```
def nuitees(t):
    for x in range(t//24+1):
        for y in range(t//45+1):
            if 24*x + 45*y == t:
                print(x, y)
```

   c. On peut alors appeler `nuitees(438)` et on trouve que Laurence a passé 7 jours dans le premier hébergement et 6 dans le second. C'est l'unique solution.

2. a. Si on note t le prix à payer, on a

$$t = 24x + 45y = 3\underbrace{(8x + 15y)}_{\in\mathbb{N}}$$

   donc t est un multiple de 3.

   b. Par l'algorithme d'Euclide étendu ou simplement en cherchant un peu, on trouve que $(2,-1)$ est une solution :

$$8 \times \mathbf{2} + 15 \times \mathbf{(-1)} = 1$$

   c. Notons $(\text{E}) : 8x + 15y = 146$. D'après la question précédente, on peut écrire que

$$\begin{aligned}(x,y) \text{ solution de } (\text{E}) &\iff 8x + 15y = 146\\ &\iff 8x + 15y = 146 \times 1\\ &\iff 8x + 15y = 146 \times \big(8 \times 2 + 15 \times (-1)\big)\\ &\iff 8(x - 146 \times 2) = -15(y + 146 \times 1)\\ &\iff 8(x - 292) = -15(y + 146)\end{aligned}$$

On en déduit que 15 divise $8(x-292)$. Or 8 et 15 sont premiers entre eux. Le théorème de Gauss affirme qu'alors 15 divise $x-292$, c'est à dire que : $\exists k \in \mathbb{Z}$, tel que $x-292=15k$.

Reste à faire la synthèse de ce raisonnement :

Si $x=292+15k$, alors :

$$\begin{aligned}(x,y) \text{ solution de } (\mathrm{E}) &\iff 8(x-292)=-15(y+146)\\ &\iff 8\times 15k=-15(y+146)\\ &\iff 8k=-y-146\\ &\iff y=-146-8k\end{aligned}$$

Finalement l'ensemble des solutions de (E) dans $\mathbb{Z}^2$ sont les couples $(292+15k;-146-8k)$ où $k \in \mathbb{Z}$.

d. Ici on cherche à résoudre : $24x+45y=438$ qui est équivalente en divisant par 3 à $8x+15y=146$ qui n'est autre que l'équation précédente. Donc (x,y) sont les entiers naturels de la forme $(292+15k;-146-8k)$.

Or $292+15k \geqslant 0 \iff k \geqslant -\dfrac{292}{15}\ (\approx -19,5)$.

De même, $-146-8k \geqslant 0 \iff k \leqslant -\dfrac{146}{8}\ (\approx -18,3)$.

Le seul entier dans cet intervalle est -19.

Ainsi $(x,y)=(292-15\times 19;-146+8\times 19)=(7,6)$ Comme le programme de la première partie l'avait démontré.

Retour sur l'exercice B12

1. Une première idée est de créer une chaîne contenant les 26 caractères de `"a"` à `"z"`. La place du caractère dans la chaîne est alors sa valeur :

```
def alphanum(lettre):
    alphabet = "abcdefghijklmnopqrstuvwxyz"
    for i in range(26):
        if alphabet[i] == lettre:
            return i
```

Une autre solution, plus informatique, est d'utiliser la valeur ASCII du caractère. Comme celle de `"a"` est 97, il suffit de soustraire 97 au code ASCII de la lettre à coder et le tour est joué :

```
def alphanum(lettre):
    return ord(lettre) - 97
```

Pour la fonction réciproque, les mêmes pistes sont envisageables :

```
def numalpha(code):
    alphabet = "abcdefghijklmnopqrstuvwxyz"
    return alphabet[code]
```

Ou avec le code ASCII :

```
def numalpha(code):
    return chr(code + 97)
```

2. a. En traduisant l'algorithme :

```
def code_lettre(caractere, a, b):
    if "a" <= caractere <= "z":
        code = alphanum(caractere)
        code = (a * code + b) % 26
        return numalpha(code)
    else:
        return caractere
```

b. Il suffit alors de coder les lettres une à une :

```
def code_chaine(texte, a, b):
    reponse = ""
    for car in texte:
        reponse = reponse + code_lettre(car, a, b)
    return reponse
```

3. a. On teste toute les possibilités de clés :

```
def trouve_cle(a, b):
   for c in range(26):
     for d in range(26):
        trouve = True
        for code in range(26):
           car = numalpha(code)
           if code_lettre(code_lettre(car, a, b), c, d) != car:
              trouve = False
        if trouve :
           return (c, d)
   return False
```

b. On peut répondre à l'aide d'un code :

```
for b in range(26):
    for a in range(26):
        if trouve_cle(a, b) == False:
            print((a,b), end=";")
    print("")
```

```
(0, 0);(2, 0);(4, 0);(6, 0);(8, 0);(10, 0);(12, 0);(13, 0);(14, ↙
    ↘ 0);(16, 0);(18, 0);(20, 0);(22, 0);(24, 0);
(0, 1);(2, 1);(4, 1);(6, 1);(8, 1);(10, 1);(12, 1);(13, 1);(14, ↙
    ↘ 1);(16, 1);(18, 1);(20, 1);(22, 1);(24, 1);
(0, 2);(2, 2);(4, 2);(6, 2);(8, 2);(10, 2);(12, 2);(13, 2);(14, ↙
    ↘ 2);(16, 2);(18, 2);(20, 2);(22, 2);(24, 2);
...
(0, 23);(2, 23);(4, 23);(6, 23);(8, 23);(10, 23);(12, 23);(13, ↙
    ↘ 23);(14, 23);(16, 23);(18, 23);(20, 23);(22, 23);(24, 23);
(0, 24);(2, 24);(4, 24);(6, 24);(8, 24);(10, 24);(12, 24);(13, ↙
    ↘ 24);(14, 24);(16, 24);(18, 24);(20, 24);(22, 24);(24, 24);
(0, 25);(2, 25);(4, 25);(6, 25);(8, 25);(10, 25);(12, 25);(13, ↙
    ↘ 25);(14, 25);(16, 25);(18, 25);(20, 25);(22, 25);(24, 25);
```

On conclut que la condition se porte uniquement sur a : (a,b) est inversible si et seulement si a est premier avec 26.

AU DÉTOUR DE LA BALADE...

On pourra démontrer ce résultat en terminale maths expertes à l'aide du théorème de Bezout.

Retour sur l'exercice B13

1. Pour tracer la parabole :

```
import matplotlib.pyplot as plt

N = 3
x = -N
X, Y = [], []
while x < N :
    X.append(x)
    Y.append(x**2)
    x = x + 0.01
plt.plot(X, Y, 'k-')
plt.grid()
plt.show()
```

Ici à l'avant dernière ligne, on a ajouté un `plt.grid()` qui permet d'afficher une grille sur le repère.

2. Pas de difficulté pour cette question, on insère ces lignes avant le `show` pour voir les points bleus sur le même graphique.

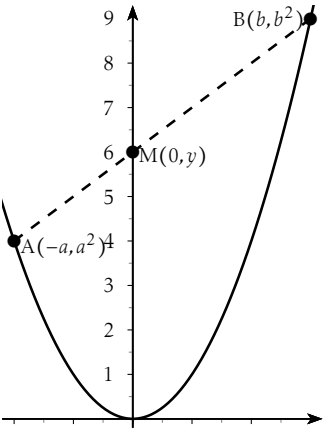
```
for i in range(2,N**2+1):
    plt.plot(0,i, 'bo')
```

3. a. Comme les extrémités sont placées sur la parabole, elles se trouvent en $(-a,(-a)^2) = (-a,a^2)$ et (b,b^2). Ainsi voici la fonction `segment` demandée :

```
def segment(a, b):
    plt.plot([-a,b], [a**2, b**2], 'g--')
```

b. Pour cette question, un petit calcul est nécessaire. Si on note $M(0,y)$ le point de l'axe cherché, $A(-a,a^2)$ et $B(b,b^2)$

Les vecteurs $\overrightarrow{AM}\begin{pmatrix} a \\ y-a^2 \end{pmatrix}$ et $\overrightarrow{AB}\begin{pmatrix} b+a \\ b^2-a^2 \end{pmatrix}$ sont colinéaires. Leur déterminant est donc nul :



$$\begin{vmatrix} a & b+a \\ y-a^2 & b^2-a^2 \end{vmatrix} = 0$$

$$\iff y\underbrace{(a+b)}_{\neq 0} = ab(a+b)$$

$$\iff y = ab$$

Donc $M(0,ab)$ et on obtient la nouvelle fonction :

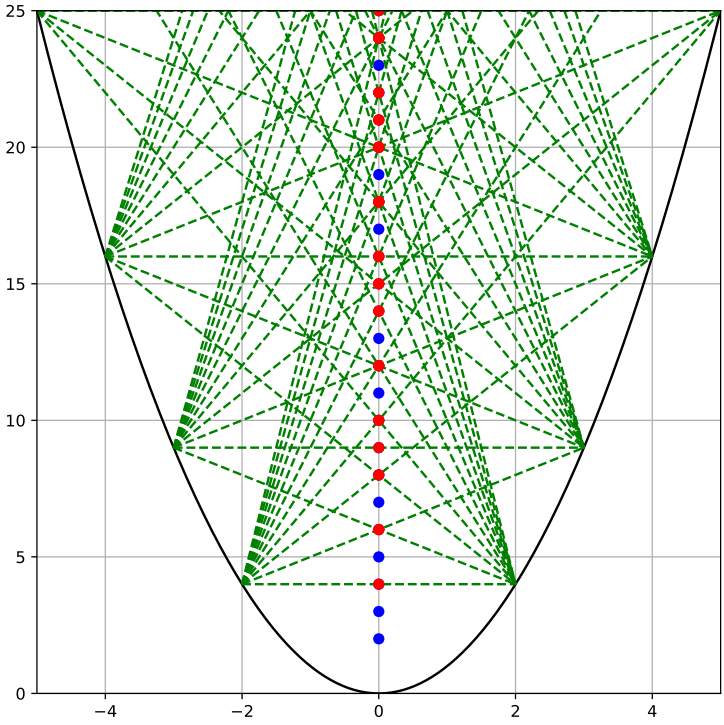
```
def segment(a, b):
    plt.plot([-a,b], [a**2, b**2], 'g--')
    plt.plot(0, a*b, 'ro')
```

c. Pour afficher tous les segments, on va utiliser une double boucle. Il est conseillé de recadrer la fenêtre. Voici le code dans son intégralité (il continue à la page suivante) :

```
from matplotlib import pyplot as plt

def segment(a, b):
    plt.plot([-a,b], [a**2, b**2], 'g--')
    plt.plot(0, a*b, 'ro')

N = 3
x = -N
X, Y = [], []
while x < N :
    X.append(x)
    Y.append(x**2)
    x = x + 0.01
plt.plot(X, Y, 'k-')
for i in range(2,N**2+1):
    plt.plot(0,i, 'bo')
```

```
for a in range(2, N**2//2+1) :
    for b in range(2, N**2//2+1) :
        segment(a,b)

plt.axis([-N,N,0,N**2])
plt.grid()
plt.show()
```

Et le résultat pour N = 5 :



4. Il semblerait que les ordonnées des points bleus soient tous les nombres premiers de l'intervalle $[0;N^2]$. En effet :

   - Les ordonnées des points rouges ne sont pas des nombres premiers car elles sont de la forme ab avec $b \geqslant 2$ et $a \geqslant 2$ donc elle possèdent un diviseur strict.
     À ce stade, on sait que les points rouges ne sont pas premiers, reste à montrer que les bleus le sont.
   - Au lieu de démontrer l'implication : "A point bleu $\Longrightarrow y_A$ est premier, on va démontrer la contraposée : Si y_A non premier Alors $A(0;y_A)$ n'est pas bleu (donc rouge). Cette implication est évidente.

Retour sur l'exercice B14

1. a. On peut remarquer qu'avec les carrés, si un couple (a,b) est solution, alors les couples $(-a,b)$, $(a,-b)$ et $(-a,-b)$ le sont aussi. Une fois qu'on a trouvé une solution, on peut en obtenir jusqu'à 3 autres dont l'une au moins est constituée d'entiers positifs. On peut donc se contenter de chercher les couples d'entiers naturels qui sont solutions.

   b. Une simple boucle suffit, on va chercher tous les diviseurs de 8633 en utilisant le reste de la division euclidienne :

```
for a in range(1, 8634) :
    if 8633 % a == 0 :
        b = 8633 // a
        print("8633=", a, "x", b)
```

   On trouve alors 4 décompositions possibles :

$$8633 = 1 \times 8633 = 89 \times 97 = 97 \times 89 = 8633 \times 1.$$

   c. $x^2 - 4y^2 = (x-2y)(x+2y)$, on doit donc résoudre :

$$(x-2y)(x+2y) = 8633$$

   Comme $x-2y$ et $x+2y$ sont entiers, on obtient :

$$(\mathrm{E}_1) \iff \begin{cases} x-2y=1 \\ x+2y=8633 \end{cases} \text{ ou } \begin{cases} x-2y=89 \\ x+2y=97 \end{cases}$$

$$\text{ou } \begin{cases} x-2y=97 \\ x+2y=89 \end{cases} \text{ ou } \begin{cases} x-2y=8633 \\ x+2y=1 \end{cases}$$

   Puisque $x-2y \leqslant x+2y$, les 2 derniers cas sont impossibles. Il reste :

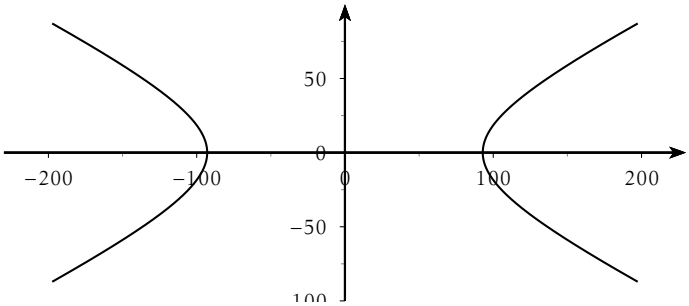
$$(\mathrm{E}_1) \iff \begin{cases} x-2y=1 \\ x+2y=8633 \end{cases} \text{ ou } \begin{cases} x-2y=89 \\ x+2y=97 \end{cases}$$

$$\iff \begin{cases} x=4317 \\ y=2158 \end{cases} \text{ ou } \begin{cases} x=93 \\ y=2 \end{cases}$$

   Finalement, on obtient 8 couples d'entiers solutions :

$$\mathcal{S} = \{(\pm 4317; \pm 2158), (\pm 93, \pm 2)\}$$

   Pour information, si on cherche l'ensemble des réels x et y, on a une infinité de solutions qui représentent les coordonnées des points d'une hyperbole représentée ci-dessous. Pour notre exercice, seuls les points à coordonnées entières sont concernés.



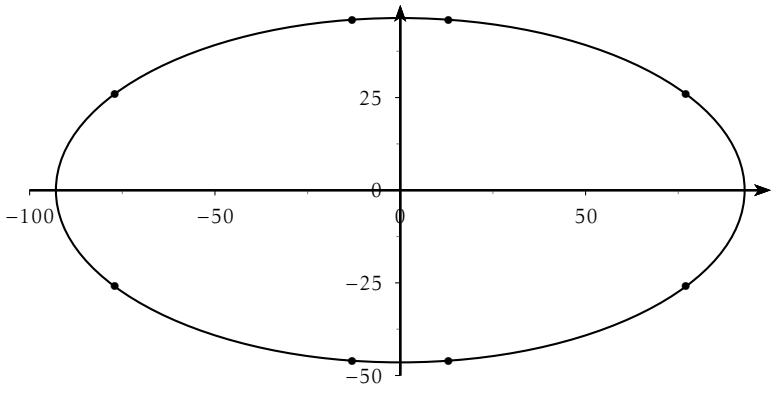
2. a. Cette fois-ci, on ne sait pas factoriser $x^2 + 4y^2$ (en fait cette expression n'est pas factorisable dans $\mathbb{R}$). Cependant le fait que l'on ajoute des quantités positives nous indique que le nombre de solutions est forcément limité. En effet, $\sqrt{8633} \approx 92,9$, ainsi si $x > 93$ ou $y > 93$, $x^2 + 4y^2$ sera strictement supérieur à 8633. Il y a donc un nombre fini de valeurs possibles pour x et pour y que nous allons tester une à une à l'aide de deux boucles (on peut limiter y à 46, mais cela ne change pas grand-chose ici).

b.
```
for x in range(94):
    for y in range(94):
        if x ** 2 + 4 * y ** 2 == 8633:
            print(x, ", ", y)
```

On trouve 8 solutions :

$$\mathcal{S} = \{(\pm 13; \pm 46), (\pm 77; \pm 26)\}$$

Encore une fois, si on avait cherché les solutions réelles de cette équation, on aurait trouvé une infinité de solutions (bornées) qui représentent les points d'une ellipse :



Retour sur l'exercice B15 En reprenant l'algorithme proposé dans l'aide, on obtient le programme suivant :

```
def somme_carres(longueur, largeur):
    print(longueur, " x ", largeur, " = ", sep="", end="")
    while longueur != largeur:
        print(largeur, "² + ", end="", sep="")
        longueur = longueur - largeur
        if longueur < largeur:
            largeur, longueur = longueur, largeur
    print(largeur, "²", sep="")
```

Retour sur l'exercice B16

1. Si n est le nombre de chiffres du nombre N, $10^{n-1} \leqslant \mathrm{N} < 10^n$ et la somme des cubes des chiffres $\mathrm{S(N)} \leqslant n \times 9^3 = 729n$. Donc
   $\mathrm{S(N)} = \mathrm{N} \Longrightarrow 10^{n-1} \leqslant 729n \Longrightarrow 10^{n-1} - 729n \leqslant 0$.
   Posons $f(x) = 10^{x-1} - 729x$, sur $]0;+\infty[$. $f'(x) = \ln(10)10^{x-1} - 729$ et
   $$\begin{aligned} f'(x) \geqslant 0 &\iff \ln(10)10^{x-1} - 729 \geqslant 0 \\ &\iff 10^{x-1} \geqslant \frac{729}{\ln 10} \\ &\iff (x-1)\ln 10 \geqslant \ln\left(\frac{729}{\ln 10}\right) \\ &\iff (x-1) \geqslant \frac{1}{\ln 10}\ln\left(\frac{729}{\ln 10}\right) \\ &\iff x \geqslant \frac{1}{\ln 10}\ln\left(\frac{729}{\ln 10}\right) + 1 = \alpha(\approx 3{,}5). \end{aligned}$$
   Ainsi on a le tableau suivant :

x	0		α		5		$+\infty$
$f'(x)$		$-$	0	$+$			
f	$\frac{1}{10}$	↘	$f(\alpha)$	↗	6355	↗	$+\infty$

   Il est donc inutile de chercher des nombres à plus de 4 chiffres.

2. Une solution consiste à utiliser le reste `n%10` pour récupérer le chiffre des unités :

```
def somme_cubes(n) :
    somme = 0
    while n > 0 :
        somme = somme + (n%10)**3
        n = n // 10
    return somme
```

Une autre solution récursive utilisant le même principe :

```
def somme_cubes(n) :
    if n == 0 :
        return 0
    else:
        return (n%10)**3 + somme_cubes(n//10)
```

3. On sait que les solutions doivent avoir moins de 4 chiffres, il suffit donc de les tester toutes ! Cela tient en 3 lignes :

```
for n in range(10**4):
    if n == somme_cubes(n):
        print(n)
```

Retour sur l'exercice B17

1. a. Comme $2^{11} = 2048$, a, b et c ne peuvent dépasser 10.

   b. On en déduit un programme qui teste toutes les possibilités :

```
for a in range(11):
   for b in range(a+1, 11):
      for c in range(b+1, 11):
         if 2**a + 2**b + 2**c == 1120:
            print("2^", a, "+2^", b, "+2^", c, "=1120", sep="")
```

   On trouve alors que $2^5 + 2^6 + 2^{10} = 1120$.

2. Parfois une vidéo est plus efficace qu'un long discours. Sur leur page youtube, Iman et David, font vivre les mathématiques à travers de nombreuses vidéos dont cet exercice s'est inspiré :

3. a. En tapant dans la console :

```
>>> bin(1120)
'0b10001100000'
```

   b. On lit (de droite à gauche)

$$1120 = 0.2^0 + 0.2^1 + 0.2^2 + 0.2^3 + 0.2^4 + 1.2^5 + 1.2^6 + 0.2^7 + 0.2^8 + 0.2^9 + 1.2^{10}$$

$$\text{Soit } 1120 = 2^5 + 2^6 + 2^{10}$$

Chapitre C

GÉOMÉTRIE

1. Quelques triangles et quadrilatères

C1 *Créer une fonction qui indique par un booléen si un quadrilatère, dont on renseigne les coordonnées des sommets, est un parallélogramme ou non.*

C2 *Soient A, B, C trois points dans un repère orthonormé de coordonnées respectives* (x_A, y_A), (x_B, y_B) *et* (x_C, y_C).

1. a. *Écrire une fonction qui reçoit 6 réels* `xA`, `yA`, `xB`, `yB`, `xC` *et* `yC` *et qui calcule et renvoie l'angle géométrique* $\widehat{BAC}$ *en degrés, arrondi au centième.*

   b. *À l'aide de ce programme déterminer l'angle géométrique en degrés avec* $A(-5;-1)$, $B(-1;-6)$ *et* $C(-5;-3)$.

   c. *Même question avec* $A(-2;3)$, $B(2;-1)$ *et* $C(2\sqrt{3};1+2\sqrt{3})$. *En déduire la nature du triangle* ABC.

2. a. *À l'aide de la fonction précédente, écrire une fonction* `nature` *qui reçoit les mêmes paramètres que la fonction* `angle`, *et qui affiche la nature du triangle* ABC *à partir du calcul de ses angles.*

   b. *Donner la nature du triangle* ABC *sachant que* $A(-4;1)$, $B(-1;2)$ *et* $C(1;-4)$.

   c. *Même question avec* $A(-2;4)$, $B(4;4)$ *et* $C(4;10)$.

C3 *Dans cet exercice, on se place dans le plan muni d'un repère orthonormé et on cherche à trouver des relations de colinéarité et d'orthogonalité d'un ensemble de vecteurs donnés.*

1. *Écrire une fonction* `indice` *qui reçoit une liste* `L` *et un élément* `e` *et renvoie l'indice de la première occurrence de* `e` *dans la liste* `L`*. Si* `e` *n'est pas présent, la fonction renverra -1 par convention.*

2. *Écrire une fonction* `colineaires` *qui reçoit en paramètres 4 nombres* `x1`*,* `y1`*,* `x2` *et* `y2` *et qui renvoie un booléen indiquant si les vecteurs* $\vec{u}\begin{pmatrix} x1 \\ y1 \end{pmatrix}$ *et* $\vec{v}\begin{pmatrix} x2 \\ y2 \end{pmatrix}$ *sont colinéaires ou non.*

3. *Dans la suite, on considère l'ensemble de vecteurs*

$$\left\{\vec{u}\begin{pmatrix}-2\\4\end{pmatrix}, \vec{v}\begin{pmatrix}4\\4\end{pmatrix}, \vec{w}\begin{pmatrix}-6\\4\end{pmatrix}, \vec{t}\begin{pmatrix}8\\1\end{pmatrix}, \vec{z}\begin{pmatrix}-2\\2\end{pmatrix}, \vec{n}\begin{pmatrix}-3\\2\end{pmatrix}, \vec{m}\begin{pmatrix}-15\\10\end{pmatrix}\right\}$$

Compléter la fonction suivante pour qu'elle reçoive en paramètre le nom d'un vecteur de l'ensemble (sous forme d'une chaine) et retourne la liste de tous les vecteurs colinéaires à celui-ci.

```
def liste_col(vecteur) :
    noms = ['u', 'v', 'w', 't', 'z', 'n', 'm']
    X = [-2, 4, -6, 8, -2, -3, -15]
    Y = [4, 4, 4, 1, 2, 2, 10]
    reponse = []
    i = indice( ..... , ..... )
    for j in range( ..... ) :
        if colineaires(X[ ..... ], Y[ ..... ], X[ ..... ], Y[ ..... ]) :
            reponse.append( ..... )
    return reponse
```

4. *Que faudrait-il changer pour obtenir la liste des vecteurs orthogonaux à un vecteur donné de l'ensemble ? Dresser une table des relations entre les vecteurs.*

C4 *Dans le plan complexe ramené à un repère orthonormé* $(O;\vec{u},\vec{v})$*, on considère trois points* A, B *et* C *distincts deux à deux d'affixes respectives* z_A*,* z_B *et* z_C*.*

1. a. *Écrire une fonction qui reçoit en paramètres l'affixe des deux points* A *et* B *et renvoie l'affixe du vecteur* $\overrightarrow{AB}$*.*

b. *En déduire une seconde fonction ayant les même paramètres mais qui renvoie* $\|\overrightarrow{AB}\|$.

2. *On considère le quotient de complexes* $\mathcal{Z} = \dfrac{z_C - z_A}{z_B - z_A}$. *En utilisant le module et un argument de* $\mathcal{Z}$, *écrire une fonction qui reçoit en paramètres l'affixe des trois points 2 à 2 distincts* A, B *et* C *et renvoie une chaîne de caractères indiquant le fait que le triangle soit isocèle en* A, *rectangle en* A, *équilatéral ou quelconque (on pourra avoir recours à la fonction* `arg` *de la fiche sur les complexes).*

3. *Créer une banque de tests pertinents pour vérifier les cinq situations possibles.*

2. Et des droites

C5 *Le plan étant muni d'un repère orthonormé, on donne deux droites* (d_1) *et* (d_2) *d'équations cartésiennes respectives* $d_1 : ax + by + c = 0$ *et* $d_2 : dx + ey + f = 0$ *où* a, b, c, d, e *et* f *sont des entiers.*

1. *Écrire une fonction qui reçoit ces 6 nombres* a, b, c, d, e *et* f *et qui renvoie une chaîne de caractères parmi* `"strictement parallèles"`, `"confondues"`, `"sécantes"`, `"perpendiculaires"` *indiquant les positions relatives de ces deux droites.*

2. *Créer une fonction qui affiche la représentation graphique de la droite d'équation cartésienne* $ax + by + c = 0$ *dans la fenêtre* $[-10;10] \times [-10;10]$ *pour trois réels* a, b *et* c *donnés en paramètres et avec* $(a, b) \neq (0, 0)$. *Attention de prendre un repère orthonormé pour voir la perpendicularité !*

3. *Tracer les droites, compléter par lecture graphique le tableau en indiquant les positions relatives des droites. Vérifier ensuite à l'aide de la fonction précédemment réalisée.*

droite	*équation*
D_1	$x + 2 = 0$
D_2	$-x + 2y + 4 = 0$
D_3	$2x + y + 2 = 0$
D_4	$x - 2y + 1 = 0$
D_5	$2x + 2x + 2 = 0$
D_6	$y - 3 = 0$
D_7	$-x - y - 1 = 0$

	D_1	D_2	D_3	D_4	D_5	D_6	D_7
D_1							
D_2							
D_3							
D_4							
D_5							
D_6							
D_7							

C6 *On se place dans le plan muni d'un repère orthonormé. On cherche à réaliser une fonction qui affiche l'équation réduite de la droite* (AB) *connaissant les coordonnées des points* A *et* B.

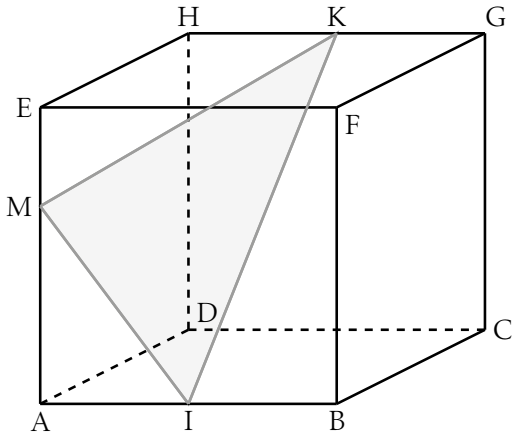
1. *Tracer sur une feuille les droites* (AB) *en utilisant les coordonnées données ci-dessous et compléter le tableau :*

A	B	*équation de la droite* (AB)
$(-1;-2)$	$(3;0)$	
$(-2;-2)$	$(-2,3)$	
$(-1;2)$	$(4;2)$	
$(-1;1)$	$(2;-2)$	

2. *En déduire une fonction qui reçoit les coordonnées de deux points* A *et* B *et qui affiche l'équation réduite de la droite* (AB).

3. Rendez-vous dans l'espace

C7 ABCDEFGH *est un cube dont l'arête mesure 8cm.* I *et* K *sont les milieux respectifs des arêtes* [AB] *et* [HG]. M *est un point de l'arête* [AE].



L'objectif de l'exercice est de déterminer la position du point M *pour que le triangle* IKM *soit rectangle en* M *selon 2 pistes différentes :*

1. *Piste informatique :*

   a. *Calculer* KI^2.

b. *On pose* AM = x. *Créer une fonction notée* `f` *qui prend pour paramètre une valeur de* x *avec* $x \in [0,8]$ *et qui renvoie le résultat de* $MI^2 + MK^2$.

c. *Tracer, à l'aide de Python, la courbe de la fonction* `f` *et conjecturer la valeur de* x *pour que le triangle* IKM *soit rectangle en* M.

2. *Piste calcul mathématique :*

a. *À l'aide des questions précédentes, déterminer une équation que doit vérifier le réel* x.

b. *Résoudre cette équation et conclure sur la valeur de* x *cherchée.*

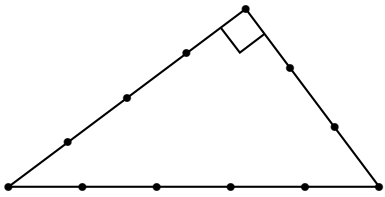
C8 *Soit un point* A *dans un repère orthonormé* $(O;\vec{i},\vec{j},\vec{k})$.

1. *Créer une fonction qui reçoit l'abscisse, l'ordonnée et la cote d'un point* A *et renvoie un booléen indiquant si ce point est ou non sur la sphère de centre* O *et de rayon* 23.

2. *Vérifier que ce code fonctionne en comparant avec une calculatrice dans les situations suivantes :* A(3;14;18), A(10;13;0). *Que constate-t-on pour* A(6,44;0;22,08) *? Émettre une hypothèse sur la cause concernant cette constatation.*

4. Des classiques

C9 *La corde à 13 nœuds ou corde des druides.*



Il s'agit d'une corde connue depuis le temps de l'Égypte ancienne et formée de 12 intervalles de même longueur délimités par 13 nœuds. Elle permet de manière très pratique de réaliser des constructions, en particulier des angles droits pour réaliser des édifices. On cherche à réaliser d'autres cordes de druide.

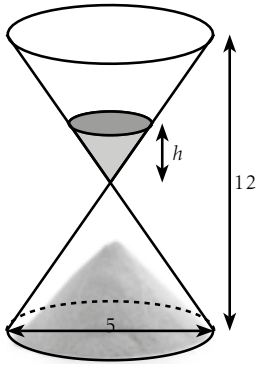
1. *Justifier qu'avec cette corde à 13 nœuds on peut en effet construire des triangles rectangles.*

2. *Proposer une fonction qui, connaissant le nombre total de nœuds, renvoie* 0 *si l'on ne peut pas construire de triangle rectangle avec cette corde et* 1 *lorsque c'est possible en affichant dans ce cas la taille des segments à prendre.*

3. *Peut-on imaginer une corde sur le même principe avec moins de nœuds ?*

4. *Créer un programme qui affiche* 10 *possibilités de cordes avec des nombres de nœuds différents.*

C10 *Un sablier de hauteur totale* 12*cm est constitué de deux cônes de révolution identiques. Le diamètre de chaque base est* 5*cm. Au départ, la hauteur du sable est de* h *cm dans le cône du haut où* h *est un nombre inférieur à* 6*. Le sable s'écoule régulièrement à raison de* $1,6 cm^3$ *par minute.*



1. *Créer une fonction Python qui reçoit en paramètres le rayon* r *et la hauteur* h *d'un cône de révolution et renvoie le calcul du volume de ce cône.*

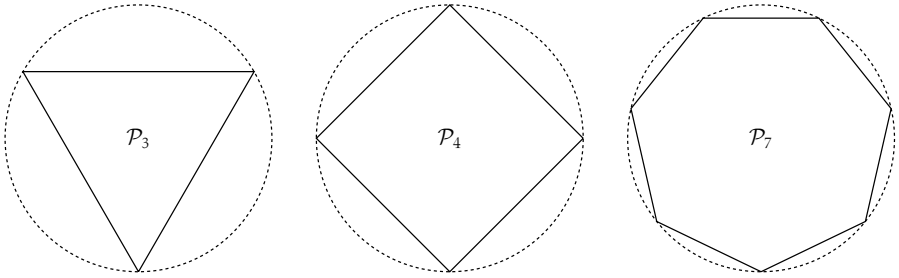
2. *En déduire une valeur approchée du volume de sable présent dans la partie haute, sachant qu'on a pu constater que le cône de sable a un diamètre de* $2,5cm$*.*

3. *En déduire la durée en secondes pour que la totalité de ce sable passe dans le cône du bas.*

5. Si vous n'êtes pas encore fatigué, des défis

Les défis de cette dernière section sont à réaliser avec le module `turtle` de Python. Une fiche est consacrée à ce module en fin de livre.

C11 *Soit* $\mathcal{C}$ *un cercle de rayon* 100*. Pour* $n \geqslant 3$*, on veut construire à l'aide du mode tortue un polygone* $\mathcal{P}_n$ *régulier à* n *côtés inscrit dans* $\mathcal{C}$ *comme l'illustrent les figures de la page suivante.*



Soit n un entier supérieur ou égal à 3.

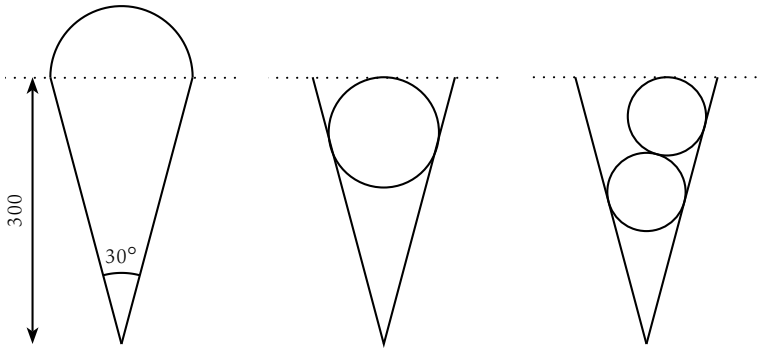
1. *Calculer les longueurs et les angles nécessaires afin d'écrire une fonction* `polynome` *qui reçoit un entier $n \geqslant 3$ et réalise la figure demandée.*

2. *Exprimer le périmètre $\mathcal{L}_n$ du polygone $\mathcal{P}_n$ en fonction de n et modifier la fonction pour qu'elle affiche cette valeur.*

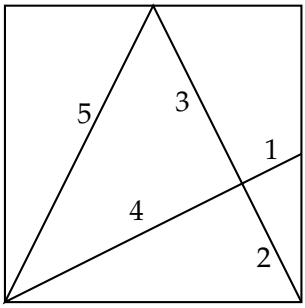
3. *Faire de même pour l'aire $\mathcal{A}_n$ du polygone $\mathcal{P}_n$.*

4. *Quelles semblent être les limites de $\mathcal{A}_n$ et $\mathcal{L}_n$? Cela était-il prévisible ?*

C12 *Comment dessiner ces 3 cornets de glace fort appétissants, sachant que la hauteur d'un cornet vide est de 300 pas et que l'angle à la base mesure 30° ?*



C13 *Après avoir calculé les longueurs et les angles nécessaires, tracer cette figure inscrite dans un carré et dont les mesures sont indiquées sur le graphique en essayant de ne passer qu'une fois sur chaque segment.*

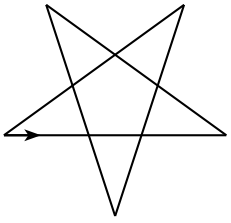


C14

D'après une idée de Benoit DUCANGE

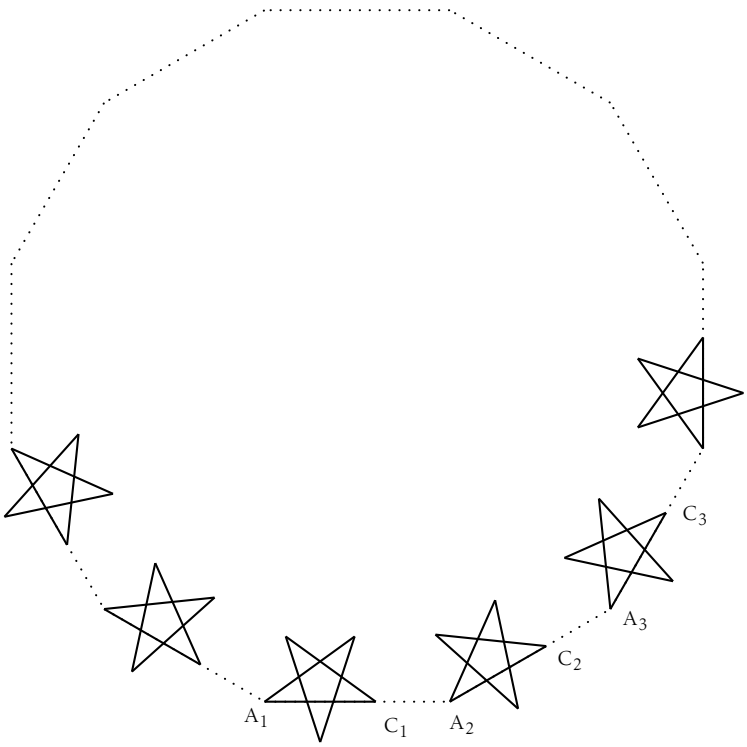
On souhaite dessiner le drapeau européen à l'aide de la tortue.

1. *Créer une fonction* `etoile` *qui dessine une étoile dont les branches mesurent 30 pas et où la tortue retrouve sa place initiale à la fin du tracé.*



2. *On choisit de disposer les étoiles sur les côtés d'un dodécagone virtuel de 50 pas de côté comme sur la figure ci-dessous. Déterminer l'angle* $\widehat{A_1A_2C_2}$.

3. *Réaliser la figure.*



6. Aide pour les exercices

Aide pour l'exercice C1 On peut caractériser un parallélogramme à l'aide d'égalités vectorielles ou encore par le fait que ses diagonales se coupent en leur milieu.

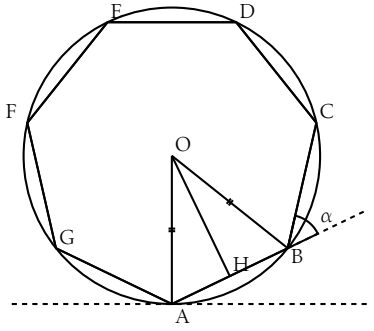
Aide pour l'exercice C2 On peut utiliser le produit scalaire ou la formule d'Al-Kashi.

Aide pour l'exercice C6 Il faut distinguer le cas où (AB) est parallèle à l'axe des ordonnées des autres cas pour lesquels on peut calculer le coefficient directeur avec la formule $m = \dfrac{y_B - y_A}{x_B - x_A}$. Reste ensuite à trouver l'ordonnée à l'origine.

Aide pour l'exercice C8 Pour qu'un point A soit sur la sphère de centre O et de rayon 23, il faut et il suffit que la distance OA soit égale à 23.

Aide pour l'exercice C10 Pour la seconde question, pensez que si on connaît le diamètre, on peut retrouver la hauteur à l'aide du cône de départ.

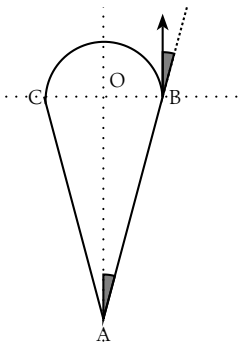
Aide pour l'exercice C11 Une petite figure devrait pouvoir nous aider... ici $n = 7$.



On y trouve beaucoup de triangles particuliers (rectangles, isocèles) permettant de faire de la trigonométrie. Pour commencer, on peut remarquer que l'on a $OA = OB = OC = 100$ et $\widehat{AOB} = \dfrac{360°}{n}$. Le but est de calculer AB et α...

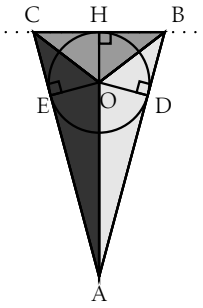
Aide pour l'exercice C12 Amateurs de trigonométrie, vous allez être servis ! Voici quelques pistes, avec une proposition de cheminement à suivre si vous êtes bloqués.

1.



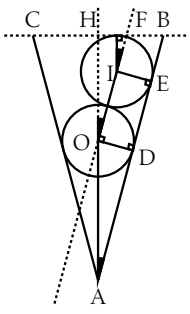
- Les angles dessinés en gris mesurent tous 15°.
- On peut calculer AB et OB.

2. L'objectif est de déterminer le rayon r du cercle à tracer. Voici une démarche basée sur l'observation que la somme des aires des 3 triangles coloriés vaut l'aire du triangle ABC.



- Calculer BC et l'aire $\mathcal{A}_{ABC}$.
- Calculer AB.
- Calculer les aires $\mathcal{A}_{ABO}$, $\mathcal{A}_{ACO}$ et $\mathcal{A}_{CBO}$ en fonction de r. En déduire r.
- Calculer AD.

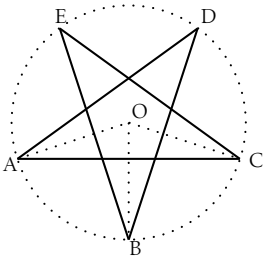
3. Les angles marqués en noir mesurent tous $15°$. On note r le rayon des cercles.



- Exprimer AO en fonction de r.
- Exprimer IF en fonction de r.
- Exprimer OF en fonction de r.
- Exprimer OH en fonction de r.
- En remarquant que AO + OH = AH, en déduire r.
- Calculer AD.

Aide pour l'exercice C14 Pour la première question.

En notant O le centre de l'étoile et A, B, C, D, E ses sommets, on sait que $\widehat{AOB} = \dfrac{360°}{5} = 72°$. On peut alors en déduire les mesures des angles $\widehat{ACO}$ puis $\widehat{ACE}$...



7. Solutions des exercices

Retour sur l'exercice C1 Si on note ABCD ce quadrilatère, on sait que :

ABCD est un parallélogramme $\iff \overrightarrow{AB} = \overrightarrow{DC} \iff \begin{cases} x_B - x_A = x_C - x_D \\ y_B - y_A = y_C - y_D \end{cases}$.

Ainsi, en faisant attention à l'ordre des points :

```
def parallelogramme(xA, yA, xB, yB ,xC, yC, xD, yD) :
    return xB - xA == xC - xD and yB - yA == yC - yD
```

Retour sur l'exercice C2

1. a. On peut par exemple utiliser le théorème d'Al-Kashi :

$$BC^2 = AB^2 + AC^2 - 2 \times AB \times AC \times \cos\left(\widehat{BAC}\right)$$

D'où on peut déduire que :

$$\cos\left(\widehat{BAC}\right) = \frac{AB^2 + AC^2 - BC^2}{2 \times AB \times AC}$$

Comme on va avoir 3 distances à calculer, on peut créer une fonction qui reçoit A et B et renvoie AB^2. On peut alors réaliser la fonction demandée :

```
from math import sqrt, acos, degrees
def distance2(x1, y1, x2, y2):
    return (x2-x1)**2 + (y2-y1)**2

def angle(xA, yA, xB, yB, xC, yC):
    a2 = distance2(xB, yB, xC, yC)
    b2 = distance2(xA, yA, xC, yC)
    c2 = distance2(xB, yB, xA, yA)

    cosA = (c2+b2-a2)/(2*sqrt(c2*b2))
    A = acos(cosA)
    return round(degrees(A), 2)
```

AU DÉTOUR DE LA BALADE...

On aurait pu aussi utiliser le produit scalaire :

$$\vec{u} \cdot \vec{v} = \left\|\vec{u}\right\| \times \left\|\vec{v}\right\| \times \cos\left(\vec{u}, \vec{v}\right)$$

Ce qui permet aussi d'extraire le cosinus.

b. $\widehat{BAC} \approx 38,66°$:

```
>>> angle(-5, -1, -1, -6, -5, -3)
38.66
```

c. De la même manière :

```
>>> angle(-2, 3, 2, -1, 2*sqrt(3), 1+2*sqrt(3))
60.0
>>> angle(2, -1, 2*sqrt(3), 1+2*sqrt(3), -2, 3)
60.0
>>> angle(2*sqrt(3), 1+2*sqrt(3), -2, 3, 2, -1)
60.0
```

Le triangle ABC semble donc équilatéral (aux erreurs d'arrondi près)

2.

```
def nature(xA, yA, xB, yB, xC, yC):
    A = angle(xA, yA, xB, yB, xC, yC)
    B = angle(xB, yB, xC, yC, xA, yA)
    C = angle(xC, yC, xA, yA, xB, yB)
    if A == B == C:
        print("Le triangle est équilatéral")
    else :
        if A == B:
            if C == 90:
                print("Le triangle est isocèle et rectangle en C")
            else:
                print("Le triangle est isocèle en C")
        elif B == C:
            if A == 90:
                print("Le triangle est isocèle et rectangle en A")
            else:
                print("Le triangle est isocèle en A")
        elif C == A:
            if B == 90:
                print("Le triangle est isocèle et rectangle en B")
            else:
                print("Le triangle est isocèle en B")
        elif A == 90:
            print("Le triangle est rectangle en A")
        elif B == 90:
            print("Le triangle est rectangle en B")
        elif C == 90:
            print("Le triangle est rectangle en C")
        else:
            print("Le triangle est quelconque")
```

Que l'on peut raccourcir (en informatique, on dit que l'on a **factorisé** le code) :

```
def nature(xA, yA, xB, yB, xC, yC):
    A = angle(xA, yA, xB, yB, xC, yC)
    B = angle(xB, yB, xC, yC, xA, yA)
    C = angle(xC, yC, xA, yA, xB, yB)
    if A == B == C:
        print("Le triangle est équilatéral")
    else:
        phrase = ""
        if A == B:
            phrase = "isocèle en C"
        elif B == C:
            phrase = "isocèle en A"
        elif A == C:
            phrase = "isocèle en B"
        if phrase != "" and max(A, B, C) == 90:
            phrase = "rectangle et " + phrase
        elif A == 90:
            phrase = "rectangle en A"
        elif B == 90:
            phrase = "rectangle en B"
        elif C == 90:
            phrase = "rectangle en C"
        else:
            phrase = "quelconque"
        print("Le triangle est " + phrase)
```

On peut alors répondre aux deux dernières questions :

```
>>> nature(-4, 1, -1, 2, 1, -4)
Le triangle est rectangle en B
>>> nature(-2, 4, 4, 4, 4, 10)
Le triangle est rectangle et isocèle en B
```

AU DÉTOUR DE LA BALADE...

Ici on a fait le choix de proposer une solution à l'aide des angles (en prenant garde d'arrondir). En terminale, on pourrait utiliser les nombres complexes, ou même simplement les longueurs si on réalise cette activité en seconde.

Retour sur l'exercice C3

1. Pour cette question, on réalise un parcours de la liste et on renvoie la position dès que l'élément en question est trouvé. Si à la fin de la boucle l'élément n'a pas été trouvé, on renvoie -1 :

```
def indice(L, e):
    for i in range(len(L)):
        if L[i] == e:
            return i
    return -1
```

2. La deuxième question est une application du cours avec le calcul du déterminant :

```
def colineaires(x1, y1, x2, y2) :
    return x1 * y2 - x2 * y1 == 0
```

3. Voici une solution :

```
def liste_col(vecteur) :
    noms = ['u', 'v', 'w', 't', 'z', 'n', 'm']
    X = [-2, 4, -6, 8, -2, -3, -15]
    Y = [4, 4, 4, 1, 2, 2, 10]
    reponse = []
    i = indice(noms, vecteur)
    for j in range(len(noms)) :
        if colineaires(X[i], Y[i], X[j], Y[j]) :
            reponse.append(noms[j])
    return reponse
```

4. Pour tester l'orthogonalité, il faudrait modifier le test de la fonction `colineaires` en `x1 * x2 + y1 * y2 == 0` pour tester si le produit scalaire est nul.

   On obtient alors ces relations :

	$\vec{u}$	$\vec{v}$	$\vec{w}$	$\vec{t}$	$\vec{z}$	$\vec{n}$	$\vec{m}$
$\vec{u}$	$\parallel$			$\perp$			
$\vec{v}$		$\parallel$					
$\vec{w}$			$\parallel$			$\parallel$	$\parallel$
$\vec{t}$	$\perp$			$\parallel$			
$\vec{z}$					$\parallel$		
$\vec{n}$			$\parallel$			$\parallel$	$\parallel$
$\vec{m}$			$\parallel$			$\parallel$	$\parallel$

Où $\parallel$ signifie colinéaire.

AU DÉTOUR DE LA BALADE...

Ici, ce n'est pas pertinent de modifier la définition de `colineaires` *pour définir l'orthogonalité, mieux vaut placer en paramètre le test que l'on souhaite réaliser ainsi :*

```
def colineaires(x1, y1, x2, y2) :
    return x1 * y2 - x2 * y1 == 0

def orthogonaux(x1, x2, y1, y2):
    return x1 * x2 + y1 * y2 == 0

def liste(vecteur, relation) :
    noms = ['u', 'v', 'w', 't', 'z', 'n', 'm']
    X = [-2, 4, -6, 8, -2, -3, -15]
    Y = [4, 4, 4, 1, 2, 2, 10]
    reponse = []
    i = indice(noms, vecteur)
    for j in range(len(noms)) :
        if relation(X[i], Y[i], X[j], Y[j]) :
            reponse.append(noms[j])
    return reponse
```

Ainsi on peut appeler

```
>>> liste('t', orthogonaux)
['u']
>>> liste('n', colineaires)
['w', 'n', 'm']
```

Cependant, même si informatiquement c'est bien pratique, il est compliqué pour des élèves de lycée de placer une fonction en paramètre, d'autant que celle-ci renverra un booléen servant de test.

Retour sur l'exercice C4

1. Pas de difficulté pour cette question, si on se souvient que $z_{\overrightarrow{AB}} = z_B - z_A$:

```
def affixe(zA, zB):
    return zB-zA

def norme(zA, zB):
    return abs(affixe(zA, zB))
```

2. Pour cette seconde question, le cours sur les nombres complexes nous indique que $|\mathcal{Z}| = \dfrac{AC}{AB}$ et que $\arg(\mathcal{Z}) \equiv \left(\overrightarrow{AB}, \overrightarrow{AC}\right)[2\pi]$. Ainsi on peut coder avec la fonction `arg` de la fiche sur les complexes :

```
from math import acos, pi

def arg(z):
    r = abs(z)
    angle = acos(z.real/r)
    if z.imag >= 0:
        return angle
    else:
        return -angle

def nature(zA, zB, zC):
    Z = (zC-zA) / (zB-zA)
    module, angle = abs(Z), arg(Z)
    if module == 1:
        if abs(angle) == pi / 3:
            return "Le triangle est équilatéral"
        elif abs(angle) == pi / 2:
            return "Le triangle est rectangle et isocèle en A"
        else:
            return "Le triangle est isocèle en A"
    elif abs(angle) == pi / 2:
        return "Le triangle est rectangle en A"
    else :
        return "Le triangle est quelconque"
```

3. Pensez qu'un triangle peut être rectangle et isocèle ! Mais déjà en testant un triangle censé être équilatéral :

```
>>> nature(-2, 1+1j*sqrt(3), 1-1j*sqrt(3))
'Le triangle est quelconque'
```

Cela est toujours dû aux problèmes d'arrondi, il faudrait par exemple remplacer tous les tests d'égalité type `a == b` par des inégalités `abs(b-a) < 10**-7`. On peut ainsi faire prendre conscience aux élèves des limites des tests d'égalité impliquant des nombres flottants.

Retour sur l'exercice C5

1. On sait que si une droite $\mathcal{D}$ a pour équation : $ax + by + c = 0$ alors le vecteur $\vec{n}\begin{pmatrix} a \\ b \end{pmatrix}$ est normal à $\mathcal{D}$ et $\vec{u}\begin{pmatrix} -b \\ a \end{pmatrix}$ est un vecteur directeur de $\mathcal{D}$. Ainsi en considérant 2 droites d'équations respectives $d_1 : ax + by + c = 0$ et $d_2 : dx + ey + f = 0$, notons $\vec{u}_1\begin{pmatrix} -b \\ a \end{pmatrix}$ et $\vec{u}_2\begin{pmatrix} -e \\ d \end{pmatrix}$ les vecteurs directeurs de ces droites :

SI $\vec{u}_1$ colinéaire à $\vec{u}_2$ ALORS :
    SI (a,b,c) proportionnel à (d,e,f) ALORS :
        Les droites sont confondues
    SINON :
        Les droites sont strictement parallèles.
    FIN SI
SINON SI $\vec{u}_1 \perp \vec{u}_2$ ALORS :
    Les droites sont perpendiculaires.
SINON :
    Les droites sont sécantes.
FIN SI

Ce qui donne en utilisant le determinant et le produit scalaire à bon escient :

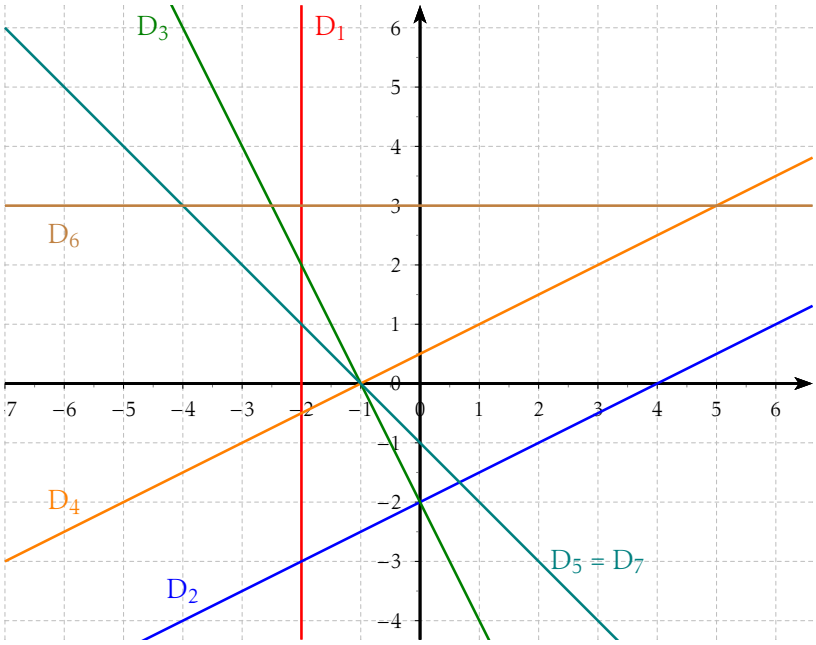
```
def positions(a, b, c, d, e, f):
    if a * e - b * d == 0:
        if a * f == c * d:
            return "confondues"
        else :
            return "strictement parallèles"
    elif b*e + a*d == 0:
        return "perpendiculaires"
    else:
        return "sécantes"
```

2. Pour tracer la droite dans la fenêtre demandée, nous allons choisir deux points A et B de la droite d'abscisses respectives $x_A = -10$ et $x_B = 10$. Or cela n'est pas possible si la droite est verticale, il faut donc distinguer 2 cas :

```
import matplotlib.pyplot as plt

def trace(a, b, c):
    if b == 0 :
        xA, xB = -c / a, -c / a
        yA, yB = -10, 10
    else:
        xA, xB = -10, 10
        yA = (-a * xA - c) / b
        yB = (-a * xB - c) / b
    plt.plot([xA, xB], [yA, yB])

trace(1, 0, 2)  # D1
trace(-1, 2, 4) # D2
plt.show()
```

3. On obtient alors le tableau suivant (le symbole × signifie ici sécantes) :

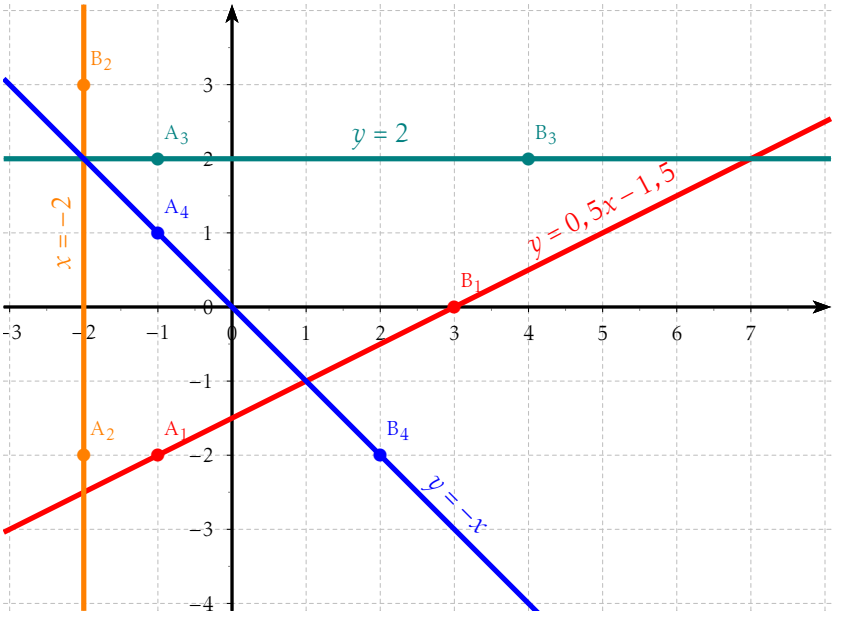
	D_1	D_2	D_3	D_4	D_5	D_6	D_7
D_1	=	×	×	×	×	⊥	×
D_2	×	=	⊥	//	×	×	×
D_3	×	⊥	=	⊥	×	×	×
D_4	×	//	⊥	=	×	×	×
D_5	×	×	×	×	=	×	=
D_6	⊥	×	×	×	×	=	×
D_7	×	×	×	×	=	×	=



Retour sur l'exercice C6

1. On complète le tableau :

	A	B	équation de la droite (AB)
①	$(-1;-2)$	$(3;0)$	$y = 0,5x - 1,5$
②	$(-2;-2)$	$(-2,3)$	$x = -2$
③	$(-1;2)$	$(4;2)$	$y = 2$
④	$(-1;1)$	$(2;-2)$	$y = -x$



2. On distingue donc deux cas :

   - Si $x_A = x_B$, la droite est parallèle à l'axe des ordonnées. Dans ce cas l'équation réduite de (AB) est : $x = x_A$.
   - Sinon, on sait que la droite (AB) a pour équation $y = mx + p$ où m est le coefficient directeur donné par la formule $m = \dfrac{y_B - y_A}{x_B - x_A}$ et p l'ordonnée à l'origine que l'on peut trouver en utilisant le fait que $A \in (AB)$. Ainsi $y_A = mx_A + p$ et donc $p = y_A - mx_A$.

```
def droite(xA, yA, xB, yB):
    if xA == xB :
        print("(AB) : x =", xA)
    else :
        m = (yB - yA) / (xB - xA)
        p = yA - m * xA
        if p < 0 :
            print("(AB) : y=", m, "x", p)
        else :
            print("(AB) : y=", m, "x+", p)
```

Le dernier `if` n'étant là que pour éviter d'afficher $y = 0,5x + -1,5$.

Retour sur l'exercice C7

1. a. KIAH étant un rectangle,
on a $KI^2 = AH^2 = AD^2 + DH^2 = 8^2 + 8^2 = 128$

b. On calcule chacune des longueurs :

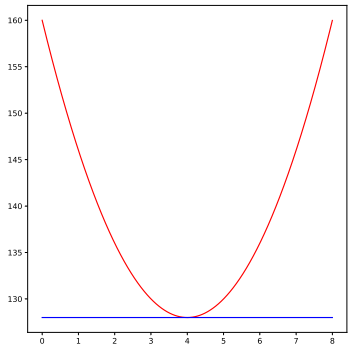
- $MI^2 = MA^2 + AI^2 = x^2 + 4^2 = x^2 + 16$
- $MK^2 = ME^2 + EK^2 = ME^2 + EH^2 + HK^2 = (8-x)^2 + 8^2 + 4^2$
$= 8^2 - 16x + x^2 + 80 = x^2 - 16x + 144$

Ainsi : $MI^2 + MK^2 = 2x^2 - 16x + 160$, on peut donc définir la fonction :

```
def f(x):
    return 2*x**2 - 16*x + 160
```

c. Pour tracer la courbe :

```
import matplotlib.pyplot as plt
# Tracé de la courbe
X, Y = [], []
x = 0
while x <= 8:
    X.append(x)
    Y.append(f(x))
    x = x + 0.001
plt.plot(X, Y, 'r')

# Droite d'équation y = 128
plt.plot([0,8], [128, 128], 'b')
plt.show()
```



Il semblerait que $f(x)$ soit égal à 128 une unique fois : pour $x = 4$

2. Par calcul :

$$\begin{aligned} 2x^2 - 16x + 160 = 128 &\iff 2x^2 - 16x + 32 = 0 \\ &\iff x^2 - 8x + 16 = 0 \\ &\iff (x-4)^2 = 0 \\ &\iff x = 4 \end{aligned}$$

Ainsi $x = 4$ est bien l'unique solution. On a donc M milieu de $[AE]$

Retour sur l'exercice C8 On sait que

$$A(x_A, y_A, z_A) \in \mathcal{S}(O, 23) \iff OA = 23 \iff \sqrt{x_A^2 + y_A^2 + z_A^2} = 23$$

Pour éviter le calcul d'une racine carrée, on peut tester la condition équivalente $x_A^2 + y_A^2 + z_A^2 = 23^2 = 529$. Dans la correction en plus du booléen renvoyé, on a ajouté l'affichage d'un texte :

```
def sur_sphere(x, y, z):
    if x ** 2 + y ** 2 + z ** 2 == 23 ** 2:
        print("A(", x, ";", y, ";", z, ") est sur la sphère")
        return True
    else:
        print("A(", x, ";", y, ";", z, ") n'est pas sur la sphère")
        return False
```

Si on ne souhaite pas l'affichage, on peut alors coder en une ligne :

```
def sur_sphere(x, y, z):
    return x ** 2 + y ** 2 + z ** 2 == 23 ** 2
```

Le dernier exemple ne fonctionne pas ! En effet : $\sqrt{6,44^2+0^2+22,08^2} = 23$, pourtant notre programme affiche que A n'est pas sur la sphère ...

Regardons de plus près en tapant dans la console le calcul.

```
>>> 6.44**2 + 0**2 + 22.08**2
528.9999999999999
```

L'erreur provient donc d'une erreur d'arrondi. C'est une notion importante qu'il faut avoir en tête. Autant Python est très efficace lorsqu'il travaille sur des entiers, autant les calculs sur les flottants (décimaux) amènent souvent à des erreurs d'arrondi du fait de leur représentation en machine. Ce problème se retrouve aussi dans beaucoup de langages. Souvenez-vous qu'un ordinateur travaille en base deux et non en base dix comme nous. Le nombre $6,44 = 2^2 + 2^1 + \frac{1}{2^2} + \frac{1}{2^3} + \frac{1}{2^4} + \frac{1}{2^9} + \frac{1}{2^{11}} + \ldots$ n'est pas simple à représenter en binaire.

Une alternative serait de remplacer le test $OA^2 = 529$ par $|OA^2 - 529| < \varepsilon$ où ε serait une tolérance à préciser par l'utilisateur. Par exemple :

```
def sur_sphere(x, y, z):
    epsilon = 10**-7
    return abs(x ** 2 + y ** 2 + z ** 2 - 23 ** 2) < epsilon
```

Retour sur l'exercice C9

1. Comme il y a 12 intervalles, on peut dessiner un triangle dont les côtés mesurent 5, 4 et 3 ($5+4+3=12$) qui, comme chacun sait, est un triangle rectangle connu bien avant le théorème de Pythagore !

2. Le problème n'est pas si simple car il faut tester un certain nombre de possibilités. Notons `hypo` la longueur du plus grand côté du triangle réalisé lorsque c'est possible. On a alors $1 \leqslant$ `hypo` $\leqslant n-3$. Pour chaque éventualité de `hypo`, on va tester toutes les longueurs possibles pour

les autres côtés. Par exemple pour $n = 13$ et `hypo`$= 5$, seule la solution $3, 4, 5$ donne un triangle rectangle :

```
def possible(n):
    """ Affiche si une corde à n noeuds est une corde de druide. """
    for hypo in range(1, n-2):
        for cote1 in range(1, hypo):
            cote2 = (n-1) - hypo - cote1
            if 0 < cote2 < hypo and hypo**2 == cote1**2 + cote2**2:
                print(cote1, cote2, hypo)
                return 1
    return 0
```

Notez que si vous ne mettez pas la condition `0 < cote2 < hypo`, le programme vous dira que vous pouvez fabriquer une corde à 19 nœuds en prenant comme côtés $12, -9$ et 15.

AU DÉTOUR DE LA BALADE...

Les puristes auraient sûrement préféré le renvoi d'un booléen. Selon le niveau des élèves, il n'est pas nécessaire de multiplier les difficultés.

3. Le plus dur étant fait, il reste à tester, pour les valeurs de n inférieures à 13, si on peut ou non fabriquer une corde de druide à n nœuds en utilisant la fonction précédente. On constate que ce n'est pas possible...

```
for i in range(13):
    if possible(i) == 0:
        print("Impossible avec", i, "noeuds")
    else:
        print("Possible avec", i, "noeuds")
```

4. Même principe pour cette dernière question, cette fois-ci on va utiliser une boucle TANT QUE puisque l'on ne connaît pas le nombre de nœuds qu'il faudra atteindre pour la $10^{\text{ième}}$ corde de druide (s'il en existe 10) :

```
i = 14
nb = 0
while nb < 10:
    if possible(i) == 1:
        print("On peut construire une corde à", i, "noeuds")
        nb = nb + 1
    i = i + 1
```

On trouve les possibilités suivantes : $25, 31, 37, 41, 49, 57, 61, 71, 73$ et 81.

Retour sur l'exercice C10

1. On applique la formule :

```
from math import pi

def volumecone(r, h):
        return pi * r**2 * h / 3
```

2. Si $d = 2,5$, on en déduit d'après le théorème de Thalès que $h = 3$ et on peut donc calculer le volume du demi-sablier avec la fonction précédente :

```
>>> volumecone(2.5/2, 3)
4.908738521234052
```

On peut aussi remarquer que le volume du cône occupé par le sable est une réduction de coefficient $1/2$ du cône de hauteur 6cm. Le volume est donc multiplié par $(1/2)^3$. On en déduit que le volume de sable est environ $4,91\text{cm}^3$.

3. Il s'agit de commencer par procéder au choix de l'unité de temps. L'énoncé nous donne une vitesse d'écoulement en cm^3 par minute, mais la seconde semble un meilleur choix puisqu'on veut un résultat « à la seconde près ».

```
>>> v = volumecone(1.25, 3)
>>> v / 1.6 * 60
184.07769454627692
```

Il faut environ 3 minutes et 4 secondes pour que la totalité du sable passe dans le cône du bas.

Retour sur l'exercice C11

1. Avec les notations de la figure disponible dans la section aide, on trouve successivement que :

   - $\widehat{\text{AOB}} = \dfrac{360°}{n}$
   - Le triangle AOB est isocèle, la hauteur (OH) est aussi la bissectrice de l'angle $\widehat{\text{AOB}}$, ainsi :

$$\widehat{\text{AOH}} = \frac{360°}{2n} = \frac{180°}{n}$$

- Dans le triangle AHO rectangle en H :

$$\mathrm{AH} = \mathrm{AO}\sin\left(\widehat{\mathrm{AOH}}\right) = 100\sin\left(\frac{180°}{n}\right)$$

- Comme le triangle AOB est isocèle, la hauteur (OH) est aussi médiatrice de [AB], d'où

$$\mathrm{AB} = 2\mathrm{AH} = 200\sin\left(\frac{180°}{n}\right)$$

- Comme AOB est isocèle, les angles à la base sont de même mesure, ainsi :

$$\widehat{\mathrm{ABO}} = \frac{180° - \widehat{\mathrm{AOB}}}{2} = \frac{180° - \dfrac{360°}{n}}{2} = 90° - \frac{180°}{n}$$

- Puisque $\widehat{\mathrm{ABO}} = \widehat{\mathrm{OBC}}$:

$$\alpha = 180° - 2 \times \widehat{\mathrm{ABO}} = 180° - 2\left(90° - \frac{180°}{n}\right) = \frac{360°}{n}$$

Reste à déterminer l'angle de rotation β entre le tracé du cercle et le premier côté du polygone :

$$\beta = 90° - \widehat{\mathrm{OAB}} = 90° - \left(90° - \frac{180°}{n}\right) = \frac{180°}{n}$$

On peut alors réaliser la fonction. Attention aux unités pour calculer le sinus !

```
from turtle import *
from math import *

def polygone(n):
    beta = 180/n
    alpha = 360/n
    AB = 200*sin(radians(180/n))

    circle(100)
    left(beta)
    for i in range(n):
        forward(AB)
        left(alpha)

    mainloop()
```

AU DÉTOUR DE LA BALADE...

En réalité, il existe déjà un paramètre dont nous n'avons pas parlé dans la fiche de cours sur ***turtle****, il permet justement de dessiner des polygones sans aucun calcul, il s'agit du paramètre* ***steps*** *:*

```
from turtle import *

def polygone(n):
    circle(100)
    circle(100, steps=n)

    mainloop()
```

Bien évidemment, ce n'est plus le même objectif pédagogique.

2. Le périmètre du polygone est assez facile à trouver :

$$\mathcal{L}_n = n \times \mathrm{AB} = 200n \sin\left(\frac{180°}{n}\right)$$

3. De la même manière, on peut calculer l'aire en multipliant par n l'aire du triangle AOB, on trouve

$$\mathcal{A}_n = n\frac{\mathrm{AB} \times \mathrm{OH}}{2} \text{ avec } \mathrm{OH} = 100\cos\left(\frac{180°}{n}\right)$$

On peut alors ajouter quelques lignes en fin de programme pour répondre au problème posé :

```
from turtle import *
from math import *

def polygone(n):
    beta = 180/n
    alpha = 360/n
    AB = 200*sin(radians(180/n))

    circle(100)
    left(beta)
    for i in range(n):
        forward(AB)
        left(alpha)

    OH = 100 * cos ( pi / n)
    print("Le périmètre est de", n*AB)
    print("L'aire est de", n*AB*OH/2)

polygone(5)
mainloop()
```

4. Pour $n = 96$, on trouve un périmètre d'environ $628,206$ et une aire de $31393,502$. La figure attire notre attention par le fait qu'un polygone avec autant de côtés s'approche très fortement d'un disque de périmètre $200\pi \approx 628,319$ et d'aire $\pi \times 100^2 = 10000\pi \approx 62831,853$. C'est d'ailleurs ce polygone à 96 côtés qu'a utilisé Archimède pour donner une bonne valeur approchée du nombre π.

   En terminale spécialité mathématiques, on peut prouver ce résultat en utilisant le fait que $\lim\limits_{x\to 0} \dfrac{\sin x}{x} = 1$. En exprimant les angles en radians (indispensable si on veut utiliser les fonctions sinus et cosinus en mathématiques), on trouve :

$$\begin{aligned}
\lim_{n\to+\infty} \mathcal{L}_n &= \lim_{n\to+\infty} 200n \sin\left(\frac{\pi}{n}\right) \text{ en posant } x = \frac{1}{n} \\
&= \lim_{x\to 0} 200\frac{1}{x}\sin(x\pi) \\
&= \lim_{x\to 0} 200\frac{\sin(x\pi)}{x} \\
&= \lim_{x\to 0} 200\pi\frac{\sin(x\pi)}{x\pi} \\
&= 200\pi \text{ soit } 2\pi r
\end{aligned}$$

   De la même manière :

$$\begin{aligned}
\lim_{n\to+\infty} \mathcal{A}_n &= \lim_{n\to+\infty} n\frac{\text{AB}\times\text{OH}}{2} \\
&= \lim_{n\to+\infty} \frac{1}{2}\times\mathcal{L}_n\times\text{OH} \\
&= \lim_{n\to+\infty} \frac{1}{2}\times\mathcal{L}_n\times 100\cos\left(\frac{\pi}{n}\right) \\
&= \lim_{n\to+\infty} 50\times\mathcal{L}_n\times\cos\left(\frac{\pi}{n}\right) \\
&= 50\times 200\pi\times 1 \\
&= 10000\pi \text{ soit } \pi r^2
\end{aligned}$$

> Un thème dédié spécifiquement au nombre π est proposé dans les pages centrales du livre.

Retour sur l'exercice C12

1. Avec les notations données dans l'aide, on trouve

$$\text{AB} = \frac{\text{AO}}{\cos\left(\widehat{\text{OAB}}\right)} = \frac{300}{\cos(15)} \text{ et } \text{OB} = 300\times\tan(15).$$

D'où le programme suivant (les 5 premières lignes permettent juste de faire descendre la tortue avant de commencer le dessin) :

```
from turtle import *
from math import *

up()
right(90)
forward(200)
left(90)
down()
a = radians(15)
AB = 300 / cos(a)
OB = 300 * tan(a)
left(90+15)
forward(AB)
back(AB)
right(30)
forward(AB)
left(15)
circle(OB, 180)
mainloop()
```

2. Avec les notations de l'aide, on trouve successivement :

- $BC = 2BH = 2 \times 300 \times \tan(15)$
- $\mathcal{A}_{ABC} = \dfrac{BC \times AH}{2}$
- $AB = \dfrac{300}{\cos(15)}$
- $\mathcal{A}_{ACO} = \mathcal{A}_{ABO} = \dfrac{AB \times r}{2}$ et $\mathcal{A}_{BCO} = \dfrac{BC \times r}{2}$
- Finalement : $2\mathcal{A}_{ABO} + \mathcal{A}_{BCO} = \mathcal{A}_{ABC}$

$$2 \times \frac{AB \times r}{2} + \frac{BC \times r}{2} = \frac{BC \times AH}{2}$$
$$2 \times AB \times r + BC \times r = BC \times AH$$
$$r(2AB + BC) = BC \times AH$$
$$r = \frac{BC \times AH}{2AB + BC}$$

- Enfin, dans le triangle AOD,

$$AD = AO \times \cos(15) = (AH - r) \times \cos(15).$$

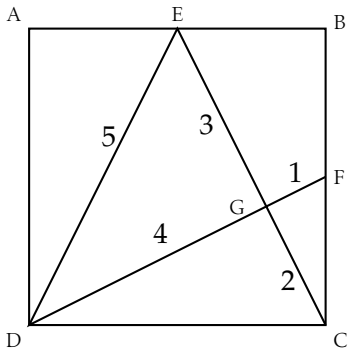
```
from turtle import *
from math import *

up()
right(90)
forward(200)
left(90)
down()

a = radians(15)
AH = 300
BC = 2 * AH * tan(a)
AB = AH / cos(a)
r = BC*AH / (2*AB + BC)
AD = (AH-r) * cos (a)

left(90+15)
forward(AB)
back(AB)
right(30)
forward(AB)
back(AB-AD)
circle(r)

hideturtle()
mainloop()
```

3. On garde encore les notations de l'aide pour ce dernier cas, on trouve successivement :

   - $AO = \dfrac{r}{\sin(15)}$
   - $IF = \dfrac{r}{\cos(15)}$
   - $OF = 2r + \dfrac{r}{\cos(15)}$
   - $OH = OF \times \cos(15) = 2r\cos(15) + r$
   - On déduit que : $\dfrac{r}{\sin(15)} + 2r\cos(15) + r = 300$ d'où
     $$r\left(\frac{1}{\sin(15)} + 2\cos(15) + 1\right) = 300 \text{ et}$$
     $$r = \frac{300}{1/\sin(15) + 2\cos(15) + 1}$$
   - Enfin, $AC = \dfrac{r}{\tan(15)}$

```
from turtle import *
from math import *

up()
right(90)
forward(200)
left(90)
down()

a = radians(15)
r = 300 / (1/sin(a) + 2*cos(a) + 1)
AD = r / tan(a)
AB = 300 / cos(a)
left(90+15)
forward(AB)
back(AB)
right(30)
forward(AB)
back(AB-AD)
circle(r)
forward(2*r)
circle(r)

hideturtle()
mainloop()
```

Retour sur l'exercice C13 Pour simplifier les explications, nommons certains points comme sur la figure.



Réfléchissons directement à l'ordre dans lequel il faut relier les points. Seuls des points F et C partent un nombre impair de chemins. Il faut donc commencer et terminer le tracé par ces 2 points. En effet, lorsqu'on "arrive" et qu'on "repart" d'un point, on trace 2 segments. Si bien que de tous les points à l'intérieur du trajet doivent partir un nombre pair de chemins. Choisissons par exemple le trajet C – B – A – D – C – E – D – F.

- ED = EC, donc E est sur la médiatrice de [CD] et donc au milieu de [AB]. Notons $\ell = \mathrm{AE}$. Comme AED est rectangle, on a $\mathrm{AE}^2 + \mathrm{AD}^2 = 5^2$ c'est à dire $\ell^2 + (2\ell)^2 = 25$. On trouve finalement $\ell^2 = 5$, puis $\ell = \sqrt{5}$.

- Dans le triangle CBE, on trouve $\sin\left(\widehat{\mathrm{ECB}}\right) = \dfrac{\mathrm{EB}}{\mathrm{CE}} = \dfrac{\sqrt{5}}{5}$. On utilisera la fonction `asin` sur Python pour obtenir une bonne valeur approchée de l'angle en radians, qu'il faudra ensuite convertir en degrés. Dans la suite on note $\alpha = \widehat{\mathrm{ECB}} \approx 27°$.

- On a $\widehat{\text{DCE}} = 90 - \alpha$ et $\widehat{\text{DEC}} = 180 - 2\widehat{\text{DCE}} = 180 - 2(90 - \alpha) = 2\alpha$.
- Pour finir $\widehat{\text{EDF}} = 90 - 2\alpha$.

On peut alors réaliser le programme, où l'on a multiplié les longueurs par 50 pour une meilleure lisibilité.

```
from turtle import *
from math import *
l = 50*sqrt(5)
for i in range(4):
    left(90)
    forward(2*l)

alpha = degrees(asin(sqrt(5)/5))

left(90+alpha)
forward(50*5)

DEC = 2*alpha
left(180-DEC)
forward(50*5)

EDF = 90-2*alpha
left(180-EDF)
forward(50*5)

mainloop()
```

Retour sur l'exercice C14 Avec les notations données en aide, on trouve successivement :

- $\widehat{\text{AOB}} = \dfrac{360°}{5} = 72°$.
- $\widehat{\text{ACO}} = \dfrac{180° - \widehat{\text{AOB}}}{2} = \dfrac{180° - 72°}{2} = 54°$.
- Enfin $\widehat{\text{ACE}} = 2 \times 54° = 108°$.

Il faut donc tourner entre chaque branche de $180° - 108° = 72°$. D'où la fonction :

```
def etoile() :
    for branche in range (5):
        forward(30)
        left(144)
```

Cette fonction ne reçoit aucun argument. De plus, elle ne renvoie rien. Une telle fonction sans retour est appelée **fonction d'impression**.

On trouve par le même procédé que $\widehat{A_1A_2C_2} = 150°$ ce qui permet de terminer le programme :

```
from turtle import *

def etoile():
    for branche in range(5):
        forward(30)
        left(144)

pensize(2)
pencolor('yellow')
bgcolor('blue')
for i in range(12):
    down()
    etoile()
    up()
    forward(50)
    left(30)

mainloop()
```

Chapitre D

FONCTIONS

1. Problèmes concrets

D1 *Le directeur d'une salle de spectacle de 8000 places organise un concert. Il souhaite fixer le prix du billet pour optimiser le montant de sa recette. Une étude de marché lui apprend que :*

- *si le prix du billet est de 50 euros, il vend 3 000 billets.*
- *chaque baisse de 1 euro lui permet de vendre 170 billets supplémentaires.*

On souhaite déterminer le prix du billet qui sera entier et rendra la recette maximale.

1. *Youness a créé cette fonction pour calculer le nombre de billets vendus :*

```
def nbbillets(prix):
    return 3000 + prix * 170
```

   Sa voisine lui fait remarquer que sa formule est fausse, car on ne retrouve pas les 3000 billets vendus lorsque le prix est fixé à 50€. Corriger la fonction.

2. *En utilisant la fonction précédente, créer une fonction* `recette` *qui reçoit le prix d'un billet en paramètre et renvoie la recette du spectacle.*

3. *Écrire un script qui représente graphiquement la recette en fonction du prix du billet en euros et répondre à la question.*

4. *Étudier la fonction f définie par $f(x) = x\big(3000+170(50-x)\big)$ et retrouver le résultat de la question précédente.*

5. *Laquelle des 2 méthodes vous semble la plus fiable ?*

2. À la rencontre de vieilles connaissances

D2 1. *Écrire une fonction qui reçoit deux nombres et renvoie la distance entre ces nombres (par exemple : la distance entre 3 et 8 est 5, la distance entre 3 et -1 est 4).*

2. *En déduire une fonction qui reçoit deux nombres et affiche le plus grand des deux, sans utiliser de tests, ni d'autres fonctions que celle qui a été créée.*

3. *La fonction valeur absolue est la fonction définie sur* $\mathbb{R}$ *par* $f(x) = |x|$ *où* $|x|$ *est la distance entre* 0 *et* x*. En déduire le code à saisir pour calculer l'image d'un nombre par la fonction valeur absolue à l'aide de la fonction* `distance`*.*

4. *Tracer la courbe représentative de cette fonction sur l'intervalle* $[-8;8]$*.*

5. *Sur Python,* **`abs(x)`** *renvoie la valeur absolue d'un nombre* x*. À l'aide de cette fonction, tracer la courbe représentative de la fonction* g *définie par* $g(x) = |x^2 - 2x - 20|$ *sur l'intervalle* $[-8;8]$*.*

D3 *[Fonction homographique]*

1. *Soit* f *la fonction homographique définie sur son ensemble de définition par* $f(x) = \dfrac{2x+1}{x-2}$*. Réaliser un programme qui affiche l'image d'un nombre entré par l'utilisateur lorsque c'est possible et avertit que le calcul est impossible le cas échéant.*

2. *En posant* $y = f(x)$*, exprimer* x *en fonction de* y*. Que peut-on remarquer ?*

3. *Est-ce le cas pour toutes les fonctions homographiques ?*

D4 *Réaliser le programme suivant qui sera à modifier au fur et à mesure des questions.*

1. *On donne un nombre, l'ordinateur affiche son image par la fonction*

$$f : x \longmapsto 2x - 6$$

Tester le programme pour les valeurs 0, 1, 2, -1 et -2.

2. *Améliorer le programme pour que l'ordinateur affiche « Bravo ! Vous êtes un as du calcul mental ! » si l'image obtenue vaut 4.*

3. *Modifier le code précédent pour réaliser la même chose mais avec la fonction*

$$g : x \longmapsto \frac{x+1}{x-2}$$

4. *Modifier le programme de la question 2 pour réaliser la même chose mais avec la fonction* $h : x \longmapsto \sqrt{3x+1}$.

5. *Compléter le programme pour qu'il trace les courbes de ces 3 fonctions avec des couleurs différentes sur l'intervalle* $[3;10]$.

D5 *On considère une fonction* f *définie par l'expression* $f(x) = \dfrac{ax+b}{cx+d}$ *où* a, b, c *et* d *sont 4 réels.*

1. *Déterminer l'ensemble de définition de la fonction* f.

2. *À quelle condition sur les réels* a, b, c *et* d,

   a. *la fonction* f *est-elle une fonction affine ?*

   b. *la fonction* f *est-elle constante ?*

3. *Compléter l'algorithme suivant :*

```
FONCTION nature(a, b, c, d) :
    SI c ≠ 0 ALORS :
        SI ad − bc = 0 ALORS :
            RENVOYER "La fonction f est une fonction ....."
        SINON :
            RENVOYER "La fonction f est une fonction ....."
        FIN SI
    SINON :
        SI d = 0 ALORS :
            RENVOYER "Impossible de définir la fonction f"
        SINON :
            RENVOYER "La fonction f est une fonction ....."
        FIN SI
    FIN SI
FIN FONCTION
```

4. *Traduire l'algorithme précédent en Python.*

D6 *Dans le supérieur, on est amené à définir deux fonctions appelées respectivement cosinus hyperbolique et sinus hyperbolique sur* $\mathbb{R}$ *par* $ch(x) = \dfrac{e^x + e^{-x}}{2}$ *et* $sh(x) = \dfrac{e^x - e^{-x}}{2}$.

1. *Calculer* $ch(0)$ *et* $sh(0)$.
2. *Définir ces deux fonctions sous Python et vérifier les calculs précédents.*
3. *Afficher la courbe représentative de ces fonctions avec Python.*
4. *En prenant quelques valeurs réelles au hasard, quelle relation peut-on conjecturer entre* $ch^2(x)$ *et* $sh^2(x)$ *pour un réel* x *donné ?*
5. *Démontrer cette conjecture.*

D7 *Soit* f *la fonction définie sur* $\mathbb{R}$ *par* $f(x) = x^2$. *On cherche à connaître la "valeur moyenne" de* f *sur l'intervalle* $[0;3]$

1. ***Approche stochastique*** *: réaliser un programme en Python qui choisit aléatoirement 1000 nombres* $x_i \in [0;3]$ *et calcule la moyenne des 1000 valeurs* $f(x_i)$.

2. ***Approche déterministe*** *: Soit* $n \in \mathbb{N}^*$*, on pose* $x_i = \dfrac{3i}{n}$ *pour* $0 \leqslant i < n$*; ce qui définit* n *nombres équirépartis dans l'intervalle* $[0;3]$*. Écrire une fonction* `moyenne(n)` *qui reçoit un entier* `n` *non nul et renvoie la moyenne des* $f(x_i)$*.*

3. *Les valeurs de* `moyenne(n)` *semblent converger. Comparer les résultats avec* $\mu = \dfrac{1}{3}\displaystyle\int_0^3 f(x)\mathrm{d}x$*. Était-ce prévisible ?*

3. Attention aux racines !

D8 *[Racines réelles d'un polynôme.]*

1. *Sans calculatrice, compléter le tableau ci-dessous en indiquant les racines réelles quand elles existent de chaque polynôme du second degré :*

polynôme	*racines*
$x^2 - 5x + 6$	
$x^2 + x + 1$	
$2x^2 - 12x + 18$	
$2x^2 + 3x - 2$	

2. *Créer une fonction ayant pour paramètres trois nombres réels* a*,* b *et* c *représentant les coefficients du trinôme* $ax^2 + bx + c$ $(a \neq 0)$*, qui renvoie une liste contenant les racines réelles de ce polynôme.*

3. *Tester la fonction créée avec les exemples de la première question.*

D9 *Soit* `f` *la fonction définie sur* $\mathbb{N}$ *en Python par :*

```
def f(n) :
    if n > 20 :
        n = n - 15
    if n % 3 == 0 :
        n = n * 2
    else :
        n = n * 3
    return n
```

1. *Écrire en fonction de* `n` *les 4 expressions qu'il est possible d'obtenir pour* `f(n)`*.*

2. *La fonction* `f` *admet-elle des points fixes, c'est à dire des valeurs* α *telles que* $f(\alpha) = \alpha$ *?*

D10 *On considère la fonction ci-dessous :*

```
from math import log, exp

def f(x) :
    y = (x-2)**2 - 3
    if y > 0  :
        return y * log(y)
    else :
        x, y = y*exp(2-x), y*exp(2+x)
        return y - x
```

1. *Quel est l'ensemble de définition de cette fonction ?*
2. *La fonction est-elle continue sur cet ensemble ?*
3. *Combien l'équation* $f(x) = 0$ *admet-elle de solutions ?*
4. *Représenter à l'aide de Python la courbe de* `f` *sur l'intervalle* $[-5;5]$.

D11 *Soit* m *un réel et* $\mathcal{C}_m$ *la courbe représentative de la fonction polynomiale* f_m *définie sur* $\mathbb{R}$ *par* $f_m(x) = x^3 - mx^2 + mx - 1$. *L'objectif est d'étudier quelques caractéristiques de la famille des courbes* $\mathcal{C}_m$.

1. *Écrire une fonction* `f(m,x)` *qui pour deux réels* `m` *et* `x` *renvoie la valeur de* $f_m(x)$.
2. *Écrire une fonction* `C(m,coul)` *qui trace la courbe* $\mathcal{C}_m$ *de la couleur* `coul`, *qui est une chaîne de caractères.*

   Tracer quelques courbes en limitant l'axe des abscisses à $[-2,2]$ *et celui des ordonnées à* $[-5;5]$.

3. *Les courbes semblent passer par deux points fixes. Lesquels ? Démontrer cette conjecture.*
4. *Conjecturer, puis déterminer les solutions de l'équation* $f_m(x) = 0$ *dans les cas suivants :*

   a. $m = -1,5$ *b.* $m = -1$ *c.* $m = 0$ *d.* $m = 3$

5. *La courbe* $\mathcal{C}_m$ *possède-t-elle toujours (au moins) une tangente horizontale ? Rédiger les preuves des réponses.*

D12 *Soit f la fonction définie sur $\mathbb{R}$ par $f(x) = e^x + x - 2$.*

1. *Démontrer que f est continue et strictement croissante sur $\mathbb{R}$.*

2. *Démontrer que l'équation $f(x) = 0$ admet une unique solution, notée α, dans $\mathbb{R}$ et que $\alpha \in [0;1]$.*

3. *Parmi les propositions suivantes, quelle est la fonction en langage Python qui permet de donner une valeur approchée de α lorsque a et b sont deux réels encadrant α (avec $a < b$) :*

Proposition 1 :

```
def equation(a, b):
    while abs(b-a) >= 0.001:
        m = (a+b)/2
        if f(m) < 0:
            b = m
        else:
            a = m
    return m
```

Proposition 3 :

```
def equation(a, b):
    m = (a+b) / 2
    while abs(b-a) <= 0.001:
        if f(m) < 0:
            a = m
        else:
            b = m
    return m
```

Proposition 2 :

```
def equation(a, b):
    while abs(b-a) >= 0.001:
        if f(m) < 0:
            a = m
        else:
            b = m
    return m
```

Proposition 4 :

```
def equation(a, b):
    while abs(b-a) >= 0.001:
        m = (a+b) / 2
        if f(m) < 0:
            a = m
        else:
            b = m
    return m
```

4. *En déduire une valeur approchée de la solution de l'équation $f(x) = 0$ à 0,001 près.*

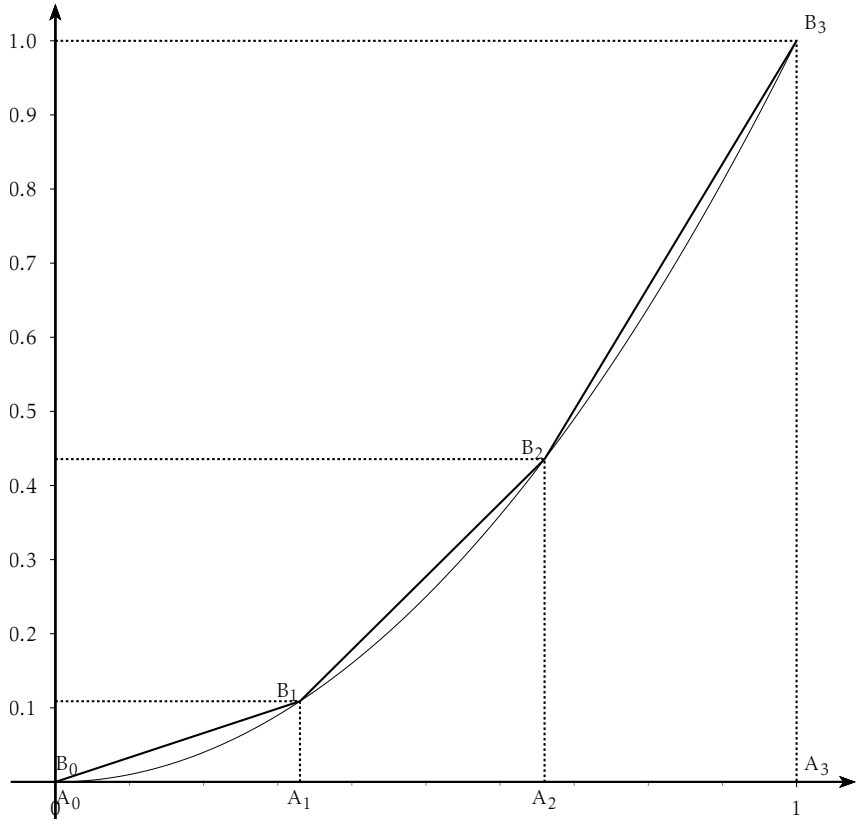
4. Si vous n'êtes pas encore fatigué, relevez les défis

D13 *On se propose de calculer la longueur d'un arc de la parabole $\mathcal{P}$ représentant la fonction $x \longmapsto x^2$ sur l'intervalle $[0;1]$ dans un repère orthonormé (O,I,J). On note ℓ cette longueur cherchée.*

1. *Justifier graphiquement que $\sqrt{2} < \ell < 2$.*

2. *Pour estimer ℓ, on propose d'approcher la courbe $\mathcal{P}$ par une succession de n segments. Soient $A_0, \ldots, A_n$ les points du segment $[OI]$ partageant*

ce segment en n segments de même longueur et $B_0, \ldots, B_n$ les points de la parabole de même abscisse respectivement que $A_0, \ldots, A_n$.



Exemple avec $n = 3$.

a. *Donner, en fonction de k et n, les coordonnées des points A_k et B_k pour $0 \leqslant k \leqslant n$.*

b. *Exprimer la longueur $B_k B_{k+1}$ en fonction de k et de n pour $0 \leqslant k < n$.*

c. *Écrire alors une fonction* `longueur(n)` *qui renvoie la longueur de la ligne brisée $B_0 B_1 \ldots B_n$ pour un nombre n donné de segments.*

3. *On choisit de prendre comme valeur de n la première valeur pour laquelle l'écart (absolu) entre* `longueur(n)` *et* `longueur(n+1)` *est plus petit que 10^{-10}. À l'aide d'un programme, déterminer la valeur de n et une valeur approchée de la longueur de la ligne brisée correspondante.*

4. *On cherche à présent une valeur exacte de ℓ. Un théorème de mathématiques affirme que si f est une fonction dérivable ayant pour dérivée la fonction f'*

continue sur un intervalle $[a;b]$*, alors la longueur de la courbe représentant* f *sur l'intervalle* $[a;b]$ *peut être calculée par*

$$\ell = \int_a^b \sqrt{1+[f'(x)]^2}\,\mathrm{d}x$$

a. *En déduire une expression de* ℓ *à l'aide d'une intégrale.*

b. *Soit la fonction* φ *définie sur* $[0;1]$ *par*

$$\varphi : x \longmapsto 2x\sqrt{1+4x^2} + \ln\left(2x+\sqrt{1+4x^2}\right).$$

Calculer φ'*.*

c. *En déduire la valeur exacte de* ℓ*.*

D14 *On se pose la question :*

« *Quels sont les entiers distincts* a *et* b *tels que* $a^b = b^a$ *?* »

1. *Écrire un script qui teste tous les entiers distincts* a *et* b *à moins de 5 chiffres. Que peut-on conjecturer ?*

2. *Après avoir écarté certains cas pouvant poser problème, déterminer une fonction* f *telle que* $a^b = b^a \iff f(a) = f(b)$*.*

3. *Répondre à la question en étudiant cette fonction* f*.*

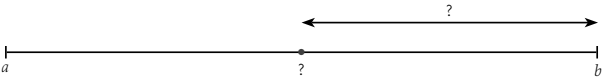
5. Aide pour les exercices

Aide pour l'exercice D2

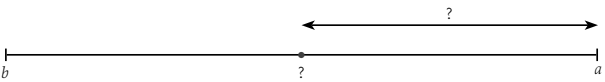
1. Une distance est toujours un nombre positif, programmez l'écart dans "le bon sens".

2. Pour la seconde question, remplacer les ? par les expressions $\dfrac{a+b}{2}$ et $\dfrac{|b-a|}{2}$ pour trouver une formule.

Si $a < b$:



Si $a > b$:



3. utiliser les questions précédentes.

4. consulter la fiche sur les tracés si besoin.

Aide pour l'exercice D3

1. Il y a une valeur interdite à cause de la division. Pensez à traiter ce cas à part.

2. Commencez par écrire $y = \dfrac{2x+1}{x-2}$ puis modifiez cette égalité pour isoler x.

Aide pour l'exercice D4 Cet exercice ne présente pas de difficulté particulière... attention cependant à bien vérifier votre programme pour les valeurs qui sont demandées dans chacun des cas... il y aura peut-être des surprises....

1. Créer une fonction qui prend pour paramètre x et qui renvoie $f(x)$.

2. Trouver la valeur qui affiche ce message.

3. Tester le programme pour les valeurs 0, 1, 2, -1 et -2 et trouver la valeur de x qui affiche le fameux message.

Aide pour l'exercice D5

1. Attention à la valeur interdite !

2a. Quelle est la définition d'une fonction affine ? En déduire une condition sur a,b,c ou d.

2b. Trouver une condition sur a,b,c et d pour que les x s'annulent dans l'expression de la fonction f.

Aide pour l'exercice D8 Calculer le *discriminant* pour *discriminer* les différents cas peut s'avérer une bonne idée...

Aide pour l'exercice D12

2. Pensez à utiliser le corollaire du théorème des valeurs intermédiaires pour justifier l'unicité de la solution de l'équation.

Aide pour l'exercice D13

1. Tracer la courbe dans le repère (O, I, J).

2. Ne pas hésiter à tester avec des valeurs de n et de k. Exploiter le fait que les points B_k appartiennent à la courbe.

Aide pour l'exercice D14 Pour la seconde question, pensez à l'utilité de la fonction logarithme népérien.

6. Solutions des exercices

Retour sur l'exercice D1

1. En effet, la baisse est de `50-prix` et non `prix` :

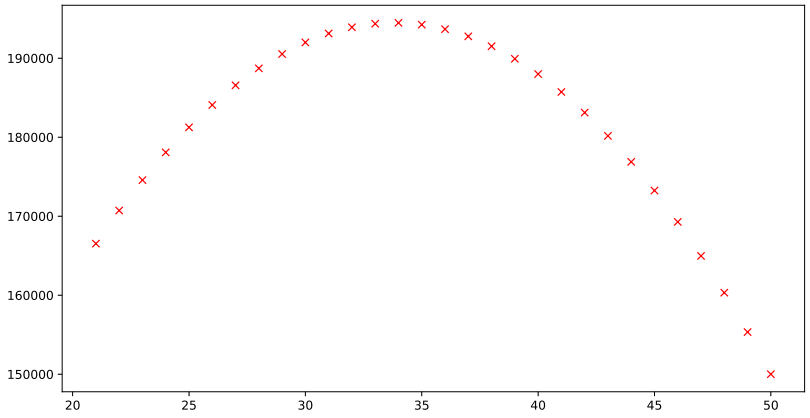
```
def nbbillets(prix):
    return 3000 + (50-prix)*170
```

2. En utilisant la fonction précédente :

```
def recette(prix):
    return nbbillets(prix) * prix
```

3. Partant du prix de 50€, on retire 1€ puis 1€ ... jusqu'à ce qu'il n'y ait plus de place dans la salle :

```
import matplotlib.pyplot as plt

p = 50
while nbbillets(p) < 8000:
    r = recette(p)
    print("A", p, "€ le billet, la recette est de", r, "€.")
    plt.plot(p, r, 'rx')
    p = p - 1
plt.show()
```



On peut conjecturer que le meilleur prix se situe autour de 35€, et à l'aide de l'affichage avec `print`, on peut préciser :

```
...
A 37 € le billet, la recette est de 192770 €.
A 36 € le billet, la recette est de 193680 €.
A 35 € le billet, la recette est de 194250 €.
A 34 € le billet, la recette est de 194480 €.
A 33 € le billet, la recette est de 194370 €.
...
```

Le meilleur prix est donc de 34€.

4. En étudiant la fonction définie par

$$f(x) = x\big(3000 + 170(50 - x)\big) = -170x^2 + 11500x$$

x	0	$\frac{575}{17}$	$+\infty$
Var f		↗	↘

$\frac{575}{17} \approx 33{,}8$, le maximum parmi les entiers se trouve donc en 33 ou 34. Une comparaison des images permet de confirmer que le meilleur prix est de 34€.

5. Dans notre situation, tous les prix sont entiers et on ne risque donc pas d'erreur de calcul due aux approximations des nombres réels.

D'autre part, le nombre de possibilités est fini, on peut donc réaliser une liste exhaustive des cas et trouver le meilleur. La preuve par l'algorithme dans ce cas et celle mathématique sont donc aussi rigoureuses l'une que l'autre.

AU DÉTOUR DE LA BALADE...

Pour l'enseignant de mathématiques qui se demande "mais à quoi bon faire encore des mathématiques ?".

Voici un exercice pouvant être proposé à vos élèves. Il devrait susciter leur engagement :

J'ai acheté des cahiers à 2€50 l'un et des crayons à 1€20 l'un. J'ai payé 54€30. Combien ai-je acheté de cahiers et de crayons ? Voici 2 propositions d'algorithmes :

```
POUR cahier DE 1 à 22 FAIRE :
    POUR crayon DE 1 à 46 FAIRE :
        SI cahier × 2,50 + crayon × 1,20 = 54,30 ALORS :
            AFFICHER "On peut acheter",cahier,"cahiers et",crayon"crayons."
        FIN SI
    FIN POUR
FIN POUR
```

```
POUR cahier DE 1 à 22 FAIRE :
    reste ⟵ 54,3 − cahier × 2,5
    crayon ⟵ reste ÷ 1,2
    SI crayon EST ENTIER ALORS :
        AFFICHER "On peut acheter",cahier,"cahiers et",crayon"crayons."
    FIN SI
FIN POUR QUE
```

1. *Ces 2 algorithmes sont-ils justes ?*
2. *Les programmer.*
3. *Que se passe-t-il ?*

Retour sur l'exercice D2

1. Une première solution

```
def distance(a, b):
    if a < b:
        return b - a
    else:
        return a - b
```

Bien sûr, si vous connaissez déjà la valeur absolue, c'est immédiat, mais cette fonction est justement présentée dans les questions suivantes :

```
def distance(a, b):
    return abs(b-a)
```

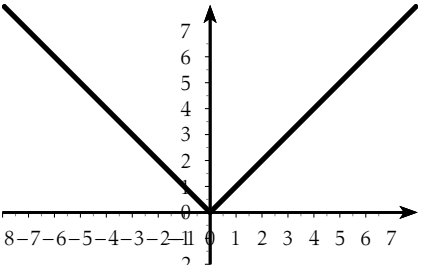
2. En s'appuyant sur le schéma proposé dans l'aide, on trouve que la plus grande valeur (notée m) entre a et b peut être calculée par la formule $m = \frac{a+b}{2} + \frac{|b-a|}{2}$, ainsi on obtient le programme :

```
def plusgrand(a,b):
    return (a + b + distance(a, b)) / 2
```

3. Il suffit de saisir `distance(a, 0)` pour calculer $|a|$.

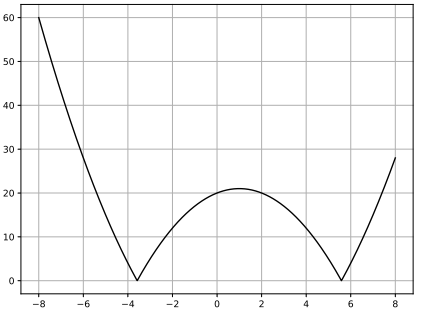
4. On peut s'inspirer du code présent sur la fiche des graphiques, voici une solution utilisant la troisième méthode :

```
from numpy import linspace
import matplotlib.pyplot as plt

X = linspace(-8, 8, 1000)
Y = [distance(0,x) for x in X]
plt.plot(X,Y)
plt.show()
```



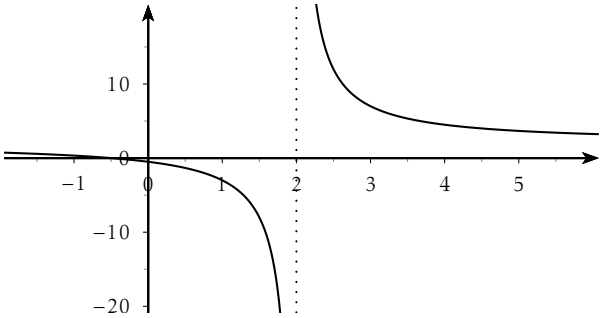
5. Même principe en changeant de fonction :

```
from numpy import linspace
import matplotlib.pyplot as plt

def g(x):
    return abs(x**2-2*x-20)

X = linspace(-8, 8, 1000)
Y = [g(x) for x in X]
plt.plot(X,Y)
plt.grid()
plt.show()
```



Retour sur l'exercice D3

1. Il s'agit d'une fonction homographique, définie sur $\mathbb{R}\backslash\{2\}$.




```
x = float(input("De quel nombre souhaitez-vous l'image ?"))
if x == 2:
    print(x, "n'est pas dans l'ensemble de définition de f.")
else:
    f = (2*x + 1) / (x - 2)
    print("L'image de", x, " est", f)
```

Pour rappel l'instruction `float` permet de transformer la chaîne de caractères saisie en nombre réel pour effectuer le calcul.

2. Soit $x \neq 2$ et $y \in \mathbb{R}$.

$$\begin{aligned} y = \frac{2x+1}{x-2} &\iff y(x-2) = 2x+1 \\ &\iff yx - 2x = 2y + 1 \\ &\iff x(y-2) = 2y+1, \text{ si } y \neq 2 \\ &\iff x = \frac{2y+1}{y-2} \end{aligned}$$

et si $y = 2$, $y = \dfrac{2x+1}{x-2} \iff 2(x-2) = 2x+1 \iff -4 = 1$ (impossible).

Ainsi tous les réels autres que 2 ont un unique antécédent, f est une bijection de $\mathbb{R}\backslash\{2\}$ dans $\mathbb{R}\backslash\{2\}$

AU DÉTOUR DE LA BALADE...

La fonction homographique f vérifie $f = f^{-1}$, on pourra remarquer que sa courbe a pour axe de symétrie la droite d'équation $y = x$. On peut aussi calculer $f\big(f(x)\big)$.

3. Il suffit de trouver un contre exemple.

Retour sur l'exercice D4

1.
```
def f(x) :
    return 2 * x - 6

x = float(input("Valeur de x : "))
y = f(x)
print("L'image de",x,"par f est",y)
```

2. Il suffit d'ajouter les deux lignes :
```
def f(x):
    return 2 * x - 6

x = float(input("Valeur de x : "))
y = f(x)
if y == 4:
    print("Bravo, vous êtes un as du calcul mental !")
print("L'image de", x, "par f est", y)
```

On a $f(x) = 4 \iff 2x - 6 = 4 \iff 2x = 10 \iff x = 5$.

3. La première idée est simplement de modifier la définition de la fonction, mais si on entre $x = 2$, on risque d'obtenir une erreur. Pour cela on va gérer le problème dans la définition même de la fonction en renvoyant l'image lorsque celle-ci peut être calculée et le texte "ERR" dans le cas contraire.
```
def g(x):
    if x != 2:
        return (x+1) / (x-2)
    else:
        return "ERR"

x = float(input("Valeur de x : "))
y = g(x)
if y != "ERR":
    if y == 4:
        print("Bravo, vous êtes un as du calcul mental !")
    print("L'image de", x, "par g est", y)
else:
    print("Calcul impossible.")
```

On a $g(x) = 4 \iff \begin{cases} x \neq 2 \\ \dfrac{x+1}{x-2} = 4 \end{cases} \iff \begin{cases} x \neq 2 \\ x + 1 = 4(x-2) \end{cases}$

$\iff \begin{cases} x \neq 2 \\ -3x = -9 \end{cases} \iff x = 3$

4. Même principe pour la fonction h. Pour éviter les `if` imbriqués, on utilise cette fois un `elif` :

```
from math import sqrt

def h(x):
    if 3*x + 1 >= 0:
        return sqrt(3*x + 1)
    else:
        return "ERR"

x = float(input("Valeur de x : "))
y = h(x)
if y == "ERR":
    print("Calcul impossible.")
elif y == 4:
    print("Bravo, vous êtes un as du calcul mental !")
else:
    print("L'image de", x, "par h est", y)
```

On a $h(x) = 4 \iff \begin{cases} 3x+1 \geqslant 0 \\ \sqrt{3x+1} = 4 \end{cases} \iff \begin{cases} 3x+1 \geqslant 0 \\ 3x+1 = 16 \end{cases} \iff x = 5.$

5. Comme toutes les valeurs de l'intervalle $[3;10]$ sont dans les ensembles de définition des fonctions f, g et h, on peut se passer des tests.

```
from math import sqrt
import matplotlib.pyplot as plt

def f(x):
    return 2*x - 6

def g(x):
    return (x+1) / (x-2)

def h(x):
    return sqrt(3*x + 1)

x = 3
X, Y1, Y2, Y3 = [], [], [], []
while x <= 10:
    X.append(x)
    Y1.append(f(x))
    Y2.append(g(x))
    Y3.append(h(x))
    x = x + 0.1

plt.plot(X, Y1, 'r-')
plt.plot(X, Y2, 'b-')
plt.plot(X, Y3, 'g-')
plt.show()
```

Si on souhaite afficher les noms des courbes, on peut modifier les 4 dernières lignes ainsi :

```
plt.plot(X, Y1, 'r-', label='Cf')
plt.plot(X, Y2, 'b-', label='Cg')
plt.plot(X, Y3, 'g-', label='Ch')
plt.legend()
plt.show()
```

Retour sur l'exercice D5

1. Si $c \neq 0$, la fonction f est définie sur $\left]-\infty;\frac{-d}{c}\right[\cup \left]\frac{-d}{c};+\infty\right[$, sinon, elle est définie sur $\mathbb{R}$ tout entier. (sauf si $c = d = 0$, mais dans ce cas la fonction n'existe pas).

2. a. f est une fonction affine (non constante) si $c = 0$ et $d \neq 0$. Bien évidemment, si on a de plus $a = 0$ cette fonction est constante...

   b. f est une fonction constante si le numérateur et le dénominateur sont proportionnels, donc si $ad - bc = 0$.

3. Voici l'algorithme complété :

```
FONCTION nature(a, b, c, d) :
    SI c ≠ 0 ALORS :
        SI ad − bc = 0 ALORS :
            RENVOYER "La fonction f est une fonction constante"
        SINON :
            RENVOYER "La fonction f est une fonction homographique"
        FIN SI
    SINON :
        SI d = 0 ALORS :
            RENVOYER "Impossible de définir la fonction f"
        SINON :
            RENVOYER "La fonction f est une fonction affine"
        FIN SI
    FIN SI
FIN FONCTION
```

4. Et sa traduction en Python :

```
def nature(a, b, c, d):
    if c != 0:
        if a * d - b * c == 0:
            return "La fonction f est une fonction constante"
        else:
            return "La fonction f est une fonction homographique"
    else:
        if d == 0:
            return "Impossible de définir la fonction f"
        else:
            return "La fonction f est une fonction affine"
```

Retour sur l'exercice D6

1. $\text{ch}(0) = 1$ et $\text{sh}(0) = 0$

2.
```
from math import exp

def ch(x):
    return (exp(x) + exp(-x)) / 2

def sh(x):
    return (exp(x) - exp(-x)) / 2
```

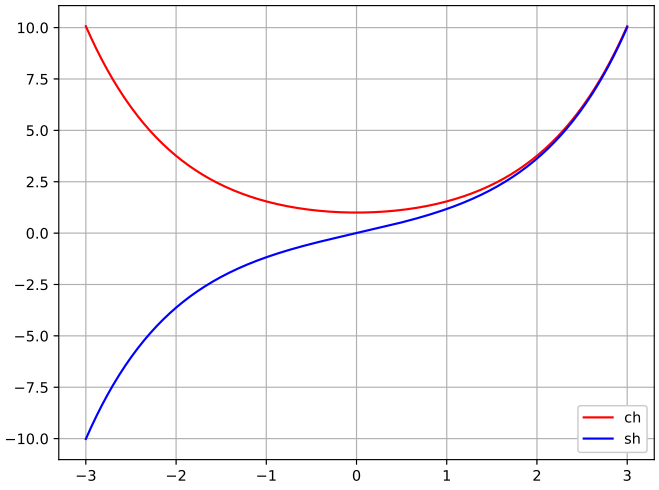
On vérifie :

```
>>> ch(0)
1.0
>>> sh(0)
0.0
```

Remarque : ces deux fonctions existent déjà dans le module `math` sous les noms **`cosh`** et **`sinh`**.

3. On peut alors tracer les deux courbes sur $[-3, 3]$ par exemple :

```
import matplotlib.pyplot as plt

x = -3
X, CH, SH = [], [], []
while x < 3 :
    X.append(x)
    CH.append(ch(x))
    SH.append(sh(x))
    x = x + 0.01
plt.plot(X, CH, 'r-', label='ch')
plt.plot(X, SH, 'b-', label='sh')
plt.grid()
plt.legend()
plt.show()
```



4. On conjecture que $\forall x \in \mathbb{R},\ \mathrm{ch}^2(x) - \mathrm{sh}^2(x) = 1$.

5. Pour tout réel x,

$$\begin{aligned} \mathrm{ch}^2(x) - \mathrm{sh}^2(x) &= \left(\frac{e^x + e^{-x}}{2}\right)^2 + \left(\frac{e^x - e^{-x}}{2}\right)^2 \\ &= \frac{\left(e^{2x} + 2 + e^{-2x}\right) - \left(e^{2x} - 2 + e^{-2x}\right)}{4} \\ &= 1 \end{aligned}$$

Retour sur l'exercice D7

1. Une boucle `for` suffira :

```
from random import uniform

total = 0
for i in range(1000) :
    x = uniform(0, 3)
    total = total + x**2
print("Moyenne", total/1000)
```

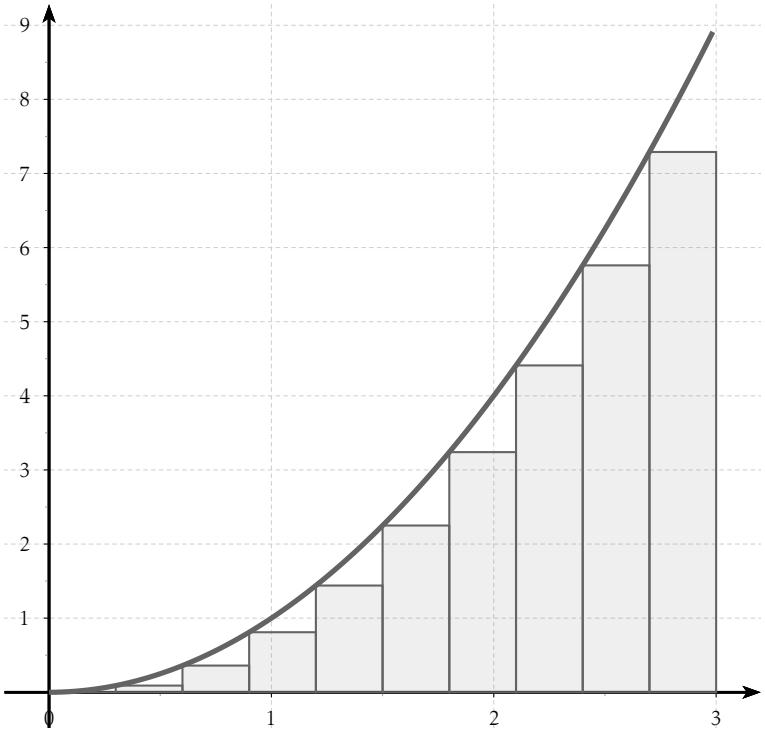
2. La fonction `moyenne` ressemble au code précédent, sauf que les x_i ne sont plus aléatoires :

```
def moyenne(n) :
    total = 0
    for i in range(n) :
        x = 3*i / n
        total = total + x**2
    return total / n
```

3. Les valeurs de `moyenne(n)` semblent converger vers 3 qui vaut bien $\frac{1}{3}\int_0^3 x^2 \mathrm{d}x$. En effet, graphiquement

$$\texttt{moyenne(n)} = \frac{1}{n}\sum_{i=0}^{n-1} f(x_i) = \frac{1}{3}\frac{3}{n}\sum_{i=0}^{n-1} f(x_i) = \frac{1}{3}\sum_{i=0}^{n-1} \underbrace{\frac{3}{n} \times f(x_i)}_{\text{rectangle gris}}$$

Donc `moyenne(n)` $\xrightarrow{n\to+\infty} \frac{1}{3}\int_0^3 x^2 \mathrm{d}x$



AU DÉTOUR DE LA BALADE...

Cet exercice permet de mieux appréhender la notion de valeur moyenne d'une fonction dont la formule peut parfois sembler "parachutée".

Retour sur l'exercice D8

1. Voici les réponses :

polynôme	racines
$x^2 - 5x + 6$	2 et 3
$x^2 + x + 1$	pas de racine réelle
$2x^2 - 12x + 18$	3
$2x^2 + 3x - 2$	-2 et 0,5

2. Il suffit de calculer le discriminant $\Delta = b^2 - 4ac$:

```
from math import sqrt

def racines(a, b, c):
    delta = b ** 2 - 4 * a * c
    if delta > 0:
        return [(-b-sqrt(delta))/(2*a), (-b+sqrt(delta))/(2*a)]
    elif delta == 0:
        return [-b/(2*a)]
    return []
```

À noter, que le dernier `return` ne nécessite pas de test, étant donné que si Python arrive à cette ligne, c'est qu'il n'a encore rien renvoyé (au premier `return`, Python quitte la fonction). D'ailleurs le `elif` pourrait être remplacé par un `if` pour les mêmes raisons.

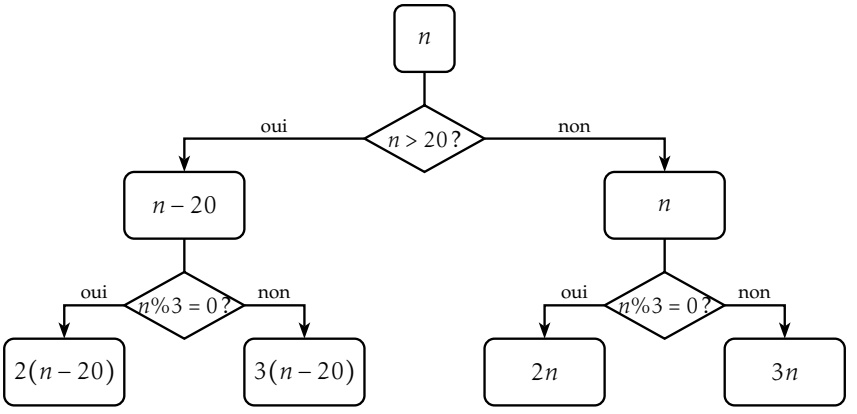
3. **AU DÉTOUR DE LA BALADE...**

La première question permet de se fabriquer un jeu de tests intéressants, en particulier les 2 derniers permettent de se rendre compte de l'erreur classique d'oublier les parenthèses au dénominateur.

Il peut d'ailleurs être intéressant d'imaginer régulièrement différents cas pour tester les programmes.

Retour sur l'exercice D9

1. Plusieurs cas possibles, on représente le contenu de la variable `n` dans le rectangle arrondi.



Les 4 expressions possibles sont :

$$f(n) = 2(n-20),\ f(n) = 3(n-20),\ f(n) = 2n \text{ et } f(n) = 3n$$

2. Recherche des points fixes.
Analyse : on résout $f(n) = n$ avec les 4 expressions possibles :

   - $2(n-20) = n \iff n = 40$
   - $3(n-20) = n \iff 2n = 60 \iff n = 30$
   - $2n = n \iff n = 0$
   - $3n = n \iff n = 0$

   Les points fixes, s'il y en a, sont donc parmi les valeurs $\{0; 30; 40\}$. Cependant rien n'assure que pour ces valeurs, l'algorithme prendra le chemin menant à l'expression souhaitée.

   Synthèse : On vérifie donc si ces valeurs sont bien des points fixes ou non.

```
>>> f(0)
0
>>> f(30)
30
>>> f(40)
60
```

   Seuls 0 et 30 sont des points fixes !

Retour sur l'exercice D10

1. f est définie sur $\mathbb{R}$, car seul le calcul de $\ln y$ pourrait poser souci. Or il n'a lieu que lorsque $y > 0$.

2. Analysons la condition pour que $y > 0$:
   $y > 0 \iff (x-2)^2 - 3 > 0 \iff x \in]-\infty; 2-\sqrt{3}[\cup]2+\sqrt{3}; +\infty[$. Ainsi :

   - Sur $]-\infty; 2-\sqrt{3}[\cup]2+\sqrt{3}; +\infty[$ la fonction f est définie par
   $$f(x) = \left((x-2)^2 - 3\right)\ln\left((x-2)^2 - 3\right).$$
   - Sur $[2-\sqrt{3}; 2+\sqrt{3}]$ la fonction f est définie par
   $$f(x) = \left((x-2)^2 - 3\right)\left(e^{2+x} - e^{2-x}\right)$$

   En étudiant ensuite les limites en $2+\sqrt{3}$ et en $2-\sqrt{3}$ à gauche et à droite, on obtient bien 0. La fonction f est continue sur $\mathbb{R}$.

3. Résolvons $f(x) = 0$ sur chacun des intervalles.

- Sur $]-\infty; 2-\sqrt{3}[\cup]2+\sqrt{3}; +\infty[$,

$$\begin{aligned} f(x) = 0 &\iff \underbrace{\left((x-2)^2-3\right)}_{>0} \ln\left((x-2)^2-3\right) = 0 \\ &\iff (x-2)^2 - 3 = 1 \iff x = 0 \text{ ou } x = 4 \\ &\quad \text{qui appartiennent bien à ces intervalles.} \end{aligned}$$

- Sur $I = [2-\sqrt{3}; 2+\sqrt{3}]$,

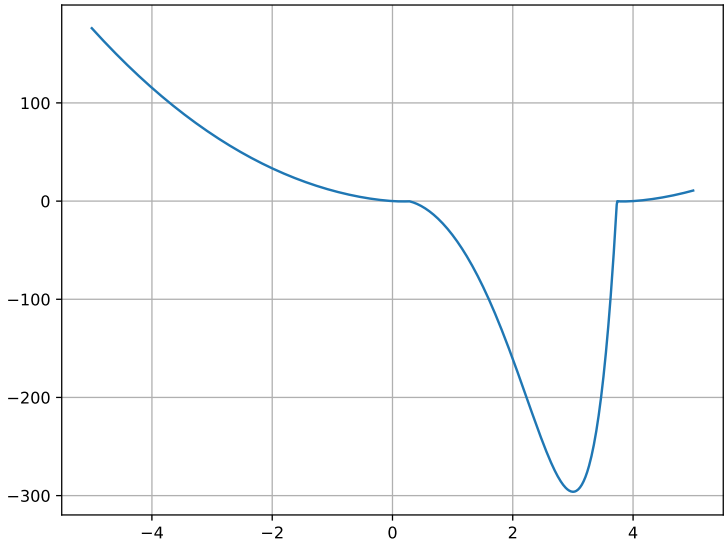
$$\begin{aligned} f(x) = 0 &\iff \left((x-2)^2-3\right)\left(e^{2+x} - e^{2-x}\right) = 0 \\ &\iff (x-2)^2 - 3 = 0 \text{ ou } e^{2+x} - e^{2-x} = 0 \\ &\iff x = 2 \pm \sqrt{3} \text{ ou } e^{2+x} = e^{2-x} \\ &\iff x = 2 \pm \sqrt{3} \text{ ou } \underbrace{x = 0}_{\notin I} \end{aligned}$$

Il y a donc 4 solutions : 0 ; $2-\sqrt{3}$; $2+\sqrt{3}$ et 4.

4. Voici un code utilisant la fonction `linspace` présentée dans la fiche sur le graphisme :

```
from math import log, exp
from numpy import linspace
import matplotlib.pyplot as plt

def f(x) :
    y = (x-2)**2 - 3
    if y > 0  :
        return y * log(y)
    else :
        x, y = y*exp(2-x), y*exp(2+x)
        return y - x

X = linspace(-5,5,1000)
Y = [f(x) for x in X]
plt.plot(X, Y)
plt.grid()
plt.show()
```

Et la sortie graphique :



Retour sur l'exercice D11

1. On crée la fonction à 2 paramètres :

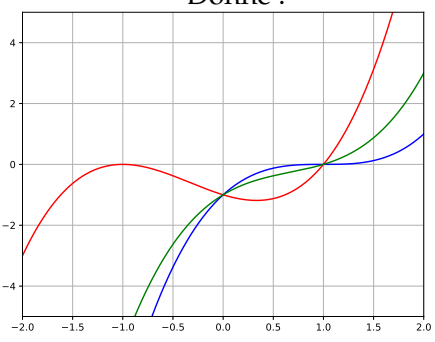
```
def f(m, x):
    return x**3 - m*x**2 + m*x - 1
```

2. On peut alors réaliser le tracé comme cela est expliqué dans la fiche sur les graphismes (troisième méthode).

```
from numpy import linspace
import matplotlib.pyplot as plt

def C(m, coul):
    X = linspace(-2, 2, 1000)
    Y = [f(m,x) for x in X]
    plt.plot(X, Y, coul)
```

```
C(-1, 'r')
C(3, 'b')
C(2, 'g')
plt.axis([-2, 2, -5, 5])
plt.grid()
plt.show()
```

Donne :



Remarque : on n'a pas placé le `plt.show()` dans la fonction volontairement, ce qui permet de réaliser plusieurs tracés sur la même fenêtre.

3. Toutes les courbes semblent passer par les points $(0,-1)$ et $(1,0)$. En effet, pour tout réel m :
   - $f(0) = 0^3 - m \times 0^2 + m \times 0 - 1 = -1$
   - $f(1) = 1^3 - m \times 1^2 + m \times 1 - 1 = 1 - m + m - 1 = 0$

4. a. $f_{-1,5}(x) = 0 \iff x^3 + 1,5x^2 - 1,5x - 1 = 0 \iff 2x^3 + 3x^2 - 3x - 2 = 0$.
   On sait que 1 est solution d'après la question précédente, cela donne l'idée de factoriser par $(x-1)$ pour la retrouver :
   $$\begin{aligned} 2x^3 + 3x^2 - 3x - 2 = 0 &\iff (x-1)(2x^2 + 5x + 2) = 0 \\ &\iff x = 1 \text{ ou } 2x^2 + 5x + 2 = 0 \\ &\iff x = 1 \text{ ou } x = -2 \text{ ou } x = -1/2 \end{aligned}$$

   b. De la même manière :
   $$\begin{aligned} f_{-1}(x) = 0 &\iff x^3 + x^2 - x - 1 = 0 \\ &\iff (x-1)(x^2 + 2x + 1) = 0 \\ &\iff x = 1 \text{ ou } x = -1 \end{aligned}$$

   c. Si $m = 0$: il y a une seule solution :
   $$\begin{aligned} f_{-1}(x) = 0 &\iff x^3 - 1 = 0 \\ &\iff (x-1)(x^2 + x + 1) = 0 \\ &\iff x = 1 \end{aligned}$$

   d. Une unique aussi dans le cas où $m = 3$, mais triple :
   $$\begin{aligned} f_{-1}(x) = 0 &\iff x^3 - 3x^2 + 3x - 1 = 0 \\ &\iff (x-1)^3 = 0 \\ &\iff x = 1 \end{aligned}$$

   e. Après quelques tentatives, il semblerait que pour $m = 1$ entre autres, la courbe $\mathcal{C}_1$ ne possède pas de tangente horizontale. En effet, par l'absurde, si c'était le cas, il existerait un réel x_0 tel que $f_1'(x_0) = 0$. Or,
   $f_1'(x) = 0 \iff 3x^2 - 2x + 1 = 0$ n'a pas de solution réelle.

Retour sur l'exercice D12

1. f est la somme de fonctions continues et dérivables sur $\mathbb{R}$. Pour tout réel x, $f'(x) = e^x + 1$ d'où $f'(x) > 0$. On en déduit que la fonction f est continue et strictement croissante sur $\mathbb{R}$.

2. $\lim\limits_{x \to -\infty} e^x + x - 2 = -\infty$ et $\lim\limits_{x \to +\infty} e^x + x - 2 = +\infty$, comme f est strictement croissante et continue sur $\mathbb{R}$, l'équation $f(x) = 0$ admet une unique solution. De plus $f(0) = -2 < 0$ et $f(1) = e - 1 > 0$ donc la solution est dans l'intervalle $[0;1]$.

3. On reconnaît l'algorithme de dichotomie, dont vous retrouverez le principe dans les pages du thème « les nombres remarquables ». L'algorithme 4 est correct.

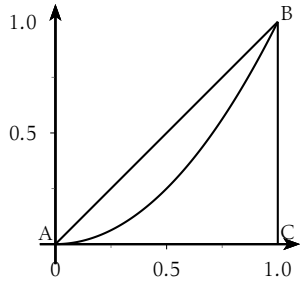
4. Une solution :

```
from math import exp

def f(x):
    return exp(x) + x - 2

def equation(a, b):
    while abs(b-a) >= 0.001:
        m = (a+b) / 2
        if f(m) < 0:
            a = m
        else:
            b = m
    return m
```

```
>>> equation(0, 1)
0.4423828125
```

On obtient $0,442$ qui est une valeur approchée par défaut à $0,001$ près de la solution.

Retour sur l'exercice D13

1. 

Graphiquement la longueur de la parabole cherchée est supérieure à la longueur AB qui vaut $\sqrt{2}$ (théorème de Pythagore) et inférieure à la longueur du chemin $AC + CB = 1 + 1 = 2$.

2. a. Notons x_k les abscisses des points A_k. L'écart entre deux valeurs x_k et x_{k+1} est constant et vaut $\frac{1}{n}$ d'où $x_k = \frac{k}{n}$. Ainsi $A_k\left(\frac{k}{n};0\right)$, et puisque les points B_k sont sur la courbe, $B_k\left(\frac{k}{n};\frac{k^2}{n^2}\right)$.

b. $B_kB_{k+1} = \sqrt{\left(\frac{k+1}{n} - \frac{k}{n}\right)^2 + \left(\frac{(k+1)^2}{n^2} - \frac{(k)^2}{n^2}\right)^2}$

$$= \sqrt{\left(\frac{1}{n}\right)^2 + \left(\frac{k^2+2k+1-k^2}{n^2}\right)^2} = \sqrt{\frac{1}{n^2} + \frac{(2k+1)^2}{n^4}}$$
$$= \sqrt{\frac{n^2}{n^4} + \frac{4k^2+4k+1}{n^4}} = \frac{\sqrt{n^2+4k^2+4k+1}}{n^2}$$

c. En réalisant une boucle pour k allant de 0 à $n-1$ on va ajouter tour à tour les longueurs B_kB_{k+1} :

```
def longueur(n):
    somme = 0
    for k in range(n):
        somme = somme + sqrt(n**2 + 4*k**2 + 4*k + 1) / (n**2)
    return somme
```

3. En reprenant la fonction, il suffit d'écrire la condition demandée :

```
n = 1
while abs(longueur(n+1) - longueur(n)) > 10 ** -10:
    n = n + 1
print(n, longueur(n))
```

On trouve alors $n = 1142$ et $\ell \approx 1,479$. On peut remarquer que la valeur absolue n'est pas nécessaire et que de plus, le calcul de chaque longueur se fait 2 fois. On pourrait donc optimiser le programme ainsi pour améliorer sensiblement le temps d'exécution.

```
n = 1
l_actuelle = longueur(1)
l_suivante = longueur(2)
while l_suivante - l_actuelle > 10 ** -10:
    n = n + 1
    l_actuelle, l_suivante = l_suivante, longueur(n+1)
print(n, l_actuelle)
```

4. a. D'après la formule, puisque $f(x) = x^2$ et $f'(x) = 2x$, on trouve :

$$\ell = \int_0^1 \sqrt{1+(2x)^2}\,dx = \int_0^1 \sqrt{1+4x^2}\,dx$$

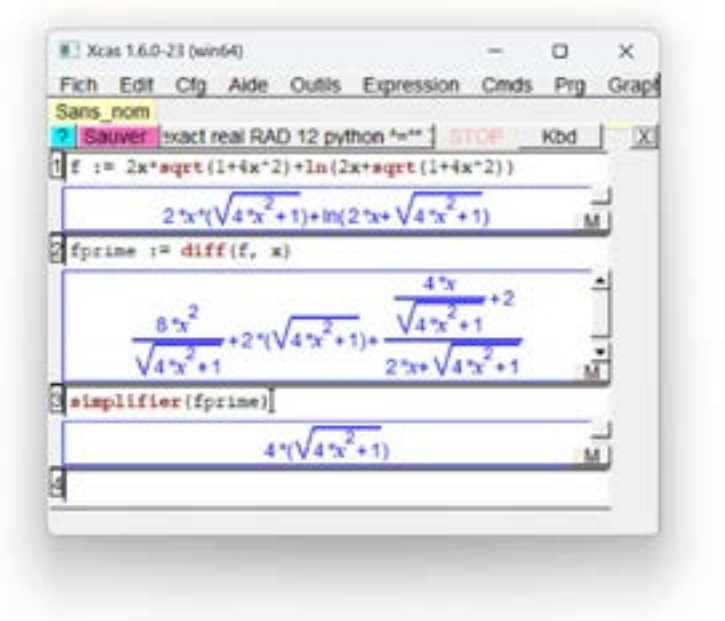
b. Le calcul peut sembler compliqué, il faut s'organiser.
Découpons φ en une somme $\varphi_1 + \varphi_2$ avec $\varphi_1(x) = 2x\sqrt{1+4x^2}$ et $\varphi_2(x) = \ln\left(2x+\sqrt{1+4x^2}\right)$. Sur l'intervalle $[0,1]$,

- $\varphi_1'(x) = 2 \times \sqrt{1+4x^2} + 2x \times \dfrac{8x}{2\sqrt{1+4x^2}}$
$= \dfrac{2(1+4x^2)+8x^2}{\sqrt{1+4x^2}} = \dfrac{16x^2+2}{\sqrt{1+4x^2}}$

- $\varphi_2'(x) = \dfrac{2 + \frac{8x}{2\sqrt{1+4x^2}}}{2x + \sqrt{1+4x^2}} = \dfrac{\frac{2\sqrt{1+4x^2}+4x}{\sqrt{1+4x^2}}}{2x + \sqrt{1+4x^2}}$
$= \dfrac{2(\sqrt{1+4x^2}+2x)}{\sqrt{1+4x^2}} \times \dfrac{1}{2x+\sqrt{1+4x^2}} = \dfrac{2}{\sqrt{1+4x^2}}$

Finalement, $\varphi'(x) = \dfrac{16x^2+2}{\sqrt{1+4x^2}} + \dfrac{2}{\sqrt{1+4x^2}} = \dfrac{16x^2+4}{\sqrt{1+4x^2}} = 4\dfrac{4x^2+1}{\sqrt{1+4x^2}}$
$= 4\sqrt{1+4x^2}$ car $4x^2+1 \geq 0$

Pour des calculs aussi laborieux, le calcul formel permet de vérifier ou d'avancer plus rapidement, par exemple avec XCas, on obtient le même résultat :



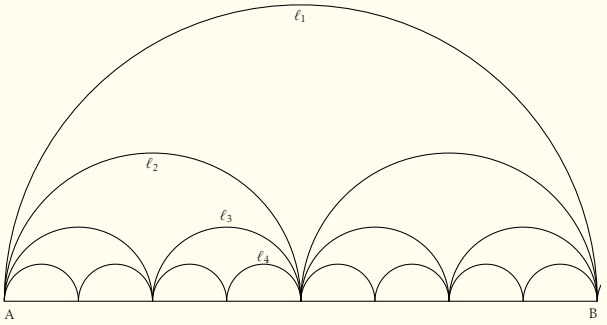
c. On est à présent capable de calculer l'intégrale cherchée :

$$\ell = \int_0^1 \sqrt{1+4x^2}\,dx = \left[\frac{\varphi(x)}{4}\right]_0^1 = \frac{1}{4}(\varphi(1) - \varphi(0))$$

$$\ell = \frac{2\sqrt{5} + \ln(2+\sqrt{5})}{4}$$

AU DÉTOUR DE LA BALADE...

Il faut faire attention à ce genre de "découpages" qui semblent converger vers la longueur souhaitée. Le grand mathématicien français Lebesgue était lui-même longtemps resté bloqué sur le paradoxe suivant :



Ici le segment [AB] *mesure* 2*cm. Il semblerait que les longueurs* ℓ_n *convergent vers cette longueur. Pourtant, on montre facilement que pour tout entier* n *non nul,* $\ell_n = \pi$. *Ainsi on aurait* $2 = \lim\limits_{n\to+\infty} \ell_n = \pi$.

D'où $\boxed{\pi = 2}$

Retour sur l'exercice D14 Avant de commencer le problème, on peut remarquer que sans modifier les solutions, on peut supposer que $a < b$.

1. Une double boucle permet de répondre à la question. Pour la seconde, on commence à `a + 1` puisqu'on a supposé $b > a$:

```
for a in range(10**5) :
    for b in range(a+1, 10**5):
        if a**b == b**a:
            print(f"{a}^{b}={b}^{a}={a**b}")
```

   On trouve une unique solution : $2^4 = 4^2 = 16$. (Ici on a utilisé des f-strings déjà présentées dans le livre pour un affichage plus simple).

2. si $a = 0$, on a $0^b = b^0$, soit $0 = 1$, donc $a > 0$ et b aussi par conséquent. On peut donc appliquer la fonction ln :

   Si $0 < a < b$, $a^b = b^a \iff \ln a^b = \ln b^a \iff b\ln a = a\ln b \iff \dfrac{\ln a}{a} = \dfrac{\ln b}{b}$.

   La fonction $f : x \longmapsto \dfrac{\ln x}{x}$ répond donc à la question.

3. Dressons le tableau de variation de cette fonction sur $[1;+\infty[$:

x	1		e		$+\infty$
Var f	0	↗	$\frac{1}{e}$	↘	0

Comme la fonction f est une bijection sur $[1;e]$ et sur $[e;+\infty[$, on déduit que si ces deux entiers a et b existent, il faut que $a \in [1;e]$ et $b \in [e;+\infty[$. Donc $a \in \{1;2\}$.

Pour $a = 1$, c'est impossible, car f ne s'annule pas une seconde fois. Pour $a = 2$, nous l'avons déjà trouvée. Ainsi $2^4 = 4^2 = 16$ est la seule possibilité.

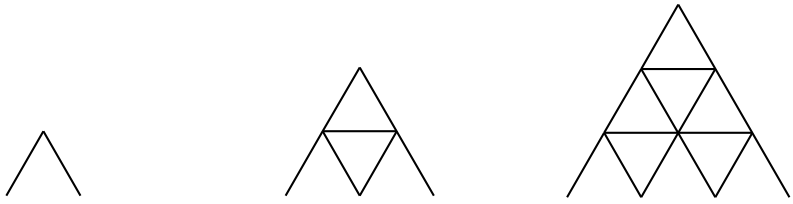
Chapitre E

MODÈLES ÉVOLUTIFS ET SUITES

1. Echauffons-nous "tout de suite"...

E1 *Une balle rebondissante est lâchée du haut d'un immeuble de 10 mètres. À chaque rebond la hauteur diminue d'un tiers. Combien de fois la personne du premier niveau dont la fenêtre est située à 1 mètre 50 du sol verra-t-elle passer la balle ?*

E2 *On se propose de réaliser un château de cartes de la manière suivante :*



Étape 1 *Étape 2* *Étape 3*

1. *Combien faut-il de cartes pour réaliser un château à la $4^{ème}$ étape ?*
2. *Si on note e l'étape d'un château, combien faut-il ajouter de cartes pour passer à un château à l'étape $e+1$?*
3. *En déduire une fonction qui reçoit en paramètre le nombre d'étapes voulu et renvoie le nombre de cartes nécessaires à la réalisation de ce château.*
4. *Créer la fonction inverse : elle reçoit en paramètre le nombre de cartes à disposition et renvoie le nombre d'étapes maximal que l'on peut réaliser.*

E3 *Axel vient de trouver son premier emploi. Il compte investir dans une automobile qui lui a tapé dans l'œil. Malheureusement elle coûte 13 900 € neuve et il ne dispose que de 2000 €. Il décide donc de l'acheter plus tard d'occasion en épargnant (sans faire de placement) chaque mois. Dans le même temps, le véhicule perd chaque mois 2% de sa valeur. Écrire une fonction qui reçoit en entrée la somme épargnée chaque mois et renvoie le nombre de mois qu'il lui faut attendre pour se payer la voiture.*

E4 *On considère la fonction ci contre :*

1. *Que renvoie* `calcul(10)` *?*
2. *Que se passe-t-il si on remplace à la 4ème ligne* `i**2` *par* `i` *? Expliquer.*
3. *En déduire une formule en fonction de n qui donne directement* `calcul(n)`*.*

```
1 def calcul(n) :
2     s = 0
3     for i in range(n+1) :
4         if i**2 % 2 == 0 :
5             s = s + i
6     return s
```

E5 *Myriam envisage d'acheter l'appartement qu'elle loue actuellement sur Lyon. Elle hésite entre attendre et économiser ou effectuer un crédit.*

1. *On suppose que Myriam a 20 000€ d'apport qu'elle place à un taux annuel de 2%. Écrire une fonction qui reçoit en paramètres la somme de départ, le nombre d'années de placement et le taux en pourcentage et renvoie la somme d'argent disponible après ces années. Utiliser cette fonction pour déterminer la somme dont disposera Myriam après 20 ans.*
2. *Myriam paie un loyer de 790 euros chaque mois. Écrire une fonction qui renvoie le montant des loyers payés sur n années complètes.*
3. *Une banque lui propose un prêt à un taux annuel de 3,6%.*
   a. *Créer une fonction qui admet pour paramètre le taux annuel T en % et qui renvoie le taux mensuel t_m en pourcentage.*

b. *Chaque mensualité d'un crédit est composée d'un remboursement d'une partie du capital emprunté et du remboursement des intérêts. La mensualité d'un crédit se calcule avec la formule suivante :*

$$m = \frac{\mathrm{C} \times t_m}{1 - (1 + t_m)^{-\mathrm{N}}}$$

où C *est le capital emprunté,* t_m *est le taux mensuel et* N *le nombre de mensualités. Créer une fonction qui admet pour paramètres* C, t_m *et* N *et qui renvoie le montant de la mensualité.*

c. *Myriam souhaite emprunter 150 000 euros sur 20 ans à un taux annuel de* 3,6%. *Quelles seront les mensualités ?*

d. *En déduire le montant des intérêts payés par Myriam.*

2. Jouons avec les suites

E6 *Un magicien propose à Anna de choisir deux nombres entiers. Anna choisit 5 et 12. Le magicien écrit ensuite au tableau une suite de nombres :*

5, 12, 17, 29, 46, 75, 121, 196, 317, 513, 830, 1343, 2173

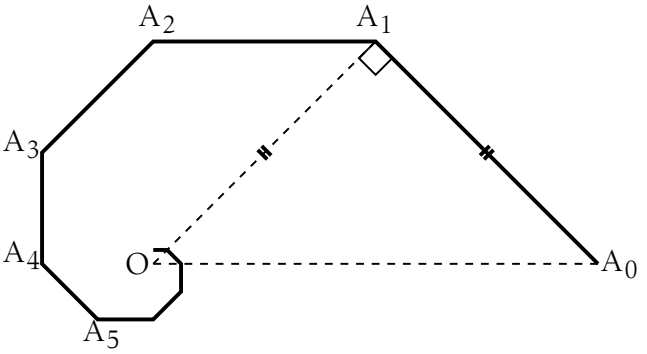
Il demande ensuite à Anna de choisir un nombre dans la liste et affirme qu'il pourra donner la somme des termes jusqu'à celui indiqué par Anna en une fraction de seconde. Anna choisit 830. Et en effet, le magicien dit tranquillement "la somme est 2161".

1. *En analysant cette suite, créer une fonction qui reçoit les 2 nombres choisis et un entier* `n` *et qui renvoie la liste de longueur* `n` *des termes générée à partir de ces 2 nombres.*

2. *Créer une fonction ayant pour paramètres une liste* `L` *et un élément* `a` ***de cette liste*** *et qui renvoie la somme des nombres de la liste* `L` *jusqu'à celui-ci (inclus).*

3. *Vérifier la réponse du magicien et justifier mathématiquement comment il a pu donner une réponse aussi rapide sans utiliser l'ordinateur.*

E7 *Dans le plan, on considère un segment* $[OA_0]$ *de 4cm de longueur. On construit la suite de points* $A_1, A_2, A_3, \ldots$ *de telle manière que, pour tout entier naturel* k*, le triangle* OA_kA_{k+1} *soit rectangle isocèle en* A_{k+1} *comme l'illustre la figure ci-dessous :*



1. *Démontrer que* $OA_1 = A_0A_1 = 2\sqrt{2}$.
2. *On réitère ainsi le procédé en traçant la suite de triangles rectangles isocèles. Déterminer une relation entre* OA_{n+1} *et* OA_n.
3. *Écrire une fonction qui prend en paramètre la valeur* n *et qui renvoie la distance* OA_n.
4. *Donner la nature de la suite* (u_n) *définie par* $u_n = OA_n$.
5. *Créer une fonction qui prend en paramètre la valeur* n *et qui renvoie la longueur* L_n *de la ligne brisée* $A_0A_1A_2A_3...A_n$*. Conjecturer la limite de* L_n.
6. *Exprimer* L_n *en fonction de* n.
7. *Déterminer par le calcul la limite de la suite* (L_n).

3. Des suites et des suites, encore des suites

E8 *Soit* (u_n) *la suite définie sur* $\mathbb{N}$ *par* $u_n = \dfrac{n^3}{n+3}$.

1. *Créer une fonction* `u` *qui reçoit un entier* n *et renvoie la valeur de* u_n*. Vérifier les résultats obtenus en calculant quelques valeurs à la main.*
2. *En déduire un programme affichant les* 1000 *premières valeurs de* u_n*. Quelle semble être la limite de* (u_n) *?*

3. *Modifier le programme précédent pour qu'il trouve le plus petit entier n tel que $u_n > 10^{10}$.*

E9 *On considère la fonction informatique ci-dessous qui reçoit en paramètre un entier naturel.*

```
def suite(n):
    u, i = 3, 0
    while i < n:
        u = 2 * u - 1
        i = i + 1
    return u
```

1. *Cette fonction renvoie le n-ième terme d'une suite. Donner l'expression de cette suite.*

2. *La modifier pour calculer le n-ième terme de la suite (v_n) définie par :*

$$\begin{cases} v_0 = 1 \\ v_{n+1} = \dfrac{v_n + 4}{3} \text{ pour tout } n \in \mathbb{N} \end{cases}$$

3. *Quelle semble être la limite de (v_n) ?*

4. *En considérant la suite (w_n) définie sur $\mathbb{N}$, par $w_n = v_n - 2$.*

   a. *Montrer que (w_n) est une suite géométrique.*

   b. *En déduire une expression de w_n en fonction de n, puis celle de v_n.*

   c. *Conclure sur la limite de (v_n).*

E10 ***D'après un exemple de la documentation AmiensPytnon***

Soit (u_n) la suite définie par :

$$\begin{cases} u_0 = 2,\ u_1 = 5 \\ \forall n \in \mathbb{N}^*, u_{n+1} = \sqrt{u_{n-1} \times u_n + 9(u_n + 1)} \end{cases}$$

1. *Écrire une fonction qui reçoit un entier n en paramètre et renvoie u_n.*

2. *Que peut-on conjecturer pour la suite (u_n) ?*

3. *Le démontrer par récurrence.*

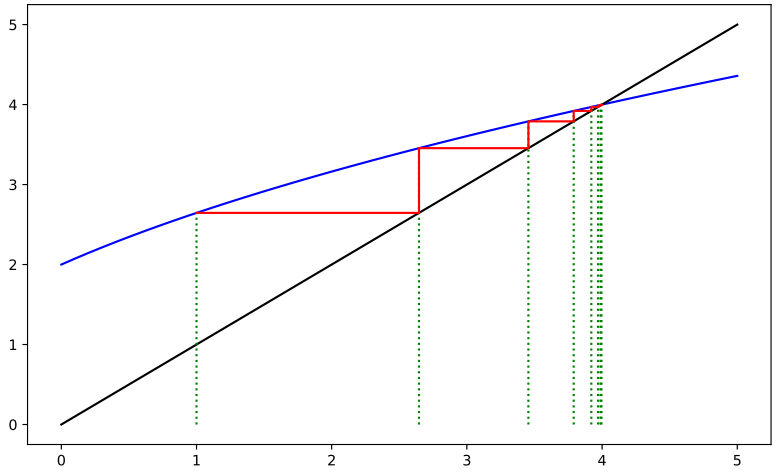
E11 *On cherche à représenter la suite définie par*

$$\begin{cases} u_0 = 1 \\ u_{n+1} = \sqrt{3u_n + 4} \text{ pour } n \in \mathbb{N} \end{cases}$$

On note f la fonction : $x \longmapsto \sqrt{3x+4}$.

1. a. *Représenter la fonction f sur l'intervalle $[0;5]$ ainsi que la droite d'équation $y = x$.*

   b. *Pour une valeur de n donnée, générer le tracé des valeurs de u_n comme on le ferait avec un papier et un crayon et afficher les premières valeurs.*



   c. *Quelles conjectures peut-on faire sur la suite (u_n) (variations et convergence) ?*

2. a. *Démontrer par récurrence que $\forall n \in \mathbb{N},\ 0 \leqslant u_n \leqslant u_{n+1} \leqslant 4$.*

   b. *En déduire une preuve de la conjecture.*

3. *On considère cette fois la suite (v_n) définie par* $\begin{cases} v_0 = 1 \\ v_{n+1} = \dfrac{8}{u_n + 1} \text{ pour } n \in \mathbb{N} \end{cases}$

   a. *Modifier le programme pour faire une conjecture sur la convergence de (v_n).*

   b. *En supposant que (v_n) converge, quelle peut être la valeur de sa limite ?*

E12 *Pour $n \in \mathbb{N}^*$, on définit* $\mathrm{H}_n = 1 + \dfrac{1}{2} + \dfrac{1}{3} + \cdots + \dfrac{1}{n} = \displaystyle\sum_{k=1}^{n} \frac{1}{k}$.

1. *Parmi les fonctions Python suivantes, quelle est celle qui permet de calculer et renvoyer H_n ?*

proposition 1 :

```
def H(n):
    S = 0
    for i in range(n):
        S = 1/(i+1)
    return S
```

proposition 3 :

```
def H(n):
    i = 0
    while S < n:
        S = S + 1/(i+1)
    return S
```

proposition 2 :

```
def H(n):
    S = 0
    for i in range(n):
        S = S + 1/(i+1)
    return S
```

proposition 4 :

```
def H(n):
    i = 0
    while i < n:
        S = S + 1/(i+1)
    return S
```

2. *Représenter sur un graphique avec des points rouges, les valeurs de H_n en fonction de n pour les 100 premières valeurs de n.*

3. *On reconnaît (normalement) l'allure d'une courbe ressemblant à celle de la fonction logarithme népérien. Tracer en vert cette courbe sur le même graphique, ainsi qu'en bleu les termes de la suite correspondant à l'écart entre les deux courbes : pour $n \in \mathbb{N}^*$, $u_n = \mathrm{H}_n - \ln n$.*

4. *Que peut-on conjecturer pour la suite (u_n) ?*

5. a. *Soit k un entier supérieur ou égal à 1, justifier que* $\dfrac{1}{k} \geqslant \displaystyle\int_k^{k+1} \frac{\mathrm{d}t}{t}$.

   b. *En déduire que pour tout $n \in \mathbb{N}^*$, $u_n \geqslant 0$.*

   c. *Étudier les variations de la suite (u_n).*

   d. *En déduire que (u_n) converge vers un réel que l'on notera γ.*

AU DÉTOUR DE LA BALADE...

*Pour information H_n s'appelle **série harmonique** et le nombre γ est appelé **constante d'Euler** et vaut environ 0,577.*

E13 *Pour $n \in \mathbb{N}$, on considère la quantité* $\mathrm{I}_n = \displaystyle\int_0^1 (1-x)^n e^x \,\mathrm{d}x$.

1. *Une première conjecture :*

   a. *Calculer* I_0.

   b. *Pour $n \in \mathbb{N}$, démontrer que* $\mathrm{I}_{n+1} = (n+1)\mathrm{I}_n - 1$.

   c. *Déduire de cette formule de récurrence une fonction Python qui renvoie* I_n *pour un entier n donné en paramètre.*

   d. *Que peut-on conjecturer pour la limite de* I_n *?*

2. *Analyse critique :*

   a. *À l'aide d'un programme Python, afficher les courbes représentant les fonctions :* $f_n : x \longmapsto (1-x)^n e^x$ *sur l'intervalle* $[0,1]$ *pour* $0 \leqslant n \leqslant 9$.

   b. *Que dire par rapport à la conjecture émise en 1. ?*

3. *Calcul de la limite de* I_n *:*

   a. *Après avoir encadré la fonction* $x \longmapsto e^x$ *sur* $[0,1]$, *donner un encadrement de* I_n.

   b. *En déduire la limite de* I_n.

4. *Une explication : soit* J_n *une autre suite de premier terme* $\mathrm{J}_0 = e - 1 + \varepsilon$ *où* ε *est un réel donné et vérifiant la même relation de récurrence que* I_n, *soit* $\forall n \in \mathbb{N} : \mathrm{J}_{n+1} = (n+1)\mathrm{J}_n - 1$. *On note* E_n *l'écart entre* J_n *et* I_n.

   a. *Calculer* E_0.

   b. *Exprimer* E_{n+1} *en fonction de* E_n.

   c. *En déduire une expression de* E_n *en fonction de n et* ε.

   d. *Quelle est la limite de* (E_n), *puis de* (J_n) *?*

   e. *Conclure.*

E14 *On définit les suites* (u_n) *et* (v_n) *sur l'ensemble des entiers naturels par*

$$\begin{cases} u_0 = 0 \\ v_0 = 1 \end{cases} \quad \text{et } \forall n \in \mathbb{N}, \quad \begin{cases} u_{n+1} = \dfrac{u_n + v_n}{2} \\ v_{n+1} = \dfrac{u_n + 2v_n}{3} \end{cases}$$

1. a. *Calculer* u_1, u_2, v_1 *et* v_2.
   b. *Écrire une fonction qui reçoit en paramètre un entier naturel* n *et qui renvoie le couple* (u_n, v_n).
   c. *Quelles conjectures peut-on émettre quant aux limites et variations des suites ?*

2. *Première méthode : étude de ces suites avec les matrices.*

   a. *Pour tout entier naturel* n, *on définit le vecteur colonne* $X_n = \begin{pmatrix} u_n \\ v_n \end{pmatrix}$ *et la matrice* $A = \begin{pmatrix} 1/2 & 1/2 \\ 1/3 & 2/3 \end{pmatrix}$.
   Vérifier que pour tout entier n, $X_{n+1} = AX_n$.
   b. *Démontrer par récurrence que pour tout entier* n, $X_n = A^n X_0$
   c. *On définit les matrices* P, P' *et* B *par :*

$$P = \begin{pmatrix} 1 & -3/2 \\ 1 & 1 \end{pmatrix} \quad P' = \begin{pmatrix} 2/5 & 3/5 \\ -2/5 & 2/5 \end{pmatrix} \quad \text{et} \quad B = \begin{pmatrix} 1 & 0 \\ 0 & 1/6 \end{pmatrix}$$

   Montrer que $A = PBP'$
   d. *Démontrer par récurrence que pour tout entier naturel* n, $A^n = PB^nP'$.
   e. *On admet que pour tout entier naturel* n, $B^n = \begin{pmatrix} 1 & 0 \\ 0 & 1/6^n \end{pmatrix}$ *En déduire l'expression de la matrice* A^n *en fonction de* n.
   f. *Montrer que pour tout entier naturel* n, $X_n = \begin{pmatrix} 3/5 - 3/5(1/6)^n \\ 3/5 + 2/5(1/6)^n \end{pmatrix}$
   g. *En déduire les expressions de* u_n *et de* v_n *en fonction de* n.
   h. *En déduire les limites des suites* (u_n) *et* (v_n).

3. *Seconde méthode à l'aide de suites auxiliaires. On pose pour tout entier* n, $w_n = v_n - u_n$.

   a. *Écrire en Python une fonction qui prend en entrée un entier naturel* n *et donne en sortie* $(u_n; v_n; w_n)$.
   b. *Conjecturer la nature de la suite* (w_n), *puis démontrer cette conjecture.*
   c. *Donner l'expression de* w_n *en fonction de* n.
   d. *On pose pour tout entier* n, $t_n = 2u_n + 3v_n$. *Montrer que la suite* (t_n) *est constante.*
   e. *En déduire que* $u_n = \dfrac{3}{5} - \dfrac{3}{5 \times 6^n}$ *et* $v_n = \dfrac{3}{5} + \dfrac{2}{5 \times 6^n}$.
   f. *Retrouver les limites des suites* (u_n) *et* (v_n).

4. Si vous n'êtes pas fatigué, un peu d'originalité

E15 *On propose le processus suivant de génération de briques numérotés :*

- *Étape 0 :* | 1 | 1 |
- *Étape 1 : on écarte les deux briques précédentes et on obtient :* | 1 | 2 | 1 | *car 2 est la somme des deux briques voisines ;*
- *Étape 2 : on fait de même, on obtient :* | 1 | 3 | 2 | 3 | 1 |
- *Étape 3 :* | 1 | 4 | 3 | 5 | 2 | 5 | 3 | 4 | 1 | *... Et ainsi de suite...*

1. *Adil affirme qu'on obtient 1025 nombres à l'étape 10.*
   a. *On note u_n le nombre de cases du tableau à l'étape n. Trouver une relation de récurrence sur u_n.*
   b. *À l'aide d'un programme, commenter l'affirmation de Adil.*
2. *Emma affirme qu'à l'étape 10, le plus grand nombre est 144.*
   a. *Écrire une fonction `suivant` qui reçoit une liste représentant les briques d'une étape et renvoie la liste représentant les briques à l'étape suivante. Par exemple :*

```
>>> suivant([1,2,1])
[1, 3, 2, 3, 1]
```

   b. *À l'aide de cette fonction, dire si Emma a raison.*
3. *Aïcha dit alors aux 2 autres que s'ils ne comptent à l'étape 10 que les nombres différents, il y en a seulement 118.*
   a. *Écrire une fonction `present(L,e)` qui reçoit une liste `L` et un nombre `e` et renvoie un booléen indiquant si l'élément `e` est présent dans `L`.*
   b. *Répondre alors sur la véracité de l'affirmation de Aïcha.*

AU DÉTOUR DE LA BALADE...

*Les plus perspicaces auront remarqué que les maxima suivent **une suite de Fibonacci**. Les rangs des valeurs maximales suivent quant à eux une **suite de Jacobsthal**. Cela doit pouvoir se démontrer !*

E16

Inspiré des travaux d'E. Sebert

On s'intéresse à la modélisation de la propagation de graines dans un champ par le vent ou par des oiseaux. On suppose pour simplifier que le champ est carré et est composé de 9 parcelles toutes identiques nommées ainsi :

A	B	C
D	E	F
G	H	I

Au centre de la parcelle, on a une quantité initiale de graines et les autres parcelles n'ont aucune graine.

Étape 0 :

0	0	0
0	1	0
0	0	0

On opte pour les règles d'évolution suivantes :

- *au pas de temps $n+1$, la moitié des graines présentes au pas de temps n dans chaque parcelle se répartit de façon équitable dans les parcelles du champ voisines (4 directions) alors que l'autre moitié des graines reste sur place.*
- *les graines ne peuvent pas se disperser à l'extérieur du champ.*

1. *Montrer qu'à l'étape 1 la configuration est la suivante :*

Étape 1 :

0	$\frac{1}{8}$	0
$\frac{1}{8}$	$\frac{1}{2}$	$\frac{1}{8}$
0	$\frac{1}{8}$	0

2. *Quelle configuration obtient-on à l'étape 2 ?*
3. *À l'étape n avec n entier naturel, on note a_n la quantité de graines dans la parcelle* A, *b_n la quantité de graines dans la parcelle* B *et ainsi de suite ...*
   a. *Trouver une formule de récurrence pour exprimer les quantités a_{n+1}, b_{n+1}, c_{n+1}, ... i_{n+1} en fonction des quantités $a_n, b_n, c_n, ...i_n$.*

b. *En déduire une fonction* `evolution` *qui reçoit une liste de 9 valeurs* $[a_n, b_n, ...i_n]$ *et renvoie la nouvelle liste* $[a_{n+1}, ..., i_{n+1}]$.

c. *Que constate-t-on avec l'initialisation de l'étape 0 sur les valeurs des 9 cases ? Est-ce toujours le cas ?*

4. *On se place dans le cas de l'initialisation de l'étape 0, on ne garde ainsi que les variables* a_n, b_n *et* e_n.

   a. *On admet que ces 3 suites convergent, calculer leurs limites.*

   b. *Retrouver ces résultats*

5. *Que se passe-t-il si on initialise le tableau autrement ?*

5. Aide pour les exercices

Aide pour l'exercice E1 La réponse est 9 fois :

- 1 fois lorsqu'elle tombe des 10 mètres
- 2 fois (montée et descente) lorsqu'elle monte jusqu'à $6,67$ mètres
- 2 fois lorsqu'elle monte jusqu'à $4,44$ mètres
- 2 fois lorsqu'elle monte jusqu'à $2,96$ mètres
- 2 fois lorsqu'elle monte jusqu'à $1,98$ mètres

Aide pour l'exercice E3 On peut utiliser deux variables : l'une pour emmagasiner la somme économisée, l'autre pour calculer le prix de la voiture au fil des mois.

Aide pour l'exercice E6

1. Trouver la relation entre un terme donné et les précédents.

3. Regarder la somme jusqu'au nombre souhaité et le terme situé 2 rangs plus loin dans la liste....

Aide pour l'exercice E8 Vérifier par exemple que $u_1 = 0.25$ et que $u_6 = 24$.

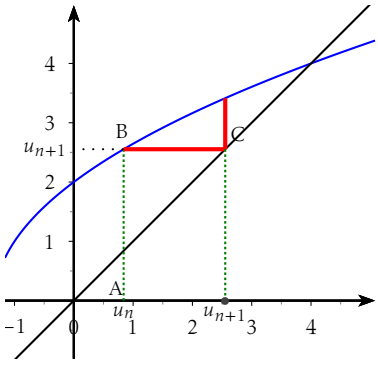
Aide pour l'exercice E10 Vous devriez trouver une suite arithmétique

Aide pour l'exercice E11

1. Pour faire le tracé, on peut à chaque fois que l'on passe de u_n à u_{n+1} :

   - Tracer en trait discontinu le segment vertical $[AB]$ avec $A(u_n, 0)$ et $B(u_n, f(u_n))$.

   - Placer dans une liste les points $B(u_n, f(u_n))$ et $C(f(u_n), f(u_n))$. Ainsi quand à la fin du programme on tracera la liste des points, on obtiendra "l'escalier".



Aide pour l'exercice E12

2. Le plus simple pour ne pas recalculer la somme pour chaque valeur de H_n est de remarquer que si $n > 1$, $H_n = H_{n-1} + \dfrac{1}{n}$.

4a. Si $t \in [k, k+1]$, on peut encadrer la fonction $t \longmapsto \dfrac{1}{t}$ sur cet intervalle avant de calculer son intégrale.

4c. On pourra étudier le signe de l'expression $\dfrac{1}{x+1} - \ln x + \ln(x+1)$.

6. Solutions des exercices

Retour sur l'exercice E1 Pour chaque hauteur obtenue au dessus de $1{,}5$ mètre, on voit 2 fois la balle, sauf la première fois lorsqu'elle est lâchée de 10 mètres, où on ne la voit qu'à la descente. C'est pour ne pas compter deux fois la première descente que l'on initialise la variable `vue` à -1.

```
h = 10
vue = -1
while h > 1.5:
    vue = vue + 2
    h = h * 2 / 3
print("On voit", vue, "fois la balle.")
```

Retour sur l'exercice E2

1. Pour l'étape 4, il faut 26 cartes.
2. Pour créer une nouvelle étape, il faut ajouter :
   - 2 cartes "en pointe" par niveau.
   - 1 carte horizontale par niveau.
   - 2 cartes "en pointe" pour le dernier niveau.

   Soit $2 \times e + e + 2 = 3e + 2$ cartes.
3. D'où la fonction suivante :

```
def nbCartes(etape):
    cartes = 2
    for e in range(1, etape):
        cartes = cartes + 3 * e + 2
    return cartes
```

4.

```
def nbNiveaux(cartesMax):
    e = 0
    cartes = 0
    while cartes <= cartesMax:
        cartes = cartes + 3 * e + 2
        e = e + 1
    return e - 1
```

Retour sur l'exercice E3 En notant `epargne` la somme disponible après placement et `prix`, la valeur du véhicule au cours des mois, on obtient la fonction suivante : (on rappelle que diminuer un montant de 2% revient à le multiplier par $0,98$)

```
def attente(s) :
    """
    En entrée : s est la somme épargnée chaque mois
    En sortie : Le nombre de mois à épargner
    """
    prix = 13900
    epargne = 2000
    mois = 0
    while epargne < prix:
        mois = mois + 1
        epargne = epargne + s
        prix = prix * 0.98
        print("A la fin du mois", mois, "il a", epargne, "et la voiture ↙
            ↳ coûte", prix)
    return mois
```

On a ajouté un `print` dans la fonction pour suivre les calculs, il n'est bien entendu pas obligatoire.

Retour sur l'exercice E4

1. `calcul(10)` renvoie 30.

2. La ligne `if i**2 % 2 == 0` signifie « Si i^2 est pair » qui est équivalente à « Si i est pair ». En effet un entier et son carré ont toujours la même parité (essayez de le démontrer !). Donc le changement proposé ne modifie pas le résultat.

3. D'après la question précédente, la fonction `calcul(n)` renvoie la somme de tous les nombres pairs de l'intervalle $[0;n]$. Si on note p la partie entière de $n/2$ alors :

   - Si n est pair : $n = 2p$ et `calcul(n)` $= 0+2+4+\ldots+2p$
   - Si n est impair : $n = 2p+1$ et encore `calcul(n)` $= 0+2+4+\ldots+2p$

   $$\begin{aligned} \text{Or } \texttt{calcul(n)} &= 0+2+4+\ldots+2p \\ &= 2(1+2+3+\ldots+p) \\ &= 2\frac{p(p+1)}{2} \\ &= p(p+1) \end{aligned}$$

   Ainsi, on peut écrire plus simplement :

```
def calcul(n) :
    p = n // 2
    return p * (p+1)
```

Retour sur l'exercice E5

1. Ajouter t% revient à multiplier par $1+\frac{t}{100}$:

```
def epargne(s, n, t):
    for i in range(n):
        s = s * (1 + t/100)
    return s
```

   Pour répondre à la question :

```
>>> epargne(20000, 20, 2)
29718.947919567087
```

2. Attention aux unités, il y a 12 mois dans 1 an :

```
def loyers(n):
    l = 790 * 12 * n
    return l
```

3. a. Si t_m est le taux mensuel et T le taux annuel alors on sait que $\left(1+\frac{t_m}{100}\right)^{12} = 1+\frac{T}{100}$, d'où $t_m = 100\times\left(\left(1+\frac{T}{100}\right)^{\frac{1}{12}}-1\right)$.

```
def tauxmensuel(T):
        return( (1+T/100) ** (1/12) - 1 ) * 100
```

b. Il suffit de traduire la formule :

```
def mensualite(C, t_m, N):
    return (C * t_m / 100) / (1 - (1 + t_m/100) ** (-N) )
```

c. En utilisant la fonction, il y a $N = 20\times 12$ mensualités avec un taux mensuel à calculer :

```
>>> mensualite(150000, tauxmensuel(3.6), 20*12)
873.1750672497817
```

Soit des mensualités d'environ 873,18€.

d. Sur 20 ans, Myriam aura remboursé $873,18\times 20\times 12 = 209563,2$. Sur 150 000 euros empruntés, le montant des intérêts payés est d'environ 59 563,2 euros.

Retour sur l'exercice E6

1. Si on veut tenir compte des valeurs 0 et 1 de n (ce qui ne présente pas grand intérêt dans notre cas) :

```
def suite(a, b, n):
    if n == 0:
        return []
    if n == 1:
        return [a]
    reponse = [a, b]
    for i in range(n-2) :
        l = len(reponse)
        reponse.append(reponse[l-1] + reponse[l-2])
    return reponse
```

2. On réalise un calcul de somme :

```
def somme(L, a) :
    s = 0        # On initialise la somme
    for element in L :
        s = s + element
        if element == a :
            return s
```

À noter que si a $\notin$ L, la fonction termine la boucle for et ne renvoie rien (None).

3. On vérifie l'exactitude de la prédiction en tapant dans la console :

```
>>> somme(suite(5, 12, 20), 830)
2161
```

Pour revenir à l'astuce de notre magicien, on remarque que la suite de nombres est construite d'une certaine façon : Si $u_1 = a$ et $u_2 = b$ sont les deux premières valeurs, on a ensuite pour tout entier naturel n non nul : $u_{n+2} = u_n + u_{n+1}$ On a donc une somme de termes d'une suite de type Fibonacci. En ajoutant terme à terme ces égalités :

$$\begin{array}{rcrcr}
u_1 & + & u_2 & = & \cancel{u_3} \\
u_2 & + & \cancel{u_3} & = & \cancel{u_4} \\
u_3 & + & \cancel{u_4} & = & u_5 \\
\ldots & & & & \\
u_{n-2} & + & u_{n-1} & = & \cancel{u_n} \\
u_{n-1} & + & \cancel{u_n} & = & \cancel{u_{n+1}} \\
u_n & + & \cancel{u_{n+1}} & = & u_{n+2} \\
\hline
u_1 + u_2 + u_3 + \ldots + u_n & + & u_2 & = & u_{n+2}
\end{array}$$

D'où : $u_1 + u_2 + u_3 + \ldots + u_n = u_{n+2} - u_2$

Le magicien a donc effectué $2173 - 12 = 2161$.

Retour sur l'exercice E7

1. Le triangle OA_0A_1 est rectangle isocèle en A_1 et $OA_0 = 4$. Si on note $\ell = OA_1 = A_OA_1$, on a d'après le théorème de Pythagore : $\ell^2 + \ell^2 = 4^2$, d'où $2\ell^2 = 16$ soit $\ell = \sqrt{8} = 2\sqrt{2}$.

2. De la même manière, pour $n \in \mathbb{N}$,

OA_nA_{n+1} est isocèle rectangle donc $\begin{cases} OA_n^2 = OA_{n+1}^2 + A_{n+1}A_n^2 \\ OA_{n+1}^2 = A_{n+1}A_n^2 \end{cases}$

Donc $OA_{n+1}^2 = A_{n+1}A_n^2 = \dfrac{1}{2}OA_n^2$ et on en déduit que $OA_{n+1} = \dfrac{1}{\sqrt{2}}OA_n$.

3. En programmant cette relation de récurrence :

```
from math import sqrt

def OA(n):
    u = 4
    for i in range(n):
        u = u / sqrt(2)
    return u
```

4. La suite (u_n) est une suite géométrique de raison $\frac{1}{\sqrt{2}}$ et de premier terme $u_0 = 4$. On en déduit que pour tout entier n, $u_n = 4 \times \left(\frac{1}{\sqrt{2}}\right)^n$.

5. On a $L_n = \sum_{k=1}^{n} A_{k-1}A_k = \sum_{k=1}^{n} OA_k = \sum_{k=1}^{n} u_k$:

```
def L(n):
    total = 0
    for k in range(1, n+1):
        total = total + OA(k)
    return total
```

6. On a vu que $L_n = \sum_{k=1}^{n} u_k$, donc :

$$\begin{aligned} L_n &= \sum_{k=1}^{n} 4 \times \left(\frac{1}{\sqrt{2}}\right)^k = 4\sum_{k=1}^{n}\left(\frac{1}{\sqrt{2}}\right)^k = 4\sum_{k=0}^{n-1}\left(\frac{1}{\sqrt{2}}\right)^{k+1} \\ &= \frac{4}{\sqrt{2}}\sum_{k=0}^{n-1}\left(\frac{1}{\sqrt{2}}\right)^k = \frac{4}{\sqrt{2}}\frac{1-\left(\frac{1}{\sqrt{2}}\right)^n}{1-\frac{1}{\sqrt{2}}} \\ &= \frac{4}{\sqrt{2}-1}\left(1-\left(\frac{1}{\sqrt{2}}\right)^n\right) = \frac{4}{\sqrt{2}-1}\left(1-\frac{1}{2^{n/2}}\right). \end{aligned}$$

7. Comme $\left|\frac{1}{\sqrt{2}}\right| < 1$, on a $\lim_{n\to+\infty}\left(\frac{1}{\sqrt{2}}\right)^n = 0$ et $\lim_{n\to+\infty} L_n = \frac{4}{\sqrt{2}-1}$

Retour sur l'exercice E8

1. Cette fonction est très simple, elle ne nécessite ni boucle, ni test :

```
def u(n) :
    return n ** 3 / (n + 3)
```

2. Il suffit d'ajouter une boucle POUR :

```
for n in range(1000):
    print("u", n, "=", u(n), sep="")
```

La suite (u_n) semble diverger vers $+\infty$.

3. Au lieu d'une boucle POUR, nous allons faire une boucle TANT QUE cette fois-ci :

```
n = 0
while u(n) <= 10 ** 10:
    n = n + 1
print(n)
```

Retour sur l'exercice E9

1. Le programme calcule le $n^{\text{ième}}$ terme de la suite définie par $u_0 = 3$ et pour tout entier n, $u_{n+1} = 2u_n - 1$.

2. Il suffit de modifier la deuxième ligne en initialisant `u` à 1 et de changer la formule de la $4^{\text{ième}}$ ligne en

```
u = (u + 4) / 3:
```

```
def suite(n):
    u, i = 1, 0
    while i < n:
        u = (u + 4) / 3
        i = i + 1
    return u
```

3. La limite semble être 2.

4. a. $\forall n \in \mathbb{N},\ w_{n+1} = v_{n+1} - 2 = \dfrac{v_n + 4}{3} - 2 = \dfrac{v_n - 2}{3} = \dfrac{w_n}{3}$.
   La suite (w_n) est géométrique de premier terme $w_0 = v_0 - 2 = -1$ et de raison $\dfrac{1}{3}$.

   b. Puisque (w_n) est une suite géométrique, pour tout $n \in \mathbb{N}$, $w_n = -1 \times \left(\dfrac{1}{3}\right)^n = \dfrac{-1}{3^n}$. On en déduit $v_n = w_n + 2 = \dfrac{-1}{3^n} + 2$.

   c. $\lim\limits_{n\to+\infty} 3^n = +\infty$, et $\lim\limits_{n\to+\infty} \dfrac{-1}{3^n} = 0$ donc $\lim\limits_{n\to+\infty} v_n = 2$.

Retour sur l'exercice E10

1. Cette fonction s'écrit très facilement de manière récursive, à condition d'exprimer u_n en fonction des termes précédents :

$$\forall n \in \mathbb{N}, n \geqslant 2,\ u_n = \sqrt{u_{n-2} \times u_{n-1} + 9(u_{n-1} + 1)}$$

```
from math import sqrt

def u(n):
    if n == 0:
        return 2
    elif n == 1:
        return 5
    else :
        return sqrt( u(n-2)*u(n-1) + 9*(u(n-1)+1) )
```

Cette première idée, pourtant très lisible, n'est cependant pas efficace car elle recalcule un grand nombre de fois la même chose (le calcul de u_{20} commence à être long !). Voici une seconde solution itérative qui consiste à conserver en mémoire les 2 dernières valeurs calculées :

	a	b	
n	u_n	u_{n+1}	u_{n+2}
0	2	5	$8 = \sqrt{2\times 5+9(5+1)}$
1	5	8	$11 = \sqrt{5\times 8+9(8+1)}$
2	8	11	$14 = \sqrt{8\times 11+9(11+1)}$
…	…	…	…

```
def u(n):
    a, b = 2, 5
    for i in range(n) :
        a, b = b, sqrt( a*b + 9*(b+1) )
    return a
```

2. Quelle que soit la solution envisagée précédemment, avec une boucle, on obtient les premières valeurs :

```
for n in range(20):
    print(u(n), end=' / ')
```

```
2 / 5 / 8 / 11 / 14 / 17 / 20 / 23 / 26 / 29 / 32 / 35 / 38 / ↙
    ↘ 41 / 44 / 47 / 50 / 53 / 56 / 59...
```

La suite semble arithmétique de raison 3.

3. Il ne semble pas simple de démontrer que la différence $u_{n+1} - u_n$ est constante. Nous allons plutôt montrer à l'aide d'une **récurrence double** que

$$\forall n \in \mathbb{N},\ u_n = 2 + 3n$$

- Pour $n = 0$: $2 + 3 \times 0 = 2 = u_0$, l'égalité est vérifiée au rang 0.
- Pour $n = 1$: $2 + 3 \times 1 = 5 = u_1$, l'égalité est vérifiée au rang 1.
- Supposons que pour un $n \in \mathbb{N}$ donné, on ait : $\begin{cases} u_n = 2 + 3n \\ u_{n+1} = 2 + 3(n+1) \end{cases}$, alors :

$$\begin{aligned} u_{n+2} &= \sqrt{u_n \times u_{n+1} + 9(u_{n+1} + 1)} \\ &= \sqrt{(2+3n) \times \big(2 + 3(n+1)\big) + 9\big(2 + 3(n+1) + 1\big)} \\ &= \sqrt{9n^2 + 48n + 64} \\ &= \sqrt{(3n+8)^2} \\ &= |3n+8| \\ &= 3n + 8 \\ &= 2 + 3(n+2) \text{ L'égalité est vraie au rang } n+2. \end{aligned}$$

Par récurrence, $\forall n \in \mathbb{N},\ u_n = 2 + 3n$, la suite (u_n) est donc bien une suite arithmétique de premier terme 2 et de raison 3.

Retour sur l'exercice E11

1. a. On va créer une liste de 1000 valeurs de x variant dans l'intervalle $[0;5]$ et calculer les valeurs de $f(x)$ correspondantes. Pour la droite d'équation $y = x$, c'est encore plus simple, puisqu'il suffit de relier les points de coordonnées $(0,0)$ et $(5,5)$:

```
import matplotlib.pyplot as plt
from math import sqrt

def f(x):
    return sqrt(3*x + 4)

X, Y = [], []
x = 0
while x < 5:
    X.append(x)
    Y.append(f(x))
    x = x + 0.1

plt.plot(X, Y, 'b-')
plt.plot([0, 5], [0, 5], 'k-')
plt.show()
```

b. On va ajouter les tracés indiqués dans l'aide. Les traits verticaux en pointillés sont tracés au fur et à mesure, les points de "l'escalier" sont stockés dans une liste pour être tracés et reliés en fin de boucle. On passe d'une valeur de u_n à u_{n+1} en écrasant à chaque tour de boucle `u` avec l'affectation `u = f(u)` :

```
import matplotlib.pyplot as plt
from math import sqrt

def f(x):
    return sqrt(3*x + 4)

# Tracé de la courbe de f et droite d'équation : y = x
X, Y = [], []
x = 0
while x < 5:
    X.append(x)
    Y.append(f(x))
    x = x + 0.1

plt.plot(X, Y, 'b-')
plt.plot([0, 5], [0, 5], 'k-')

n = int(input('Rang ?'))
u = 1
X, Y = [], []
for i in range(n+1):
    plt.plot([u, u], [0, f(u)], 'g:')
    X.append(u)
    Y.append(f(u))
    X.append(f(u))
    Y.append(f(u))
    print('U', i, "=", u, sep="")
    u = f(u)
plt.plot(X, Y, 'r-')

plt.show()
```

c. On conjecture que la suite (u_n) est croissante et qu'elle converge vers 4.

2. a. Pour $n \in \mathbb{N}$, on note $\mathcal{P}_n$ la proposition « $0 \leqslant u_n \leqslant u_{n+1} \leqslant 4$ ».

- $u_1 = \sqrt{3+4} = \sqrt{7} \approx 2,6$. Ce qui justifie que la proposition $\mathcal{P}_0$ est vraie puisque l'on a bien $0 \leqslant u_0 \leqslant u_1 \leqslant 4$.

- Si, pour un $n \in \mathbb{N}$ donné, $\mathcal{P}_n$ est vraie, alors :

$$\begin{array}{ccccccc} 0 & \leqslant & u_n & \leqslant & u_{n+1} & \leqslant 4 & \\ & & & & & & \text{On multiplie par } 3>0 \\ 0 & \leqslant & 3u_n & \leqslant & 3u_{n+1} & \leqslant 12 & \\ & & & & & & \text{On ajoute } 4 \\ 4 & \leqslant & 3u_n+4 & \leqslant & 3u_{n+1}+4 & \leqslant 16 & \\ & & & & & & \text{On applique la fonction} \\ & & & & & & \sqrt{\cdot} \text{ croissante sur } \mathbb{R}_+ \\ 2 & \leqslant & \sqrt{3u_n+4} & \leqslant & \sqrt{3u_{n+1}+4} & \leqslant 4 & \\ & & & & & & \text{En particulier,} \\ 0 & \leqslant & u_{n+1} & \leqslant & u_{n+2} & \leqslant 4 & \end{array}$$

  La proposition $\mathcal{P}_{n+1}$ est donc vraie à son tour.

- Par le principe de récurrence, la proposition $\mathcal{P}_n$ est vraie pour tout $n \in \mathbb{N}$.

b. La question précédente nous montre que la suite (u_n) est croissante et majorée (et même bornée). Elle converge donc vers un réel que l'on notera ℓ. Or la suite (u_{n+1}) converge elle aussi vers ℓ par unicité de la limite. ℓ doit vérifier l'équation :

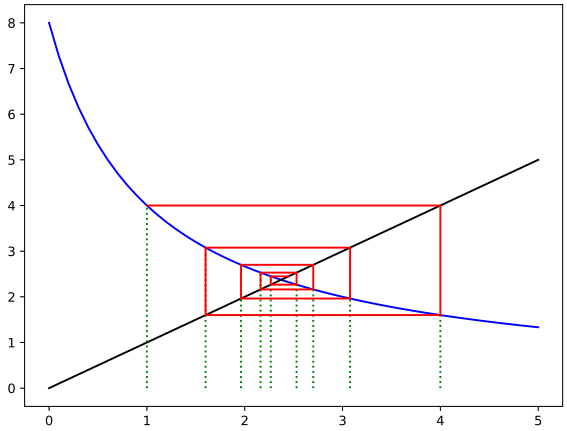
$$\begin{array}{ll} \ell = \sqrt{3\ell+4} & \text{, il faut donc que} \\ \ell^2 = 3\ell+4 & \text{ou encore} \\ \ell = -1 \text{ ou } \ell = 4 & \end{array}$$

Seule la valeur 4 peut convenir puisque $\ell \geqslant 0$. Ainsi la suite (u_n) converge vers 4.

3. a. Pour la suite (v_n), il suffit de modifier la définition de la fonction f :

```
def f(x) :
    return 8 / (x+1)
```

On obtient cette représentation :



Cette fois-ci la suite n'est plus monotone, mais semble cependant converger vers une valeur proche de $2,37...$ La preuve sera possible après le bac.

b. Si elle converge, notons ℓ sa limite. On a, par le même raisonnement que précédemment :

$$\ell = \frac{8}{\ell+1} \iff \ell^2 + \ell = 8 \iff \ell = \frac{-1 \pm \sqrt{33}}{2}$$

Les valeurs de v_n étant clairement positives au vu de sa définition, seule $\ell = \dfrac{-1+\sqrt{33}}{2}$ convient et est donc la limite cherchée.

Retour sur l'exercice E12

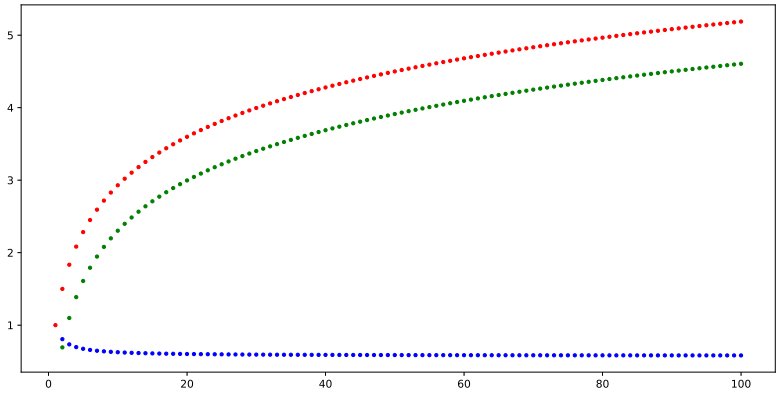
1. C'est la proposition n°2.

2. On peut utiliser la fonction de la question précédente, mais c'est un peu dommage, car il faut recalculer la somme à chaque fois. En utilisant la variable `H` pour accumuler les quantités $\frac{1}{k}$, on obtient :

```
import matplotlib.pyplot as plt

N = 100
H = 1
plt.plot(1, 1, 'r.')
for k in range(2, N+1):
    H = H + 1/k
    plt.plot(k, H, 'r.')

plt.show()
```

3. On complète le script précédent avec les 2 autres tracés :

```
import matplotlib.pyplot as plt
from math import log

N = 100
H = 1
plt.plot(1, 1, 'r.')
for k in range(2, N+1):
    H = H + 1/k
    plt.plot(k, H, 'r.')
    plt.plot(k, log(k), 'g.')
    plt.plot(k, H-log(k), 'b.')

plt.show()
```



4. L'écart entre les 2 courbes semble devenir constant. La suite (u_n) semble être décroissante et converger.

5. a. On a ainsi les inégalités suivantes :

Soit $k \geqslant 1$. Si

$$k \leqslant t \leqslant k+1$$

La fonction inverse étant décroissante sur $]0;+\infty[$

$$\frac{1}{k+1} \leqslant \frac{1}{t} \leqslant \frac{1}{k}$$

En intégrant ces fonctions sur l'intervalle $[k;k+1]$

$$\int_k^{k+1} \frac{1}{k+1}\mathrm{d}t \leqslant \int_k^{k+1} \frac{1}{t}\mathrm{d}t \leqslant \int_k^{k+1} \frac{1}{k}\mathrm{d}t$$

Après calcul ($\frac{1}{k}$ étant constant) :

$$\frac{1}{k} \geqslant \int_k^{k+1} \frac{1}{t}\mathrm{d}t$$

b. En utilisant l'inégalité démontrée, on a pour $n > 1$:

$$\begin{aligned} H_n &= 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n} \\ &\geqslant \int_1^2 \frac{1}{t}\,dt + \int_2^3 \frac{1}{t}\,dt + \ldots + \int_n^{n+1} \frac{1}{t}\,dt \end{aligned}$$

en appliquant la relation de Chasles,

$$= \int_1^{n+1} \frac{1}{t}\,dt = \Big[\ln t\Big]_1^{n+1} = \ln(n+1) - \ln 1 = \ln(n+1) \geqslant \ln n$$

D'où l'inégalité voulue : $u_n = H_n - \ln n \geqslant 0$.

La suite (u_n) est donc minorée.

c. Soit $n > 1$.

$$\begin{aligned} u_{n+1} - u_n &= (H_{n+1} - \ln(n+1)) - (H_n - \ln n) \\ &= H_{n+1} - H_n - \ln(n+1) + \ln n \\ &= \cancel{1} + \frac{\cancel{1}}{\cancel{2}} + \ldots + \frac{\cancel{1}}{\cancel{n}} + \frac{1}{n+1} - \cancel{1} - \frac{\cancel{1}}{\cancel{2}} - \ldots - \frac{\cancel{1}}{\cancel{n}} - \ln(n+1) + \ln n \\ &= \frac{1}{n+1} - \ln(n+1) + \ln n \end{aligned}$$

Posons $f(x) = \dfrac{1}{x+1} - \ln(x+1) + \ln x$, f est définie sur $[1;+\infty[$ et étudions ses variations. Sur $[1;+\infty[$,

$$f'(x) = -\frac{1}{(x+1)^2} - \frac{1}{x+1} + \frac{1}{x} = \frac{1}{x(x+1)^2} > 0.$$

x	1 $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ $+\infty$
$f'(x)$	$+$
f	$\frac{1}{2} - \ln 2$ $\nearrow$ 0

Pour la limite, on peut remarquer que lorsque $x \to +\infty$,

$$f(x) = \frac{1}{x+1} - \ln\left(\frac{x+1}{x}\right) = \frac{1}{x+1} - \ln\left(1 + \frac{1}{x}\right)$$

et $\lim\limits_{x\to+\infty} \underbrace{\dfrac{1}{x+1}}_{\to 0} \underbrace{- \ln\left(1 + \dfrac{1}{x}\right)}_{\to 0} = 0.$

Ainsi pour tout $n > 1$, $u_{n+1} - u_n < 0$ ce qui prouve que la suite est décroissante.

d. Finalement la suite (u_n) est décroissante et minorée, elle est donc convergente.

Retour sur l'exercice E13

1. a. $I_0 = \int_0^1 (1-x)^0 e^x \mathrm{d}x = \int_0^1 e^x \mathrm{d}x = \left[e^x\right]_0^1 = e - 1.$

b. À l'aide d'une intégration par parties : si $n \in \mathbb{N}$, on pose :

$$u(x) = (1-x)^{n+1} \quad v(x) = e^x$$
alors
$$u'(x) = -(n+1)(1-x)^n \quad v'(x) = e^x$$

Les fonctions u et v étant dérivables et de dérivées continues sur l'intervalle $[0;1]$, on a

$$\begin{aligned} I_{n+1} &= \int_0^1 (1-x)^{n+1} e^x \mathrm{d}x && \text{par IPP,} \\ &= \left[(1-x)^{n+1} e^x\right]_0^1 + (n+1)\int_0^1 (1-x)^n e^x \mathrm{d}x \\ &= -1 + (n+1)I_n. \end{aligned}$$

c. En appliquant le relation de récurrence, on peut proposer une fonction récursive en remarquant que si $n > 0$, $I_n = -1 + nI_{n-1}$:

```
from math import exp

def I(n):
    if n == 0:
        return exp(1) - 1
    else:
        return -1 + n * I(n-1)
```

ou une version itérative :

```
def I(n):
    reponse = exp(1) - 1
    for k in range(1, n+1):
        reponse = -1 + k * reponse
    return reponse
```

d. En réalisant une boucle...

```
>>> for n in range(100):
...     print(n, I(n))
...
0 1.718281828459045
1 0.7182818284590451
```

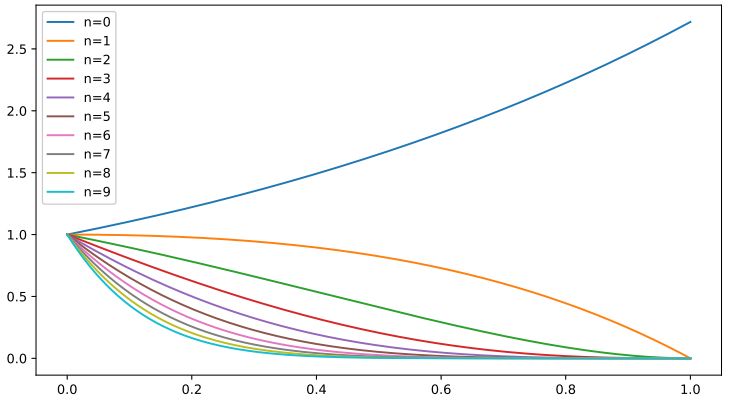
```
 2 0.4365636569180902
 ...
95 -1.4933501293152828e+132
96 -1.4336161241426716e+134
97 -1.3906076404183915e+136
98 -1.3627954876100237e+138
99 -1.3491675327339235e+140
```

... on peut conjecturer que la suite diverge vers $-\infty$.

2. a. On peut créer une fonction pour cela :

```
from math import exp
from numpy import linspace
import matplotlib.pyplot as plt

def listeY(n) :
    X = linspace(0, 1, 1000)
    return [(1-x)**n * exp(x) for x in X]

X = linspace(0, 1, 1000)
plt.plot(X, listeY(0), label='n=0')
plt.plot(X, listeY(1), label='n=1')
plt.plot(X, listeY(2), label='n=2')
plt.plot(X, listeY(3), label='n=3')
plt.plot(X, listeY(4), label='n=4')
plt.plot(X, listeY(5), label='n=5')
plt.plot(X, listeY(6), label='n=6')
plt.plot(X, listeY(7), label='n=7')
plt.plot(X, listeY(8), label='n=8')
plt.plot(X, listeY(9), label='n=9')
plt.legend()
plt.plot()
plt.show()
```

Bien sûr, ce programme pourrait être amélioré par une boucle `for` ! Dans tous les cas, on obtient ce graphique (plus joli en couleur) :



b. Le graphique précédent illustre que les fonctions intégrées sont toutes positives (cela se prouve facilement)... Il est donc impossible que l'aire sous la courbe diverge vers $-\infty$...

3. a. Par encadrements successifs, on trouve que :

$$\begin{array}{lccccc} & 0 & \leqslant & x & \leqslant & 1 \\ \Longrightarrow & 1 & \leqslant & e^x & \leqslant & e \\ & & & \text{car } x \mapsto e^x \text{ est croissante sur } [0;1] & & \\ \Longrightarrow & (1-x)^n & \leqslant & (1-x)^n e^x & \leqslant & e(1-x)^n \\ & & & & & \text{car } (1-x)^n \geqslant 0 \\ \Longrightarrow & \displaystyle\int_0^1 (1-x)^n \mathrm{d}x & \leqslant & \displaystyle\int_0^1 (1-x)^n e^x \mathrm{d}x & \leqslant & \displaystyle\int_0^1 e(1-x)^n \mathrm{d}x \\ & & & & & \text{par croissance de l'intégrale} \\ \Longrightarrow & \dfrac{1}{n+1} & \leqslant & \mathrm{I}_n & \leqslant & \dfrac{e}{n+1} \end{array}$$

b. Comme $\lim\limits_{n\to+\infty} \dfrac{1}{n+1} = 0$, on déduit par le théorème d'encadrement que $\lim\limits_{n\to+\infty} \mathrm{I}_n = 0$.

4. a. $\mathrm{E}_0 = \mathrm{J}_0 - \mathrm{I}_0 = \varepsilon$.

b. $\forall n \in \mathbb{N}$,

$$\begin{array}{rcl} \mathrm{E}_{n+1} & = & \mathrm{J}_{n+1} - \mathrm{I}_{n+1} = ((n+1)\mathrm{J}_n - 1) - ((n+1)\mathrm{I}_n - 1) \\ & = & (n+1)(\mathrm{J}_n - \mathrm{I}_n) = (n+1)\mathrm{E}_n. \end{array}$$

c. On montre par récurrence immédiate que $\mathrm{E}_n = n!\varepsilon$.

d. Si $\varepsilon > 0, \mathrm{E}_n \longrightarrow +\infty$ ainsi que J_n et si $\varepsilon < 0, \mathrm{E}_n \longrightarrow -\infty$ ainsi que J_n

e. Ainsi (sans tenir compte des autres erreurs d'approximation dans les calculs des I_n successifs), si la valeur de I_0 n'est pas exactement initialisée à $e-1$, la suite diverge. Or il est assez peu probable que ce nombre irrationnel soit stocké de manière exacte en mémoire...

Retour sur l'exercice E14

1. a. $u_1 = \dfrac{1}{2}$; $v_1 = \dfrac{2}{3}$; $u_2 = \dfrac{7}{12}$; $v_2 = \dfrac{11}{18}$.

b. Au cas où vous seriez tenté par la simplicité, cette première idée ne fonctionne pas :

```
def suites(n):
    u = 0
    v = 1
    for i in range(n):
        u = (u + v) / 2
        v = (u + 2*v) / 3
    return (u, v)
```

En effet, lors de l'affectation `u = (u + v) / 2` la variable `u` est écrasée et modifiée posant des problèmes pour le calcul de `v` qui est effectué ensuite.
Une seconde idée, qui elle fonctionne, est d'utiliser une variable temporaire, ici `t`, pour mémoriser le résultat de la future valeur de `u` tout en conservant la valeur d'origine pour finir le calcul de `v`. Une fois le calcul terminé, on peut alors mettre à jour la valeur de `u` :

```
def suites(n):
    u, v = 0, 1
    for i in range(n):
        t = (u + v) / 2
        v = (u + 2*v) / 3
        u = t
    return (u, v)
```

Enfin, on peut utiliser à profit l'affectation simultanée réalisable en Python :

```
def suites(n):
    u, v = 0, 1
    for i in range(n):
        u, v = (u + v) / 2, (u + 2*v) / 3
    return (u, v)
```

c. On peut conjecturer que la suite (u_n) est croissante et (v_n) est décroissante. Elles semblent toutes deux converger vers $0,6$.

2. Avec les matrices

a. On vérifie bien par un produit matriciel que $X_{n+1} = AX_n$

b. Pour tout entier n, on pose $\mathcal{P}_n$ la proposition : « $X_n = A^n X_0$ ».

- Initialisation : $A^0 X_0 = I_2 X_0 = X_0$ donc la proposition $\mathcal{P}_0$ est vraie.
- Soit n un entier naturel fixé. On suppose que la proposition $\mathcal{P}_n$ est vraie, c'est à dire $X_n = A^n X_0$. Il en résulte :

$$X_{n+1} = AX_n \overset{\text{H.R}}{=} A \times A^n X_0 = A^{n+1} X_0.$$

Donc $\mathcal{P}_{n+1}$ est vraie à son tour.

- On peut donc conclure que la propriété $X_n = A^n X_0$ est vraie pour tout entier naturel n.

c. On vérifie par calcul que $A = PBP'$

AU DÉTOUR DE LA BALADE...

On remarquera que les matrices P *et* P′ *sont inverses l'une de l'autre et sont les matrices de passage résultant de la diagonalisation de la matrice* A.

d. Soit $\mathcal{Q}_n$ « $A^n = PB^nP'$ ».

- Pour $n = 0$, $PB^0P' = P \times I_2 \times P' = P \times P' = I_2 = A^0$ où $I_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ est la matrice identité. Ainsi $\mathcal{Q}_0$ est vraie.
- On suppose la proposition $\mathcal{Q}_n$ vraie pour un entier naturel n donné. Alors

$$\begin{aligned} A^{n+1} &= A \times A^n = (PBP')(PB^nP') \\ &= PB(P'P)B^nP' = PBI_2B^nP' \\ &= PBB^nP' = PB^{n+1}P'. \end{aligned}$$

  Donc la proposition est vraie au rang $n+1$.
- On en conclut que pour tout entier naturel, $A^n = PB^nP'$.

e. $A^n = PB^nP'$ et par produit des matrices, on obtient :

$$A^n = \begin{pmatrix} 2/5 + 3/5(1/6)^n & 3/5 - 3/5(1/6)^n \\ 2/5 - 2/5(1/6)^n & 3/5 + 2/5(1/6)^n \end{pmatrix}$$

f. Pour tout entier naturel n, $X_n = A^nX_0$ avec $X_0 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$

On en déduit que pour tout entier naturel n,

$$X_n = \begin{pmatrix} 3/5 - 3/5(1/6)^n \\ 3/5 + 2/5(1/6)^n \end{pmatrix}$$

g. On en déduit que pour tout entier naturel n,

$$u_n = \frac{3}{5} - \frac{3}{5 \times 6^n} \text{ et } v_n = \frac{3}{5} + \frac{2}{5 \times 6^n}.$$

h. On en conclut que $\lim\limits_{+\infty} u_n = \frac{3}{5} = \lim\limits_{+\infty} v_n$

3. À l'aide de suites auxiliaires

a. Il n'y a que le `return` à modifier dans la fonction précédente :

```
def suites(n):
    u, v = 0, 1
    for i in range(n):
        u, v = (u + v) / 2, (u + 2*v) / 3
    w = v - u
    return (u, v, w)
```

b. La suite (w_n) semble géométrique. pour tout n un entier naturel,

$$w_{n+1} = v_{n+1} - u_{n+1} = \frac{u_n + 2v_n}{3} - \frac{u_n + v_n}{2}$$

$$= \frac{2u_n - 3u_n + 4v_n - 3v_n}{6} = \frac{1}{6}\left(v_n - u_n\right) = \frac{1}{6}w_n.$$

On en déduit que la suite (w_n) est géométrique de raison $\frac{1}{6}$ et de premier terme $w_0 = v_0 - u_0 = 1$.

c. Pour tout entier n, $w_n = \left(\frac{1}{6}\right)^n = \frac{1}{6^n}$

d. Pour tout entier n,

$$t_{n+1} = 2u_{n+1} + 3v_{n+1} = 2 \times \frac{u_n + v_n}{2} + 3 \times \frac{u_n + 2v_n}{3}$$

$$= u_n + v_n + u_n + 2v_n = 2u_n + 3v_n = t_n$$

e. Pour tout entier n, $t_n = 2u_n + 3v_n = t_0 = 3$ car la suite est constante. Les termes des suites (u_n) et (v_n) vérifient donc le système :

$$\begin{cases} v_n - u_n = \dfrac{1}{6^n} \\ 2u_n + 3v_n = 3 \end{cases}$$

En le résolvant, on obtient bien $u_n = \frac{3}{5} - \frac{3}{5 \times 6^n}$ et $v_n = \frac{3}{5} + \frac{2}{5 \times 6^n}$.

f. Et comme précédemment, on conclut que $\lim\limits_{+\infty} u_n = \lim\limits_{+\infty} v_n = \frac{3}{5}$

Retour sur l'exercice E15

1. a. Pour passer d'une liste à b briques à la suivante, on ajoute $b - 1$ valeurs dans les intervalles (il y a un intervalle de moins que le nombre de briques).

Ainsi la suite (u_n) est définie par $\begin{cases} u_0 = 2 \\ u_{n+1} = 2u_n - 1 \end{cases}$

b. Une simple boucle pour i allant de 0 à 10 permet de répondre à la question :

```
u = 2
for i in range(11):
    print("A l'étape", i, "il y a", u, "nombres")
    u = 2*u - 1
```

On trouve bien 1025 nombres, Adil a donc raison.

AU DÉTOUR DE LA BALADE...

*La suite (u_n) est une suite **arithmético-géométrique**. Bien que son étude ne soit pas au programme du lycée, le calcul de $u_n - 1$ peut ici aider à réaliser une conjecture que l'on peut ensuite démontrer par récurrence.*

2. a. Pour la fonction `suivant`, on traduit l'algorithme :

```
def suivant(T):
    L = [T[0]]       # On place déjà le 1er nombre
    n = len(T)
    for i in range(1, n) :
        L.append(T[i-1]+T[i])
        L.append(T[i])
    return L
```

b. Pour savoir si Emma a raison, on va générer le $10^{\text{ème}}$ étage :

```
T = [1, 1]
for i in range(10) :
    print((T))
    T = suivant(T)
print("Longueur", len(T)) # Pour vérifier la 1ère question !
print("Valeur maximale", max(T))
```

On trouve un maximum de 144, Emma a donc elle aussi dit vrai.

3. a. La fonction `present` a déjà été rencontrée dans la fiche sur les listes, vous pouvez aussi utiliser l'instruction `in`.

b. On peut alors réaliser une boucle sur toutes les valeurs de 1 à 144 et tester leur présence. On aura ainsi le nombre de briques différentes :

```
...
nb = 0
for i in range(1, 145) :
    if i in T :
        nb = nb + 1
print(nb, "nombres différents")
```

Aïcha a donné la bonne valeur, en effet on trouve bien 118 valeurs différentes.

Retour sur l'exercice E16

1. À l'étape 1, il reste la moitié au centre et l'autre moitié est partagée entre les 4 voisins, d'où les $\frac{1}{8}$.

2. À l'étape 2, on trouve :

   - Pour la case A (et C, G et I) : $2 \times \frac{1}{3} \times \frac{1}{16} = \frac{1}{24}$
   - Pour la case B (et D, F et H) : $\frac{1}{4} \times \frac{1}{2} \times \frac{1}{2} + \frac{1}{2} \times \frac{1}{8} = \frac{1}{8}$
   - Pour la case E : $\frac{1}{2} \times \frac{1}{2} + 4 \times \frac{1}{3} \times \frac{1}{2} \times \frac{1}{8} = \frac{1}{3}$

Étape 2 :

$\frac{1}{24}$	$\frac{1}{8}$	$\frac{1}{24}$
$\frac{1}{8}$	$\frac{1}{3}$	$\frac{1}{8}$
$\frac{1}{24}$	$\frac{1}{8}$	$\frac{1}{24}$

3. a. Avec les notations précédentes, on trouve que pour tout $n \in \mathbb{N}$:

$$\begin{cases} a_{n+1} = a_n/2 + b_n/6 + d_n/6 \\ b_{n+1} = b_n/2 + a_n/4 + c_n/4 + e_n/8 \\ c_{n+1} = c_n/2 + b_n/6 + f_n/6 \\ d_{n+1} = d_n/2 + a_n/4 + g_n/4 + e_n/8 \\ e_{n+1} = e_n/2 + d_n/6 + b_n/6 + f_n/6 + h_n/6 \\ f_{n+1} = f_n/2 + c_n/4 + i_n/4 + e_n/8 \\ g_{n+1} = g_n/2 + d_n/6 + h_n/6 \\ h_{n+1} = h_n/2 + g_n/4 + i_n/4 + e_n/8 \\ i_{n+1} = i_n/2 + h_n/6 + f_n/6 \end{cases}$$

   b. Il suffit de traduire le système précédent :

```
def evolution(L) :
    a, b, c, d, e, f, g, h, i = L
    return [a/2 + b/6 + d/6,
            b/2 + a/4 + c/4 + e/8,
            c/2 + b/6 + f/6,
            d/2 + a/4 + g/4 +e/8,
            e/2 + d/6 + b/6  + f/6 + h/6,
            f/2 + c/4 + i/4 + e/8,
            g/2 + d/6 + h/6,
            h/2 + g/4 + i/4 + e/8,
            i/2 + h/6 + f/6]
```

c. Une simple boucle permet de réaliser ces conjectures :

```
L = [0, 0, 0, 0, 1, 0, 0, 0, 0]
for n in range(10) :
    print("Etape", n, L)
    L = evolution(L)
```

On constate que les cases A, C, G et I ont toujours les mêmes valeurs. De la même manière, les cases B, D, F et H. Ce qui n'est pas surprenant avec les symétries des équations et de l'initialisation. Bien sûr, si on change les valeurs initiales, ce n'est plus vrai.

4. a. S'il y a symétrie des valeurs, alors pour $n \in \mathbb{N}$,

$$\begin{cases} a_{n+1} = a_n/2 + b_n/6 + b_n/6 \\ b_{n+1} = b_n/2 + a_n/4 + a_n/4 + e_n/8 \\ e_{n+1} = e_n/2 + b_n/6 + b_n/6 + b_n/6 + b_n/6 \end{cases}$$

Soit

$$\begin{cases} a_{n+1} = a_n/2 + b_n/3 \\ b_{n+1} = a_n/2 + b_n/2 + e_n/8 \\ e_{n+1} = 2b_n/3 + e_n/2 \end{cases}$$

Si ces suites convergent alors en notant a, b et e les limites respectives, on obtient :

$$\begin{cases} a = a/2 + b/3 \\ b = a/2 + b/2 + e/8 \\ e = 2b/3 + e/2 \end{cases} \iff \begin{cases} a = \dfrac{e}{2} \\ \\ b = \dfrac{3e}{4} \end{cases}$$

Il nous manque donc une équation... Fort heureusement, il nous reste une information : $4a + 4b + e = 1$ (la proportion de graines est toujours de 1 au total sur les 9 cases).

Finalement, on trouve : $a = \dfrac{1}{12}$; $b = \dfrac{1}{8}$ et $e = \dfrac{1}{6}$.

AU DÉTOUR DE LA BALADE...

Cet exercice peut être enrichi en proposant des variantes. Il n'y a pas de raison à ce que les graines ne se répartissent pas au delà des 9 parcelles : on peut imaginer des parcelles plus grandes ou ajouter une direction de vent...

Enfin, on peut illustrer l'évolution de la répartition des graines avec ce code :

```
import matplotlib.pyplot as plt

L = [0, 0, 0, 0, 1, 0, 0, 0, 0]
for n in range(50) :
    print("Etape", n, L)
    M = [L[0:3], L[3:6], L[6:9]]
    plt.imshow(M, vmin = 0, vmax = 1, cmap='Reds')
    plt.pause(0.5)
    L = evolution(L)
plt.show()
```

Chapitre F

STATISTIQUES ET PROBABILITÉS

1. Ah, ce fameux baccalauréat !

F1 *Louna souhaite faire une estimation de sa moyenne au bac, elle décide de réaliser un simulateur de bac.*

Matières	*Coefficients*
Enseignement scientifique	*6*
Histoire-Géographie	*6*
LV A	*6*
LV B	*6*
Ens. spécialité 1	*8*
EPS	*6*
EMC	*2*
Français écrit	*5*
Français oral	*5*
Philosophie	*8*
Grand oral	*10*
Ens. spécialité 2	*16*
Ens. spécialité 3	*16*

Créer une fonction prenant en paramètre une liste de notes (dans l'ordre du tableau) et qui renvoie la moyenne des notes obtenues au bac.

2. Quelques études statistiques

F2 *On dispose d'une liste* X *de valeurs classées par ordre croissant et d'une liste* N *de même taille correspondant aux effectifs.*

1. *Réaliser 4 fonctions permettant, pour cette série, de calculer et renvoyer :*
   a. *l'effectif total ;*
   b. *la moyenne ;*
   c. *la liste des effectifs cumulés croissants ;*
   d. *la médiane.*
2. *Tester les fonctions avec les deux séries suivantes. La première représente la répartition des médailles d'or aux Jeux Olympiques de l'été 1900, la seule année où la France était arrivée première avec un total de 25 médailles d'or ! La deuxième représente cette fois-ci cette répartition aux JO de Tokyo en 2020 (qui se sont d'ailleurs déroulés en 2021). Comparer les résultats affichés par le programme avec ceux qui ont été calculés "à la main".*

En 1900

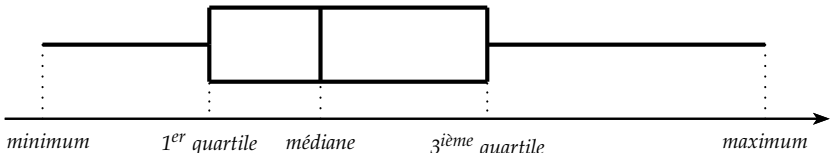
Nb médailles d'or	0	1	2	4	5	6	15	19	25
Nb de pays	7	5	2	1	1	2	1	1	1

Aux JO de Tokyo

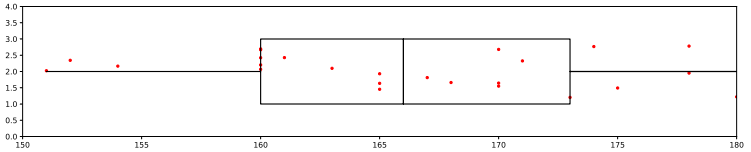
Nb médailles d'or	0	1	2	3	4	6	7	10	17	20	22	27	38	39
Nb de pays	28	22	11	11	5	2	4	4	1	1	1	1	1	1

F3 *On donne une série de valeurs, ordonnées ou non, placées dans une liste.*

1. *Écrire une fonction qui renvoie les 5 caractères de dispersion de la série : le minimum et le maximum, le premier et le troisième quartile et la médiane.*
2. *Adapter ce programme pour qu'il dessine le diagramme en boîte (ou diagramme de Tukey) de la série avec la convention suivante :*



3. *Afin de visualiser la répartition des valeurs dans cette boîte, améliorer le programme pour que soient aussi représentées dans un nuage de points les valeurs de la liste* L.



4. *On a réalisé une enquête auprès de 60 élèves de 4ème et de 3ème afin de connaître leurs tailles (en centimètres). Voici, ci-dessous, le résultat obtenu. Pour récupérer ces données qui ont déjà été saisies dans un programme Python, taper* TAILLES.

   Filles : 155, 155, 168, 169, 165, 159, 167, 165, 165, 167, 165, 160, 160, 163, 163, 165, 166, 156, 168, 167, 166, 160, 170, 156, 159, 169, 159, 161, 164, 160, 172, 156, 156, 160
   Garçons : 160, 154 ,170, 160, 160, 171, 160, 160, 165, 168, 170, 151, 165, 178, 161, 163, 175, 178, 152, 174, 186, 180, 170, 165, 173, 167.

   Utiliser le programme réalisé précédemment pour comparer ces deux séries statistiques.

F4 *On souhaite réaliser une étude statistique sur les entrées de cinéma. Pour cela on a récupéré sur le site* `https://www.data.gouv.fr` *le nombre d'entrées au cinéma tous les mois du 1er Janvier 1980 au 21 décembre 2022. Ces données ont été rassemblées et placées dans un fichier que l'on peut charger à l'aide de ces deux lignes :*

```
import numpy as np
donnees = np.load('cine.npy')
```

On obtient alors une matrice `donnees` *de 43 lignes et 13 colonnes. La colonne 0 contient le numéro de l'année, la colonne 1 les entrées en janvier de 1980 à 2022, jusqu'à la colonne 12 qui contient les entrées de décembre. Ce fichier est présent sur le site du livre en tapant le code* CINE.

1. *Combien y-a-t-il eu d'entrées en Mars 2000 ?*

2. *Écrire un script qui trace la courbe représentant le nombre d'entrées en fonction de l'année. Que constate-t-on ?*

3. *On souhaite affiner le graphique en affichant sous forme d'un diagramme en bâtons le nombre d'entrées au fil des mois de 1980 à 2022. Qu'observe-t-on ?*

4. *Pour étudier le nombre d'entrées en fonction des mois, on souhaite représenter par un diagramme en bâtons le nombre d'entrées moyen en fonction des différents mois. Que remarque-t-on ?*

F5 *Le site* `https://basketlfb.com` *de la ligue féminine de basketball, propose une rubrique avec un grand nombre de statistiques par joueuses (une ligne par match et par joueuse).*

A partir des informations de la saison 2022-2023, on a constitué un fichier tableur téléchargeable avec le code `BASKET`*. Voici ce que contiennent les colonnes :*

0	*Numéro de la joueuse*
1	*Temps de jeu en minutes*
2	*Nombre de points sur la rencontre*
3	*Nombre de rebonds*
4	*Nombre de passes décisives*
5	*Nombre de paniers à 2 points réussis*
6	*Nombre de paniers à 2 points tentés*
7	*Nombre de paniers à 3 points réussis*
8	*Nombre de paniers à 3 points tentés*
9	*Tirs réussis en %*
10	*Nombre de lancers francs réussis*
11	*Nombre de lancers francs tentés*

Le numéro des joueuses a arbitrairement été fixé ainsi :

n°	*nom*	*date de naiss.*	*Taille*	*Poste*
1	*Marine Fauthoux*	*23 janvier 2001*	*1m74*	*1*
2	*Isabelle Yacoubou*	*21 avril 1986*	*1m90*	*5*
3	*Shakayla Thomas*	*05 mai 1996*	*1m82*	*4*
4	*Carla Leite*	*16 avril 2004*	*1m75*	*1*
5	*Tiffany Clarke*	*05 février 1991*	*1m84*	*4*
6	*Alexis Peterson*	*20 juin 1995*	*1m65*	*1*
7	*Oderah Chidom*	*09 juillet 1995*	*1m93*	*5*
8	*Tima Pouye*	*07 avril 1999*	*1m75*	*2*

On peut charger le tableau au format matrice à l'aide des 2 lignes suivantes :

```
import numpy as np
stats = np.load('basket.npy')
```

On pourra alors accéder à la donnée de la ligne `i` *et colonne* `j` *avec l'appel* `L[i,j]` *comme cela est expliqué dans la fiche sur les matrices.*

À l'aide de Python, on souhaite réaliser quelques analyses statistiques.

1. *Quelle joueuse a disputé le moins de matchs ?*

   a. *Réaliser une fonction* `nb_matchs(j, tab)` *qui reçoit le numéro de la joueuse et le tableau de données en paramètres et renvoie le nombre de matchs disputés par la joueuse n°* `j`.

   b. *Répondre à la question.*

2. *La série « Passes décisives ».*

   a. *En s'inspirant de la question précédente, déterminer la valeur modale de la série "Passes décisives", c'est à dire la valeur présente le plus souvent pour les 8 joueuses concernées.*

   b. *Améliorer le script pour obtenir également la moyenne des passes décisives.*

3. *Quelle est la joueuse qui est la plus performante ?*

   a. *Écrire une fonction* `total(j, col, tab)` *qui reçoit le numéro d'une joueuse, la colonne à sommer et le tableau de statistiques et renvoie le total de la colonne* `col` *en se limitant aux lignes de la joueuse* `j`.

   b. *En déduire un script qui donne un tableau par joueuse avec :*

      Q1. *Le nombre de paniers à 2 points marqués.*
      Q2. *Le nombre de paniers à 2 points réussis par heure jouée.*
      Q3. *Le pourcentage de paniers à 2 points réussis.*
      Q4. *Le nombre de paniers à 3 points marqués.*
      Q5. *Le nombre de paniers à 3 points réussis par heure jouée.*
      Q6. *Le pourcentage de paniers à 3 points réussis.*

3. Simulation et modélisation

F6

D'après une idée de Paul Stienne

1. *Parmi les propositions suivantes, lesquelles permettent de simuler le lancer d'un dé cubique équilibré à 6 faces numérotées de 1 à 6 (un dé classique en fait !) ?*

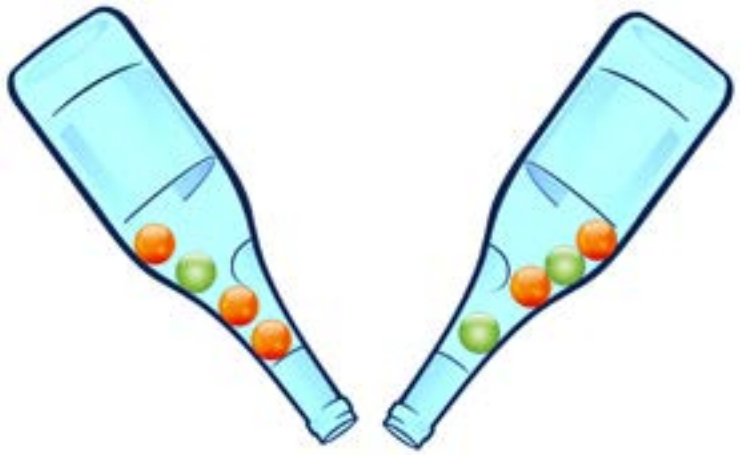
   a. `de = 6 * floor(random())`
   b. `de = floor (6 * random())`
   c. `de = floor(6 * random()) + 1`
   d. `de = floor(6 * random() + 1)`
   e. `de = round(5 * random()) + 1`
   f. `de = floor(10 * random()) % 6 + 1`
   g. `de = floor(12 * random()) % 6 + 1`
   h. `de = floor(6 * random() **2) + 1`
   i. `de = 2 * floor(3 * random()) + 1`
   j. `de = floor(3*random()) + floor(4*random()) +1`

2. *Imaginer un programme pour vérifier vos conjectures.*

F7 *On dispose de deux bouteilles :*

- *Une bouteille A contenant 3 boules oranges et 1 boule verte.*
- *Une bouteille B contenant 2 boules oranges et 2 boules vertes.*

On retourne les bouteilles et on sort de chacune les deux premières boules par le goulot. On se pose la question : « dans quelle bouteille la probabilité d'obtenir deux boules de la même couleur est-elle la plus grande ? »



1. *Avec une simulation.*

   a. *Écrire une fonction* `tirage(bout)` *qui reçoit un caractère* `"A"` *ou* `"B"`*, effectue un tirage dans la bouteille* `bout` *et renvoie un couple représentant les 2 premières boules obtenues. Par exemple :*

```
>>> tirage('A')
('Orange', 'Orange')
```

   b. *Créer une fonction* `frequences(bout, N)` *qui simule* `N` *tirages dans la bouteille* `bout` *et renvoie la liste de l'évolution de la fréquence de réalisation de l'événement « les 2 boules sont de la même couleur » au cours des* `N` *tirages.*

   c. *Représenter, sur le même graphique, la simulation avec les 2 bouteilles.*

   d. *Conclure*

2. *Retrouver ces résultats par le calcul.*

F8 *On décide de choisir au hasard un chiffre parmi les 2000 premières décimales du nombre* π*. Est-ce une situation d'équiprobabilité ? Sinon, quel chiffre a la plus grande probabilité d'être choisi ? On pourra utiliser le programme en Python* `pi.py` *en tapant le code* PI *sur le site du livre pour télécharger un programme comportant les 2000 premières décimales du nombre* π *sous forme d'une chaîne de caractères.*

F9 *Une urne contient des boules numérotées de 1 à 9. Pour tout entier naturel* n *tel que* $1 \leqslant n \leqslant 9$*, il y a exactement* n *boules dans l'urne comportant le nombre* n*. On en tire successivement deux en remettant la boule tirée dans l'urne. On note* X *l'écart absolu entre les 2 boules.*

1. *Montrer qu'il y a 45 boules dans l'urne.*

2. *Écrire un programme Python qui modélise le contenu de cette urne à l'aide d'une liste.*

3. *Créer un programme Python qui simule 10 000 parties et estime la loi de* X *et l'espérance* $\mathrm{E}(\mathrm{X})$*.*

4. *Modifier le programme pour obtenir les valeurs exactes de la loi de probabilité.*

5. *Écrire un programme qui calcule la valeur exacte de* $\mathrm{E}(\mathrm{X})$*.*

F10 *Le jour de la fête de la science, des étudiants organisent un jeu : le joueur lance 3 dés cubiques (rouge, vert, bleu). S'il arrive à construire un triangle dont les côtés mesurent les valeurs des faces obtenues, il gagne, sinon il perd.*

1. *Créer une fonction recevant les valeurs des trois dés et renvoie la chaîne* `"gagné"` *ou* `"perdu"`.

2. *Créer un programme qui affiche les tirages possibles de manière exhaustive en précisant si ce sont des jeux gagnants ou non. En déduire la probabilité p de gagner à ce jeu.*

3. *On réalise à présent un échantillon de* N *parties.*

   a. *Ecrire une fonction* `echantillon` *qui reçoit en entrée un entier* N *et renvoie la fréquence de parties gagnées sur ces* N *parties.*

   b. *Afficher 10 fois* `echantillon(20)`, `echantillon(100)` *et* `echantillon(1000)`. *Que constate-t-on ?*

4. *On répète 100 fois cette simulation d'un échantillon de taille* N *et on représente les 100 fréquences obtenues à l'aide du programme suivant :*

```
import matplotlib.pyplot as plt
from random import *
from math import *

def echantillon(N):
    votre fonction !

N = int(input("Nombre de parties dans une simulation : "))
for i in range(100):
    f = echantillon(N)
    if 111/216 - 1/sqrt(N) <= f <= 111/216 + 1/sqrt(N):
        plt.plot(i, f, 'g.')
    else:
        plt.plot(i, f, 'ro')
plt.axis(ymin=0, ymax=1)
plt.show()
```

   a. *Saisir et exécuter le programme pour* N = 20.

   b. *À quoi correspondent les points verts et les points rouges ?*

   c. *Exécuter plusieurs fois le programme pour différentes valeurs de* N. *Que constate-on ? Pouvait-on le prévoir ?*

F11 *À l'occasion de la fête du 14 Juillet, une commune organise un jeu consistant à tirer des boules dans une urne contenant 3 boules bleues, 3 boules blanches et 2 boules rouges. La partie coûte 5 €. Chaque boule tirée rapporte une certaine somme d'argent. Un élève a simulé le déroulement d'une partie dans un programme qui affiche le gain algébrique du joueur lorsque le jeu est terminé.*

```
from random import *
U = ['Bleu'] * 3 + ['Blanc'] * 3 + ['Rouge'] * 2
T = []
x = -5
b = 'Rien'
while b not in T:
    T.append(b)
    b = choice(U)
    if b == 'Bleu':
        x = x + 2
    elif b == 'Rouge':
        x = x + 5
print(x)
```

1. *À la lecture du programme :*
   a. *Combien rapporte chaque boule suivant sa couleur ?*
   b. *Le tirage est-il avec ou sans remise ?*
   c. *Quand le jeu s'arrête-t-il ?*
2. *On note* X *la variable aléatoire représentant le gain algébrique d'une partie.*
   a. *Quelles sont les valeurs pouvant être prises par* X *?*
   b. *Modifier le programme précédent pour qu'il affiche une estimation de l'espérance mathématique de* X *sur un échantillon de 1000 parties.*
3. *Donner la loi de probabilité de la variable aléatoire* X *(on pourra utiliser un arbre pondéré).*
4. *Calculer alors l'espérance de* X *et la comparer avec la valeur trouvée lors de la simulation.*

4. Des classiques intemporels

F12 *On cherche à répondre à la question : « dans une classe, à partir de combien d'élèves a-t-on plus d'une chance sur deux qu'au moins 2 élèves fêtent leur anniversaire le même jour ? »*

1. *Expérimentation avec Python.*

   a. *Écrire une fonction* `jours(n)` *qui reçoit un entier* `n` *et renvoie une liste modélisant les jours où chacun des* `n` *élèves est né.*

   b. *Écrire une fonction* `jumeauxdejour(L)` *qui reçoit une liste de jours et renvoie un booléen indiquant s'il y a dans la liste (au moins) 2 fois le même jour.*

   c. *On note* A_n *l'événement « Dans la classe de* n *élèves, il y a des jumeaux de jour ». À l'aide d'une simulation sur 1000 classes estimer* $P(A_n)$ *puis tracer la courbe donnant* $P(A_n)$ *en fonction de* n *pour* $n \in [\![1;100]\!]$.

2. *Démonstration.*

   a. *Déterminer* $P(A_n)$ *en fonction de* n.

   b. *En déduire le nombre d'élèves pour qu'il y ait plus d'une chance sur deux qu'au moins 2 élèves fêtent leur anniversaire le même jour.*

F13 *Galilée a rédigé vers 1620 un mémoire sur les jeux de dés pour répondre à une demande du Duc de Toscane. Il est ainsi l'un des premiers avec Cardan à avoir écrit sur le "calcul des hasards". À la cour de Florence, de nombreux jeux de société étaient alors pratiqués. Parmi ceux-ci, l'un faisait intervenir la somme des numéros sortis lors du lancer de trois dés. Le Duc de Toscane avait constaté que la somme 10 était obtenue légèrement plus souvent que la somme 9, ce qui lui paraissait inexplicable puisqu'il y a autant de façons d'écrire 10 que 9 comme somme de trois entiers compris entre 1 et 6.*

$9 = 1 + 2 + 6$	$10 = 1 + 3 + 6$
$9 = 1 + 3 + 5$	$10 = 1 + 4 + 5$
$9 = 1 + 4 + 4$	$10 = 2 + 2 + 6$
$9 = 2 + 2 + 5$	$10 = 2 + 3 + 5$
$9 = 2 + 3 + 4$	$10 = 2 + 4 + 4$
$9 = 3 + 3 + 3$	$10 = 3 + 3 + 4$

1. *Réaliser un programme qui simule 50 lancers de 3 dés et affiche s'il y a eu plus de 9, plus de 10 ou autant de 9 que de 10.*

2. *Faire fonctionner plusieurs fois le programme et déterminer combien de fois la découverte du Duc de Toscane s'avère vraie.*

3. *On cherche à calculer la probabilité des événements A : "la somme obtenue est 9" et B : "la somme obtenue est 10". Pour que ce soit plus simple, on suppose que les dés sont de couleurs différentes : un rouge, un bleu et un vert.*

   a. *Lister à l'aide d'un programme les triplets (face rouge, face bleue, face verte) possibles lors d'un lancer. Combien y en a-t-il ?*

   b. *Modifier le programme pour qu'il compte le nombre d'issues donnant une somme de 9 et le nombre d'issues donnant une somme de 10.*

   c. *En déduire les probabilités cherchées. Quelle est la faille dans le raisonnement du Duc de Toscane ?*

F14 *On cherche à écrire un programme qui demande un entier naturel n et affiche le développement de $(a+b)^n$ en fonction de a et b.*

1. *Vérifier que* $(a+b)^3 = a^3 + 3a^2b + 3ab^2 + b^3$.

   Il existe une formule mathématique pour généraliser ces "identités remarquables" appelée formule du ***binôme de Newton****, si a et b sont 2 nombres réels et n un entier naturel :*

$$(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^{n-k}b^k$$

$$= \binom{n}{0}a^n + \binom{n}{1}a^{n-1}b^1 + \binom{n}{2}a^{n-2}b^2 + \ldots + \binom{n}{n}b^n$$

   Par exemple pour $n = 3$, on obtient bien :

$$(a+b)^3 = \binom{3}{0}a^3b^0 + \binom{3}{1}a^2b^1 + \binom{3}{2}a^1b^2 + \binom{3}{3}a^0b^3$$

$$= 1a^3b^0 + 3a^2b^1 + 3a^1b^2 + 1a^0b^3$$

$$= a^3 + 3a^2b + 3ab^2 + b^3$$

2. *Le **triangle de Pascal** permet de calculer les coefficients binomiaux :*

$n=0$	1						
$n=1$	1	1					
$n=2$	1	2	1				
$n=3$	1	3	3	1			
$n=4$	1	4	6	4	1		
$n=5$	1	5	10	10	5	1	
$n=6$	1	6	15	20	15	6	1

Si n et p sont deux entiers tels que $0 \leqslant p < n$,

$$\binom{n+1}{p+1} = \binom{n}{p+1} + \binom{n}{p}$$

En utilisant une matrice, écrire une fonction qui reçoit un entier naturel n et renvoie la $n^{ème}$ ligne du triangle de Pascal sous forme d'une liste.

3. *En déduire une fonction qui reçoit un entier n strictement positif et qui affiche le développement de $(a+b)^n$.*

F15

1. *En reprenant l'exercice présent dans la fiche concernant le module math, écrire une fonction* `CoefBinom(n,k)` *:*

   - *<u>En entrée</u> : n et k sont deux entiers vérifiant $0 \leqslant k \leqslant n$.*
   - *<u>En sortie</u> : l'entier $\binom{n}{k}$ défini par $\binom{n}{k} = \dfrac{n!}{k!(n-k)!}$.*

2. *Utiliser cette fonction pour afficher la loi de probabilité d'une variable aléatoire X suivant une loi binomiale de paramètres (n,p).*

3. *Même question mais en cumulant les valeurs de manière à afficher cette fois-ci la fonction de répartition $P(X \leqslant k)$ pour $0 \leqslant k \leqslant n$.*

4. *Utiliser le programme pour résoudre le dilemme suivant : "Un vendeur achète un lot de 200 ampoules bon marché à un grossiste qui lui assure que 96% des ampoules fonctionnent. Après avoir vendu l'ensemble des ampoules, le vendeur s'aperçoit qu'il a eu 15 retours de clients pour des ampoules défectueuses. Que penser de l'affirmation du grossiste ?"*

F16 *On cherche à simuler le choix d'une main (c'est à dire une série de 5 cartes) dans un jeu de poker simplifié en piochant successivement et sans remise 5 cartes dans un jeu de 52 cartes. On veut alors déterminer la probabilité d'avoir un full, c'est-à-dire une main comportant 3 cartes de même valeur et deux autres cartes de même valeur (cette dernière étant distincte de la première valeur).*

1. *Créer un programme qui génère une liste* `jeu` *de valeurs représentant les 52 cartes du jeu de cartes. Pour notre problème, inutile de coder l'information sur la couleur (trèfle, carreau, cœur ou pique) de la carte. On créera juste une liste contenant 4 cartes de chaque valeur 'As', '2', '3', ..., '10', 'Valet', 'Dame', 'Roi'.*

2. *Compléter le programme pour créer une liste* `main` *qui contient une main de 5 cartes distinctes.*

3. *Écrire une fonction* `full(L)` *qui reçoit une liste de 5 valeurs et renvoie un booléen indiquant si* `L` *représente un full. À cet effet, on pourra utiliser la fonction* **`sorted`** *de Python qui trie une liste :*

```
>>> main
['Valet', '3', 'Valet', '3', '9']
>>> sorted(main)
['3', '3', '9', 'Valet', 'Valet']
```

4. *Adapter alors l'ensemble pour simuler 10000 tirages d'une main de poker et donner la fréquence de l'événement "Obtenir un full".*

5. *Comparer ce résultat avec la probabilité de l'événement.*

5. Si vous n'êtes pas fatigué, les derniers défis

F17 *Avec un jeu de 52 cartes, on propose le jeu suivant : on tire des cartes une à une sans les remettre jusqu'à obtenir un valet. On note* X *la variable aléatoire représentant le nombre de cartes tirées avant l'obtention de la paire.*

1. *Justifier que* X *ne peut prendre qu'un nombre fini de valeurs.*

2. *Réaliser un programme qui simule 10 000 parties et affiche une estimation de la loi de probabilité de* X.

3. *À l'aide de diagrammes en barres, comparer les fréquences obtenues aux probabilités après les avoir déterminées.*

F18 ***Inspiré d'un article de E. Janvresse et T. de la Rue***

Le 2 août 2005, un avion d'Air France sort de piste lors de son atterrissage à Toronto et s'enflamme. Le 6 août, un avion de Tuninter tombe en mer à proximité de Palerme. Le 14 août, un avion d'Hélios Airways percute une montagne près d'Athènes. Le 16 août, un avion de West-Carribean s'écrase au Vénézuéla. Et le 23 août, un avion de Tans s'écrase en Amazonie. Cet été-là, cette série noire a suscité

l'inquiétude générale : était-elle liée à l'augmentation du trafic aérien ? Trahissait-elle une soudaine hausse du risque d'accidents s'expliquant par le relâchement des contrôles et de la maintenance ? Etait-ce le vieillissement de la flotte aérienne ? Pour répondre à cette interrogation légitime, il peut être intéressant de se poser la question :

"Sur une année, quelle est la probabilité pour que se produise une succession d'au moins 5 accidents d'avion en 22 jours" ?

Voici les données dont nous disposions en 2005 :

- *La probabilité qu'un avion ait un accident est de* $\frac{1}{500000}$.
- *Il y a 20 000 vols par an.*
- *Les accidents se produisent de manière indépendante.*

N'ayant pas les outils mathématiques pour résoudre ce problème, on se propose de travailler par simulation.

1. *Écrire une fonction* `somme` *qui reçoit en paramètres une liste* L *et un entier* `deb` *et renvoie la somme des 22 termes consécutifs :*

$$L[deb] + L[deb+1] + ... + L[deb+21]$$

2. *Écrire un programme qui génère une liste* `L` *de 365 valeurs représentant la simulation du nombre d'avions qui ont eu un accident par jour sur une année.*

3. *Ecrire une fonction* `anneeNoire` *:*
   En entrée : Une liste de 365 valeurs représentant le nombre d'accidents par jour d'une année
   En sortie : 1 si c'est une année noire, 0 sinon

4. *Réaliser un programme qui simule sur 500 ans et affiche la fréquence d'années noires.*

5. *Conclure.*

6. Aide pour les exercices

Aide pour l'exercice F2 Seul le calcul de la médiane risque de poser problème. Sachant que l'on a déjà les effectifs cumulés croissants, le problème est ainsi plus simple. Trois cas peuvent se produire (on note N l'effectif total) :

- **cas 1 :** Si N est impair, notons $m = \frac{N+1}{2}$. Il faut prendre la $m^{\text{ième}}$ valeur de la série, on va donc parcourir la liste des effectifs cumulés croissants jusqu'à trouver un effectif supérieur ou égal à m.
- **cas 2 :** Si N est pair, notons $m = \frac{N}{2}$. La médiane est obtenue en réalisant la moyenne de la $m^{\text{ième}}$ et de la $(m+1)^{\text{ième}}$ valeur de la série. Comme pour le cas précédent, on peut parcourir la liste des effectifs cumulés croissants jusqu'à trouver un effectif supérieur ou égal à m. Là, deux cas se présentent :
  - **cas 2a :** Soit l'effectif cumulé croissant est strictement supérieur à m, ce qui signifie que la $m^{\text{ième}}$ et la $(m+1)^{\text{ième}}$ valeurs sont identiques, et dans ce cas la médiane est la $m^{\text{ième}}$ valeur.
  - **cas 2b :** Soit l'effectif cumulé croissant est égal à m et dans ce cas on fait la moyenne de la valeur correspondante et la valeur suivante.

Ci-dessous les 3 cas présentés sur des exemples :

cas 1 :

x_i	1	3	⑤	8	TOTAL
n_i	2	3	5	1	11
ECC	2	5	**10**	11	-

$m = \frac{11+1}{2} = 6$, $med = 5$

cas 2 :

x_i	1	3	⑤	8	TOTAL
n_i	2	3	5	4	14
ECC	2	5	**10**	14	-

$m = \frac{14}{2} = 7$, $med = 5$

cas 3 :

x_i	1	③	⑤	8	TOTAL
n_i	4	3	5	2	14
ECC	2	**7**	**12**	14	-

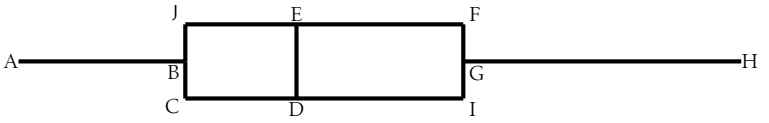
$m = \frac{14}{2} = 7$, $med = \frac{3+5}{2} = 4$

Que N soit pair ou impair, on peut remarquer que l'on a toujours

$$m = \left\lfloor \frac{N+1}{2} \right\rfloor.$$

Aide pour l'exercice F3 1. Le premier quartile est la valeur de la série pour laquelle les fréquences cumulées croissantes sont supérieures ou égales à 25%. Il faut donc prendre la (`N // 2`) ième valeur si N est un multiple de 4 ou la (`N // 2 + 1`) ième sinon. Attention, la numérotation des listes commence à 0, la $n^{\text{ième}}$ valeur de la liste L n'est donc pas L[n]...

2. Pour tracer la boîte, on peut relier successivement les points dans cet ordre (ou un autre !) : ABCDEFGHGIDEJB



3. Pour placer les points on pourra, pour aérer le graphique, prendre une ordonnée aléatoire puisque celle-ci ne représente rien.

Aide pour l'exercice F6 Il faut s'assurer que :

- D'une part, on ne tire que des valeurs entre 1 et 6.
- D'autre part, ces valeurs sont équi-réparties

Aide pour l'exercice F8 Pour compter le nombre d'apparitions de chaque chiffre, deux possibilités sont envisageables :

- Créer 10 variables ($nb0, nb1, ..., nb9$)
- Pour i allant de 0 à 9, parcourir le nombre et compter les occurrences du chiffre i.
- Créer une liste de compteurs

Attention aux types des objets quand vous les comparez (comparez deux nombres ou deux chaînes).

Aide pour l'exercice F10

1. Pensez qu'afin de pouvoir tracer un triangle, il faut que la longueur du plus grand côté soit strictement plus petite que la somme des longueurs des deux autres. C'est ce que l'on appelle l'inégalité triangulaire.

2. En réalisant 3 boucles imbriquées, vous serez en mesure de lister toutes les possibilités.

3. Pour la fonction `échantillon`, réalisez d'abord un programme dans lequel N est fixé, ce programme sera ensuite transformé en fonction.

Aide pour l'exercice F11

1. Une fois n'est pas coutume, prendre un exemple et faire fonctionner manuellement l'algorithme peut donner des idées (le choix est aléatoire) :

T	b	x	b ∈ T ?
[]	'Rien'	-5	NON
['Rien']	'Bleu'	$-5+2=-3$	NON
['Rien','Bleu']	'Blanc'	-3	NON
['Rien','Bleu','Blanc']	'Bleu'	$-3+2=-1$	OUI

x vaut -1 à la fin du programme. Le programme s'est arrêté parce que 'Bleu' était déjà dans la liste.

2. Il suffit d'ajouter une boucle et de calculer la moyenne des gains obtenus.

Aide pour l'exercice F12

1a. On pourra supposer qu'une année comporte 365 jours. Une date d'anniversaire correspond donc au choix d'un jour dans l'année.

1b. Une double boucle sera sûrement utile pour parcourir la liste des dates.

Aide pour l'exercice F15 Pour la dernière question, on peut supposer que l'affirmation du grossiste est vraie et chercher la probabilité qu'il y ait 15 retours ou plus.

Aide pour l'exercice F16

2. Pensez à retirer la carte du jeu une fois tirée...

3. Pour tester si on a un full ou non, il peut être intéressant d'ordonner la liste pour ensuite tester si elle est de la forme AAABB ou AABBB.

5. Pour trouver la probabilité, vous pouvez commencer par calculer la probabilité d'un événement tel que "Obtenir un full de 6 par les 7", c'est-à-dire avoir trois 6 et deux 7.

Aide pour l'exercice F17

- Pour la simulation, simulez une liste de 52 cartes comme vous le souhaitez et utilisez le `remove` pour retirer une a une les cartes tirées.
- Pour le calcul des probabilités, modéliser la situation à l'aide d'un arbre si besoin.

Aide pour l'exercice F18

1. Cette question ne devrait pas poser de problème. Voici quelques calculs qui peuvent vous permettre de vérifier votre fonction :
   - `somme([1]*50, 3)` renvoie 22.
   - `somme([0, 1, 2]*50, 3)` renvoie 21.
   - `somme([0, 1, 2]*50, 2)` renvoie 22.
   - `somme([0, 1, 2]*50, 1)` renvoie 23.
2. Le plus simple est de suivre les données dont on dispose : par jour décollent 20 000 avions, chacun ayant une "chance" sur 500 000 d'avoir un accident.
3. Pour savoir s'il s'agit d'une année noire, on va parcourir la liste du nombre d'accidents jour par jour et regarder si dans les 4 jours qui suivent on a un total de plus de 5 accidents.

7. Solutions des exercices

Retour sur l'exercice F1 On peut, pour cette fonction, utiliser 2 accumulateurs : le premier pour calculer la somme des notes coefficientées, le second pour mémoriser la somme des coefficients.

```
def moyenne(L):
    coefs = [6, 6, 6, 6, 8, 6, 2, 5, 5, 8, 10, 16, 16]
    total = 0
    somme_coefs = 0
    for i in range(len(L)) :
        total = total + L[i] * coefs[i]
        somme_coefs = somme_coefs + coefs[i]
    return total / somme_coefs
```

Retour sur l'exercice F2

1a.

```
def effectif(X, N):
    Total = 0
    for n in N:
        Total = Total + n
    return Total
```

1b.

```
def moyenne(X, N):
    Total, Pays = 0, 0
    for i in range(len(X)) :
        Total = Total + X[i] * N[i]
        Pays = Pays + N[i]
    return Total/Pays
```

On trouve une moyenne d'environ 4,23 médailles par pays.

1c.

```
def ECC(X, N):
    Total = 0
    liste = []
    for n in N:
        Total = Total + n
        liste.append(Total)
    return liste
```

1d.

```
def mediane(X, N):
    E = ECC(X, N)
    Total = effectif(X, N)
    i = 0
    milieu = (Total+1) // 2
    while E[i] < milieu:
        i = i + 1
    if Total % 2 == 1:
        med = X[i]
    elif ECC[i] > milieu:
        med = X[i]
    else:
        med = (X[i]+X[i+1]) / 2
    return med
```

2. Avec les programmes et à la main, on trouve pour le premier tableau

Nb médailles d'or	0	①	2	4	5	6	15	19	25	TOT.
Nb de pays	7	5	2	1	1	2	1	1	1	21
ECC	7	**12**	14	15	16	18	19	20	21	–

Et une médiane de 1

Aux JO de Tokyo

Nb méd. or	0	①	2	3	4	6	7	10	17	20	22	27	38	39
Nb de pays	28	22	11	11	5	2	4	4	1	1	1	1	1	1
ECC	28	**50**	61	72	77	79	83	87	88	89	90	91	92	93

Retour sur l'exercice F3

1. Pas de difficulté pour la première question. En s'appuyant sur la remarque de l'aide, on distingue 2 cas pour calculer les quartiles selon que `N` est multiple de 4. Pour la médiane, on discute selon que `N` est pair ou non.

```
def infos(serie) :
    L = sorted(serie) # Nouvelle liste
    N = len(L)
    mini = L[0]
    maxi = L[len(L)-1]
    if N % 4  == 0 :
        Q1 = L[ceil(N / 4 - 1)]
        Q3 = L[ceil(3 * N / 4 - 1)]
    else :
        Q1 = L[N // 4]
        Q3 = L[(3*N) // 4]
    if N % 2 == 0 :
        med = (L[N//2 -1 ] + L[N//2] ) / 2
    else :
        med = L[N//2]
    return mini, Q1, med, Q3, maxi
```

Pour ne pas avoir à distinguer le cas où `N` est multiple de 4 ou non, on peut utiliser la fonction **`ceil`** du module `math` qui arrondit un nombre à l'entier supérieur le plus proche.

2. Avec les notations proposées dans l'aide, on a :

Point	A	B	C	D	E	F	G	H	I	J
Abscisse	*mini*	Q_1	Q_1	*Me*	*Me*	Q_3	Q_3	*maxi*	Q_3	Q_1
Ordonnée	2	2	1	1	3	3	2	2	1	3

On peut donc compléter le code en utilisant la fonction précédente :

```
import matplotlib.pyplot as plt

L = [155, 155, 168, 169, 165, 159, 167, 165, 165, 167, 165, 160, ↙
    ↳ 160, 163, 163, 165, 166, 156, 168, 167, 166, 160, 170, 156, ↙
    ↳ 159, 169, 159, 161, 164, 160, 172, 156, 156, 160]
mini, Q1, med, Q3, maxi = infos(L)
X = [mini, Q1, Q1, med, med, Q3, Q3, maxi, Q3, Q3, med, med, Q1, Q1]
Y = [2, 2, 1, 1, 3, 3, 2, 2, 2, 1, 1, 3, 3, 2]
plt.plot(X, Y, 'k-')
plt.axis([mini-1, maxi+1, 0, 4])
plt.show()
```

3. On peut aussi tracer les points de la série statistique ainsi :

```
for x in L:
    plt.plot(x, uniform(1.2, 2.8), 'r.')
```

4. Pour comparer les deux séries, on peut regarder les diagrammes en boîte. Il est préférable d'avoir la même échelle pour les deux, ainsi on va modifier la ligne : `plt.axis([mini-1, maxi+1, 0, 4])` en `plt.axis([150, 180, 0, 4])`.

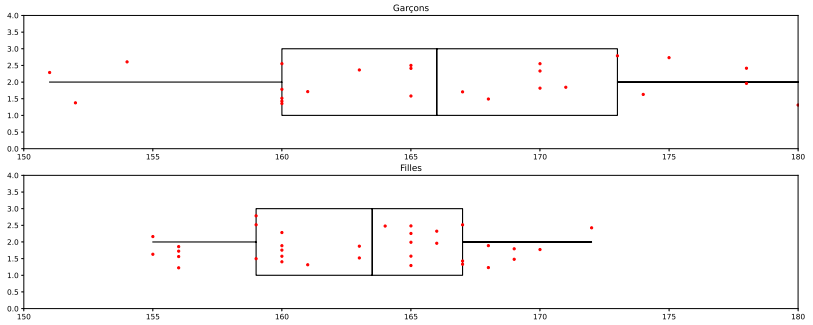
La correction qui suit présente une nouvelle fonction : **`subplot`**. Lorsqu'on appelle cette fonction, on utilise comme paramètre un nombre à 3 chiffres qui permet de découper la fenêtre graphique en sous-fenêtres :

- Le chiffre des centaines représente le nombre de colonnes
- Le chiffre des dizaines représente le nombre de lignes
- Le chiffre des unités représente le numéro du graphique actif (celui où l'on trace)

Ici on voudrait avoir les 2 diagrammes l'un sous l'autre. On appellera donc **`subplot(121)`** pour tracer le premier graphique et **`subplot(122)`** pour le second. Reste à transformer en fonction le précédent programme qui affiche un diagramme de manière à pouvoir l'utiliser deux fois. On obtient le programme final suivant :

```
import matplotlib.pyplot as plt
from random import uniform

def boite(L):
    for x in L:
        plt.plot(x, uniform(1.2, 2.8), 'r.')
    mini, Q1, med, Q3, maxi = infos(L)
    X = [mini, Q1, Q1, med, med, Q3, Q3, maxi, Q3, Q3, med, med, ↙
        ↳ Q1, Q1]
    Y = [2, 2, 1, 1, 3, 3, 2, 2, 2, 1, 1, 3, 3, 2]
    plt.plot(X, Y, 'k-')
    plt.axis([mini-1, maxi+1, 0, 4])
    plt.axis([150, 180, 0, 4])

H = [160, 154, 170, 160, 160, 171, 160, 160, 165, 168, 170, 151, ↙
    ↳ 165, 178, 161, 163, 175, 178, 152, 174, 186, 180, 170, 165, ↙
    ↳ 173, 167]
F = [155, 155, 168, 169, 165, 159, 167, 165, 165, 167, 165, 160, ↙
    ↳ 160, 163, 163, 165, 166, 156, 168, 167, 166, 160, 170, 156, ↙
    ↳ 159, 169, 159, 161, 164, 160, 172, 156, 156, 160]
plt.subplot(211)
plt.title('Garçons')
boite(H)
plt.subplot(212)
plt.title('Filles')
boite(F)
plt.show()
```

Et le graphique correspondant :



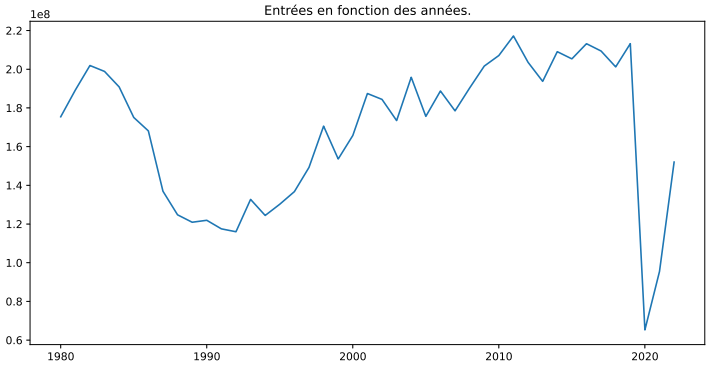
Retour sur l'exercice F4

1. Le nombre d'entrées en Mars 2000 (environ 14 millions) se trouve :
   - à la ligne 20 (car 2000 = 1980 + 20)
   - à la colonne 3 (car mars est le troisième mois de l'année)

```
>>> donnees[20, 3]
14438051
```

2. Pour répondre à cette question, on va calculer le nombre total d'entrées, année par année (on aurait d'ailleurs pu réaliser une fonction pour cela) :

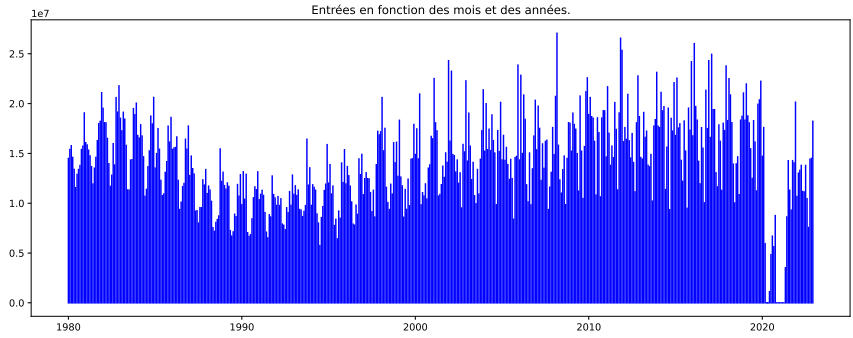
```
entrees = []
for annee in range(1980, 2023):
    total = 0
    for m in range(1, 13) :
        total = total + donnees[annee-1980, m]
    entrees.append(total)
plt.plot(range(1980, 2023), entrees)
plt.title("Entrées en fonction des années.")
plt.show()
```



On peut voir que dans les années 90, il y avait beaucoup moins d'entrées au cinéma. On remarque aussi la chute du nombre d'entrées autour des années 2020 et l'impact de la crise du COVID.

3. Pour être plus précis, nous allons tracer chaque barre au fur et à mesure des mois. Le calcul `(m-1)/12` permet de décaler chaque barre :

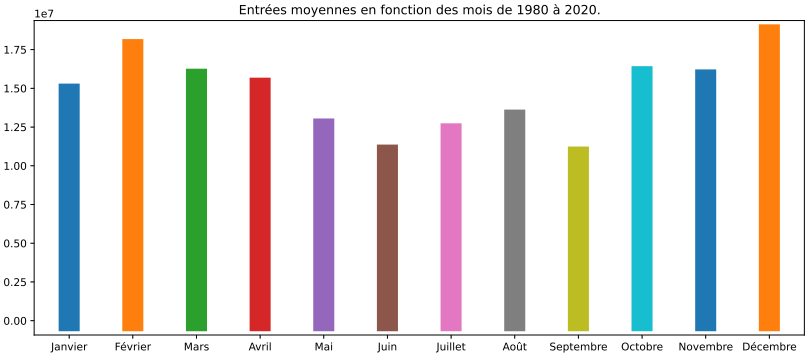
```
entrees = []
for annee in range(1980, 2023):
    for m in range(1, 13):
        x = annee + (m-1)/12
        plt.plot([x, x], [0, donnees[annee-1980, m]], 'b-')
plt.title("Entrées en fonction des mois et des années.")
plt.show()
```



On retrouve les 2 périodes de confinement liées au COVID. On peut aussi remarquer une certaine périodicité dans les motifs.

4. Pour cette dernière question, il suffit d'intervertir les deux boucles pour sommer sur les colonnes :

```
mois = ['Janvier', 'Février', 'Mars', 'Avril', 'Mai', 'Juin', ↙
    ↳ 'Juillet', 'Août', 'Septembre', 'Octobre', 'Novembre', ↙
    ↳ 'Décembre']
for m in range(1, 13) :
    total = 0
    for annee in range(1980, 2021):
        total = total + donnees[annee-1980, m]
    plt.plot([m, m], [0, total/41], linewidth = 20 )
    plt.xticks(range(1, 13), mois)

plt.title("Entrées moyennes en fonction des mois de 1980 à 2020.")
plt.show()
```



On peut penser que les mois de décembre et février avec les vacances sont propices aux sorties au cinéma, contrairement aux périodes estivales pendant lesquelles on peut avoir moins envie d'aller dans les salles obscures.

Sur ce dernier exemple, on peut remarquer deux nouvelles idées pour améliorer la présentation des graphiques :

- le paramètre **`linewidth`** dans le `plot` qui permet de jouer sur l'épaisseur des lignes.
- la fonction **`xticks`** qui reçoit une liste d'ordonnées et une liste de chaînes de caractères d'autant d'éléments et qui permet de mettre des étiquettes sur les axes. La fonction **`yticks`** existe aussi.

Retour sur l'exercice F5

1. a. Les informations se trouvent dans la colonne 0 :

```
def nb_matchs(j, tab):
    n = len(tab) # Nombre de lignes
    nb = 0
    for i in range(n) :
        if tab[i, 0] == j :
            nb = nb + 1
    return nb
```

b. Il y a 8 joueuses, on peut se contenter d'afficher le nombre de matchs de chacune :

```
for j in range(1, 9):
    nb_m = nb_matchs(j, stats)
    print("La joueuse n°", j, "a joué", nb_m, "matchs.")
```

2. On peut comme précédemment créer une fonction qui compte le nombre de fois où il y a eu `x` passes :

```
def nb_fois(x, tab) :
n = len(tab) # Nombre de lignes
nb = 0
for i in range(n) :
        if tab[i, 4] == x :
        nb = nb + 1
return nb
```

   a. On peut considérer, sans prendre trop de risques, qu'il y a moins de 1000 passes décisives dans un match et compter ainsi :

```
for n in range(1000) :
    nf = nb_fois(n, stats)
    if nf != 0 :
        print(n, nf)
```

   La valeur modale correspond donc à une passe (42 fois).

   b. On peut aussi totaliser pour obtenir une moyenne :

```
nbp, nbtot = 0, 0
for n in range(1000) :
    nf = nb_fois(n, stats)
    nbtot = nbtot + nf
    nbp = nbp + n * nf
print("Moyenne", nbp / nbtot)
```

   la valeur moyenne est d'environ 2,5.

3. a. Le code pour le calcul de `total` ressemble au code des deux fonctions précédentes :

```
def total(j, col, tab) :
        n = len(tab) # Nombre de lignes
t = 0
for i in range(n) :
        if tab[i, 0] == j :
        t = t + tab[i, col]
return t
```

b. On peut alors afficher les informations demandées :

```
for j in range(1, 9):
    tps = total(j, 1, stats) / 60 # en heure
    p2_r = total(j, 5, stats)
    p2_t = total(j, 6, stats)
    p3_r = total(j, 7, stats)
    p3_t = total(j, 8, stats)
    print("La joueuse n°", j, " :")
    print("    temps de jeu (h)", tps)
    print("    paniers 2 pts réussis", p2_r)
    print("    paniers 2 pts réussis à l'heure", p2_r / tps)
    print("   $%$ paniers 2 pts réussis", p2_r / p2_t * 100)
    print("    paniers 3 pts réussis", p3_r)
    print("    paniers 3 pts réussis à l'heure", p3_r / tps)
    print("   $%$ paniers 3 pts réussis ", p3_r / p3_t * 100)
```

Qui donne ces résultats :

joueuse	temps (h)	Q1	Q2	Q3	Q4	Q5	Q6
1	10,1	65	6,4	49,6	30	3,0	28,3
2	8,35	119	14,3	54,6	0	0,0	0,0
3	12,7	179	14,1	47,6	2	0,2	11,1
4	9,4	51	5,4	50,0	22	2,3	32,4
5	10,3	145	14,0	57,5	0	0,0	0,0
6	12,5	89	7,1	46,8	40	3,2	47,1
7	10,95	131	12,0	54,1	6	0,5	24,0
8	8,8	68	7,7	40,2	24	2,7	33,8

Suivant ce qu'on retient comme critère, on peut analyser les résultats de différentes façons. Ainsi, la joueuse 3 est celle qui a eu un temps de jeu le plus important et un nombre de tirs réussis à 2 points le plus élevé. La joueuse 6 est performante pour les paniers à 3 points avec 47,1% de réussite. La joueuse 2 est celle qui a le plus grand nombre de paniers à 2 points réussis. La joueuse 5 est celle qui a un taux de réussite le plus élevé sur les paniers à 2 points.

Retour sur l'exercice F6 Pour obtenir une simulation convenable, il nous faut réaliser le tirage d'un entier pouvant prendre toutes les valeurs de 1 à 6 (et seulement ces valeurs) de manière équiprobable. Commençons par la seconde question, pour comprendre les différents cas que nous analyserons ensuite. Pour se donner une première idée, on peut réaliser un grand nombre de lancers (ici 10 000) et calculer la fréquence d'apparition de chacune des faces.

- Si on n'est pas trop à l'aise avec les listes, il nous faut 6 variables. Si `de` prend une autre valeur que 1, 2, 3, 4, 5 ou 6, le programme s'arrête et affiche cette valeur. Sinon, il s'arrête au bout de 10 000 lancers et affiche les fréquences. Pour tester, il faut remplace la ligne 6 par la formule proposée dans l'énoncé :

```
1  from random import random
2  from math import floor
3  nb1, nb2, nb3, nb4, nb5, nb6 = 0, 0, 0, 0, 0, 0
4  i = 0
5  while i < 10000:
6      de = floor(6 * random() + 1)
7      if de == 1:
8          nb1 = nb1 + 1
9      elif de == 2:
10         nb2 = nb2 + 1
11     elif de == 3:
12         nb3 = nb3 + 1
13     elif de == 4:
14         nb4 = nb4 + 1
15     elif de == 5:
16         nb5 = nb5 + 1
17     elif de == 6:
18         nb6 = nb6 + 1
19     else:
20         print("Valeur tirée ", de, ". Impossible !")
21         i = 20000
22     i = i + 1
23 if i < 20000:
24     print("Fréquence des 1", round(nb1/i, 2))
25     print("Fréquence des 2", round(nb2/i, 2))
26     print("Fréquence des 3", round(nb3/i, 2))
27     print("Fréquence des 4", round(nb4/i, 2))
28     print("Fréquence des 5", round(nb5/i, 2))
29     print("Fréquence des 6", round(nb6/i, 2))
```

- Une seconde solution est d'utiliser une liste de 6 valeurs, ce qui raccourcit le code.

```
from random import random
from math import floor
nb = [0] * 6
i = 0
while i < 10000:
    de = floor(6 * random() + 1)
    if 1 <= de <= 6 :
        nb[de-1] = nb[de-1] + 1
    else:
        print("Valeur tirée ", de, ". Impossible !")
        i = 20000
    i = i + 1
if i < 20000:
    for j in range(6) :
        print("Fréquence des", j+1, ":", round(nb[j]/i, 2))
```

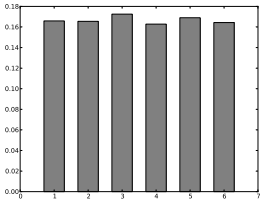
Attention dans le dernier script, en indice 0 on stocke le nombre de fois où on a obtenu 1 et ainsi de suite !

Voici les résultats obtenus après avoir exécuté le programme, pour que ce soit plus visuel, ils ont été présentés avec des diagrammes en bâtons lorsque toutes les valeurs tirées correspondaient au résultat d'un dé (vous pouvez essayer de programmer ce rendu !).

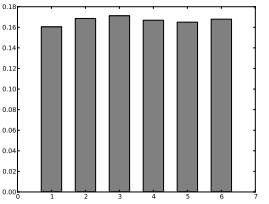
Proposition a :
Valeur tirée 0.
Impossible !

Proposition b :
Valeur tirée 0.
Impossible !

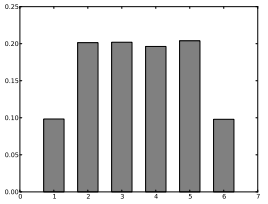
Proposition c :



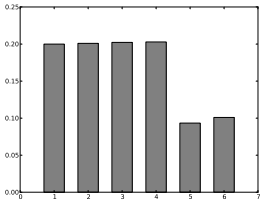
Proposition d :



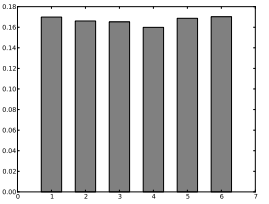
Proposition e :



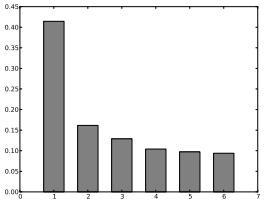
Proposition f :



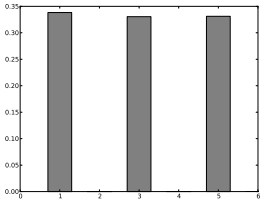
Proposition g :



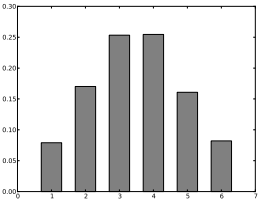
Proposition h :



Proposition i :

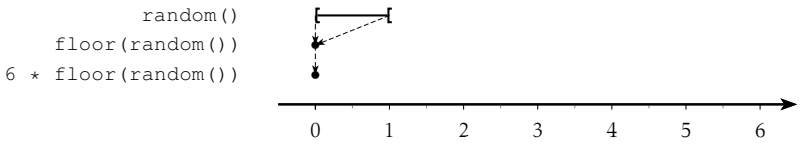


Proposition j :



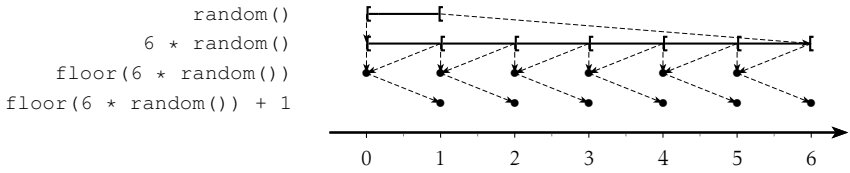
À première vue, il semblerait que seules les propositions c, d et g conviennent. Expliquons les graphiques obtenus.

a. Tous les tirages donnent 0 puisque l'on prend la partie entière d'un nombre de l'intervalle $[0;1[$.

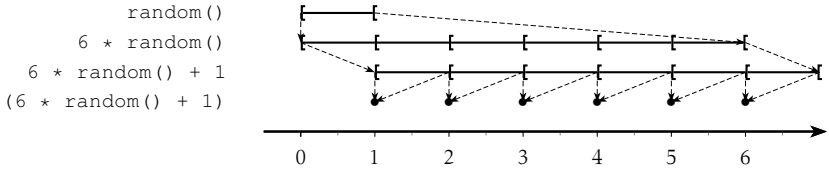


b. Le nombre 0 peut en effet apparaître. Par exemple lorsque `random()` renvoie 0.1, on a `de = floor(6*0.1) = floor(0.6) = 0`.

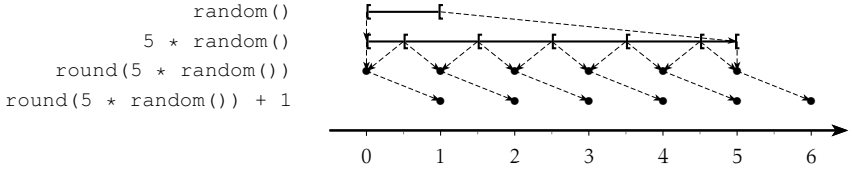
c. Cette proposition fonctionne comme l'illustre le schéma ci-dessous :



d. La proposition d ressemble à la précédente, sauf que l'on ajoute 1 avant de prendre la partie entière.



e. Cette proposition fonctionne comme l'illustre le schéma ci-dessous :



f. Comme on l'a vu dans la proposition b, `floor(10 * random())` donne un entier entre 0 et 9 choisi de manière équiprobable. Notons n ce nombre.

n	0	1	2	3	4	5	6	7	8	9
$n\%6$	0	1	2	3	4	5	0	1	2	3
$(n\%6)+1$	1	2	3	4	5	6	1	2	3	4

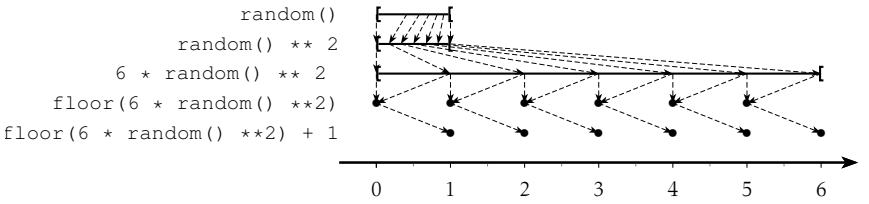
Ce qui explique que les valeurs 1, 2, 3 et 4 apparaissent deux fois plus souvent que 5 et 6.

g. Sur le même principe que la proposition f, `floor(12 * random())` donne un entier entre 0 et 11 choisi de manière équiprobable. Notons n ce nombre.

n	0	1	2	3	4	5	6	7	8	9	10	11
$n\%12$	0	1	2	3	4	5	0	1	2	3	4	5
$(n\%6)+1$	1	2	3	4	5	6	1	2	3	4	5	6

Cette fois-ci les 6 valeurs apparaissent de manière équiprobable, cette proposition convient donc.

h. Encore une fois, nous allons faire un schéma pour illustrer pourquoi cette proposition n'est pas satisfaisante :



Sur le schéma, on voit que le résultat 1 va apparaître plus souvent que le 6. En effet, si on note x la valeur retournée par `random()` et E la fonction partie entière, nous obtenons

$$
\begin{array}{l|l}
de = 1 & de = 6 \\
\iff \mathrm{E}(6x^2)+1=1 & \iff \mathrm{E}(6x^2)+1=6 \\
\iff \mathrm{E}(6x^2)=0 & \iff \mathrm{E}(6x^2)=5 \\
\iff 0 \leqslant 6x^2 < 1 & \iff 5 \leqslant 6x^2 < 6 \\
\iff 0 \leqslant x^2 < \dfrac{1}{6} & \iff \dfrac{5}{6} \leqslant x^2 < 1 \\
\iff 0 \leqslant x < \sqrt{\dfrac{1}{6}} \approx 0,41 & \iff 0,91 \approx \sqrt{\dfrac{5}{6}} \leqslant x < 1
\end{array}
$$

On peut donc s'attendre à ce que le 1 apparaisse environ 4 fois plus souvent que le 6, comme le laissait pressentir le graphique.

i. La commande `floor(3 * random()` simule le choix d'un entier dans l'ensemble $\{0,1,2\}$ de manière équiprobable. En multipliant par deux ce résultat et en ajoutant 1, on obtient alors un nombre de l'ensemble $\{1,3,5\}$; ce qui ne permet pas de simuler un lancer de dé.

j. Enfin le graphique de la dernière simulation nous fait penser que l'on obtient bien tous les résultats mais de manière non équiprobable. En effet, on a vu que `floor(3 * random())` donne un entier de $\{0,1,2\}$ de manière équiprobable, notons x ce nombre. De même y =`floor(4 * random())` donne un entier de $\{0,1,2,3\}$. Comme on a $de = x + y + 1$. Regardons dans un tableau les valeurs possibles pour la variable de en plaçant sur la première colonne les valeurs de x et sur la première ligne les valeurs de y.

	0	1	2	3
0	1	2	3	4
1	2	3	4	5
2	3	4	5	6

On comprend alors pourquoi le 3 apparaît trois fois plus souvent que le 1 ou que le 6.

Retour sur l'exercice F7

1. a. Pour la simulation d'un tirage, on va avoir recours à une liste dont les éléments sont les boules, à l'aide de la fonction `choice`, on va tirer une boule que l'on supprimera de la bouteille avec la fonction `remove` :

```
from random import choice

def tirage(bout) :
    if bout == "A" :
        bouteille = ['Orange']*3 + ['Verte']
    else :
        bouteille = ['Orange', 'Verte']*2
    b1 = choice(bouteille)
    bouteille.remove(b1)
    b2 = choice(bouteille)
    return b1, b2
```

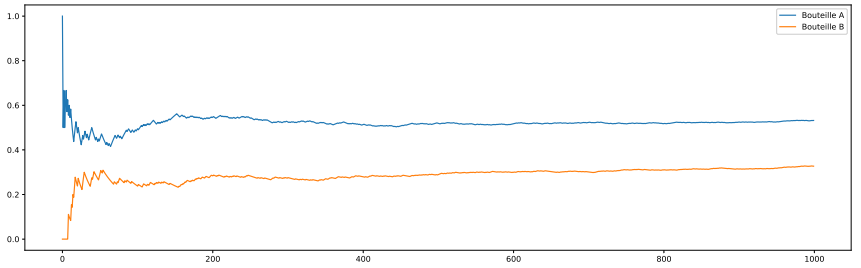
b. Pour cette fonction, on compte le nombre de succès avec la variable `nb` :

```
def frequences(bout, N) :
    """
    En entrée : la bouteille : "A" ou "B" et le nombre de ↙
        ↳ tirages
    En sortie : La liste des fréquences
    """
    nb = 0
    Fr = []
    for i in range(N):
        boule1, boule2 = tirage(bout)
        if boule1 == boule2:
            nb = nb + 1
        Fr.append(nb/(i+1))
    return Fr
```

c. Pour représenter les courbes, on peut utiliser le mot clé `label` qui permet de mettre une légende à un graphique, il faut alors écrire **`legend()`** avant le `show()` pour l'afficher.

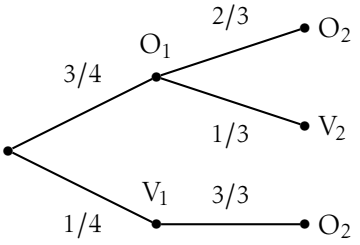
```
import matplotlib.pyplot as plt
plt.plot(frequences("A", 1000), label="Bouteille A")
plt.plot(frequences("B", 1000), label="Bouteille B")
plt.legend()
plt.show()
```

d. Il semble que la probabilité de tirer 2 boules de la même couleur, soit de $1/2$ dans la bouteille A et de $1/3$ dans la bouteille B.



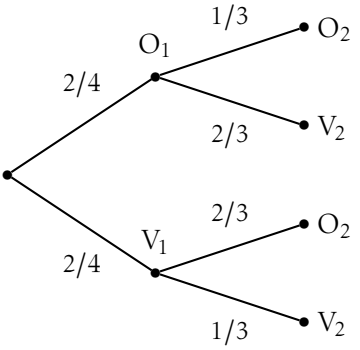
2. Si on cherche à calculer la probabilité de l'événement C "avoir 2 boules de la même couleur", un arbre peut nous aider. En notant O_i l'événement « obtenir une boule orange au i-ème tirage » et V_i pour les boules vertes :

Bouteille A :



$$p(C) = p(O_1 \cap O_2) = p(O_1) \times p_{O_1}(O_2) = \frac{3}{4} \times \frac{2}{3} = \frac{1}{2}$$

Bouteille B :



$$p(C) = \frac{2}{4} \times \frac{1}{3} + \frac{2}{4} \times \frac{1}{3} = \frac{1}{3}$$

Ce qui corrobore la conjecture émise suite à notre simulation.

Retour sur l'exercice F8 Le plus simple est de parcourir 10 fois le nombre en s'intéressant à chaque tour à un chiffre différent :

```
for i in range(10):
    nb = 0
    for chiffre in P:
        if chiffre == str(i):
            nb = nb + 1
    print("Fréquence d'apparition du chiffre", i, ":", nb/2000)
```

On s'aperçoit que les deux chiffres qui ont le plus de chance d'être choisis sont le 1 et le 9. Néanmoins les fréquences observées sont très proches. D'ailleurs plus on augmente le nombre de décimales, plus on est convaincu que toutes ces fréquences tendent vers $\frac{1}{10}$. À notre connaissance, cette conjecture n'est pas démontrée à ce jour.

Retour sur l'exercice F9

1. Le nombre de boules est $1+2+3+\ldots+9=\frac{9\times 10}{2}=45$

2. Par exemple :

```
U = []
for n in range(1,10) :
    for i in range(n) :
        U.append(n)
```

   Plus court, mais moins efficace :

```
U = []
for n in range(1,10) :
    U = U + [n]*n
```

3. $X(\Omega)=[\![0;8]\!]$, on va donc créer une liste `L` de neuf 0 telle que `L[i]` contiendra le nombre de fois où l'écart `i` a été constaté :

```
L = [0]*9
N = 10000
for _ in range(N) :
    b1 = U[randint(0,44)]    # Ou encore b1 = choice(U)
    b2 = U[randint(0,44)]
    ecart = abs(b2-b1)
    L[ecart] = L[ecart] + 1
L = [x/N for x in L]         # On passe des effectifs aux fréquences
print("Estimation des probabilités : ")
print(L)
e = 0
for i in range(9) :
    e = e + i * L[i]
print("Estimation de l'espérance :", e)
```

Par exemple, on obtient :

```
Estimation des probabilités :
[0.137, 0.245, 0.186, 0.153, 0.114, 0.078, 0.047, 0.027, 0.008]
Estimation de l'espérance : 2.482
```

4. Si on souhaite la loi de probabilité, il faut dénombrer les possibilités et non simuler. Comme il va être très laborieux de dénombrer à la main, on adapte le programme principal en réalisant une boucle **pour** sur toutes les possibilités de `b1` et `b2` dans l'urne au lieu de les tirer au hasard :

```
L = [0]*9
nb_tirages = 0
for b1 in U :
    for b2 in U :
        ecart = abs(b2-b1)
        L[ecart] = L[ecart] + 1
        nb_tirages = nb_tirages + 1
print(L)
```

On trouve

```
[285, 480, 392, 308, 230, 160, 100, 52, 18]
2025
```

Ainsi, en divisant ces valeurs par le nombre de tirages possibles ($2025 = 45^2$), on obtient :

k	0	1	2	3	4	5	6	7	8
$P(X=k)$	$\frac{285}{45^2}$	$\frac{480}{45^2}$	$\frac{392}{45^2}$	$\frac{308}{45^2}$	$\frac{230}{45^2}$	$\frac{160}{45^2}$	$\frac{100}{45^2}$	$\frac{52}{45^2}$	$\frac{18}{45^2}$
$\approx$	0,14	0,24	0,19	0,15	0,11	0,08	0,05	0,03	0,01

On retrouve un résultat cohérent avec notre simulation.

5. Avec la liste obtenue, on peut calculer l'espérance :

```
e = 0
for i in range(9) :
    e = e + i*L[i]
print(e)
print(e / nb_tirages)
```

On trouve $E(X) = \frac{5016}{2025} \approx 2,48$

Retour sur l'exercice F10

1. Si on note a, b et c les longueurs des 3 côtés du triangle, on peut trouver divers tests pour vérifier la constructibilité du triangle :

   - $a < b + c$ et $b < a + c$ et $c < b + a$
   - $\begin{cases} grand = \max\{a, b, c\} \\ petit = \min\{a, b, c\} \\ moyen = a + b + c - grand - petit \\ grand < petit + moyen \end{cases}$

     en remarquant que : $petit + moyen = a + b + c - grand$:
   - $\begin{cases} grand = \max\{a, b, c\} \\ grand < a + b + c - grand \end{cases}$ ou encore $\begin{cases} grand = \max\{a, b, c\} \\ 2 \times grand < a + b + c \end{cases}$

     ou en une seule ligne :
   - $2 \times \max\{a, b, c\} < a + b + c$

   Ce qui donne :

```
def triangle(de1, de2, de3):
    if 2 * max(de1, de2, de3) < de1 + de2 + de3:
        return "Gagné"
    else:
        return "Perdu"
```

2. On pourrait se contenter de faire une boucle pour chacun des dés afin d'afficher ce qui est demandé, mais au vu du nombre de possibilités, nous allons ajouter un compteur `jeux` qui comptera le nombre de jeux différents et un autre `gagne` qui comptera le nombre de parties gagnées pour ne pas avoir à le faire manuellement.

```
jeux, gagne = 0, 0
for de1 in range(1, 7):
    for de2 in range(1, 7):
        for de3 in range(1, 7):
            jeux = jeux + 1
            if triangle(de1, de2, de3) == "Gagné":
                print(de1, de2, de3, "-> Gagné")
                gagne = gagne + 1
            else:
                print(de1, de2, de3, "-> Perdu")
print(gagne, "parties gagnantes pour", jeux, "parties différentes")
```

Le programme affiche "111 parties gagnantes pour 216 parties différentes". Comme nous sommes en situation d'équiprobabilité, on déduit par exhaustivité des cas que la probabilité de l'événement "Gagner une partie" est $p = \dfrac{111}{216}$.

AU DÉTOUR DE LA BALADE...

Informatiquement, il aurait été préférable d'utiliser un booléen en sortie de la fonction `triangle`, cependant cette activité étant plus à destination des classes de seconde, la manipulation de booléen peut s'avérer être une difficulté supplémentaire qui n'est pas utile ici.

3. Pas de difficulté particulière pour cette question. Cette fois-ci les 3 dés sont tirés au hasard :

```
def echantillon(N):
    gagne = 0
    for i in range(N):
        de1 = randint(1, 6)
        de2 = randint(1, 6)
        de3 = randint(1, 6)
        if triangle(de1, de2, de3) == "Gagné":
            gagne = gagne + 1
    return gagne / N
```

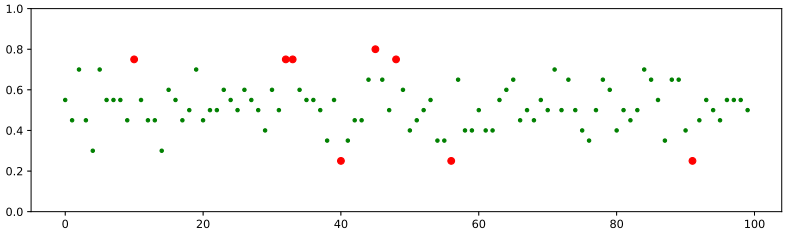
Voici les résultats obtenus (arrondis à 2 chiffres après la virgule) pour quelques valeurs de N :

N = 20	0,5	0,65	0,45	0,7	0,35	0,65	0,6	0,4	0,6
N = 50	0,5	0,48	0,58	0,56	0,5	0,48	0,46	0,52	0,5
N = 100	0,45	0,55	0,53	0,43	0,52	0,46	0,48	0,58	0,54
N = 1000	0,52	0,49	0,48	0,52	0,51	0,52	0,52	0,53	0,51
N = 10000	0,51	0,51	0,51	0,51	0,51	0,52	0,51	0,52	0,51

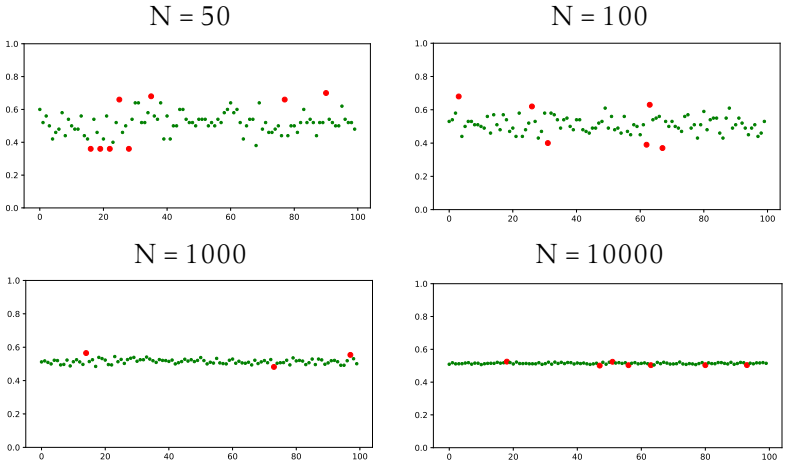
On constate que plus N est grand, plus les fréquences se stabilisent autour d'une valeur proche de 0,51 (valeur approchée de la probabilité à savoir $\frac{111}{216}$). C'est ce que l'on appelle en mathématique la loi des grands nombres.

D'autre part, lorsque N n'est pas très grand, on constate que les fréquences observées fluctuent beaucoup. C'est ce que l'on va essayer de traduire graphiquement dans la suite de cet exercice.

4. Voici un graphique obtenu pour $N = 20$.



Les points verts correspondent à des fréquences de gain dans l'intervalle $\left[p - \frac{1}{\sqrt{N}} \; ; \; p + \frac{1}{\sqrt{N}} \right]$. Les autres sont représentés sous forme de billes rouges.



Lorsque N augmente, les fréquences se "resserrent" autour de la probabilité p, cependant le nombre de points situés hors zone verte reste stable (souvent moins de 5). L'intervalle $\left[p - \frac{1}{\sqrt{N}} \; ; \; p + \frac{1}{\sqrt{N}} \right]$ s'appelle **l'intervalle de fluctuation au seuil de 95%**. En effet, dans le cas comme ici où $N > 25$ et $p \in [0,2;0,8]$ les fréquences obtenues sur les échantillons de taille N ont une probabilité supérieure à $0,95$ d'être dans cet intervalle (d'où les quelques points présents hors-zone).

Retour sur l'exercice F11

1. a. Une boule bleue rapporte 2 €, une boule rouge 5€, une boule blanche ne rapporte rien.

   b. La liste `U` n'est pas modifiée au fur et à mesure du programme (il n'y a pas de `remove` ou de `pop`). Il s'agit donc d'un tirage sans remise.

   c. La variable `T` est une liste où l'on ajoute au fur et à mesure les boules tirées. On sort de la boucle `while` dès qu'une boule de même couleur est déjà dans la liste. Ainsi, le jeu s'arrête lorsque l'on a deux boules de la même couleur.

2. a. Notons que l'on peut arrêter le jeu lorsque :

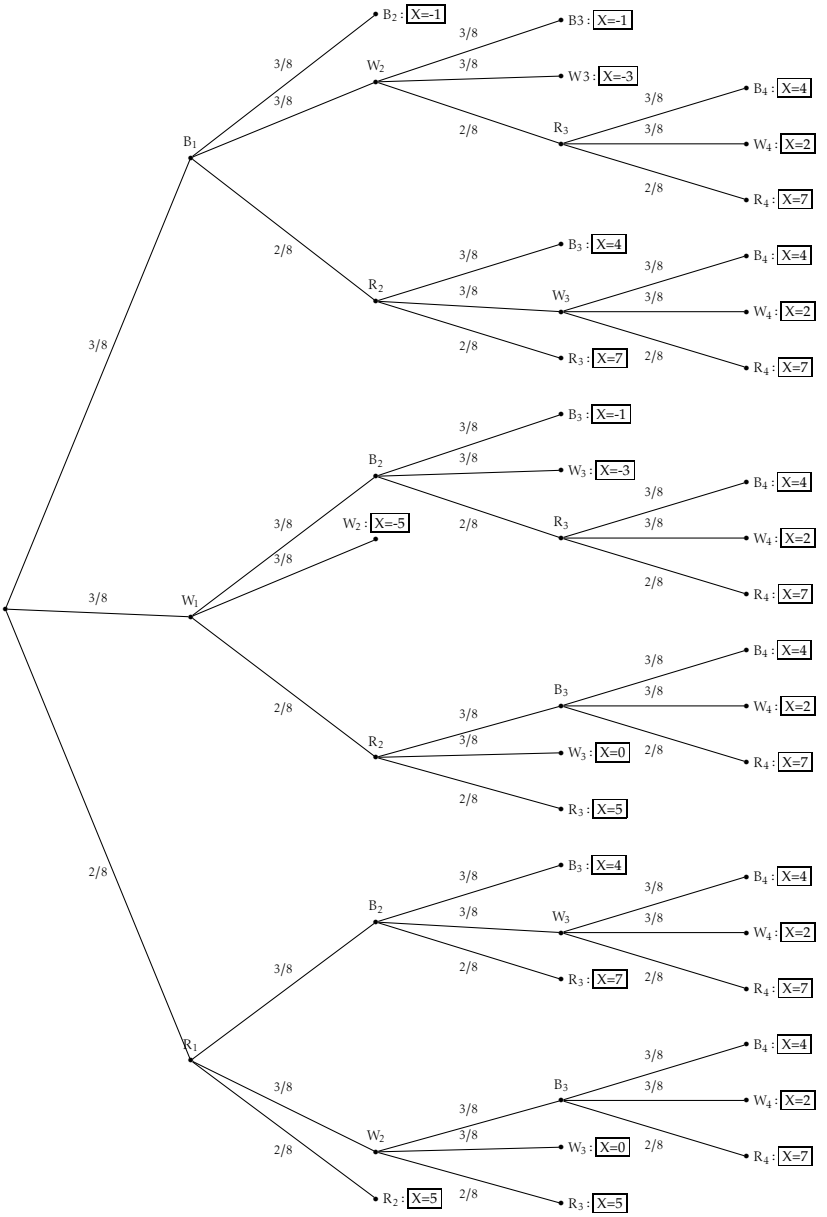
   - On a 2 boules blanches. Dans ce cas, on peut avoir :
     - Rien d'autre : $X = -5$
     - 1 boule bleue : $X = -5 + 2 = -3$
     - 1 boule rouge : $X = -5 + 5 = 0$
     - 1 boule bleue et une rouge : $X = -5 + 2 + 5 = 2$
   - On a 2 boules bleues. Dans ce cas, on peut avoir (en ne s'intéressant qu'aux boules qui rapportent de l'argent) :
     - Rien : $X = -5 + 2 \times 2 = -1$
     - 1 rouge : $X = -5 + 2 \times 2 + 5 = 4$
   - On a 2 boules rouges. Dans ce cas, on peut avoir (en ne s'intéressant qu'aux boules qui rapportent de l'argent) :
     - Rien : $X = -5 + 2 \times 5 = 5$
     - 1 bleue : $X = -5 + 2 \times 5 + 2 = 7$

   Ainsi $\Omega(X) = \{-5; -3; -1; 0; 2; 4; 5; 7\}$

b. On peut modifier le programme en ajoutant une boucle POUR afin de répéter 10000 fois le jeu. La variable x représente cette fois-ci, la somme accumulée au cours des différentes parties.

```
from random import *
U = ['Bleu'] * 3 + ['Blanc'] * 3 + ['Rouge'] * 2
x = 0
N = 10000
for i in range(N):
    T = []
    x = x-5
    b = 'Rien'
    while b not in T:
        T.append(b)
        b = choice(U)
        if b == 'Bleu':
            x = x + 2
        elif b == 'Rouge':
            x = x + 5
print(x/N)
```

3. Notons B_i l'événement "**la boule tirée au $i^{\text{ème}}$ tirage est bleue**", et de la même manière W_i et R_i pour les boules blanches et rouges. Réalisons un arbre des possibles à la page suivante :



On obtient alors la loi de probabilité en calculant les probabilités de chaque chemin :

k	-5	-3	-1	0	2	4	5	7
$p(X = k)$	$\frac{9}{64}$	$\frac{27}{256}$	$\frac{63}{256}$	$\frac{9}{128}$	$\frac{81}{1024}$	$\frac{153}{1024}$	$\frac{7}{64}$	$\frac{51}{512}$

4. L'espérance de X est donc :

$$E(X) = -5 \times \frac{9}{64} - 3 \times \frac{27}{256} - 1 \times \frac{63}{256} + \ldots + 5 \times \frac{7}{64} + 7 \times \frac{51}{512} = \frac{47}{64}$$

Retour sur l'exercice F12

1. a. On peut faire quelques simplifications pour la modélisation :

   - On considère que toutes les années comportent 365 jours.
   - On suppose aussi qu'il y a autant de naissances tous les jours de l'année.

   On peut alors proposer la fonction :

```
from random import randint

def jours(n) :
    eleves = []
    for i in range(n):
        eleves.append(randint(1,365))
    return eleves
```

   ou

```
from random import randint

def jours(n) :
    return [randint(1,365) for i in range(n)]
```

   b. Cette question n'est pas si évidente. Voici des idées qui fonctionnent :

   - La première : on regarde valeur par valeur si on retrouve une valeur identique dans la suite (d'où la double boucle). Dès que c'est trouvé, on renvoie `True`. Si à la fin du parcours aucun double n'a été trouvé, on renvoie `False` :

```
def jumeauxdejour(L) :
    for i in range(len(L)) :
        for j in range(i+1, len(L)) :
            if L[i] == L[j] :
                return True
    return False
```

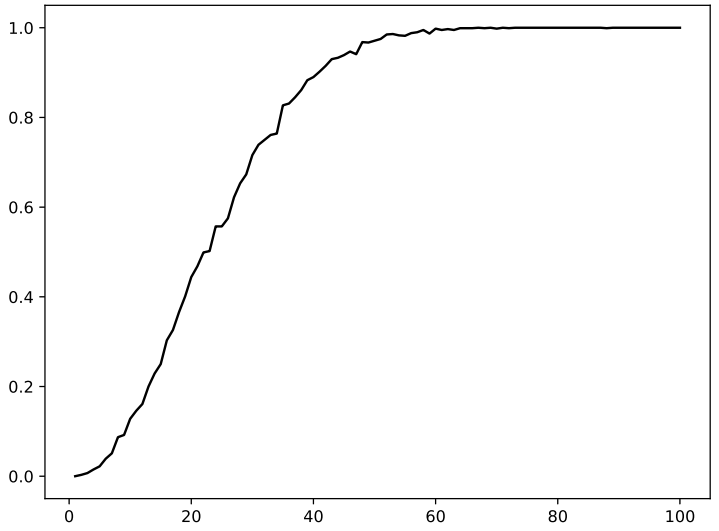
- Une seconde idée peut être de trier la liste, puis de regarder si 2 valeurs identiques se suivent. Il n'y a cette fois qu'une boucle, mais le tri prend du temps !

```
def jumeauxdejour(L) :
    L = sorted(L)
    for i in range(len(L)-1):
        if L[i] == L[i+1]:
            return True
    return False
```

c. Pour cette question, on stocke dans une liste `N` le nombre d'élèves dans la classe et dans `P` la probabilité d'avoir deux élèves nés le même jour (d'ailleurs ici c'est une fréquence !)

```
from random import randint
N, P = [], []
for n in range(1,101) :
    nb = 0
    for s in range(1000) :
        classe = jours(n)
        if jumeauxdejour(classe) :
            nb = nb + 1
    N.append(n)
    P.append(nb/1000)
plt.plot(N,P)
plt.show()
```

On obtient un graphique similaire à celui-ci :



Un peu après 20 élèves, on passe au dessus du seuil de 0,5 concernant le fait d'avoir 2 élèves nés le même jour.

2. a. Il est plus simple de raisonner par l'événement contraire : $\overline{A_n}$: « Les n élèves sont nés des jours différents ». Les dates d'anniversaires peuvent être modélisées par des n-uplets $(j_1, j_2, ..., j_n)$ où chaque $j_i \in [\![1;365]\!]$.

- Le nombre de n-uplets est 365^n.
- Le nombre de n-uplets représentant une situation où aucun élève n'est né le même jour qu'un autre est

$$365 \times 364 \times 363 \times ... \times (365 - n + 1) = \frac{365!}{(365-n)!}$$

Comme on suppose l'équiprobabilité des possibilités, on trouve

$$P(A_n) = 1 - P(\overline{A_n}) = 1 - \frac{365!}{365^n(365-n)!}$$

AU DÉTOUR DE LA BALADE...

Il peut être intéressant ici de demander aux élèves d'ajouter au tracé précédent, celui de la courbe de $P(A_n)$ en fonction de n pour confronter l'allure des deux courbes.

b. Il faut ici résoudre $p_n = P(A_n) \geqslant \frac{1}{2}$, ce qui n'est pas évident. La suite étant croissante (par comparaison du quotient $\frac{p_{n+1}}{p_n}$ à 1 par exemple), il suffit de trouver le premier entier n vérifiant cette condition. Une boucle TANT QUE répondra donc à notre problème :

```
from math import factorial

def A(n):
    return 1 - factorial(365) / (365**n * factorial(365-n))

n = 1
while A(n) < 0.5:
    n = n + 1
print("A partir de", n, "élèves on a plus de 50% de chance ↙
    ↳ d'avoir des jumeaux de jours.")
```

On trouve que cela est vérifié à partir de 23 élèves.

Retour sur l'exercice F13

1. Pas de difficulté dans cette question, on crée deux compteurs `nb9` et `nb10` que l'on incrémente à chaque fois que l'on trouve 9 ou 10 :

```
from random import randint

nb_lancers = 50
nb9, nb10 = 0, 0
for i in range(nb_lancers) :
    de1 = randint(1,6)
    de2 = randint(1,6)
    de3 = randint(1,6)
    if de1 + de2 + de3 == 9 :
        nb9 = nb9 + 1
    if de1 + de2 + de3 == 10 :
        nb10 = nb10 + 1
if nb9 > nb10 :
    print("Plus de 9")
elif nb9 == nb10 :
    print("Egalité")
else :
    print("Plus de 10")
```

2. En effet, en exécutant plusieurs fois le programme, on constate qu'il y a légèrement moins de parties au cours desquelles le 9 est obtenu.

3. Pour répondre à cette dernière question, on a adapté le programme précédent. Cette fois-ci, on ne tire plus les 3 dés mais on va parcourir l'ensemble des possibilités. Chaque dé pouvant donner un résultat entre 1 et 6, nous réalisons 3 boucles imbriquées :

```
from random import randint

nb9, nb10, nbCas = 0, 0, 0
for deRouge in range(1, 7):
    for deBleu in range(1, 7):
        for deVert in range(1, 7):
            if deRouge + deBleu + deVert == 9:
                nb9 = nb9 + 1
            if deRouge + deBleu + deVert == 10:
                nb10 = nb10 + 1
            nbCas = nbCas + 1
print("Nombre de fois où l'on a eu 9 : ", nb9)
print("Nombre de fois où l'on a eu 10 : ", nb10)
print("Nombre total d'issues : ", nbCas)
```

```
Nombre de fois où l'on a eu 9 : 25
Nombre de fois où l'on a eu 10 : 27
Nombre total d'issues : 216
```

Chacune des issues ayant la même probabilité d'être réalisée, on trouve $p(\text{A}) = \frac{25}{216}$ et $p(\text{B}) = \frac{27}{216}$ qui sont en effet très proches, mais

on a bien $p(\text{A}) < p(\text{B})$. Le raisonnement du Duc de Toscane est faux car les événements listés ne sont pas équiprobables. En effet, le total $9 = 3 + 3 + 3$ n'arrive que pour le triplet $(Rouge, Bleu, Vert) = (3,3,3)$ alors que la somme $9 = 1 + 2 + 6$ se produit lorsque l'on a les issues $(1,2,6),(1,6,2),(2,1,6),(2,6,1),(6,1,2),(6,2,1)$ et donc 6 fois plus souvent.

Retour sur l'exercice F14

1. $(a+b)^3 = (a+b)^2(a+b) = (a^2+2ab+b^2)(a+b) = a^3+3a^2b+3ab^2+b^3$.

2. On commence par créer une matrice de zéros, sur chaque ligne, on place les 1 de début et de fin et on complète la ligne par la formule donnée :

```
from numpy import zeros

def pascal(m):
    M = zeros((m+1,m+1), dtype=int)
    M[0,0] = 1
    for n in range(m):
        M[n+1, 0] = 1
        M[n+1, n+1] = 1
        for p in range(n):
            M[n+1, p+1] = M[n, p+1] + M[n, p]
    return M[m,:]
```

AU DÉTOUR DE LA BALADE...

On peut réaliser cela simplement avec une liste, mais c'est moins simple, il faut initialiser une liste à $[1,1]$ et à chaque étape, commencer par ajouter un 1 en fin de liste et calculer les coefficients de la nouvelle ligne de la droite vers la gauche pour ne pas écraser les valeurs dont on a encore besoin. Cela donne :

```
def pascal(n):
    Coef = [1, 1]
    for ligne in range(2, n+1):
        Coef.append(1)
        colonne = len(Coef) - 2
        while colonne > 0:
            Coef[colonne] = Coef[colonne] + Coef[colonne-1]
            colonne = colonne - 1
    return Coef
```

3. Une fois la liste des coefficients obtenue, on peut afficher le résultat souhaité. On utilise `sep=""` pour ne pas espacer les résultats ainsi que `end=""` pour ne pas aller à la ligne d'un `print` à l'autre.

```
def newton(n) :
    C = pascal(n)
    print("(a+b)^", n, "=", sep="", end="")
    for k in range(n+1):
        if C[k] != 1:
            print(C[k], end="")
        if n-k > 0:
            print("a", end="")
            if n-k > 1:
                print("^", n-k, sep="", end="")
        if k > 0:
            print("b", end="")
            if k > 1:
                print("^", k, sep="", end="")
        if k != n:
            print("+", end="")
```

Retour sur l'exercice F15

1. Le code de déclaration de cette fonction tient en 2 lignes car il n'est pas nécessaire de vérifier que $0 \leqslant k \leqslant n$ puisque cela est précisé dans l'énoncé.

```
from math import factorial

def CoefBinom(n, k):
    """
    Renvoie le coefficient binomial "k parmi n"
    """
    return factorial(n) / (factorial(n-k) * factorial(k))
```

La notion de factorielle d'un nombre n'est pas un attendu du programme de la spécialité mathématiques en terminale, on peut cependant remarquer que cette fonction n'est pas ce qu'il y a de plus efficace en temps de calcul. On réalise en effet de nombreux calculs inutiles qui pourraient être simplifiés avant d'être effectués. Par exemple :

$$\binom{10}{3} = \frac{10!}{3! \times 7!} = \frac{1 \times 2 \times 3 \times 4 \times 5 \times 6 \times 7 \times 8 \times 9 \times 10}{1 \times 2 \times 3 \times 1 \times 2 \times 3 \times 4 \times 5 \times \times 6 \times 7}$$

On pourrait donc optimiser ainsi :

```
from math import factorial

def CoefBinom(n, k):
    numerateur = 1
    for j in range(n-k+1, n+1):
        numerateur = numerateur * j
    return numerateur / factorial(k)
```

2. En utilisant la fonction programmée, il suffit alors d'appliquer la formule vue en classe : $p(X=k) = \binom{n}{k} p^k (1-p)^{n-k}$ pour $0 \leqslant k \leqslant n$.

```
def afficheLoi(n,p) :
    for k in range(n+1) :
        prob = CoefBinom(n,k) * p**k * (1-p)**(n-k)
        print("P(X=", k, ") = ", prob, sep="")
```

3. Pour répondre à la question, on peut créer une nouvelle variable (ici `cumul`) qui servira à cumuler les probabilités trouvées.

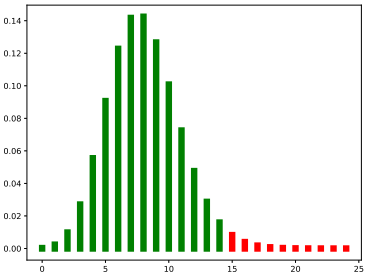
```
def afficheLoiCumul(n,p) :
    cumul = 0
    for k in range(n+1):
        prob = CoefBinom(n, k) * p**k * (1-p)**(n-k)
        cumul = cumul + prob
        print("P(X<=", k, ") = ", cumul, sep="")
```

Au passage, vous remarquerez que le dernier résultat donne pour $n = 10$ et $p = 0,3$, $P(X \leqslant 10) = 0,9999999999999992$ alors que l'on s'attendrait à avoir 1 bien évidemment. Cela est toujours dû aux problèmes d'arrondis et du cumul de ces petites erreurs. Si cela vous gêne, ajoutez un `round` pour n'avoir que 4 chiffres significatifs dans l'affichage.

4. En première réponse, on peut penser que l'engagement du grossiste n'est pas respecté car il y a 7,5% d'ampoules défectueuses, ce qui est bien supérieur aux 4% annoncés. D'un autre côté, on peut imaginer que le vendeur a joué de "malchance". Supposons donc que l'affirmation du grossiste soit vraie. Simulons le tirage de 200 ampoules dans son stock immense et notons X la variable aléatoire correspondant au nombre d'ampoules défectueuses obtenues. Le stock étant important, on peut assimiler ce tirage à un tirage avec remise, si bien que X suit une loi binomiale $\mathcal{B}(200;0,04)$. Estimons $P(X \geqslant 15)$. Avec notre programme, on va lire $P(X \geqslant 15) = 1 - P(X < 15) = 1 - P(X \leqslant 14)$. On trouve $P(X \leqslant 14) \approx 0,9848 > 0,98$. D'où $P(X \geqslant 15) \approx 0,0153 < 2\%$.

Avec un risque d'erreur inférieur à 2%, on peut affirmer que l'engagement du grossiste n'est pas respecté.

Le test réalisé ici s'appelle un **test unilatéral**.



Retour sur l'exercice F16

1. Voici une idée : on crée une liste `valeurs` avec les 13 cartes d'une couleur. Il faudra veiller à mettre les nombres sous forme de texte pour pouvoir les ordonner ensuite. On les recopie 4 fois ensuite avec une boucle :

```
valeurs = ['Valet', 'Dame', 'Roi', 'As']
for i in range(2, 11):
    valeurs.append(str(i))

jeu = []
for v in valeurs:
    for c in range(4):
        jeu.append(v)
```

Plus court, on peut "multiplier par 4" la liste `valeurs` :

```
valeurs = ['Valet', 'Dame', 'Roi', 'As']
for i in range(2, 11):
    valeurs.append(str(i))
jeu = jeu * 4
```

Encore plus court, on peut créer la liste ainsi :

```
jeu=([str(i) for i in range(2,11)] + ['Valet','Dame','Roi','As'])*4
```

2. Pour la réalisation de la liste `main`, on va partir d'une liste vide et répéter 5 fois :

   - tirer une carte,
   - l'ajouter à la liste `main`,
   - la retirer de la liste `jeu`.

```
from random import *
jeu=([str(i) for i in range(2,11)] + ['Valet','Dame','Roi','As'])*4

main = []
for i in range(5):
    carte = choice(jeu)
    main.append(carte)
    jeu.remove(carte)
```

On peut aussi utiliser la fonction `sample(L,`k`)` du module `random` qui a le même effet : choisir k valeurs d'une liste L sans remise.

```
from random import *
main = sample(jeu, 5)
```

3. L'aide nous propose une méthode, ordonner la main obtenue et vérifier si elle est composée de 3 cartes identiques suivies de 2 autres identiques ou inversement :

```
def full(L) :
    L = sorted(L)
    if L[0]==L[1] and L[3]==L[4] and (L[2]==L[1] or L[2]==L[3]):
        return True
    else:
        return False
```

ou plus court comme d'habitude :

```
def full(L) :
    L = sorted(L)
    return L[0]==L[1] and L[3]==L[4] and (L[2]==L[1] or L[2]==L[3])
```

4. Reste à faire la partie simulation, voici le code complet :

```
from random import *

def full(L) :
    L = sorted(L)
    return L[0]==L[1] and L[3]==L[4] and (L[2]==L[1] or L[2]==L[3])

N = 10**5
NbFull = 0
for i in range(N):
    jeu=([str(i) for i in range(2,11)] + ['V','D','R','A']) * 4
    main = []
    for j in range(5):
        carte = choice(jeu)
        main.append(carte)
        jeu.remove(carte)
    if full(main):
        NbFull = NbFull + 1
print("Fréquence d'obtention d'un full", NbFull/N)
```

Vous remarquerez que les valeurs de la fréquence sont assez fluctuantes. On peut augmenter N à 10^6 pour obtenir des résultats plus stables (mais cela risque d'être long selon les performances de votre ordinateur).

5. En suivant l'aide proposée, le nombre de tirages donnant un full de 6 par les 7 est :

$$\underbrace{\binom{4}{3}}_{\text{3 six parmi 4}} \times \underbrace{\binom{4}{2}}_{\text{2 sept parmi 4}} = 24$$

Nombre de tirages donnant un full quelconque :

$$\underbrace{24}_{\text{un full}} \times \underbrace{13}_{\text{1ère carte répétée 3 fois}} \times \underbrace{12}_{\text{2ère carte répétée 2 fois}} = 3744$$

Nombre de mains différentes :

$$\binom{52}{5} = 2598960$$

Puisqu'il y a équiprobabilité, la probabilité cherchée est :

$$p = \frac{3744}{2598960} = \frac{6}{4165} \approx 0,00144$$

AU DÉTOUR DE LA BALADE...

Ici on a choisi d'utiliser les valeurs 'Dame' et 'Roi' pour les figures, obligeant à mettre les nombres sous forme de caractères aussi pour pouvoir ordonner les valeurs. Si vous ne souhaitez pas ajouter cette difficulté, codez les cartes de 1 à 13 (11 pour Valet, 12 pour Dame et 13 pour Roi).

Retour sur l'exercice F17

1. Comme les cartes ne sont pas remises en jeu, il est clair qu'on va finir par en tirer un... Si on n'a vraiment pas de chance, il faudra attendre la 49ème carte pour tirer un valet. Ainsi $\Omega(X) = \{1, ..., 49\}$

2. Voici une proposition de solution utilisant les variables suivantes :

   - On utilise une liste `nb` de 50 compteurs (le premier ne sert à rien).
   - À chaque partie, `Jeu` est une liste initialisée avec les 52 cartes (remarquez que les cartes sont des nombres ou des chaînes de caractères, mais on aurait aussi pu choisir de mettre 11, 12, 13 pour Valet, Dame, Roi).
   - À chaque tirage, si la carte tirée n'est pas un Valet, on la retire de la liste `Jeu` et on incrémente le compteur `nb_cartes`.
   - Lorsque la partie s'arrête, on incrémente alors le compteur d'indice `nb_cartes`.

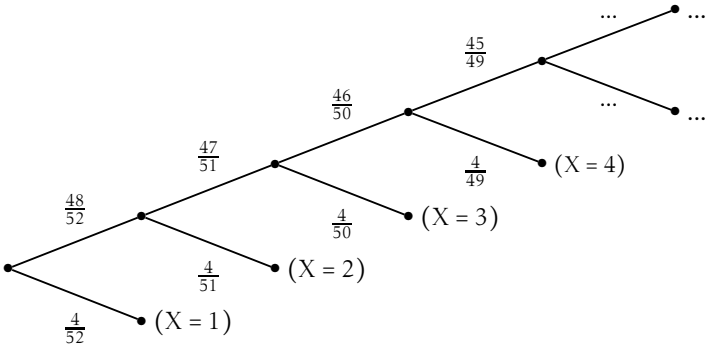
   Reste à diviser les différents résultats obtenus par le nombre de parties en fin de programme pour obtenir les fréquences cherchées :

```
from random import *
N = 10000
nb = [0] * 50
for i in range(N):
    Jeu = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, "V", "D", "R"] * 4
    nb_cartes = 1
    carte = choice(Jeu)
    while carte != "V":
        Jeu.remove(carte)
        carte = choice(Jeu)
        nb_cartes = nb_cartes + 1
    nb[nb_cartes] = nb[nb_cartes] + 1
for i in range(50):
    nb[i] = nb[i] / N
print(nb)
```

3. Regardons les premiers calculs (qu'il faudrait justifier à l'aide des probabilités conditionnelles !)

$$P(X=1)=\frac{4}{52}\ ;\ P(X=2)=\underbrace{\frac{48}{52}}_{\text{perdu}}\times\underbrace{\frac{4}{51}}_{\text{gagné}}\ ;$$

$$P(X=3)=\underbrace{\frac{48}{52}}_{\text{perdu}}\times\underbrace{\frac{47}{51}}_{\text{perdu}}\times\underbrace{\frac{4}{50}}_{\text{gagné}}$$

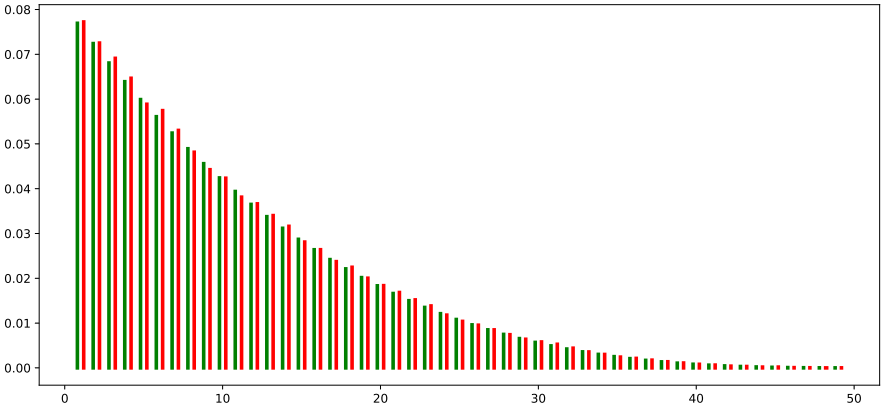


On peut donc de proches en proches calculer ces quantités :

```
def P(k) :
    prob = 1
    perdantes = 48
    for i in range(1, k): # On perd k-1 fois
        prob = prob * perdantes / (perdantes + 4)
        perdantes = perdantes - 1
    prob = prob * 4 / (perdantes + 4)
    return prob
```

On peut alors afficher les diagrammes en barres des fréquences obtenues par simulation (en rouge) et les probabilités théoriques calculées (en vert) :

```
from matplotlib.pyplot import *
for k in range(1, 50):
    plot([k-0.2, k-0.2], [0, P(k)], 'g-', linewidth=3)
    plot([k+0.2, k+0.2], [0, nb[k]], 'r-', linewidth=3)
show()
```



Remarquez ici l'ajout de l'option `linewidth` qui précise l'épaisseur du trait pour obtenir des barres assez larges.

Retour sur l'exercice F18

1.
```
def somme(L, j):
    s = 0
    for i in range(22):
        s = s + L[i+j]
    return s
```

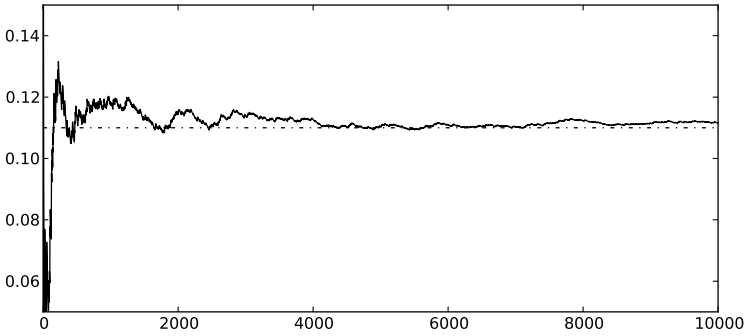
2.
```
from random import randint
L = []
for jour in range(365):
    nba = 0
    for avion in range(20000):
        if randint(1, 500000) == 1:
            nba = nba + 1
    L.append(nba)
```

3.
```
def anneeNoire(L):
    j = 0
    while j < 365 - 21 and somme(L, j) < 5:
        j = j + 1
    if j < 365 - 21:
        return True
    else:
        return False
```

4. Reste à tout assembler. La simulation est très lente du fait du grand nombre de calculs à réaliser :

```
nb = 0
for simulation in range(1, 501):
    L = []
    for jour in range(365):
        nba = 0
        for avion in range(20000):
            if randint(1, 500000) == 1:
                nba = nba + 1
        L.append(nba)

    if anneeNoire(L):
        nb = nb + 1
        print('ANNEE NOIRE ! ', end="")
    print("Fréquences d'année noires : ", end="")
    print(nb, '/', simulation, '=', nb/simulation)
```

5. Sur un grand nombre de simulations, on obtient des valeurs qui se stabilisent autour de 0,11 et cela correspond bien au nombre avancé dans l'article dont est inspiré l'exercice. Finalement, il n'est donc pas exceptionnel que sur une période de 22 jours, on constate 5 accidents d'avions...



AU DÉTOUR DE LA BALADE...

Les plus avertis auront remarqué que le nombre de crashs par jour suit une loi binomiale $\mathcal{B}\left(20000; \frac{1}{500000}\right)$. *On peut simuler le tirage de 365 valeurs suivant cette loi de manière bien plus efficace que ce que nous avons fait :*

```
from scipy import stats
L = stats.binom.rvs(20000, 1/500000, size=365)
```

L'événement étant rare, on pourrait d'ailleurs remplacer cette loi binomiale par une loi de Poisson.

Partie 2

THÈMES D'ÉTUDE

NOMBRES REMARQUABLES

Sur notre chemin, nous rencontrons régulièrement certains nombres se distinguant des autres, jouant un rôle clé, possédant certaines propriétés et qui apparaissent curieusement dans certaines formules.

Partie I - Le nombre $\sqrt{2}$

Très tôt, les mathématiciens ont cherché différents algorithmes pour obtenir des valeurs précises de $\sqrt{2}$. On sait depuis l'époque de Pythagore que le nombre $\sqrt{2}$ est un irrationnel. Cela avait à l'époque d'ailleurs causé la mort des personnes qui avaient découvert cette effroyable nouvelle qui semblait contractuellement impossible. De tels nombres étaient appelés nombres incommensurables (que l'on ne peut mesurer)...

Les algorithmes qui suivent donnent donc des techniques pour approximer le nombre $\sqrt{2}$ à l'aide de fractions. L'usage des fractions n'étant pas présenté dans ce livre, on se contentera de valeurs approchées pour nos programmes.

1. **La méthode du balayage**
   Le principe du balayage est simple. À partir des variations de la fonction carrée et en remarquant que $\sqrt{2}^2 = 2$, on déduit une méthode facile à programmer :

Algorithme

x	1	$\sqrt{2}$	2
$x \mapsto x^2$	1	2	4

- Initialiser x à 1.
- Tant que $x^2 < 2$, ajouter 10^{-6} à x
- Afficher l'arrondi de x à 6 chiffres après la virgule en fin de boucle.

a. Programmer cet algorithme.
b. Combien de fois le programme a-t-il utilisé la boucle TANT QUE pour afficher $1,414214$?

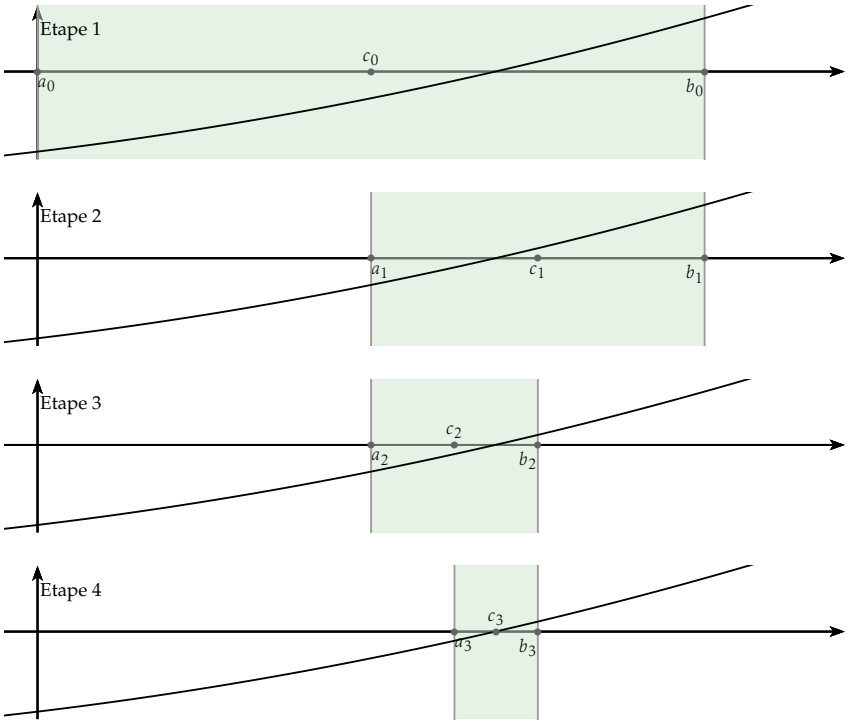
Ainsi on peut affirmer que :

$$1.414213 < \sqrt{2} < 1.414214$$

L'énorme inconvénient de cette méthode est sa lenteur. Pire, si on veut une décimale de plus, il faut multiplier par 10 le temps de calcul... Nous allons donc tenter de trouver d'autres algorithmes pour approcher $\sqrt{2}$ et nous pourrons alors comparer à chaque fois le nombre d'opérations nécessaires pour "faire aussi bien ou mieux que le balayage".

2. **La méthode de dichotomie**

Cet algorithme est au programme des classes de terminale. Il permet de résoudre une équation du type $f(x) = 0$. Voici le principe en image :



Algorithme

Connaissant un encadrement de la solution $\alpha \in [a;b]$:

- on calcule le milieu c du segment $[a;b]$,
- on compare $f(c)$ avec 0 et selon le cas :
  - ◇ on recommence avec l'intervalle $[a;c]$
  - ◇ ou avec l'intervalle $[c;b]$
- On arrête quand la précision est suffisante.

a. Programmer cette méthode avec la fonction f définie sur $\mathbb{R}_+$ par $f(x) = x^2 - 2$. Vous aurez sûrement remarqué qu'elle s'annule en $\alpha = \sqrt{2}$ et on pourra initialiser l'algorithme en constatant que $\alpha \in [1;2]$.

b. Compter le nombre de fois où l'on a exécuté la boucle.

3. **Méthode de Théon de Smyrne**

Voici le principe de cette méthode (que l'on admettra) :

Algorithme

Partant de $p = q = 1$,

- On calcule $p' = p + 2q$ et $q' = p + q$
- Alors $\frac{p'}{q'}$ est une meilleure approximation de $\sqrt{2}$ que $\frac{p}{q}$

a. Programmer cette méthode jusqu'à obtenir une meilleure approximation de $\sqrt{2}$ qu'avec la méthode de balayage.

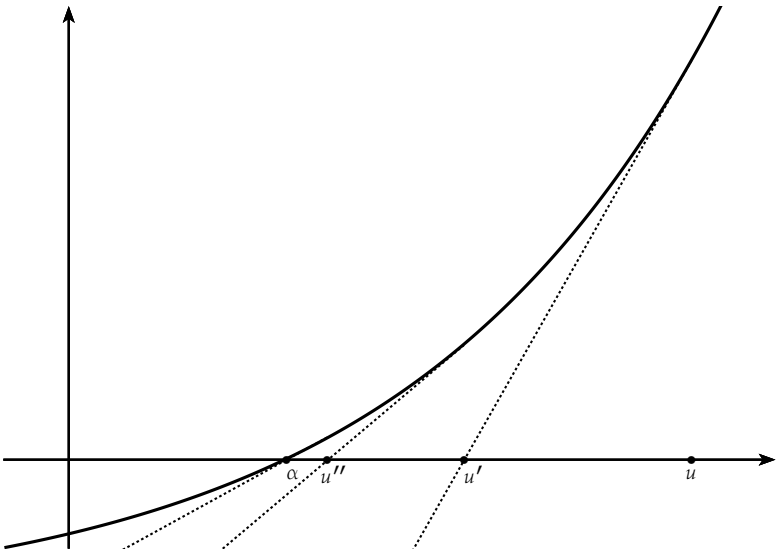
b. Ajouter un compteur pour déterminer le nombre de passages dans la boucle.

4. **Algorithme de Héron**

L'algorithme que Héron avait à l'époque déduit de considérations graphiques est en réalité un cas particulier de la méthode de Newton démontrée bien plus tard.

Algorithme

On cherche à résoudre l'équation $f(x) = 0$. On note α une solution. Connaissant un nombre u "pas trop éloigné" de α, on construit le nombre u' comme étant l'abscisse du point d'intersection de la tangente à la courbe représentant f au point d'abscisse u et de l'axe des abscisses. On montre alors que, sous certaines conditions, u' est une meilleure approximation de α que u.



Appliquons ce raisonnement à la fonction f définie sur $[0;+\infty[$ par $f(x) = x^2 - 2$.

a. Donner l'équation de la tangente en u à la courbe de f.

b. En déduire que $u' = \frac{u}{2} + \frac{1}{u}$.

c. Programmer cette méthode avec toujours le même test d'arrêt.

d. Compter le nombre de passages dans la boucle.

Partie II - Le nombre e

Dans cette partie, on s'intéresse au nombre d'Euler, noté e que l'on retrouve dans la fonction exponentielle.

On considère la suite (u_n) définie pour tout entier naturel n non nul par

$$u_n = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \ldots + \frac{1}{n!} = \sum_{k=0}^{n} \frac{1}{k!}$$

1. a. Vérifier que $u_0 = 1$ et calculer u_1 et u_2.
   b. Écrire un script qui affiche les 100 premières valeurs de u_n.
   c. Quelles conjectures peut-on faire sur les variations et la convergence de la suite (u_n) ?
   d. Montrer la conjecture sur le sens de variation de (u_n).

2. Soit $n \geqslant 1$ fixé. On considère la fonction f_n définie sur $[0;1]$ par

$$f_n(x) = \left(\sum_{k=0}^{n} \frac{x^k}{k!}\right) e^{-x} = \left(1 + \frac{x}{1} + \frac{x^2}{2!} + \cdots + \frac{x^n}{n!}\right) e^{-x}$$

   a. Calculer $f_n(0)$ et vérifier que $f_n(1) = u_n e^{-1}$
   b. Calculer $f_n'(x)$ sur $[0;1]$ et dresser le tableau de variations de f_n.
   c. En déduire que pour tout entier naturel $n \geqslant 1$, $u_n \leqslant e$.
   d. En posant $g_n(x) = f_n(x) + \dfrac{x}{n!}$, démontrer par un raisonnement analogue que $e - \dfrac{e}{n!} \leqslant u_n$.
   e. En déduire que $\forall n > 1$, $e - \dfrac{e}{n!} \leqslant u_n \leqslant e$. Conclure sur la limite de (u_n) et modifier le programme précédent pour obtenir un encadrement de la limite avec une amplitude 10^{-4}.

3. On cherche à présent à obtenir les 100 premières décimales de la limite de cette suite.
   a. Est-ce possible avec le programme tel qu'il est ?
   b. Reprendre les fonctions de l'exercice B10 sur les calculs avec fractions pour obtenir les valeurs de u_n sous forme de fractions irréductibles et ainsi répondre à la question.

Partie III - Le nombre π

Qui n'a jamais rencontré le nombre π ? Qu'on le veuille ou non, c'est un nombre qui apparaît régulièrement au détour de tout chemin que l'on soit mathématicien ou non. Les décimales de π ont été l'objet de recherche de nombreux savants depuis près de 4000 ans. Une des plus anciennes approximations de π se trouve sur le célèbre papyrus Rhind.

1. Une approche par les aires :
   Dans le papyrus Rhind, écrit par le scribe Ahmès vers 1650 avant J.C, on trouve la règle appelée la règle du **neuvième**.

   Règle du neuvième

   Si tu veux calculer une aire circulaire, ôte un neuvième au diamètre, multiplie le résultat par lui-même.

   a. Appliquer cette règle au calcul de l'aire d'un disque de diamètre 9cm. Quelle approximation est alors faite en effectuant ce calcul d'aire ? Donner une valeur approchée au centième de l'erreur commise.

   b. Créer une fonction qui donne en sortie l'aire d'un disque de diamètre fixé avec cette règle.

   c. Créer une fonction qui donne une valeur approchée de l'erreur commise en fonction du diamètre donné en paramètre.

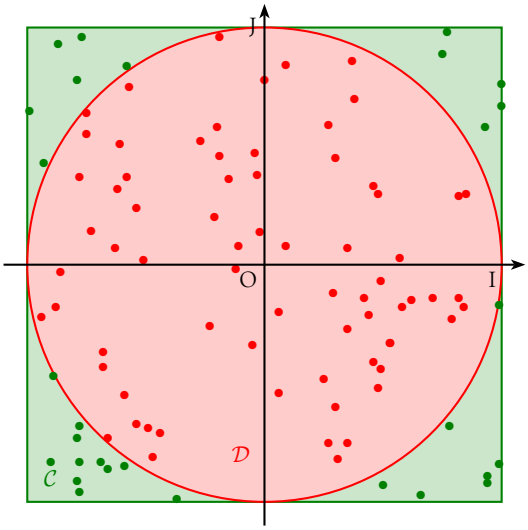
   d. Tester avec d=12cm, d=18cm et d=27cm.

   e. Montrer que l'utilisation de cette règle équivaut à calculer l'aire $A = \frac{256}{81} \times R^2$ où R est le rayon du cercle.

AU DÉTOUR DE LA BALADE...

Cet exercice permet ainsi de comprendre l'affirmation du papyrus Rhind "L'aire du disque de diamètre 9 coudées est celle du carré de 8 coudées de côté". On retrouve une solution approchée du problème de quadrature du cercle.

2. Une approche probabiliste d'estimation : la méthode de Monte Carlo. Cette méthode peut être présentée dès la classe de première : dans un repère orthonormé (O, I, J), on considère le disque $\mathcal{D}$ de centre O et de rayon 1 et $\mathcal{C}$ le carré de centre O et de côté 2.



a. Que vaut le rapport $\dfrac{\text{Aire}_{\mathcal{C}}}{\text{Aire}_{\mathcal{D}}}$?

On place aléatoirement un grand nombre de points à l'intérieur du carré $\mathcal{C}$ et on décide d'estimer ce rapport par le ratio :

$$\frac{\text{Nombre de points de } \mathcal{C}}{\text{Nombre de points de } \mathcal{D}}.$$

b. Écrire un script Python qui tire au hasard 1000 points dont les abscisses et les ordonnées sont dans l'intervalle $[-1, 1]$ et affiche au fur et à mesure les valeurs de ce rapport.

c. Modifier l'algorithme précédent pour obtenir une valeur approchée de π.

AU DÉTOUR DE LA BALADE...

Le site Eduscol (`https://eduscol.education.fr/`) propose dans son document d'accompagnement des programmes de la classe de première une belle présentation de cet algorithme avec des animations !

AUTOUR DES NOMBRES PREMIERS

PREMIERS

On rappelle qu'un entier naturel est dit **premier** s'il possède **exactement deux** diviseurs positifs. Bien que ces nombres soient rencontrés dès le collège, ils constituent encore aujourd'hui l'un des plus mystérieux défis de l'histoire des mathématiques. En effet, on sait qu'il existe une infinité de nombres premiers, mais ils deviennent plus rares au fur et à mesure que le nombre de chiffres nécessaires pour les écrire augmente. Peut-on trouver une formule pour générer ces nombres premiers ? Quel serait l'intérêt d'un algorithme permettant de dire si un nombre très grand est un nombre premier ? Y a-t-il des régularités dans leur apparition ?

Parmi tous ceux qui ont essayé de dompter ces nombres, nous rencontrerons Carl Friedrich Gauss (1777-1855), Pafnouti Tchebychev (1821-1894), Sophie Germain(1776-1831) et Marin Mersenne (1588-1648) au cours de notre promenade historique au pays des nombres premiers.

Partie I - Tests de primalité

On considère la fonction ci-dessous qui, recevant un entier naturel n non nul, renvoie le booléen `True` si n est premier et `False` sinon.

```
FONCTION estPremier(n) :
    d ← 2
    reponse ← VRAI
    TANT QUE d < n FAIRE :
        SI n mod d = 0 ALORS :
            reponse ← FAUX
        FIN SI
        d ← d + 1
    FIN TANT QUE
    RENVOYER reponse
```

1. a. Tester à la main cette fonction pour $n = 5$, $n = 9$ et $n = 1$. Renvoie-t-elle toujours le résultat attendu ? Que faut-il ajouter pour que ce soit le cas ?

   b. Écrire la fonction corrigée en Python.

c. Pour $n \in \mathbb{N}^*$ donné, combien de calculs de restes sont effectués systématiquement ?

2. On se propose d'améliorer l'algorithme précédent en quittant la boucle dès qu'un diviseur est trouvé.

   a. Modifier la fonction en Python.

   b. Que dire du nombre de calculs de restes pour cette nouvelle version ?

3. On cherche à limiter encore le nombre de tours de boucle :

   a. Soient n, p, q des entiers naturels. Montrer que si $n = pq$, alors $p \leq \sqrt{n}$ ou $q \leq \sqrt{n}$.

   b. En déduire une amélioration possible de la fonction.

4. On considère à présent la fonction ci-dessous :

```
FONCTION estPremier(n) :
    p ⟵ 0 ; a ⟵ 2 ; b ⟵ 3 ; k ⟵ 1
    TANT QUE a + p × n ⩽ √n FAIRE :
        SI (n mod a)(n mod b) = 0 ALORS :
            p ⟵ 1
        SINON :
            a ⟵ 6k − 1
            b ⟵ 6k + 1
            k ⟵ k + 1
        FIN SI
    FIN TANT QUE
    RENVOYER (p = 0) ET (n > 1)
```

   a. Justifier que si m est un nombre premier strictement supérieur à 3, alors $m \equiv \pm 1 [6]$

   b. Les affirmations suivantes sont-elles vraies ? (Justifier)

      I. À chaque tour de boucle, les valeurs de a et b sont des nombres premiers.

      II. Dès que p vaut 1, on sort de la boucle.

      III. Si $n \geqslant 2$ et n n'est pas premier alors $p = 1$ à la sortie de boucle.

   c. Combien de calculs de restes, au maximum, sont effectués avec cette version ?

   d. Programmer cette fonction en Python.

Partie II - Répartition des nombres premiers

On sait que l'ensemble des nombres premiers est infini, intéressons-nous à leur répartition. Gauss a travaillé sur cette répartition des nombres premiers. Pour cela, il établit une fonction notée π définie de $\mathbb{N}$ dans $\mathbb{N}$ qui, pour un entier n donné, renvoie le nombre de nombres premiers inférieurs ou égaux à n.

1. **Nombre de nombres premiers**

   a. Que valent $\pi(0)$, $\pi(1)$ et $\pi(10)$?

   b. Programmer cette fonction `pi` qui reçoit un entier naturel n et renvoie la quantité $\pi(n)$.

   c. Tracer la courbe de cette fonction.

   d. On appelle encadrements à la **Tchebychev** un encadrement de la forme
   $$\alpha \frac{n}{\ln n} \leqslant \pi(n) \leqslant \beta \frac{n}{\ln n}$$
   On peut par exemple démontrer qu'à partir d'un certain rang,
   $$\frac{7}{8} \frac{n}{\ln n} \leqslant \pi(n) \leqslant \frac{9}{8} \frac{n}{\ln n}$$
   Tracer les courbes représentant les trois fonctions $n \longmapsto \frac{7}{8} \frac{n}{\ln n}$, $n \longmapsto \pi(n)$ et $n \longmapsto \frac{9}{8} \frac{n}{\ln n}$ pour $2 \leqslant n \leqslant 10000$ afin de visualiser cet encadrement.

2. **Écart entre 2 nombres premiers consécutifs**

   a. Écrire une fonction `suivant` qui reçoit en paramètre un nombre p et renvoie le plus petit nombre premier strictement plus grand que p.

   b. Représenter graphiquement les 1000 premiers écarts entre nombres premiers consécutifs.

   c. Pourquoi tous les écarts (sauf le premier) sont-ils pairs ?

On remarque qu'il y a quand même "beaucoup" de nombres premiers espacés seulement de 2 au final.

Partie III - Quelques nombres premiers célèbres

Les mathématiciens recherchent depuis toujours une formule qui à tout entier n associerait le n-ième nombre premier. Plusieurs d'entre eux ont malgré tout trouvé des procédures qui permettent de "fabriquer" des nombres premiers.

1. **Les nombres de Sophie Germain**
   Sophie Germain travaille sur le théorème de Fermat et à cette occasion étudie des nombres premiers particuliers.

   > Un nombre premier de **Sophie Germain** est un nombre premier p tel que $2p + 1$ soit aussi un nombre premier.

   Par exemple : 5 est un nombre premier de Sophie Germain car 5 est premier et $2 \times 5 + 1 = 11$ est aussi un nombre premier.

   On conjecture qu'il existe une infinité de nombres premiers de Sophie Germain. À l'heure actuelle cette question reste ouverte. En mars 2010 le plus grand connu est $183027 \times 2^{265440} + 1$. Il a 79911 chiffres.

   a. Écrire une fonction `germain(p)` qui reçoit un entier `p` et renvoie un booléen indiquant si ce nombre est un nombre premier de Sophie Germain.

   b. Il existe 37 nombres premiers de Sophie Germain inférieurs à 1000, les trouver.

   Sophie Germain énonce vers 1825 que le théorème de Fermat est vérifié pour les nombres premiers de Sophie Germain.

2. **Les nombres de Mersenne**

   > Si p est un entier supérieur ou égal à 2, on appelle nombre de Mersenne le nombre $M_p = 2^p - 1$.

   Mersenne affirme que pour $p \leqslant 257$, il existe 11 nombres de Mersenne M_p qui sont premiers. Euler réussit à prouver que $2^{31} - 1$ est bien premier. En 1947, la liste est complétée avec trois nouvelles valeurs de p.

a. Déterminer les 8 premiers nombres de Mersenne à l'aide d'une fonction Python.

b. Pour $x \in \mathbb{R}$ et $n \in \mathbb{N}^*$, développer l'expression

$$(x-1)\left(1+x+x^2+\cdots+x^{n-1}\right) = (x-1)\sum_{k=0}^{n-1} x^k$$

c. En déduire que si M_p est premier alors p est premier.

d. La réciproque est-elle vraie ?

e. Que constate-on si on cherche à trouver les 3 nombres de Mersenne manquant sur les 11 annoncés ?

AU DÉTOUR DE LA BALADE...

Les nombres de Mersenne permettent d'obtenir des nombres premiers gigantesques. On utilise souvent pour savoir s'ils sont premiers le test de primalité de Lucas-Lehmer.
Au 10 juin 2009, 47 nombres premiers de Mersenne sont connus.

BEAU

Nous vous proposons maintenant une petite pause pour admirer les beaux paysages de notre promenade mathématique.

Partie I - L'ASCII-Art

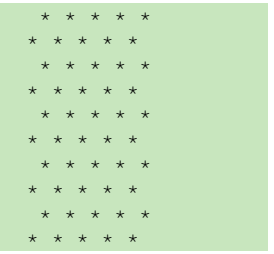
L'art ASCII consiste à réaliser des images uniquement à l'aide de lettres et caractères spéciaux contenus dans le code ASCII.

Source : https://www.asciiart.eu/

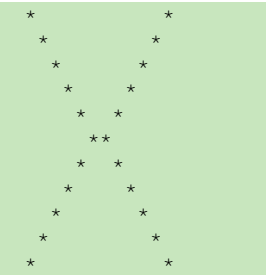
Voici quelques motifs à réaliser :

Motif 1 :

Motif 2 :



Motif 3 :



1. Réaliser ces motifs à l'aide de boucles.

2. Une fois les programmes créés, les rendre les plus courts possibles.

Partie II - Avec la tortue

Tableaux chaotiques

Chaotiques et pourtant totalement déterministes, ces 4 figures semblent de très beaux gribouillages pleinement aléatoires. Ce n'est pas le cas, ils sont le fruit de l'algorithme ci-dessous :

```
x est un réel
POUR c PARCOURANT LES DÉCIMALES DE x FAIRE :
    | AVANCER DE 10 PAS
    | TOURNER A GAUCHE DE c × 10 DEGRES.
```

Tableau 1

Tableau 2

Tableau 3

Tableau 4

Ces 4 tableaux ont été réalisés avec les 2000 décimales des nombres e, π, $\frac{22}{7}$ et $\sqrt{2}$. Retrouver à quel nombre correspond chaque tableau. Pensez à utiliser le fichier NOMBRES en Python qui contient les premières décimales de ces 4 nombres sous forme d'une chaîne de caractères.

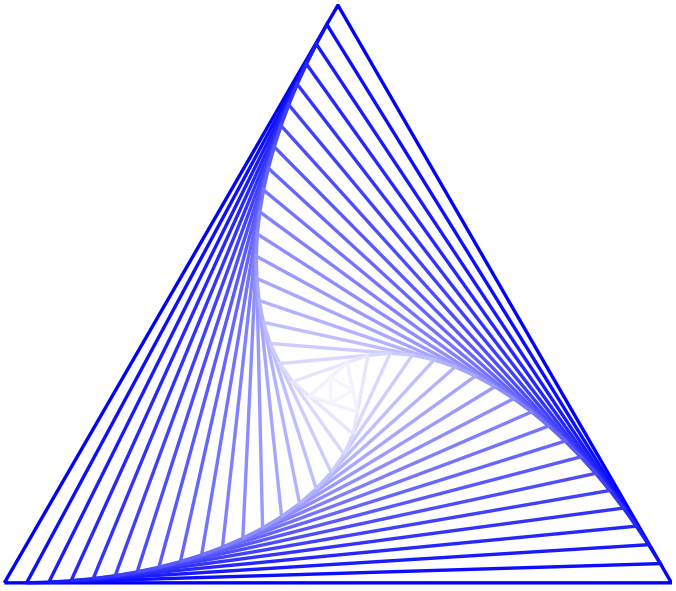
Motifs triangulaires

La question suivante utilise les relations métriques dans le triangle. On aura ainsi recours au **théorème d'Al-Kashi** :

Soit ABC un triangle.

$$BC^2 = AB^2 + AC^2 - 2 \times AB \times AC \times \cos\left(\widehat{BAC}\right)$$

On cherche à réaliser cette figure :

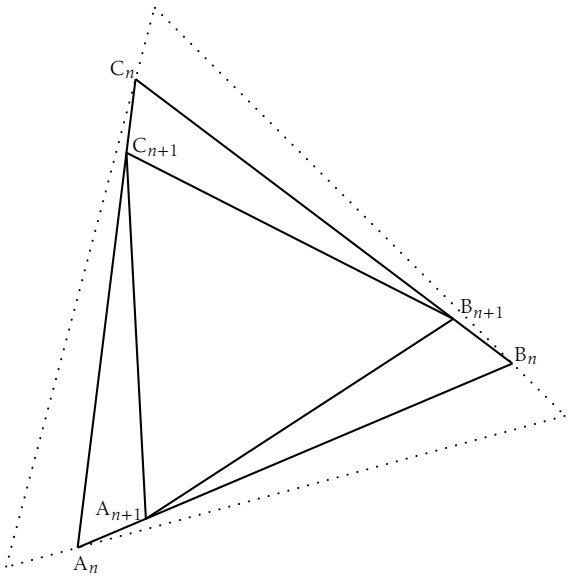


L'algorithme suivant permet de l'obtenir :

Algorithme

- On trace un triangle équilatéral $A_0B_0C_0$ de 300 unités de côté.
- Pour une valeur de n donnée, ayant tracé le triangle $A_nB_nC_n$, on trace le triangle $A_{n+1}B_{n+1}C_{n+1}$ ainsi :
  - $A_{n+1} \in [A_nB_n]$, $B_{n+1} \in [B_nC_n]$, et $C_{n+1} \in [C_nA_n]$
  - $A_nA_{n+1} = B_nB_{n+1} = C_nC_{n+1} = 10$
- On arrête le tracé lorsque le triangle obtenu à un côté mesurant moins de 10 unités.

Le passage de l'étape n à l'étape $n+1$ est illustré par la figure ci dessous. On admet que, par symétrie, on obtient un triangle équilatéral à chaque étape. On note ℓ_n la longueur d'un côté à l'étape n.



1. Calculer ℓ_{n+1} en fonction de ℓ_n pour un entier naturel n.

2. On note $\alpha_n = B_n\widehat{A_{n+1}B_{n+1}}$. Montrer que $\cos(\alpha_n) = \dfrac{l_n - 15}{l_{n+1}}$

3. Réaliser alors un script permettant à la tortue de dessiner cette figure.

Partie III - Les tables modulaires

Voici un algorithme de représentation des tables de multiplication dans des ensembles peu ordinaires.

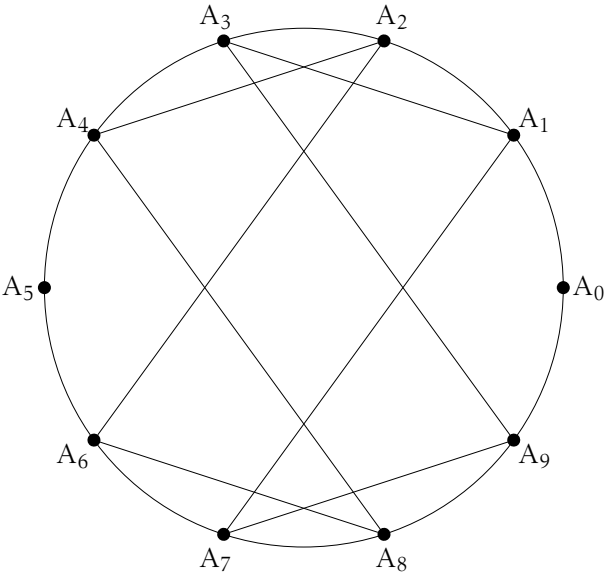
Soient n et p deux entiers naturels supérieurs à 3.

Dessin de la table de p module n

- $A_0A_1...A_{n+1}$ est un polygone régulier à n côtés ;
- Tracer le cercle circonscrit à ce polygone ;
- Pour toute valeur de $k \in [\![0;n[\![$:
  - Calculer k', le reste de la division de kp par n.
  - Tracer le segment $[A_kA_{k'}]$

Un premier exemple : la table des 7 modulo 10 :

- $0 \times 7 = 0 \equiv 0[10]$
- $1 \times 7 = 7 \equiv 7[10]$
- $2 \times 7 = 14 \equiv 4[10]$
- $3 \times 7 = 21 \equiv 1[10]$
- $4 \times 7 = 28 \equiv 8[10]$
- $5 \times 7 = 35 \equiv 5[10]$
- $6 \times 7 = 42 \equiv 2[10]$
- $7 \times 7 = 49 \equiv 9[10]$
- $8 \times 7 = 56 \equiv 6[10]$
- $9 \times 7 = 63 \equiv 3[10]$



En vous aidant de la fiche sur les graphiques (application 30) :

1. Dessiner le cercle de centre O et de rayon 1.
2. Écrire une fonction `table(p,n)` qui représente graphiquement la table de p modulo n.

Il y aurait de nombreuses choses à dire sur ces graphiques... mais la plupart des preuves sont difficilement accessibles même avec la spécialité mathématique. Cet exercice sera approfondi dans un prochain tome consacré à l'enseignement supérieur.

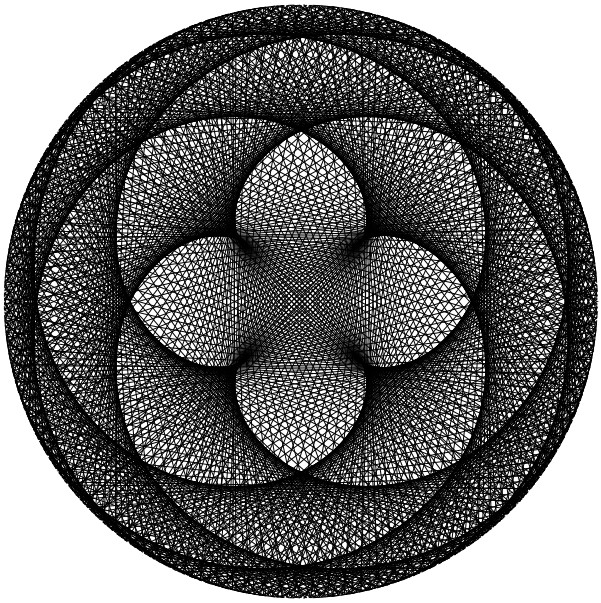


Table de 399 modulo 993

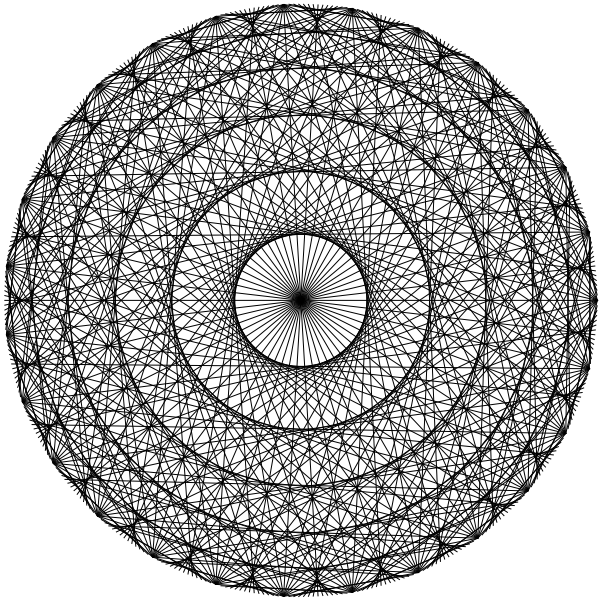


Table de 28 modulo 378

CRYPTOGRAPHIE

CRYPTO

Pour finir notre balade, nous vous proposons un thème incontournable quand on aime "jouer" avec les mathématiques : la cryptographie qui n'est autre que l'art des messages secrets. On ne considère ici que des chaînes de caractères constituées de lettres minuscules (non accentuées) et d'espaces.

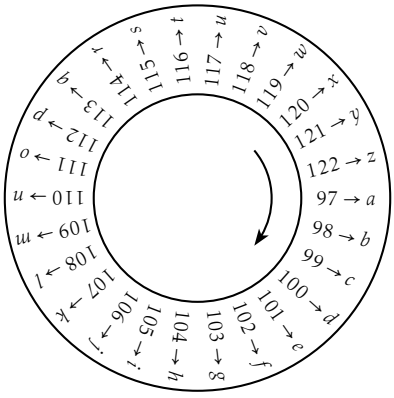
Partie I - Le codage Jules-César

Comme nous l'avons vu précédemment, en informatique chaque caractère est associé à un nombre entier grâce à la table **ASCII**. Par exemple le caractère `'a'` correspond au code 97, `'b'` à 98...

En Python, la fonction `ord(c)` renvoie le code ASCII du caractère `c` et de même la fonction `chr(x)` renvoie le caractère associé à un entier `x`. Par exemple :

```
>>> ord('f')
102
>>> chr(100)
'd'
```

Le codage de César est la méthode de cryptage d'un texte la plus rudimentaire que l'on puisse imaginer. Il a été utilisé par Jules César (mais il existait auparavant) pour certaines de ses correspondances. Le principe est de décaler les lettres de l'alphabet vers la droite de une ou plusieurs positions. Par exemple, en décalant les lettres de une position, le caractère `a` se transforme en `b`, le `b` en `c`, ..., le `z` en `a`.



Le texte `'ave cesar'` devient ainsi `'bwf dftbs'`. Enfin, si le décalage est négatif, on décale les lettres vers la gauche (par exemple, avec un décalage de -1, `'a'` devient `'z'`, `'b'` devient `'a'` ,...)

1. Que devient le texte `'ave cesar'` avec un décalage de -2 ?
2. Écrire une fonction `decalage` qui, recevant une lettre `l` et un entier `n` (positif ou négatif), renvoie la lettre obtenue en décalant `l` de `n` positions.

```
>>> decalage('a', 2)
'c'
>>> decalage('a', -1)
'z'
```

3. Écrire une fonction `codage_cesar` qui, recevant une chaîne de caractères `texte` et un entier `n` (positif ou négatif), renvoie la chaîne obtenue en décalant chaque lettre de `texte` de `n` positions.

```
>>> codage_cesar('ave cesar', 2)
'cxg eguct'
```

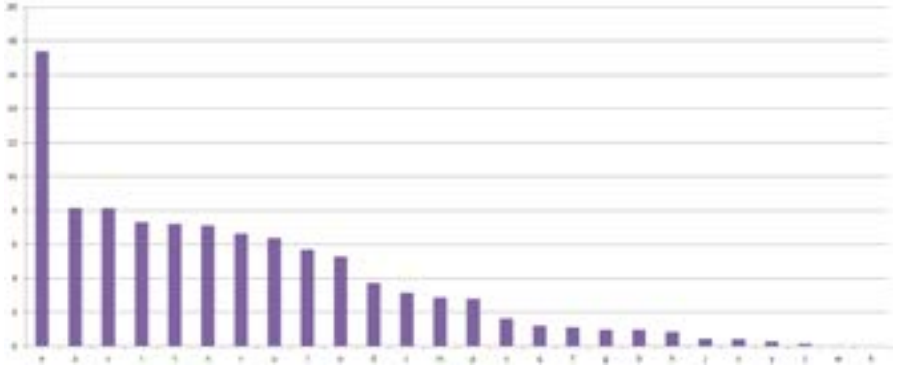
4. Écrire une fonction `decodage_cesar` qui, recevant une chaîne de caractères `texte` et un entier `n` (positif ou négatif), décode une chaîne codée par la fonction précédente.

```
>>> decodage_cesar('cxg eguct', 2)
'ave cesar'
```

Partie II - La cryptanalyse pour décoder...

Pour réaliser le décodage précédent, il faut connaître la valeur du décalage. Une manière de la déterminer automatiquement est d'essayer de deviner cette valeur. La branche des mathématiques qui s'intéresse au décodage de messages secrets s'appelle la *cryptanalyse*. L'approche la plus couramment employée est de regarder la fréquence d'apparition de chaque lettre dans le texte crypté. Ainsi la lettre la plus fréquente dans un texte suffisamment long en français est la lettre `e`.

Distribution des lettres (%) dans un texte en français.



1. Écrire une fonction `compter` qui, recevant une chaîne de caractères `texte` et une lettre `l`, renvoie le nombre d'occurrences de `l` dans `texte`.

```
>>> compter('a', 'ave cesar')
2
```

2. Écrire une fonction `lettre_frequence_max` qui, recevant une chaîne de caractères `texte`, renvoie la lettre la plus fréquente de `texte`.

```
>>> lettre_frequence_max('ave cesar')
'a'
```

3. Écrire une fonction `decodage_cesar_auto` qui, recevant une chaîne de caractères `texte`, la décode automatiquement.

```
>>> texte = 'ytzyj qf lfzqj jxy inanxjj js ywtnx ufwynjx itsy q ↙
    ↳ zsj jxy mfgnyjj ufw qjx gjqljx q
fzywj ufw qjx fvznyfnsx qf ywtnxnjrj ufw hjzc vzn ifsx qjzw ↙
    ↳ qfslzj xj strrjsy hjqyjx jy ifsx qf
stywj lfzqtnx'
>>> decodage_cesar_auto(texte)
'toute la gaule est divisee en trois parties dont l une est ↙
    ↳ habitee par les belges l autre par les
aquitains la troisieme par ceux qui dans leur langue se nomment ↙
    ↳ celtes et dans la notre gaulois'
```

Partie III - Chiffre de Vigenère

L'objectif de ce système est de contrer la faille du système précédent, à savoir que chaque lettre est toujours codée par le même symbole. Le principe est le suivant : on n'utilise cette fois plus une constante pour décaler les lettres, mais un mot.

Image wikipédia

On décale alors la première lettre du message d'un nombre de rangs correspondant à la position dans l'alphabet de la première lettre de la clé. Sur le même principe, on décale la deuxième lettre du message en utilisant la deuxième lettre du mot clé et ainsi de suite. Lorsque l'on est arrivé à la dernière lettre de la clé, on reprend ensuite à la première.
L'exemple ci-contre illustre le codage de la phrase "*il est mechant monsieur brochant*" avec la clé "*diner*".

- 1 : La lettre à coder
- 2 : Sa position dans l'alphabet
- 3 : La lettre de la clé
- 4 : Sa position dans l'alphabet
- 5 : La somme des 2 positions
- 6 : La somme corrigée
- 7 : La lettre codée

1	2	3	4	5	6	7
i	9	d	4	13	13	m
l	12	i	9	21	21	u
	0	n	14	14	14	n
e	5	e	5	10	10	j
s	19	r	18	37	10	j
t	20	d	4	24	24	x
	0	i	9	9	9	i
m	13	n	14	27	0	
e	5	e	5	10	10	j
c	3	r	18	21	21	u
h	8	d	4	12	12	l
a	1	i	9	10	10	j
n	14	n	14	28	1	a
t	20	e	5	25	25	y
	0	r	18	18	18	r
m	13	d	4	17	17	q
o	15	i	9	24	24	x
n	14	n	14	28	1	a
s	19	e	5	24	24	x
i	9	r	18	27	0	
e	5	d	4	9	9	i
u	21	i	9	30	3	c
r	18	n	14	32	5	e
	0	e	5	5	5	e
b	2	r	18	20	20	t
r	18	d	4	22	22	v
o	15	i	9	24	24	x
c	3	n	14	17	17	q
h	8	e	5	13	13	m
a	1	r	18	19	19	s
n	14	d	4	18	18	r
t	20	i	9	29	2	b

La somme est corrigée si elle dépasse 26, ce qui signifie dans ce cas que l'on doit refaire un tour d'alphabet. On prend comme convention que l'espace est en position 0 dans l'alphabet.

La phrase codée est "*munjjxi juljayrqxax iceetvxqmsrb*".

1. Écrire une fonction `position` qui renvoie la position d'une lettre minuscule dans l'alphabet et 0 pour les autres caractères.
2. Écrire une fonction `lettre` qui a l'effet inverse : pour un entier entre 0 et 26, elle renvoie la lettre correspondante.
3. Écrire une fonction `codeV` qui reçoit un message clair et une clé et renvoie le message codé.
4. Écrire une fonction `decodeV` qui a l'effet inverse : recevant un message codé et une clé, elle renvoie le message clair.

Partie 3

FICHES DE COURS

La console Python

Il existe une multitude de solutions pour coder en Python, certaines sont plus ou moins simples à mettre en place sur la machine. Dans ce livre, nous avons fait le choix d'Edupython, une distribution libre et gratuite, facilement téléchargeable sur le site

`https://edupython.tuxfamily.org/`

Pour coder en Python, il faut un **interpréteur Python** qui se chargera de lire et exécuter les lignes de vos codes et d'un éditeur de texte appelé **IDE** (**E**nvironnement de **D**éveloppement **I**ntégré). Edupython offre l'avantage de regrouper cela avec une distribution portable et utilisable sans privilège sur un réseau.

Malheureusement, EduPython qui utilise l'excellent IDE PyScripter ne fonctionne que sous windows. Si vous utilisez un autre système d'exploitation ou si la distribution Edupython ne vous convient pas, d'autres solutions existent :

- **Thonny** : basé sur le même principe qu'Edupython mais avec un IDE plus minimaliste.
- **Pyzo** : multi-plateforme, mais vous devrez installer séparément l'interpréteur Python. L'IDE est assez rudimentaire.
- **Spyder** : une bonne alternative multi-plateforme.

D'autres solutions en lignes peuvent aussi s'avérer bien utiles, en tapant dans votre moteur de recherche préféré `"Python en ligne"` vous trouverez de nombreux sites contenant plus ou moins de publicité !

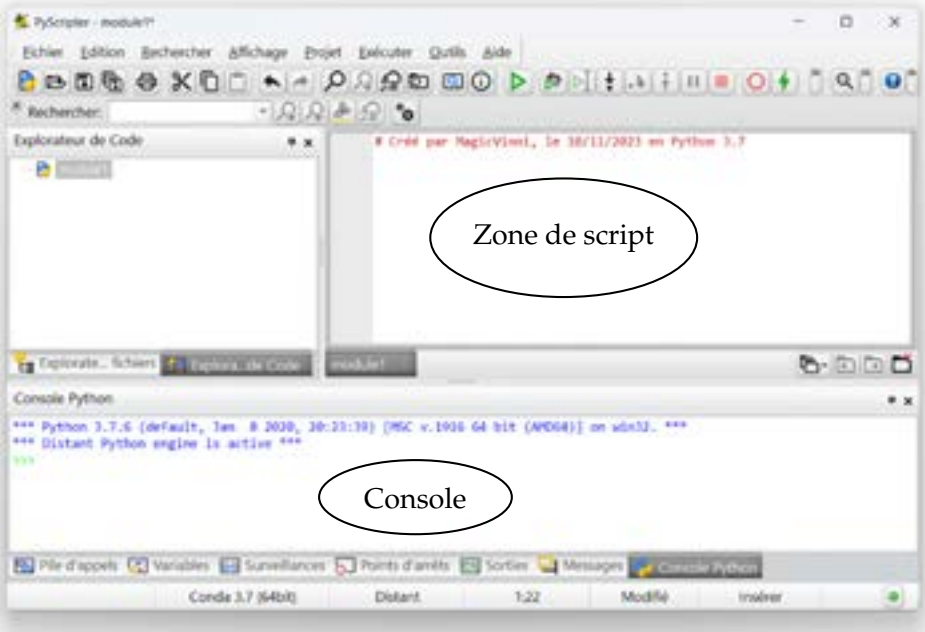
La console d'Edupython

Dans EduPython, le logiciel PyScripter fait office d'éditeur. Si vous utilisez d'autres distributions qu'Edupython, les autres éditeurs ont un fonctionnement similaire.

L'interface est découpée en plusieurs zones bien distinctes :

- La zone de script permet de saisir des programmes plus longs et de les exécuter ensuite.

- La console permet d'exécuter en direct des commandes Python, c'est aussi dans cette console que s'afficheront les résultats de vos programmes ou que nous appellerons les fonctions pour les tester.



Dans la zone console de Python, on peut donc saisir des instructions qui seront exécutées immédiatement. Cette saisie se fait après les >>> que l'on appelle des **chevrons**.

Ainsi, on peut déjà, dans un premier temps, utiliser cette console comme une calculatrice. Voici quelques opérations possibles :

Opérateur	Effet
`+`, `-`, `*`	Respectivement la somme, la différence et le produit
a `/` b	Quotient décimal dans la division de a par b
a `//` b	Quotient entier dans la division de a par b
a `%` b	Reste entier dans la division de a par b
a `**` b	Résultat de a^b. *Attention, le symbole ^ en Python a une autre signification.*

Les quelques exemples qui suivent vont vous permettre de prendre la console Python en main en jouant avec les nombres. Vous pouvez utiliser les touches [↑] et [↓] pour vous déplacer dans l'historique de ce que vous avez déjà tapé au fur et à mesure.

Application 1 *Écrire deux séquences de calcul (vraiment) différentes pour afficher* `35` *en utilisant exactement 5 fois la touche* [5] *et autant que nécessaire les touches* [+], [-], [*], [/], [%], [(] *et* [)].

Quelques remarques supplémentaires :

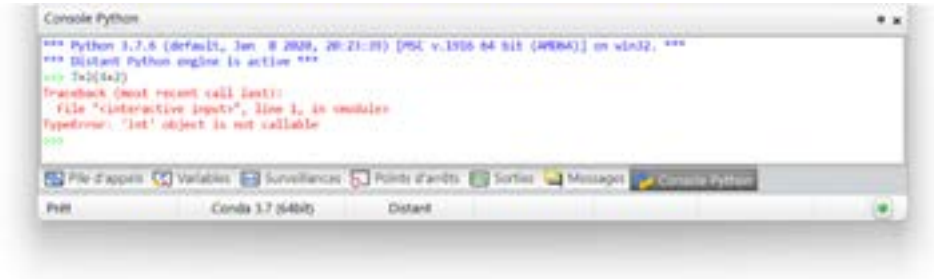
- Le quotient q et le reste r de la division entière sont définis en Python grâce au théorème suivant :

> **SI** a et b sont deux nombres entiers relatifs avec b non nul
> **ALORS** il existe un unique couple d'entiers (q, r) vérifiant :
>
> $$\begin{cases} a = b \times q + r \\ |r| < |b| \\ r \text{ et } b \text{ sont de même signe} \end{cases}$$

Attention, ce n'est pas la définition habituelle utilisée en cours de mathématiques (pour les nombres négatifs).

- Le calcul de `a ** b` (c'est à dire a^b) peut être effectué même lorsque b n'est pas entier. Dans ce cas, si $a > 0$ et $b \in \mathbb{R}$, Python utilise la formule vue en terminale : $a^b = e^{b \ln a}$ (si $a < 0$, c'est abordable dans l'ensemble des nombres complexes mais ne figure pas dans les programmes du lycée).

- Le calcul suivant : `7+2(4+2)` n'a pas de sens : Python n'interprète pas les multiplications implicites, il faut donc taper `7+2*(4+2)`.



Si vous connaissez les puissances d'exposant réel, vous pouvez faire l'exercice qui suit :

Application 2 *Qu'affichent les calculs suivants ?*

A. `16 ** 0.5`
B. `16 ** 0.25`
C. `27 ** 1 / 3`
D. `27 ** (1 / 3)`

Comme en mathématiques, Python respecte certaines priorités opératoires. En informatique, ce concept porte le nom de **précédence** des opérateurs. Dans le cas du langage Python, les précédences sont définies ainsi (par ordre croissant de priorité) :

+, -	Addition et soustraction
*, /, //, %	Multiplication, division, quotient et reste
$+x$, $-x$	Positif, négatif
**	Exponentiation

Application 3 *Qu'affichent les calculs suivants ? Après avoir cherché les réponses, ne pas hésiter à tester avec la console pour vérifier.*

A. `7 // 2`	E. `-14 // (-3)`	I. `1 - 5 * 2`
B. `4 % 2`	F. `5 ** 0 + 7`	J. `5 - 3 / - 2`
C. `3 ** 3`	G. `2 ** - 2`	K. `- 13 % 3`
D. `-5 // 2`	H. `-2 ** 10`	L. `13 % - 3`

Variables et affectations

L'avantage d'utiliser un langage de programmation pour effectuer des calculs réside dans le fait de pouvoir stocker les informations en mémoire. On peut, lorsque l'on réalise un programme ou un algorithme, utiliser des **variables** pour stocker les valeurs. Le fait de donner un nom à un calcul, un texte ou un autre objet s'appelle **l'affectation** .

En Algorithmique	On lit ...	En Python
$a \leftarrow 3$	a reçoit la valeur 3	`a = 3`

D'ailleurs en Python, les variables ne varient pas réellement, il ne s'agit que de noms donnés à des quantités.

Dans la console Python, lorsque l'on affecte une valeur à une variable, le contenu de celle-ci n'est pas affiché. Il faut taper le nom de la variable pour afficher sa valeur.

Au moment d'une affectation, la quantité à droite du signe = est évaluée et stockée dans la variable de gauche. L'écriture `a = a + 1` a donc un sens en informatique . Cela signifie simplement que la variable `a` a augmenté de 1.

Le symbole = joue donc un rôle d'affectation et non d'égalité. D'ailleurs, l'égalité `b = a * 2` n'est plus vérifiée à la fin de notre exemple.

```
>>> a = 3
>>> b = a * 2
>>> b
6
>>> a + 1
4
>>> a
3
>>> a = a + 1
>>> a
4
>>> b
6
```

Affectations simultanées

Application 4 *On considère la suite d'instructions ci-dessous*

```
>>> x = 17
>>> y = 32
>>> x = x + y
>>> y = x - y
>>> x = x - y
```

1. *Que contiennent les variables x et y à la fin ?*
2. *Constate-t-on le même phénomène pour des valeurs quelconques de x et y ? (le démontrer)*
3. *Écrire une suite d'instructions produisant le même effet, mais sans effectuer de calculs (on pourra utiliser une troisième variable temporaire).*

Python propose une manière simple d'affecter simultanément des variables en utilisant une virgule, les deux premières lignes de l'exercice précédent peuvent être réduites en **`x, y = 17, 32`**.

Comme pour l'affectation classique, lorsque l'on procède à une affectation simultanée, les quantités de droite sont évaluées dans un premier temps, puis dans un second temps, elles sont stockées dans les variables de gauche. Ainsi les 3 dernières lignes de l'exercice précédent peuvent tout simplement être transformées en : **`x, y = y, x`**.

Application 5 *Que contiennent les variables à la fin des programmes suivants ?*

1.
```
>>> a, b = 3, 2
>>> a = a - b
>>> b, a = a + 1, 2*b
>>> b = b % 3
>>> a = a - b // 2
```

2.
```
>>> x, y, z = 1, 2, 3
>>> x, y = y, z
>>> y, z = z, x
>>> y = z
>>> z = y
```

Si la console a l'avantage d'afficher les résultats en direct, le gros inconvénient est que l'on ne peut pas sauvegarder le travail. Il faut alors tout retaper pour le modifier d'où l'intérêt d'enregistrer nos codes dans un fichier réutilisable

Python, un langage interprété

Python offre la possibilité d'exécuter un ensemble de lignes que l'on appelle **script**. Il s'agit tout simplement d'un fichier texte dont **l'extension** est `.py` et qui contient la liste des instructions à effectuer.

On peut éditer un script Python avec n'importe quel éditeur comme le bloc-note de Windows mais l'éditeur **PyScripter** présent dans EduPython présente de nombreux avantages comme la colorisation automatique des instructions pour une meilleure lecture, l'exécution du script ou d'une partie de celui-ci en un clic et bien d'autres spécificités.

Contrairement à d'autres langages comme le C, le langage Python ne compile pas les codes sources pour en faire des exécutables autonomes, mais exécute le script qui lui est fourni à l'aide d'un **interpréteur Python**. Cela présente l'avantage d'être plus rapide pour lancer les programmes (pas de phase de compilation) mais l'inconvénient majeur est la nécessité d'avoir Python sur la machine avec laquelle vous voulez exécuter le code.

Créer, enregistrer, exécuter

Créer : pour créer un nouveau script (on parle aussi parfois de module), il suffit de cliquer sur l'icône  ou d'utiliser le raccourci clavier CTRL + N .

Enregistrer : pour enregistrer un script, on peut utiliser le menu : `Fichier` → `Enregistrer` ou l'icône représentant une disquette datant de l'époque paléolithique  ou encore le raccourci clavier usuel CTRL + S . Vous pourrez alors recharger ce fichier ultérieurement.

À noter qu'après le premier enregistrement, il n'est plus utile d'enregistrer les modifications : à chaque exécution, l'IDE Pyscripter enregistre le code avant de l'exécuter.

Exécuter : pour exécuter le script actif, on clique sur la flèche verte ou on utilise le raccourci clavier CTRL + F9 .

Commenter, afficher

En créant un script, on va créer des programmes contenant un grand nombre de lignes. Pour s'y retrouver et pour en permettre la compréhension à une tierce personne, on pourra insérer des commentaires dans le programme. Ces lignes de commentaires commencent par un **croisillon** # et ne sont donc ni lues, ni interprétées par la machine.

Si on tape ce premier script et qu'on l'exécute...

En mode console :

```
>>> cote = 10
>>> aire = cote * cote
>>> aire
100
>>>
```

Dans la fenêtre de script :

```
# Mon premier programme !
cote = 10
aire = cote * cote
aire
```

... Il ne se passe rien... Dans un script Python, si on veut afficher quelque chose dans la console, on devra utiliser la fonction **`print`**.

Le précédent script pourrait être modifié ainsi pour obtenir l'affichage souhaité :

```
# Mon premier programme !
cote = 10
aire = cote * cote
print(aire)
```

`print(`v`)` : Affiche le contenu de la variable v

- Si l'on souhaite afficher plusieurs quantités, on pourra les séparer par des virgules, par exemple :

```
# Mon premier programme !
cote = 10
aire = cote * cote
print(cote, aire)
```

```
10 100
```

- On peut aussi améliorer l'affichage en mixant l'affichage de variables et celui de textes (placés entre guillemets), par exemple :

```
# Mon premier programme !
cote = 10
aire = cote * cote
print("La surface d'un carré de côté", cote, "vaut", aire)
```

```
La surface d'un carré de côté 10 vaut 100
```

Bien évidemment si vous changez la valeur de `cote` en première ligne, l'affichage de sortie est modifié en conséquence.

- Par défaut lorsque l'on place des quantités à afficher séparées par des virgules dans la fonction **`print`**, un espace joue le rôle de séparateur de données. On peut changer ce séparateur selon les besoins en précisant **`sep = ...`**, par exemple :

```
a = 3
print(a, a ** 2, a ** 3, sep="/")
```

```
3/9/27
```

- Enfin, le moteur Python va à la ligne à la fin de chaque instruction **`print`** rencontrée. On peut modifier cela en ajoutant **`end=""`** pour modifier le caractère de fin de ligne (retour chariot par défaut).

```
print("Pas de retour", end="")
print("à la ligne.")
```

```
Pas de retour à la ligne.
```

Déclaration d'une fonction

Dans un premier temps, une fonction informatique peut être vue comme en mathématiques, c'est à dire un processus qui reçoit des valeurs et en renvoie d'autres.

Lors de la déclaration d'une fonction, les valeurs d'entrée sont appelées **paramètres** de la fonction. Par exemple, on peut définir la fonction `aire_carre` qui reçoit un paramètre `c` et qui renvoie la valeur de l'aire d'un carré de côté `c` :

```
def aire_carre(c):
        return c ** 2
```

Une fois ce script exécuté, on peut alors appeler la fonction dans la console en lui donnant des **arguments** :

```
>>> aire_carre(5)
25
```

zone de script – exécution – dans la console

```
def aire_carre(c):
        return c ** 2
```

```
>>> aire_carre(5)
25
```

La déclaration d'une fonction est ainsi structurée :

- La déclaration commence par le mot clef **`def`** suivi du nom de la fonction (ici `aire_carre`), puis la liste des paramètres (un seul ici `c`) entre parenthèses et enfin on termine par deux points pour commencer la déclaration.
- L'intégralité du code de la fonction (ici juste une ligne) est décalée (on appelle cela l'**indentation**) pour indiquer à Python la fin de la déclaration de la fonction.
- La fonction se termine bien souvent par un **return** qui indique la valeur à renvoyer. Dès que l'interpréteur Python rencontre un `return`, il quitte la fonction et renvoie la valeur indiquée.

Enfin, contrairement à `print`, l'instruction `return` n'est pas une fonction et ne nécessite donc pas de parenthèses !

Quelques remarques supplémentaires sur les fonctions :

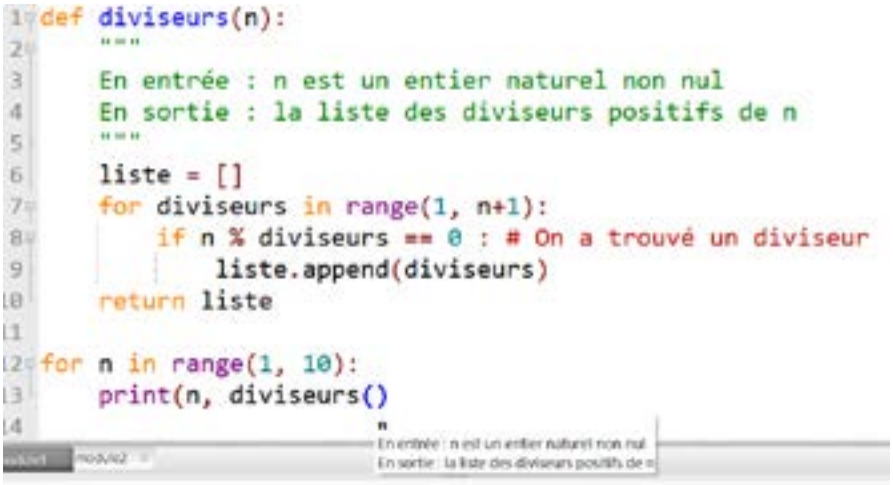
- Comme pour les variables, le nom de la fonction peut comporter plusieurs lettres suivies éventuellement de chiffres. Si vous définissez une fonction dont le nom existe déjà, c'est la dernière définition qui est prise en compte.
- Comme en mathématiques, une fonction peut faire intervenir plusieurs paramètres, il suffit alors de les lister en les séparant par une virgule.

```
def aire_rectangle(L, l) :
        return L * l
```

- On peut aussi avoir plusieurs valeurs en sortie, il suffit alors de les séparer par des virgules au moment du `return` :

```
def aire_perimetre_rectangle(L, l) :
        return L * l, 2 * (L+l)
```

Pour finir, on place souvent juste en dessous de la définition, un texte placé entre des triples guillemets qui donne la description de la fonction. Ce texte appelé **docstring** est facultatif. S'il est renseigné, il s'affiche en info-bulle lorsque l'on tape le nom de la fonction :



Pour un gain de place, nous n'écrirons pas systématiquement le docstring des fonctions dans ce livre, mais nous vous encourageons à le faire et spécifier pour chaque fonction ce qu'elle reçoit en entrée et ce qu'elle renvoie en sortie.

Quelques fonctions déjà présentes dans Python

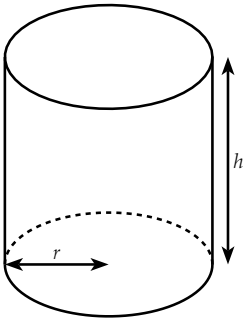
Toutes les versions de Python disposent des fonctions suivantes, elles sont appelées fonctions natives ou BUILT-IN .


Quelques fonctions :

Fonction	Description
round(x, n)	Renvoie le multiple de 10^{-n} le plus proche de x. Le deuxième argument est (un entier) facultatif (0 par défaut) et peut même être négatif si on veut arrondir à n chiffres avant la virgule : `round(3.14,1)` $\longmapsto 3.1$ `round(243.98)` $\longmapsto 244$ `round(278,-1)` $\longmapsto 280.0$
abs(x)	Renvoie la valeur absolue du nombre x, c'est à dire sa distance à 0.
min$(a, b, \ldots)$	Renvoie la plus petite valeur entre a et b (on peut mettre davantage de valeurs en les séparant par une virgule).
max$(a, b, \ldots)$	Renvoie la plus grande valeur entre a et b (on peut mettre davantage de valeurs en les séparant par une virgule).

APPLICATION 6 *On définit dans le script la constante* `pi = 3.14` *qui servira de valeur approchée du nombre* π.

1. *Écrire une fonction* `volume(r,h)` *qui reçoit deux nombres* `r` *et* `h` *et renvoie le volume du cylindre représenté ci-contre.*
2. *Écrire une fonction* `aire(r,h)` *qui reçoit deux nombres* `r` *et* `h` *et renvoie l'aire totale du cylindre.*
3. *Écrire une fonction* `cylindre(r,h)` *qui reçoit deux nombres* `r` *et* `h` *et renvoie l'aire et le volume sous forme d'un couple.*



APPLICATION 7 *On considère la fonction :*

```
def arrondi(x):
    return round(x) - round(x+1)
```

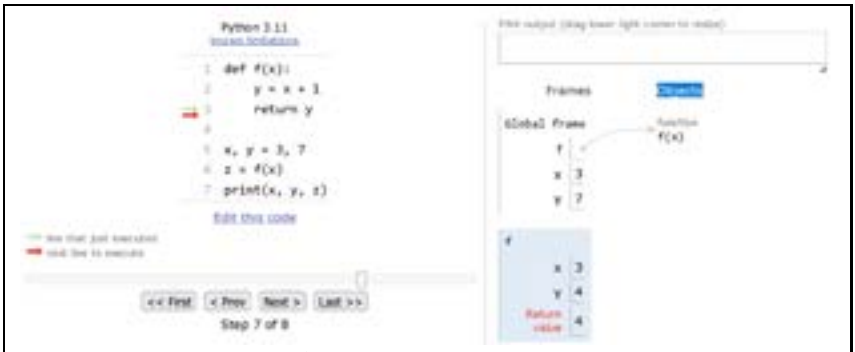
1. *Que renvoient les appels* `arrondi(0.3)`, `arrondi(1.9)`, *ainsi que* `arrondi(-3.1)` *et* `arrondi(-3.9)` *?*
2. *Que peut-on conjecturer pour la valeur de* `arrondi(x)` *où* `x` *est un nombre quelconque ?*
3. *Que renvoient les appels* `arrondi(0.5)` *et* `arrondi(1.5)` *?*

Vous êtes surpris ? Exécutez étapes par étapes les calculs réalisés par Python pour tenter de comprendre... sinon rendez-vous dans la partie correction.

Portée des variables

Lorsque l'on définit une fonction, les variables créées à l'intérieur de celle-ci sont des **variables locales** : une fois sorties de la fonction, elles n'existent plus.

Regardons les deux dernières étapes de l'exécution d'un petit code via le site pythontutor : `pythontutor.com/`



Étape juste avant le `return`

Fin du programme

Quelques autres exemples :

```
def f(x) :
    x = x + 1
    return x

x, y = 3, 7
z = f(x)
print(x, y, z)
```

Affiche

```
3 7 4
```

```
def f(x) :
    x = y + 1
    return x

x, y = 3, 7
z = f(x)
print(x, y, z)
```

Affiche

```
3 7 8
```

```
def f(x) :
    y = y + 1
    return y

x, y = 3, 7
z = f(x)
```

Affiche

```
Traceback (most recent call last):
  File "<module3>", line 7, in <module>
  File "<module3>", line 3, in f
UnboundLocalError:
   local variable 'y' referenced ↙
       ↳ before assignment
```

Ce qu'il faut retenir :

- Les **variables globales** (celles du script) sont accessibles en lecture, mais pas en écriture dans une fonction : on peut lire leurs valeurs mais pas les modifier.
- Les variables locales sont détruites au moment de la sortie de la fonction, il n'y a donc pas de conflit avec une variable globale qui porterait le même nom.
- Avec EduPython, à chaque exécution d'un script, l'ensemble des variables globales est effacé. Si on souhaite n'exécuter qu'un morceau de script (exemple modifier la définition d'une fonction) on peut sélectionner à la souris la partie à exécuter et utiliser l'outil "exécuter la sélection" avec l'icône  .

Fonctions récursives

Comme en mathématiques, on peut parfois définir des fonctions de manière inductive. Par exemple, pour un réel a non nul et un entier naturel n on veut donner la définition de a^n, on peut écrire :

$$a^n = \underbrace{a \times a \times \cdots \times a}_{n \text{ fois}}$$

Cette définition, bien que claire et couramment utilisée, n'est pas des plus rigoureuses... comment faire des démonstrations avec une définition qui comporte des ... ?

On peut ainsi donner une définition mathématiquement plus rigoureuse (mais certainement moins claire !) :

$$\forall a \in \mathbb{R}^*, n \in \mathbb{N},\ a^n = \begin{cases} 1 \text{ si } n = 0 \\ a \times a^{n-1} \text{ sinon.} \end{cases}$$

Cette définition est dite inductive, car elle utilise la définition de puissance dans sa propre définition. Elle permet en effet de calculer n'importe quelle puissance entière, par exemple :

$$\begin{aligned} 3^4 &= 3 \times 3^3 \\ &= 3 \times 3 \times 3^2 \\ &= 3 \times 3 \times 3 \times 3^1 \\ &= 3 \times 3 \times 3 \times 3 \times 3^0 \\ &= 3 \times 3 \times 3 \times 3 \times 1 \\ &= 81 \end{aligned}$$

Pour la programmer en Python, nous allons avoir besoin de faire un test. Pour davantage de précisions, vous pourrez prendre connaissance de la fiche qui suit, dédiée au test. On obtient alors presque une traduction mot à mot de la définition :

```
def puissance(a, n):
    if n == 0:
        return 1                  # Cas d'arrêt
    else:
        return puissance(a, n-1) # Appel récursif
```

Cette précédente fonction est une **fonction récursive** c'est à dire une fonction :

- qui contient des cas d'arrêt ;
- qui s'appelle elle-même ;
- où les appels successifs finissent par aboutir aux cas d'arrêt (sinon on obtient une boucle infinie).

Application 8 *On considère la suite* (u_n) *définie sur* $\mathbb{N}$ *par*

$$\begin{cases} u_0 = 2 \\ u_{n+1} = 2u_n - 1, \ \forall n \in \mathbb{N} \end{cases}$$

Écrire une fonction qui reçoit un entier naturel n *et renvoie la valeur de* u_n *à l'aide d'une fonction récursive.*

Les tests

Grâce aux tests, nous allons pouvoir quitter les structures très linéaires de nos fonctions en ajoutant des **structures conditionnelles**. Cette fiche prend comme fil rouge un exemple de recherche d'une formule la plus rentable à choisir pour pratiquer la natation en fonction du nombre d'entrées :

Une piscine municipale propose trois formules de tarifs :

- formule 1 : Une entrée au prix de 3 €,
- formule 2 : Un abonnement annuel à 9 €, puis 1 € l'entrée,
- formule 3 : Un abonnement annuel à 20 €, puis les entrées à volonté.

Si ... alors ...

Pour simplifier le problème, on ne s'intéresse qu'aux 2 premières formules. On veut écrire une fonction qui reçoit le nombre d'entrées et renvoie le numéro de la formule la plus intéressante.

Remarquons que le prix à payer ne peut être identique pour les deux tarifs puisque l'équation $3n = 9+n$ n'a pas de solution entière. L'algorithme pourrait donc s'écrire ainsi :

```
FONCTION choix1ou2(n : ENTIER) :
    p1 ← n × 3
    p2 ← 9 + n × 1
    SI p1 < p2 ALORS :
        RENVOYER 1
    FIN SI
    SI p1 > p2 ALORS :
        RENVOYER 2
    FIN SI
FIN FONCTION
```

Ce qui donne en Python :

```
def choix1ou2(n) :
    p1 = 3 * n
    p2 = 9 + n
    if p1 < p2:
        return 1
    if p2 < p1:
        return 2
```

Quelques commentaires sur le code précédent :

- Le test "SI" se traduit en Python par l'instruction **`if`** : lorsque Python rencontre une instruction **`if`**, il effectue l'ensemble des instructions **indentées** (c'est à dire décalées) qui suivent lorsque la condition est vérifiée.
- Le ALORS n'apparaît pas en Python, mais est remplacé par deux points **`":"`**
- Comme pour les fonctions, l'**indentation** peut être obtenue à l'aide de la touche  ou en utilisant des espaces. Elle est faite automatiquement dès qu'on valide une ligne commençant par `if` et terminée par **`":"`**

Cette contrainte de mise en page nous oblige à présenter les scripts de manière ergonomique et permet ainsi à celui qui les relit de s'y retrouver facilement.

Pour éclairer nos propos, voici deux fonctions très semblables :

```
def aire_carre(c) :
    if c > 0:
        aire = c * c
        return aire
```

```
def aire_carre(c) :
    if c > 0:
        aire = c * c
    return aire
```

Les deux fonctions ont le même comportement pour des appels avec un argument positif. Avec celle de gauche, l'appel `aire_carre(-1)` ne renvoie rien (aucune instruction `return` n'est rencontrée) alors qu'avec la fonction de droite, le même appel déclenche une erreur :

UnboundLocalError : local variable 'aire' referenced before assignment.

En effet, la dernière ligne étant alignée avec le reste du programme principal, elle est toujours exécutée et Python ne peut renvoyer la variable `aire` qu'il ne connaît pas.

Opérateurs de comparaison

Nous avons déjà vu les symboles > et < pour comparer deux expressions numériques, en voici d'autres qui vont s'avérer utiles par la suite :

Symbole	Signification
==	Opérateur de test d'égalité.
> et <	Opérateur de test d'inégalités strictes habituelles.
<= et >=	Opérateur de test d'inégalités larges $\leqslant$ et $\geqslant$.
! =	Opérateur de test de non-égalité (« différent de »).

APPLICATION 9 *Réaliser une fonction qui reçoit l'âge d'une personne et renvoie un booléen indiquant si cette personne est majeure ou non.*

Si ... alors ... sinon ...

Reprenons notre exemple de comparaison des tarifs de piscine. L'algorithme que nous avons précédemment présenté n'est pas optimal. En effet, lorsque Python exécutera le programme, il testera si $p1 < p2$ puis quel que soit le résultat, testera ensuite si $p2 < p1$. Or ce second test est inutile dans la mesure où, si l'une des inégalités est vérifiée, l'autre ne peut l'être et inversement (ici il ne peut y avoir égalité). Il est donc préférable d'utiliser un SI...ALORS...SINON dans ce cas :

```
FONCTION choix1ou2(n : ENTIER) :
    p1 ← n × 3
    p2 ← 9 + n × 1
    SI p1 < p2 ALORS :
        RENVOYER 1
    SINON :
        RENVOYER 2
    FIN SI
FIN FONCTION
```

Ce qui donne en Python :

```
def choix1ou2(n):
    p1 = 3 * n
    p2 = 9 + n
    if p1 < p2:
        return 1
    else:
        return 2
```

if *condition* **:** ... **else :** ... effectue l'ensemble des instructions indentées lorsque la *condition* est vérifiée, sinon effectue les instructions alternatives indentées.

On remarque que le `else` est aligné avec le `if` qui lui correspond. Comme pour le `if`, l'indentation des différents blocs se fait automatiquement lorsque l'on appuie sur la touche ENTREE après les deux points.

Notez enfin que l'on peut imbriquer les tests à la manière d'un arbre de décision qui se dessinerait avec les indentations. Dans l'exemple suivant, on considère la fonction f définie sur $\mathbb{R}$ par $f(x) = ax^2 + bx + c$. On se demande si f admet ou non des extrémums.

```
def extrema(a, b, c) :
    if a == 0:
        if b == 0:
            print("f est constante sur IR.")
        else:
            print("f n'a pas d'extrémums sur IR")
    else:
        x = - b / (2*a)
        if a > 0:
            print("f a un minimum en", x, ", mais pas de maximum sur IR")
        else:
            print("f a un maximum en", x, ", mais pas de minimum sur IR")
```

Disjonction de cas : SINON, SI...

Toujours avec notre exemple concernant les tarifs d'entrées à la piscine, nous allons aborder d'autres cas à traiter et afficher une phrase en plus de renvoyer le tarif le plus avantageux.

FONCTION tarif(n) :
    $p_1 \leftarrow n \times 3$
    $p_2 \leftarrow 9 + n \times 1$
    SI $p1 < p2$ ALORS :
        AFFICHER "Le tarif 1 est plus avantageux"
        RENVOYER 1
    SINON, SI $p2 < 20$:
        AFFICHER "Le tarif 2 est plus avantageux"
        RENVOYER 2
    SINON, SI $p2 = 20$:
        AFFICHER "Les tarifs 2 et 3 sont aussi avantageux"
        RENVOYER 3
    SINON :
        AFFICHER "Le tarif 3 est plus avantageux"
        RENVOYER 3
    FIN SI

Ce qui donne en Python :

```
def tarif(n) :
    p1 = 3 * n
    p2 = 9 + n
    if p1 < p2:
        print("Le tarif 1 est plus avantageux")
        return 1
    elif p2 < 20:
        print("Le tarif 2 est plus avantageux")
        return 2
    elif p2 == 20:
        print("Les tarifs 2 et 3 sont aussi avantageux")
        return 3
    else:
        print("Le tarif 3 est plus avantageux")
        return 3
```

elif : le terme `elif` est une contraction de "else if" (sinon, si...) et permet de raisonner par disjonction de cas.

- On peut enchaîner autant d'instructions **elif** que nécessaire.
- On peut éventuellement terminer la série de **elif** par un **else** pour traiter tous les cas restants.

Opérateurs logiques

Opérateur	Signification
A **and** B	Renvoie VRAI si les conditions A et B sont vérifiées.
A **or** B	Renvoie VRAI si au moins l'une des conditions A ou B est vérifiée.
A ^ B	Renvoie VRAI si exactement une des conditions A ou B est vérifiée (mais pas les deux, on dit qu'il s'agit d'un "ou exclusif").
not A	Renvoie le contraire de la véracité de la condition A.

APPLICATION 10 *On souhaite réaliser une fonction qui demande le numéro du mois et renvoie un booléen indiquant si ce mois comporte 31 jours ou non.*

1. *Écrire une première version de cette fonction qui n'utilise qu'un seul `if` au maximum.*
2. *Chercher une version qui n'utilise qu'un seul `if` et seulement 3 opérateurs logiques (`and`, `or` ....) au maximum.*
3. *Chercher enfin une version qui n'utilise qu'un seul `if` au maximum et aucun opérateur logique !*

Python interprète correctement les tests de doubles inégalités.

Dans ce cas, il traduit $A < B < C$ par $(A < B)$ et $(B < C)$.

```
if 2 <= x < 4:
    print("Oui")
else :
    print("Non")
```

Le programme ci-dessus affiche "Oui" si et seulement si $x \in [2;4[$.

La boucle TANT QUE

La boucle la plus simple à comprendre est la boucle :

TANT QUE ... FAIRE : ...

Cette instruction sera traduite par **`while`** en Python.

`while` *cond* **:** exécute une instruction ou un bloc d'instructions tant que la condition *cond* est vérifiée.

- Comme pour le `if`, l'ensemble des instructions à exécuter est indenté.
- La boucle peut donc ne jamais être exécutée si d'entrée la condition n'est pas remplie.

Voici un premier exemple pour commencer. On cherche à calculer le plus grand diviseur commun à 75 et 45 par la méthode des différences étudiée en classe de troisième. Le PGCD est la dernière différence non nulle, ici 15.

a	b	différence
75	45	30
45	30	15
30	15	**15**
15	15	0

En Python, on pourrait traduire l'algorithme ainsi :

```
a, b = 75, 45
print("Le PGCD de", a, "et", b, "vaut ", end="")
while a != b:
    if a > b:
        a, b = b, a-b
    else:
        a, b = a, b-a
print(a)
```

Application 11

1. *Qu'affiche cet algorithme ?*
2. *Le programmer en Python.*

$n \leftarrow 1$
TANT QUE $n \leqslant 100$ FAIRE :
  $n \leftarrow 2 \times n$
FIN TANT QUE
AFFICHER n.

Application 12 *On considère la fonction ci-contre. Que se passe-t-il si on saisit dans la console :*

1. `mystere(2,10**4)`
2. `mystere(3,81)`
3. `mystere(1,0)`
4. `mystere(0,1)`

Une fois la conjecture effectuée, écrire et exécuter la fonction pour vérifier la réponse.

```
def mystere(u, L):
    n = 0
    while u <= L:
        u = u ** 2
        n = n + 1
    return n
```

La boucle for

Parfois, on connaît à l'avance le nombre de répétitions de la boucle (par exemple lancer 3 dés, tirer un échantillon de taille 1000, ...). Dans ce cas, on peut utiliser la commande **`for`** pour gagner du temps.

Boucle TANT QUE :

```
n = 0
while n < 5:
    ...instructions...
    n = n + 1
```

Boucle POUR :

```
for n in range(5):
    ...instructions...
```

Les comportements de ces deux scripts sont identiques, avec la boucle **`for`**, il n'y a pas à gérer l'initialisation et l'incrémentation de la variable.

`for` *var* **`in range(`***debut*, *fin*, *pas*`):**

Les paramètres *debut* et *pas* sont optionnels. Cela permet de réaliser une boucle en faisant parcourir à la variable *var* tous les entiers :

- de l'intervalle $[0; fin[$ si un seul paramètre est renseigné,
- de l'intervalle $[debut; fin[$ si 2 paramètres sont renseignés,
- dans l'intervalle $[debut; fin[$ mais en réalisant une suite arithmétique de raison *pas* si les 3 paramètres sont renseignés.

À noter que :

- Comme pour le `while`, l'instruction se termine par deux points et les instructions à répéter doivent être indentées.
- L'intervalle parcouru est semi-ouvert :

```
for i in range (1,5) :
        ...
```

  dans cet exemple, `i` prend successivement les valeurs 1, 2, 3, 4.

- En cas d'incohérence, la boucle n'est pas exécutée. Par exemple :

```
for n in range (100,110,-2):
    ...
```

- Contrairement à la boucle TANT QUE, la variable du compteur parcourt, quoiqu'il arrive, les valeurs demandées, on ne peut pas la modifier en cours de boucle. Le programme ci-dessous affiche les 10 premiers entiers de 0 à 9 :

```
for i in range(10):
    print(i)
    i = i * 2
```

APPLICATION 13 *On considère la suite* (u_n) *définie sur* $\mathbb{N}$ *par*

$$\begin{cases} u_0 = 2 \\ u_{n+1} = 2u_n - 1, \ \forall n \in \mathbb{N} \end{cases}$$

Écrire une fonction qui reçoit un entier naturel n *et renvoie la valeur de* u_n *à l'aide d'une boucle* `for`.

APPLICATION 14 1. *Traduire cette fonction en Python.*

```
FONCTION calcule(n)
    s ← 0
    POUR i ALLANT DE 1 à n FAIRE :
        s ← s + i
    FIN POUR
    RENVOYER s.
FIN FONCTION
```

2. *Que renvoie-t-elle si on lui donne en argument un entier naturel ?*
3. *La modifier pour calculer* $\frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \cdots + \frac{1}{1000^2}$.

Avec la boucle POUR, on peut ne pas se limiter aux suites arithmétiques et accéder à d'autres "objets" :

```
for i in (1, 4, 5, 0, 9, 1):
        print(i)
```

```
for i in ["a", "e", "i", "o", "u", "y"]:
        print(i)
```

```
for lettre in "Python":
        print(lettre)
```

- des listes de nombres ou d'autres objets. On les met alors entre parenthèses ou entre crochets,
- une à une, les lettres d'un mot...

Le module math

Lorsque l'on programme, inutile de toujours vouloir tout réinventer, même si reprogrammer les fonctions de bases peut s'avérer formateur.

Un certain nombre de fonctions regroupées dans des **modules** sont présentes dans Python. Elles permettent de gagner du temps... à condition de bien connaître les fonctionnalités et contraintes de chacune pour ne pas avoir de mauvaises surprises.

Différentes manières d'importer un module

Prenons l'exemple du module **`math`** que nous allons présenter plus en détail ensuite. Il contient, vous l'aurez deviné, de nombreuses fonctions mathématiques, entre autres la fonction **`sqrt`** qui calcule et renvoie la racine carrée d'un réel positif.

On va **importer** le module math dans Python de manière à pouvoir utiliser cette fonction. Cela peut se faire de différentes manières :

méthode 1 :

```
>>> import math
>>> math.sqrt(4)
2.0
```

méthode 2 :

```
>>> from math import sqrt
>>> sqrt(4)
2.0
```

méthode 3 :

```
>>> from math import *
>>> sqrt(4)
2.0
```

méthode 4 :

```
>>> import math as M
>>> M.sqrt(4)
2.0
```

Quelques détails sur ces différentes méthodes :

- Avec la méthode 1, on importe le module, mais le nom de la fonction sera précédé du nom du module, ce qui s'avère pratique lorsque deux fonctions portant le même nom existent dans deux modules différents. Cette méthode permet **l'auto-complétion**, lorsque l'on commence à taper `math.`, la liste des fonctions disponibles est affichée (voir capture suivante)

- Avec la méthode 2, on importe la fonction `sqrt` du module `math` et on peut alors l'utiliser comme toutes les autres fonctions de Python. On peut séparer les noms des fonctions que l'on veut importer par une virgule s'il y en a plusieurs.

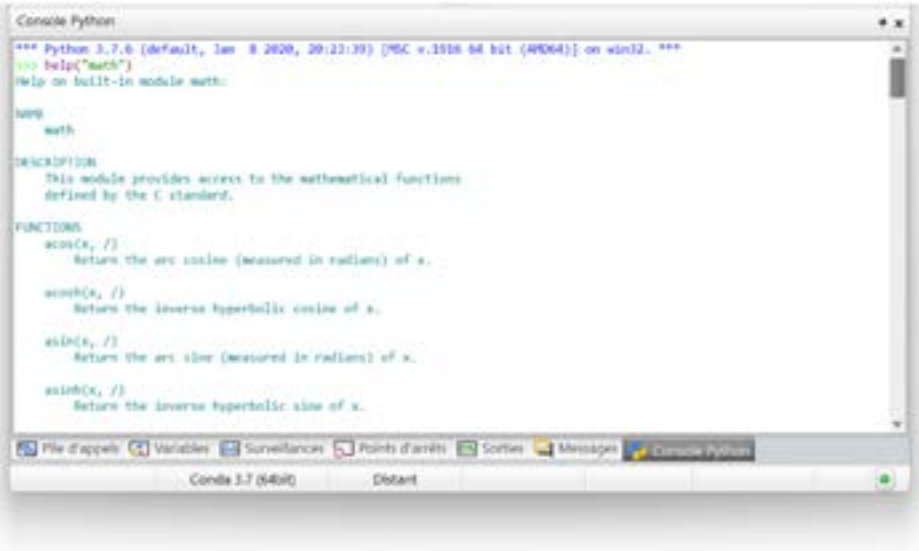
- Même principe pour la méthode 3, mais on a directement importé toutes les fonctions du module `math`.

- Enfin la méthode 4 permet de donner un nom (on dit **alias**) au module, ce qui est utile pour les modules qui ont un nom assez long.

Le module math

Le module **`math`** est un module présent dans toutes les distributions de Python (comme pour les fonctions, on parle de module BUILT-IN).

Si vous voulez avoir la liste et la description des fonctions existantes dans ce module, vous pouvez saisir dans la console `help("math")`.



Encore une fois, c'est tout l'intérêt d'avoir renseigné les doc-strings dans les fonctions : on dispose immédiatement d'une description de ces mêmes fonctions.

Si vous souhaitez juste connaître la liste des fonctions disponibles, vous pouvez utiliser la fonction **dir** comme le montre l'exemple ci-contre :

```
>>> dir("math")
['__doc__',
 '__name__',
 '__package__',
 'acos',
 'acosh',
 'asin',
 'asinh',
 'atan',
 'atan2',
 'atanh',
 'ceil',
 'copysign',
 'cos',
 ........
 'sqrt',
 'tan',
 'tanh',
 'trunc']
```

https ://docs.python.org/fr/3.7/library/math.html

Détaillons donc quelques-unes de ces fonctions qui nous seront utiles au lycée :

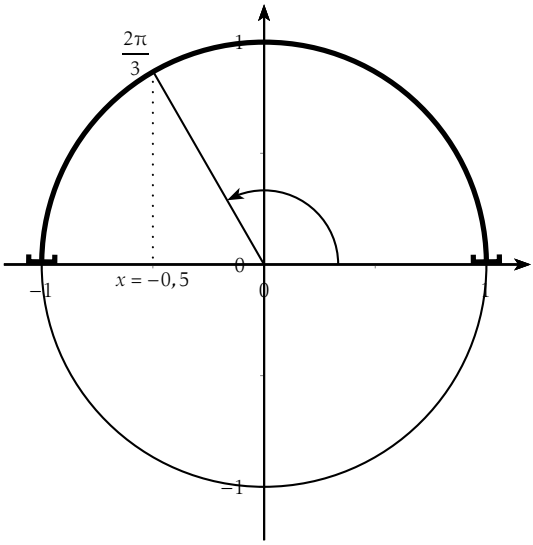
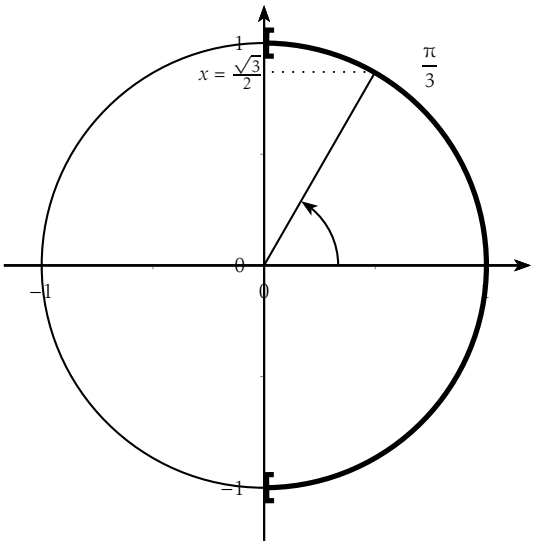
Quelques fonctions de base :

Fonction	Description
sqrt(x)	Renvoie $\sqrt{x}$ si $x \geqslant 0$ et une erreur si $x < 0$.
floor(x)	Renvoie la partie entière du nombre x au sens mathématique, c'est-à-dire le plus grand entier inférieur ou égal au nombre x. Ainsi `floor(3.9)` renvoie 3 et `floor(-3.9)` renvoie -4.
trunc(x)	Renvoie la troncature de x. Ainsi `trunc(3.9)` renvoie 3 et `trunc(-3.9)` renvoie -3.

Fonctions trigonométriques :

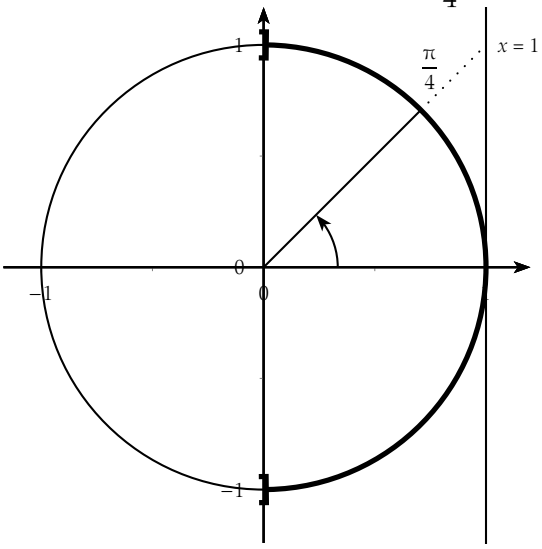
Fonction	Description
cos(a) **sin**(a) **tan**(a)	Représentent les 3 fonctions trigonométriques cosinus, sinus et tangente. Attention, le nombre a représente une mesure d'angle en radians.

Fonctions trigonométriques réciproques :

Fonction	Description
`acos``(`x`)`	Renvoie l'unique nombre de l'intervalle $[0;\pi]$ dont le cosinus vaut x. 
`asin``(`x`)`	Renvoie l'unique nombre de l'intervalle $\left[-\frac{\pi}{2};\frac{\pi}{2}\right]$ dont le sinus vaut x. 

Fonction	Description
`atan(`x**`)`**	Renvoie l'unique nombre de l'intervalle $\left]-\frac{\pi}{2};\frac{\pi}{2}\right[$ dont la tangente vaut x.

`atan(1)` $\mapsto 0.78539816339744 \approx \frac{\pi}{4}$



Ces trois fonctions correspondent aux touches $\boxed{\cos^{-1}}$, $\boxed{\sin^{-1}}$ et $\boxed{\tan^{-1}}$ de votre calculatrice lorsque celle-ci est en mode radian.

Des constantes :

- Lorsque l'on importe le module math, la constante π (ou tout du moins une très bonne valeur approchée) est déclarée sous le nom **`pi`**.
- De même, le nombre e présenté en classe de première est déclaré sous la variable **`e`**. Attention donc au conflit de noms si vous utilisez vous-même une variable `e` dans votre programme !

```
>>> math.pi
3.141592653589793

>>> r = 3
>>> 2 * math.pi * r
18.84955592153876

>>> math.pi * r ** 2
28.274333882308138

>>> math.e
2.718281828459045
```

Conversion d'angles : en classe de seconde, l'unité d'angle utilisée est le degré. À partir de la première, on utilisera le radian. Les fonctions trigonométriques étant définies en radians, il vous faudra convertir les angles.

Fonction	Description
degrees(*a*)	Convertit en degrés un angle en radians.
radians(*a*)	Convertit en radians un angle en degrés.

Par exemple pour obtenir le cosinus de 60°, on tapera :

```
>>> from math import *
>>> cos(radians(60))
0.5000000000000001
```

Application 15 *En classe de seconde, on travaille plutôt en degrés qu'en radians. Écrire les fonctions suivantes en prenant soin de préciser dans leurs docstrings ce qui est attendu en entrée et ce qu'elles renvoient en sortie.*

1. *Les 3 fonctions* `cosD`, `sinD` *et* `tanD` *similaires aux fonctions* `cos`, `sin` *et* `tan` *mais recevant des angles en degrés.*
2. *Les 3 fonctions réciproques de ces dernières.*

D'autres fonctions pour la classe de première et terminale :

Fonction	Description
exp(*x*)	Fonction exponentielle : renvoie e^x
log(*x*)	Fonction logarithme népérien : renvoie $\ln x$
log10(*x*)	Fonction logarithme décimal souvent utilisée en physique : renvoie $\log_{10}(x)$ c'est à dire $\dfrac{\ln x}{\ln 10}$
factorial(*n*)	Renvoie $n!$ c'est-à-dire $1 \times 2 \times 3 \cdots \times n$. Par convention $0! = 1! = 1$.

Application 16 *Pour $n \in \mathbb{N}$ et $p \in \mathbb{N}$, il est possible de calculer les coefficients binomiaux à l'aide de la formule :* $\binom{n}{p} = \begin{cases} \dfrac{n!}{p!(n-p)!} & \text{si } 0 \leqslant p \leqslant n \\ 0 & \text{sinon} \end{cases}$

Écrire une fonction qui reçoit deux entiers naturels n et p et renvoie $\binom{n}{p}$.

Le module random

À l'instar du module **math**, le module **random** permet, entre autres, d'obtenir des valeurs aléatoires et ainsi de concevoir des simulations.

Lois discrètes :

Fonction	Description
randint(a,b**)**	Renvoie de manière équiprobable un nombre entier aléatoire de l'intervalle $[a;b]$ où a et b sont des entiers. Avec le tableur, on retrouve la commande équivalente `ALEA.ENTRE.BORNES(`a,b`)`. Typiquement `randint(1, 6)` permet de simuler le lancer d'un dé et `randint(0, 1)` permet de simuler le lancer d'une pièce.
rand()	Renvoie un nombre réel de l'intervalle $[0;1]$. Cette fonction correspond à la fonction `ALEA()` du tableur.
uniform(a,b**)**	Identique à la fonction `rand` mais sur l'intervalle $[a;b]$.

Au détour de la balade...

À noter qu'en ce sens, une fonction informatique est différente d'une fonction mathématique, par exemple pour la fonction définie ainsi :

```
from random import randint
def f(n) :
    return randint(0,n)
```

L'appel `f(2)` *peut renvoyer n'importe quelle valeur dans* $\{0;1;2\}$ *avec équiprobabilité, ce qui n'est pas concevable pour une fonction mathématique : tout élément de l'ensemble de définition possède une unique image.*

Application 17 *Écrire une fonction* `sommeDes(n)` *qui reçoit le nombre de dés à lancer et renvoie la somme des nombres obtenus sur les faces des dés.*

Application 18 *Écrire une fonction* `nbpile(n)` *qui reçoit un entier naturel* `n` *et simule* `n` *lancers d'une pièce équilibrée. À chaque lancer, la fonction affiche "Pile" ou "Face" et envoie en fin de simulation le nombre de "Pile" obtenu.*

Les types

En informatique, tous les objets ont un type. De nombreux types existent déjà en Python, on peut même créer ses propres types d'objets, mais c'est une autre histoire !

Voici ci-contre quelques exemples de types couramment utilisés pour faire des mathématiques.

On peut également demander de quel type (que l'on appelle **classe**) est une variable à l'aide de la fonction **`type`**.

```
>>> n = 5
>>> type(n)
<class 'int'>
>>> type(2 ** 2014)
<class 'int'>
>>> type(1/2)
<class 'float'>
>>> L = "Python"
>>> type(L)
<class 'str'>
>>> n < 1
False
>>> type(n < 1)
<class 'bool'>
```

Les entiers

En Python les entiers ont la particularité de ne pas être limités en taille contrairement à la majorité des autres langages. Par exemple, si on entre dans la console `123**123`, on aura comme affichage :

```
114374367934617190099880295228066276746218078451850229775887975052369504785666896446606568365201542169649974727730628842345343196581134895919942820874449837212099476648958359023796078549041949007807220625356526926729664064846685758382803707100766740220839267
```

Nous avons déjà présenté dès les premières fiches les opérations usuelles sur ces nombres.

Les flottants

Le type flottant permet en Python de représenter des nombres réels avec toutefois une grosse spécificité due à la représentation des nombres en machine.

- ```
  >> type(3)
  <class 'int'>
  >>> type(3.0)
  <class 'float'>
  ```

  Pour indiquer à Python qu'un nombre entier est un flottant, on ajoute `.0` ou juste un point.

- ```
  >>> a = 45 / 9
  >>> a
  5.0
  >>> type(a)
  <class 'float'>
  ```

  Une division décimale renvoie toujours un flottant même si le résultat est entier.

-
```
>>> 12e-3
0.012
```
On utilise la lettre e pour signifier $\times 10^{\cdots}$ dans l'écriture scientifique par exemple.

-
```
>>> 0.3 - 0.1 - 0.2
-2.77555756156289e-17
```
Enfin, il faut se méfier des "erreurs d'arrondis" avec les nombres flottants même simples.

  Ici le résultat affiché est environ $-2,8 \times 10^{-17}$ ce qui est très proche de 0.... mais pas le 0 escompté.

Voici quelques explications sur ces erreurs d'arrondi que vous pouvez ignorer en première lecture. Il faut retenir qu'il est préférable d'utiliser des entiers dans la mesure du possible si vous souhaitez des résultats exacts et qu'il est très risqué de tester des égalités avec des nombres flottants.

Plaçons-nous en base 10 pour commencer, puisque c'est ainsi que les êtres humains comptent. On comprend aisément qu'il va être compliqué pour une machine de stocker des nombres tels que π, $\sqrt{2}$ ou encore $\frac{1}{3}$ qui contiennent une infinité de chiffres après la virgule, en effet il va être difficile de faire entrer une infinité de chiffres dans une place de la mémoire d'un ordinateur (de taille finie).

Prenons à présent le nombre $521,345$. On peut facilement l'écrire :

$$521,345 = 5 \times 100 + 2 \times 10 + 1 \times 1 + 3 \times \frac{1}{10} + 4 \times \frac{1}{100} + 5 \times \frac{1}{1000}$$

Ce nombre semble donc facile à stocker dans la mémoire d'un ordinateur. Or, c'est loin d'être évident car Python, comme la plupart des langages, compte en base 2. Ainsi même le nombre simple $0,1$ s'écrit :

$$0,1 = \frac{1}{16} + \frac{1}{32} + \frac{1}{256} + \frac{1}{512} + \frac{1}{4096} + \frac{1}{8192} + \ldots$$

$$= 2^{-4} + 2^{-5} + +2^{-8} + 2^{-9} + 2^{-12} + 2^{-13} \ldots$$

$$= \overline{0,0001100110011001100110011001100110011001100}\ldots \text{ en base 2}$$

Il est donc impossible de placer la valeur exacte de ce nombre en mémoire. Le langage va essayer de mettre la valeur la plus proche, ce qui explique les erreurs d'arrondis que l'on rencontre parfois quand on n'utilise pas des nombres entiers.

Les chaînes de caractères

Les chaînes de caractères représentent des textes, une fiche entière est consacrée à ce sujet quelques pages plus loin.

Les booléens

Les booléens sont des variables qui ne peuvent prendre que deux valeurs : `True` ou `False` (Vrai ou Faux pour les non-anglophones !). Vous aurez sûrement remarqué la majuscule au début de ces deux mots, ne l'oubliez pas.

```
>>> b = 2 < 3
>>> b
True
>>> type(b)
<class 'bool'>
```

Sans le savoir, nous utilisons ces booléens depuis le début à chaque fois que l'on effectue un test. On peut donc utiliser les opérateurs logiques comme le montrent les tables de calculs suivantes :

A	B	not A	A and B	A or B	A ^ B
VRAI	VRAI	FAUX	VRAI	VRAI	FAUX
VRAI	FAUX	FAUX	FAUX	VRAI	VRAI
FAUX	VRAI	VRAI	FAUX	VRAI	VRAI
FAUX	FAUX	VRAI	FAUX	FAUX	FAUX

Le type de `None`

Même "rien" a un type en informatique. Par exemple si on exécute le code ci-contre, on obtient le résultat présenté. En effet, comme 1 n'est pas supérieur à 2, l'appel `demo(1)` ne renvoie "rien", la variable `a` est de la classe "NoneType" qui ne peut prendre qu'une valeur : `None`

```
def demo(x) :
    if x > 2 :
        return x

a = demo(1)
print(a)
print(type(a))
```

```
None
<class 'NoneType'>
```

Le module turtle

Python propose un module **`turtle`** qui permet de faire facilement des dessins sur une fenêtre en donnant des instructions à une tortue virtuelle.

Au détour de la balade...

Ce module est très appréciable pour travailler sur des notions algorithmiques diverses sans grands pré-requis mathématiques ou pour travailler sur des situations géométriques variées. Avec seulement quelques instructions de base (avancer, tourner), chacun est en mesure de travailler en autonomie et de tester les choix privilégiés en exécutant son code pour vérifier s'il correspond ou non à la figure souhaitée.

La structure d'un programme utilisant le mode tortue est le suivant :

```
from turtle import *

ICI VOS INSTRUCTIONS A LA TORTUE

mainloop()
```

N'oubliez pas la dernière ligne `mainloop()` sinon vous ne pourrez pas reprendre la main sur la fenêtre. Si tel est le cas, revenez sur la fenêtre contenant le script et réinitialisez le moteur Python depuis le menu :

`Exécuter → Moteur Python → Réinitialiser le moteur Python`

ou tout simplement en pressant les touches CTRL et F2.

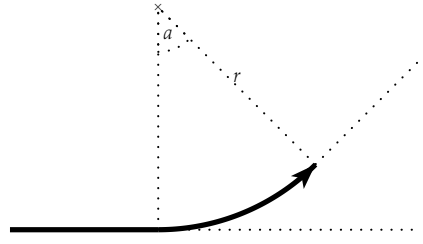
Voici quelques instructions permettant de faire de nombreuses figures avec le module `turtle`. Si vous souhaitez aller plus loin, vous pouvez toujours consulter le site officiel (en anglais) :

`https://docs.python.org/fr/3.7/library/turtle.html`

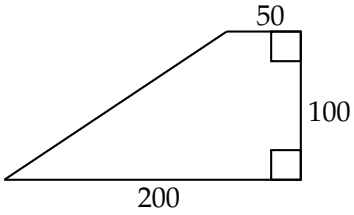
Commande	Effet
forward(*pas*)	La tortue avance de *pas*.
back(*pas*)	La tortue recule de *pas*.
left(*a*) **right**(*a*)	La tortue tourne la tête de *angle* degrés vers la gauche ou vers la droite. Attention, la tortue tourne juste la tête, mais n'avance pas !
circle(*r*, *a*)	La tortue dessine un arc de cercle (ou un cercle entier si *a* n'est pas précisé) de rayon *r* en partant de la tête de la tortue et en tournant de *a* degrés dans le sens trigonométrique. • Si $a < 0$ la tortue part en marche arrière. • Si $r < 0$ la tortue tourne dans le sens horaire.

`forward(100)`

`circle(150,45)`



APPLICATION 19 *Réaliser ce trapèze rectangle à l'aide de la tortue.*



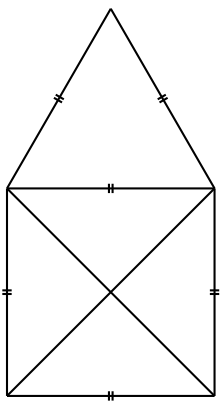
On peut aussi modifier l'apparence des traits laissés par la tortue pour alléger ou faciliter l'appropriation des constructions.

Commande	Effet
`hideturtle()` **`showturtle()`**	Permet de cacher (resp. montrer) la tortue.
`speed(`*v***`)`**	Permet de modifier la vitesse de tracé, 1 étant la plus lente et 10 la plus rapide. On peut aussi utiliser `speed(0)` pour aller le plus rapidement possible.
`up(), down()`	Ordonne à la tortue de ne plus laisser de trace derrière elle ou de reprendre le tracé.
`write(`*txt***`)`**	Écrit le texte *txt* à l'emplacement de la tortue.
`pencolor(`*coul***`)`** **`bgcolor(`***coul***`)`**	Modifie la couleur du tracé. Modifie la couleur du fond de la fenêtre.

Pour les deux dernières fonctions, la couleur *coul* peut être donnée :

- Sous forme d'une chaîne de caractères pour les couleurs de base (en anglais) : `'red','blue','yellow'` ...
- Sous forme d'un code hexadécimal comme pour les pages HTML, par exemple `'#40A497'`. Pour trouver une couleur à votre goût, rendez-vous sur le site `http://code-couleur.com/`.
- Sous forme d'un triplet de nombres de l'intervalle $[0;1]$ représentant respectivement la dose de rouge, de vert et de bleu à utiliser. Par exemple `pencolor(1, 0.5, 0)` pour tracer en orange.

Application 20 *Réaliser cette figure en veillant à ne pas passer deux fois par le même chemin. On dessinera un carré de côté* 100.



Petit cadeau : en tapant **`shape('turtle')`** votre tortue triangulaire initiale va se transformer en véritable tortue ! ... Enfin un visuel ressemblant un peu plus à cet animal fascinant.

Les listes

Les listes permettent de stocker un nombre "illimité" de données et se prêtent donc particulièrement bien à l'étude des suites, des séries statistiques et des simulations...

Une liste peut être vue comme un tableau dont le nombre de cases peut évoluer. Le terme illimité est ici exagéré dans la mesure où cela dépend néanmoins de la machine utilisée. Nous avons fait le test sur un ordinateur portable et avons pu entrer dans une liste tous les entiers de 1 à un million (mais pas jusqu'à un milliard) ce qui devrait largement nous suffire !

Les listes vues comme des tableaux

Création d'une liste :

Instruction	Effet
`liste = []`	Crée une liste vide.
`liste = [e1, e2, ...]`	Crée une liste *liste* qui contient les éléments $e1, e2, \ldots$

Dans le dernier cas, on dit que l'on a créé une liste en **extension**, c'est à dire en listant les valeurs des manière exhaustive. On peut aussi créer des listes en **compréhension** en donnant une formule :

```
>>> [i**2 for i in range(10)]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Ce qui correspond à l'écriture mathématiques $L = \left\{ i^2 \mid i \in [\![0;10[\![\right\}$

Lire et modifier les éléments d'une liste :

Instruction	Effet
`len(L)`	Renvoie le nombre d'éléments de la liste L.
`L[i]`	Renvoie l'élément d'indice i de la liste (indice 0 pour le premier).
`L[i] = el`	Stocke *el* dans la liste L au rang i s'il y a déjà un élément à ce rang.

Parcourir une liste (avec deux manières pour le faire) :

Une boucle sur les indices :

```
for i in range(len(L)):
    print(L[i])
```

Une boucle sur les éléments :

```
for e in L:
    print(e)
```

Les deux sont aussi efficaces, tout dépend des besoins.

Fonctions qui agissent sur les listes sans les modifier :

Fonction	Effet
e **in** L	Teste si l'élément *e* est dans la liste L et renvoie `True` ou `False` (VRAI ou FAUX).
min(L**), max(**L**)**	Renvoie le plus petit (le plus grand) élément de la liste L.
sorted(L**)**	Renvoie une nouvelle liste contenant les éléments ordonnés de la liste L.
choice(L**)**	Choisit au hasard un élément de la liste L (nécessite le module `random`).

Application 21 *À part la fonction* `sorted` *difficile à programmer, les 4 autres fonctions sont simples à coder et cela est formateur pour apprendre à manipuler les listes.*

1. *Sans utiliser l'instruction* `in`*, écrire une fonction* `present(L,e)` *qui reçoit une liste* `L` *et un élément* `e` *et renvoie un booléen indiquant si* `e` *est un élément de* `L`*.*
2. *Reprogrammer les fonctions* `min`*,* `max` *et* `choice` *pour des listes non vides.*

Le côté dynamique des listes

Un des gros avantages des listes est donc de pouvoir ajouter et supprimer des éléments. Cela est particulièrement adapté aux modélisations : on peut ajouter, retirer des boules dans une urne selon qu'il y a ou non remise lors des tirages ...

Ajouter / supprimer des éléments d'une liste :

Méthode	Effet
`L.append(`e`)`	Ajoute l'élément e à la fin de la liste L
`L.insert(`j`,`e`)`	Insère l'élément e au rang j de la liste L.
`L.remove(`e`)`	Supprime la première occurrence de l'élément e dans la liste L. Attention, si e n'est pas présent dans la liste, une erreur se produit. Pensez donc à vérifier que e est présent avec l'instruction `in`.
`L.pop(`i`)`	Supprime l'élément d'indice i de la liste L.

APPLICATION 22 *Qu'affichent les programmes suivants (ou déclenchent-ils une erreur) ? Tester ensuite en les codant en Python.*

1.
```
P = [2, 4, 6, 8]
print(P[2])
P.append(3)
print(P[2])
P.remove(3)
print(P)
print(len(P))
```

2.
```
f = []
for x in range(1, 6):
    f.append(3 - (x-3) ** 2)
print(min(f), max(f))
```

3.
```
L = []
for i in range(6):
    L.append(i**2)
    if i in L:
        L.remove(i)
print(L)
```

4.
```
L = [1, 1]
for i in range(6):
    print(L[0], end=' ; ')
    L.append(L[0] + L[1])
    L.pop(0)
```

Complément : les listes sont des alias

Les listes présentent l'avantage de pouvoir stocker un très grand nombre de données, mais cela a un prix... Lorsque l'on écrit `L=[1,2,3,4]`, la variable `L` ne contient pas le contenu de la liste, mais l'emplacement mémoire où se trouve la liste. On dit que la variable `L` est **un pointeur**.

L'exemple qui suit illustre le problème : lorsque l'on saisit en Python `it = fr`, on ne crée pas une copie de la liste, mais une copie de l'adresse mémoire, on dit que `it` est **un alias** de `fr`. Ainsi lorsque l'on modifie `it`, la liste `fr` est aussi modifiée, puisqu'il s'agit en fait de la même liste.

```
fr = ["Bleu", "Blanc", "Rouge"]
it = fr
it[0] = "Vert"
print("FR=", fr)
print("IT=", it)
```

Affiche :

```
FR= ['Vert', 'Blanc', 'Rouge']
IT= ['Vert', 'Blanc', 'Rouge']
```

Ceci peut être illustré via le site `https://pythontutor.com/`, un excellent support pédagogique pour exécuter du code ligne à ligne et visualiser ce qui se passe en mémoire.

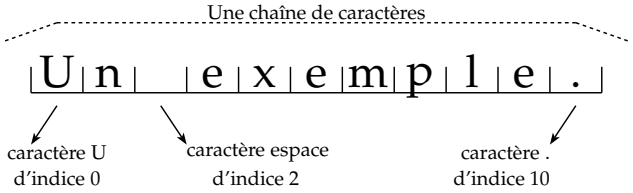
Pour éviter ce problème, on peut créer un double de la liste en tapant `it = fr[:]` ou `it = fr.copy()` qui crée un nouvel objet et recopie le contenu de la liste `fr` dans `it`, lui affectant ainsi une adresse mémoire différente de `it`.

On dit que les listes sont **des variables mutables**. Pensez-y lorsque vous réaliserez des fonctions agissant sur des listes : contrairement aux nombres qui sont des variables non mutables, si vous modifiez une liste à l'intérieur d'une fonction, cette liste se trouve modifiée aussi à l'extérieur ! Ce problème (qui n'en est pas un si on a compris le paragraphe précédent) s'appelle problème "**d'effet de bord**" qui est en réalité une mauvaise traduction du terme anglais "**side effect**" (le terme "effet secondaire" aurait peut-être été plus pertinent).

Les chaînes de caractères

L'étude des textes, appelés aussi **chaînes de caractères** ou **chaînes**, n'est pas un attendu des programmes du lycée. Cependant la manipulation de chaînes de caractères permet de faire de l'algorithmique de manière récréative... comme vous le verrez, les mathématiques ne restent jamais bien loin...

Une chaîne de caractères est donc, comme son nom peut le laisser penser, une succession de **caractères**. Un caractère pouvant être une lettre, mais aussi un chiffre ou un symbole spécial <, @, *, §, ... Pour simplifier, on peut ainsi considérer qu'un caractère est un symbole. Comme pour les listes, dans une chaîne de caractères, chaque caractère est indicé et Python commence la numérotation à 0.



Dans un programme, il faut donc différencier le nom de la variable qui contient une chaîne de son contenu. En Python, pour signifier qu'il s'agit d'une chaîne de caractères (ou d'un caractère) on met le texte entre guillemets `'ainsi'` ou `"ainsi"`.

Quelques fonctions

Il existe de très nombreuses fonctions pour travailler sur les chaînes de caractères, nous n'en donnerons que quelques-unes :

Commande	Effet
`len(`*ch*`)`	Renvoie la longueur de la chaîne *ch*.
ch`[`*i*`]`	Renvoie le caractère situé au rang *i* dans la chaîne *ch*.

Attention : contrairement aux listes, les chaînes sont des variables **non mutables**. Il est donc impossible de modifier un caractère à l'intérieur d'une chaîne.

Commande	Effet
`ch = ""`	Crée une chaîne vide nommée *ch*.
`mots[debut:fin]`	Renvoie la partie de chaîne `mots` comprise entre le caractère à la position *debut* (inclus) et celui à la position *fin* (exclu).
`mots1 + mots2`	Colle deux chaînes *mots1* et *mots2*. Le verbe utilisé est « **concaténer** ».
`mots * nb`	Recopie *nb* fois la chaîne *mots*.

Comme pour les listes, on peut parcourir les chaînes de différentes manières en réalisant une boucle `for` sur les indices ou sur les caractères.

Parcourir une chaîne de caractères :

Une boucle sur les indices :

```
for i in range(len(txt)):
    print(txt[i])
```

Une boucle sur les caractères :

```
for lettre in txt:
    print(lettre)
```

Prenons un exemple classique, on veut écrire un mot à l'envers.

- Une première idée serait d'inverser les lettres 2 à 2 (la première avec la dernière, la seconde avec l'avant-dernière....) et de s'arrêter au milieu du mot (oui... sinon, on inverse 2 fois et on revient au mot du début !). Si `mot` est la chaîne à inverser, on pourrait écrire dans la boucle `mot[i], mot[j] = mot[j], mot[i]`.

  Cette solution ne fonctionne pas car les caractères d'une chaîne de caractères ne peuvent pas être modifiés. La commande `mot[i]` permet donc de connaître (lecture) la lettre d'indice `i` du mot, mais pas de la modifier (écriture).

- La seconde idée qui, elle, va fonctionner, est d'utiliser une variable qui va nous servir à construire le mot retourné étape par étape par concaténation. En programmation, on appelle une telle variable un **accumulateur**. Le principe de l'algorithme est le suivant :

- On part d'une chaîne notée `mot`.
- On crée un accumulateur `mot1` que l'on initialise comme une chaîne vide `""`.
- On parcourt chaque lettre de `mot` et on colle cette lettre au début de `mot1`.
- Ainsi à la fin de la boucle, `mot1` contiendra les lettres de `mot` placées en sens inverse.

```
mot = "bonjour"
mot1 = ""
for i in range(len(mot)):
    mot1 = mot[i] + mot1
print(mot1)
```

mot1
b
ob
nob
jnob
ojnob
uojnob
ruojnob

Plus simplement en bouclant sur les lettres :

```
mot = "bonjour"
mot1 = ""
for lettre in mot:
    mot1 = lettre + mot1
print(mot1)
```

Application 23 *Un palindrome est un mot qui peut se lire aussi bien à l'endroit qu'à l'envers comme "kayak". En s'inspirant de l'exemple précédent, écrire une fonction qui indique si un mot est un palindrome.*

Application 24 *Écrire une fonction qui renvoie les initiales d'un nom, donné en argument, avec la règle suivante :*

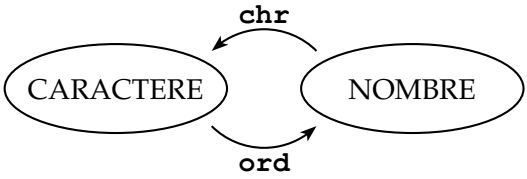
- *Vincent Maille* $\longmapsto$ `"VM"`
- *Jean-Jacques Goldmann* $\longmapsto$ `"JJG"`
- *Jean-Claude Van Damme* $\longmapsto$ `"JCVD"`

Le code ASCII

En informatique, chaque caractère est associé à un nombre appelé **code ASCII** (pour **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange). Les 128 premiers codes sont les mêmes pour toutes les machines. Par exemple le caractère 32 est l'espace, le 65 le A, ... Les caractères de 0 à 31 sont des caractères spéciaux, déplacements, retour chariot, saut de colonne... c'est pourquoi, vous ne les trouverez pas dans la table de la page suivante.

x10 \ x1	0	1	2	3	4	5	6	7	8	9
3			espace	!	"	#	$	%	&	'
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	`	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~	□	€	

On peut obtenir le code ASCII d'un caractère avec la fonction **`ord`** et, inversement, passer d'un entier au caractère correspondant à l'aide de la fonction **`chr`**.



Application 25 *En utilisant le code ASCII, écrire une fonction qui reçoit une chaîne de caractères et renvoie une chaîne dans laquelle les majuscules et minuscules ont été inversées. Par exemple la célèbre phrase de Jeff TUCHE "J'aIme LE vEau" devient "j'AiME le VeAU".*

Input et conversion de types

Pour rendre vos programmes plus interactifs, vous choisirez peut-être de demander à l'utilisateur d'entrer lui-même une valeur ou un texte à l'aide de la fonction **`input`**.

v=**input(***txt***)** : Affiche une fenêtre où figure le texte *txt* et un cadre blanc de saisie dans lequel on entrera ce qui est demandé. La réponse est alors affectée à la variable *v* sous forme d'**une chaîne de caractères**.

Un exemple qui fonctionne bien...

```
n = input("Entrer votre prénom : ")
print("Bonjour", n)
```

```
Bonjour Vincent
```

Celui-ci pose problème !

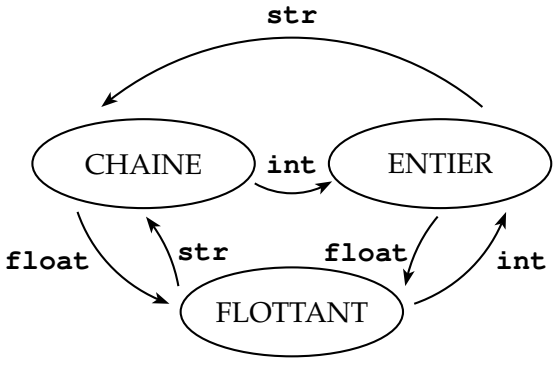
```
x = input ("Entrer un nombre")
print("Le double de", x, "est", 2 * x)
```

```
Le double de 5 est 55
```

Pour le deuxième exemple, le souci provient du fait que la variable `x` est une chaîne de caractères. Doubler une chaîne de caractères revient pour Python à la répéter deux fois comme nous l'avons vu précédemment. Il faut donc convertir `x` en variable numérique. Cela peut se faire soit à l'aide de la fonction `int` si on veut convertir en entier, soit avec la fonction `float` pour transformer en flottant.

```
x = int(input("Entrer un nombre entier"))
print("Le double de", x, "est", 2 * x)
```

```
Le double de 5 est 10
```



Attention : n'abusez pas de la fonction **`input`** même si elle donne un bel aspect à votre programme. En effet notre objectif est quand même la résolution de problèmes et vous aurez ainsi plus vite fait d'écrire des fonctions et de les appeler avec les bons paramètres dans la console (en utilisant la flèche du haut pour remonter dans l'historique) que de saisir à chaque exécution les variables pour tester le code.

À noter enfin que la fonction **`eval`** peut évaluer des expressions complexes dans la mesure où elles ont un sens.

```
>>> eval('2 * 3 + 1')
7
>>> str(1+2+3)
'6'
>>> eval('2*a')
Traceback (most recent call last):
  File "<interactive input>", line 1, in <module>
  File "<string>", line 1, in <module>
NameError: name 'a' is not defined
>>> a = 3
>>> eval('2*a')
6
```

Cependant l'utilisation de cette fonction est une mauvaise habitude, car elle exécute en fait toute la chaîne de caractères avant de stocker le résultat de l'évaluation dans la variable demandée. Imaginez que sur une machine vous demandiez un nombre à l'utilisateur et que ce dernier entre une commande Python qui efface le disque dur.... Bref, en résumé l'utilisation du `input` n'est clairement pas une priorité en classe !

Application 26 *Écrire une fonction qui reçoit un texte simple (pas de mots composés, d'apostrophes...) et qui compte et renvoie le nombre de mots dans ce texte.*

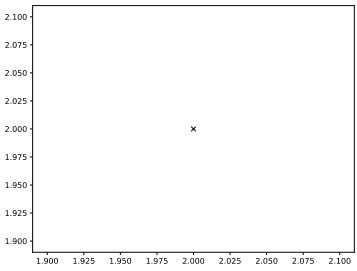
Application 27 *Écrire une fonction qui renvoie la lettre qui apparaît le plus souvent dans un texte donné.*

Les graphiques

Dans une précédente fiche, nous avons vu comment réaliser des graphiques simples à l'aide du module `turtle`, toutefois les tracés sont assez limités. En particulier, on pourrait apprécier de savoir tracer, comme sur une calculatrice, des courbes ou d'autres éléments graphiques dans un repère. C'est ce que permet le module **`matplotlib`**, plus précisément le module **`pyplot`** du package `matplotlib`.

Tracer un point

```
from matplotlib.pyplot import *
plot(2, 2, 'kx') # Tracer le point
show()           # Afficher la fenêtre
```



N'oubliez pas les parenthèses après le **`show`**, sinon, rien ne se produira...

`plot(X,Y,options)` permet :

- de placer un point de coordonnées (X, Y) si X et Y sont des nombres ;
- de tracer la courbe passant par les points (X_i, Y_i) lorsque X et Y sont des listes de même taille contenant respectivement les valeurs X_i et Y_i. La liste X vaut par défaut $[1, 2, ...n]$ si elle n'est pas renseignée.

Les options sont multiples, nous n'en présentons qu'une ici. Il s'agit d'une chaîne composée de 2 caractères, le premier représentant la couleur, le second le style de trait :

Couleur		Style	
b	bleu	-	ligne continue
g	vert	- -	tirets
r	rouge	:	pointillés
c	cyan	.	points
w	blanc	,	pixels
m	magenta	o	billes
y	jaune	x	croix
k	noir	-.	points reliés

APPLICATION 28 *Quel tracé est effectué par ce programme ?*

```
from matplotlib.pyplot import *
X, Y = [1, 1, 2, 3, 3], [1, 4, 2, 4, 1]
plot(X, Y, 'r:')
axis([0, 4, 0, 5])
show()
```

Tracer la courbe d'une fonction

Soit $f : x \longmapsto x^2-3x+1$, on souhaite afficher sa courbe représentative sur l'intervalle $[-2;3]$.

```
import matplotlib.pyplot as plt

def f(x) :
    return x**2-3*x+1
```

1ère solution

On part de deux listes vides, une pour les abscisses et l'autre pour les ordonnées. On complète alors les listes à l'aide d'une boucle.

```
X, Y = [], []
x = -2
while x <= 3 :
    X.append(x)
    Y.append(f(x))
    x = x + 0.01

plt.plot(X,Y)
plt.show()
```

2ème solution

Comme pour chaque tracé de courbe sur un intervalle $[a;b]$ il faudra prendre un grand nombre de points dans cet intervalle, on peut créer une fonction pour automatiser cela, comme dans l'application ci-dessous.

APPLICATION 29 *Écrire une fonction* `decoupe(a,b,n)` *qui "découpe" l'intervalle* $[a;b]$ *en* n *points équirépartis* ($n \geqslant 2$). *Par exemple :*

```
>>> decoupe(-2, 7, 5)
[-2.0, 0.25, 2.5, 4.75, 7.0]
```

On peut alors utiliser cette fonction pour générer la liste des abscisses, puis à partir de cette dernière celle des ordonnées. Ce qui permet de facilement réaliser le tracé.

```
X = decoupe(-2, 3, 1000)
Y = [f(x) for x in X]
plt.plot(X,Y)
plt.show()
```

3ème solution

Une fonction similaire à la fonction `decoupe` précédemment étudiée existe déjà dans le module **`numpy`** qui fait l'objet d'une fiche plus loin dans le livre. Il s'agit de la fonction **`linspace`**.

```
from numpy import linspace

X = linspace(-2, 3, 1000)
Y = [f(x) for x in X]
plt.plot(X,Y)
plt.show()
```

Attention : cette fonction ne renvoie pas exactement une liste Python, mais son usage est très ressemblant.

D'autres éléments de mise en page

Par défaut, les axes s'adaptent aux tracés réalisés en prenant les valeurs extrêmes des coordonnées des points tracés. On peut forcer les changements :

`axis` permet de régler les axes du repère de différentes manières.

Fonction	Effet
`axis()`	Renvoie une liste sous la forme $[xmin, xmax, ymin, ymax]$ contenant les extrema de la fenêtre de tracé.
`axis(L)`	Modifie les limites de la fenêtre de tracé. L est une liste de la forme $[x_{min}, x_{max}, y_{min}, y_{max}]$.
`axis(xmin=`m**`)`**	Modifie la valeur de $xmin$ sans toucher aux autres (idem avec $xmax, ymin$ et $ymax$).
`axis('equal')`	Force le repère à être orthonormal (c'est mieux surtout si on trace des cercles !)

Attention, la fonction **`axis`** doit être appelée après les **`plot`** et avant le **`show()`**.

Enfin, on peut ajouter un titre au graphique et aux axes pour préciser les quantités représentées et faciliter la compréhension.

Fonction	Effet
`xlabel(`*txt*`)`	Permet d'ajouter le titre *txt* à l'axe des abscisses
`ylabel(`*txt*`)`	(respectivement celui des ordonnées).
`title(`*txt*`)`	Permet de donner le titre *txt* au graphique.
`grid()`	Dessine une grille sur le repère.

Ce dernier exercice peut être abordé dès la classe de seconde avec le sinus et le cosinus d'un réel.

Application 30 *Réaliser une fonction qui demande un entier n ($n > 2$) et trace le polygone régulier à n côtés, de centre O et de rayon 5 en bleu ainsi que son cercle circonscrit en pointillés noirs.*

Les nombres complexes

Nativement et sans module particulier, Python connaît les nombres complexes . Ces derniers sont affichés sous la forme $a + bj$ où j est le nombre complexe de module 1 et d'argument $\pi/2$ comme en physique ou en électricité, mais que les mathématiciens notent i...
Attention lors de la saisie d'un nombre, penser à mettre toujours un coefficient devant j (éventuellement 1).

```
>>> z = 1-2j
>>> z
(1-2j)
>>> z  = complex(1,1)
>>> z
(1+1j)
>>> type(z)
<class 'complex'>
>>> abs(z)
1.4142135623730951
>>> z.conjugate()
(1-1j)
>>> z.imag
1.0
>>> z.real
1.0
```

Fonctions de base :

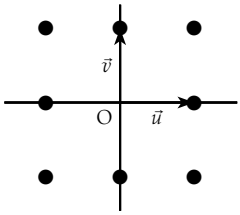
Commande	Effet
`complex(`a`,`b`)`	Renvoie le nombre complexe $a + ib$.
`abs(`z`)`	Renvoie le module de z.
z`.conjugate()`	Renvoie le conjugué de z.
z`.real` z`.imag`	Renvoient respectivement la partie réelle et la partie imaginaire du nombre complexe z.

À noter la présence de parenthèses pour le calcul du conjugué car il s'agit d'une **méthode** (c'est à dire une fonction agissant sur un **objet**), alors qu'il n'y en a pas lorsque l'on demande la partie réelle ou la partie imaginaire qui sont des **attributs** de l'objet z.

APPLICATION 31 *Soit z, un nombre complexe non nul, on appelle* ***argument principal*** *de z l'unique réel θ tel que :*

$$\begin{cases} \theta \in]-\pi;\pi], \\ |z|\cos\theta = Re(z), \\ |z|\sin\theta = Im(z) \end{cases}.$$

Vous aurez remarqué nous n'avons pas parlé de fonction qui donne l'argument d'un nombre complexe... Et pour cause, il n'en existe pas de toute faite sans importer de module.... Vous devez ainsi programmer une fonction `arg` *qui reçoit en entrée un nombre complexe non nul et retourne son argument principal. On testera cette fonction pour les 8 nombres complexes dont les images sont représentées ci-contre.*



APPLICATION 32 *La fonction* `exp` *du module* `math` *ne supporte pas les nombres complexes. Écrire une fonction* `expC` *qui reçoit un nombre complexe z et renvoie le complexe e^z.*

AU DÉTOUR DE LA BALADE...

À noter enfin qu'il existe le module ***`cmath`*** *que nous ne détaillerons pas ici et qui permet de gagner du temps : calculer une exponentielle complexe, un argument, passer de la forme algébrique à la forme trigonométrique...*

Les matrices

Le module présenté ici n'est pas un attendu des programmes officiels de mathématiques du lycée, cependant il peut être exploité au travers de l'option mathématiques expertes et va bien nous simplifier la vie.

Un premier problème réglé par `numpy`

Les listes en Python ne sont pas facilement manipulables pour travailler avec des vecteurs du plan et de l'espace. Par exemple,

si $\vec{u}\begin{pmatrix}1\\2\end{pmatrix}$, $\vec{v}\begin{pmatrix}2\\-1\end{pmatrix}$ et $\vec{w}=\vec{u}+\vec{v}$, on sait que $\vec{w}\begin{pmatrix}3\\1\end{pmatrix}\neq\begin{pmatrix}1\\2\\2\\-1\end{pmatrix}$.

```
>>> u = [1, 2]
>>> v = [2, -1]
>>> w = u + v
>>> w
[1, 2, 2, -1]
```

En effet, comme on l'a vu dans les précédentes fiches, l'opérateur + concatène les listes et n'ajoute pas les éléments entre-eux.

Le module **`numpy`** va permettre de créer de nouveaux objets appelés **tableaux** (ou **vecteurs** lorsqu'ils ont une seule dimension).

Création d'un vecteur :

Commande	Effet
`array(`*list*`)`**	Crée et renvoie un vecteur à partir de la liste *list*. Exemple : `array([1,2,3])` pour un vecteur ligne.
`linspace(`*deb*, *fin*, *nb*`)`**	Génère un vecteur de *nb* valeurs équidistantes de l'intervalle $[deb, fin]$ (par défaut, *nb* vaut 50 s'il n'est pas précisé). Exemple : `linspace(2.4,3.5,3)` donne `[2.4 2.95 3.5]`

Les opérations (qui se font coordonnées par coordonnées) :

Attention, la dernière opération n'a aucun sens mathématique !

```
>>> from numpy import array
>>> v1 = array([1, 2, 3])
>>> v2 = array([2, -1, 1])
>>> v1 + v2
[3 1 4]
>>> 3 * v1
[3 6 9]
>>> v1 * v2
[ 2 -2 3]
```

Définition d'une matrice comme une liste de listes

Pour déclarer un tableau à 2 dimensions, on peut avoir recours à une liste de listes. Par exemple pour représenter la matrice A :

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix}$$

```
>>> A = [[1,2,3],[4,5,6],[7,8,9],[10,11,12]]
>>> A
[[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]]
```

- A est une liste de 4 éléments. Ainsi `len(A)` renvoie le nombre de lignes de la matrice A.

$$A = [\ \underbrace{[1,2,3]}_{A[0]},\ \underbrace{[4,5,6]}_{A[1]},\ \underbrace{[7,8,9]}_{A[2]},\ \underbrace{[10,11,12]}_{A[3]}\ ]$$

- A[0] est à son tour une liste, elle correspond à la ligne de rang 0 : [1,2,3]. Ainsi
  - `len(A[0])` renvoie le nombre de colonnes de la matrice A
  - `A[1][2]` renvoie le nombre placé à la ligne 1 et colonne 2 (même ordre qu'en mathématiques).

```
>>> len(A)
4
>>> len(A[0])
3
>>> ligne = A[1]
>>> ligne
[4, 5, 6]
>>> ligne[2]
6
>>> A[1][2]
6
```

APPLICATION 33 *Pour créer la matrice* $A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$, *un élève a tapé dans la console :*

```
>>> A = [[0]*3]*2
>>> A[0][0] = 1
```

1. *Évaluer ce que vaut* `A`*, puis comprendre et corriger le problème.*

2. *En déduire une fonction correcte* `nulle(n,p)` *qui crée et renvoie une matrice nulle de taille* $n \times p$.

Application 34 *En utilisant l'exercice précédent, programmer les fonctions suivantes où* `A` *et* `B` *sont deux matrices :*

1. `somme(A, B)` *: qui renvoie la somme de* `A` *et de* `B`

2. `produit(A, B)` *: qui renvoie le produit de* `A` *et de* `B`

Un deuxième problème réglé par numpy

De la même manière que les listes ne sont pas idéales pour représenter les vecteurs dans le plan et l'espace, les listes de listes ne sont pas trop adaptées à la représentation des matrices. Heureusement, le module `numpy` va encore bien nous simplifier le travail !

Commande	Effet
`array(`*list***`)`**	Crée un vecteur à partir de la liste *list*. Exemple `array([[1,2],[3,4]])` pour le tableau $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$
`zeros([`*n*`,`***p*`])`**	Crée un tableau nul de taille $n \times p$.
`eye(`*n*`)`**	Crée un tableau identité de taille $n \times n$.
`+, -, *, /`	Opérations terme à terme sur 2 tableaux de même taille.
`dot(``A, B`**`)`**	Renvoie le produit matriciel A × B.
`shape(``A`**`)`**	Renvoie la taille du tableau A sous forme d'un couple d'entier.

Attention : l'opérateur * réalise, comme les autres, des opérations terme à terme... d'où l'utilité de la fonction **`dot`**

Enfin, on peut toujours accéder et modifier le coefficient de la ligne `i` et colonne `j` d'une matrice `A` en utilisant `A[i][j]`, mais on a également d'autres possibilités :

Commande	Effet
A**[**i**,** j**]**	Élément $A_{i,j}$.
A**[:,** j**]**	Vecteur ligne qui contient la $j^{\text{ème}}$ colonne de A
A**[**i**,:]**	Vecteur ligne qui contient la $i^{\text{ème}}$ ligne de A.

Ces données sont accessibles en lecture/écriture.

Aides pour les exercices des fiches

Aide pour l'application 1 Pensez qu'il peut aussi y avoir des divisions...

Aide pour l'application 8 Si $\forall n \in \mathbb{N},\ u_{n+1} = 2u_n - 1$ alors, $\forall n \in \mathbb{N}^*,\ u_n = 2u_{n-1} - 1$

Aide pour l'application 10 Utilisez les opérateurs `and` et `or` puis étudiez la parité des mois qui nous intéressent.

Aide pour l'application 11 Il suffit d'exécuter ce programme à la main pour avoir la réponse.

Aide pour l'application 12 Pour chacun des cas, on pourra compléter un tableau ressemblant à celui-ci :

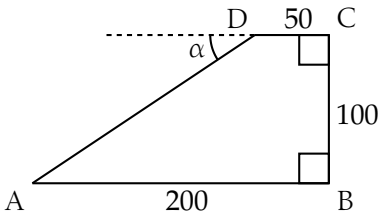
u	L	n	$u \leqslant$ L ?
…	…	…	…

Aide pour l'application 14 Pour comprendre ce que fait le programme, prenez quelques valeurs de n et regardez comment évoluent les variables `i` et `s` au cours de la boucle.

Aide pour l'application 15 On peut utiliser un tableau de proportionnalité :

radians	π	⋯
degrés	180	⋯

Aide pour l'application 19 Il vous faudra calculer la longueur AD et la mesure de l'angle α.



Aide pour l'application 21 Pour les 3 premières fonctions, le principe est le même :

- On parcourt la liste
- On traite chaque élément selon ce qui est demandé

Aide pour l'application 24 On peut parcourir le mot et garder en mémoire toutes les lettres qui se trouvent au début, après un trait d'union ou un espace.

Aide pour l'application 25 Utiliser le code ASCII des lettres et remarquer que la différence entre les lettres majuscules et minuscules est constante.

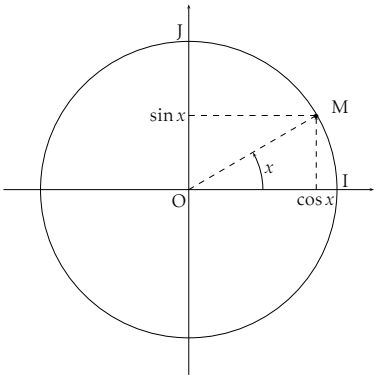
Aide pour l'application 26 Le nombre d'espaces donne une bonne information sur le nombre de mots...

Aide pour l'application 27 Avec l'exercice précédent, on a vu comment compter le nombre d'espaces. On peut facilement généraliser cela à n'importe quel caractère. Dans un second temps on va tester une à une les lettres pour trouver la plus fréquente.

Aide pour l'application 29 Remarquez la nature de la suite formée par l'ensemble des valeurs cherchées.

Aide pour l'application 30

Pensez que si, dans un repère orthonormé (O, I, J), un point M est sur le cercle de rayon 1 tel que $\left(\overrightarrow{OI}, \overrightarrow{OM}\right) = x$, alors l'abscisse de M vaut $\cos x$ et son ordonnée $\sin x$.
Pour être sur le cercle de rayon 5, il suffit de multiplier les coordonnées par 5...
Pour le cercle... un polygone à 100 côtés fera largement l'affaire !



Aide pour l'application 31 Pensez à bien vérifier pour les 8 nombres proposés car la fonction `acos` donne toujours un résultat dans $[0; \pi]$!

Aide pour l'application 33 Taper le code dans https://pythontutor.com

Solutions des exercices des fiches

Correction de l'application 1 Une solution : `(5 // 5 + 5 ) * 5 + 5`.
Une autre : `( ( 5 + 5 ) // 5 + 5) * 5 .`
Encore une : `55 - 5 * 5 + 5`.

Correction de l'application 2

A. `4.0` B. `2.0` C. `9.0` D. `3.0`

Correction de l'application 3

A. `3`
B. `0`
C. `27`
D. `-3`
E. `4`
F. `8`
G. `0.25`
H. `-1024`
I. `-9`
J. `6.5`
K. `2`
L. `-2`

Au détour de la balade...

La définition du quotient et du reste utilisée par Python diffère de celle que l'on utilise habituellement, en particulier pour les entiers négatifs. En mathématiques, le quotient et le reste d'une division euclidienne sont définis ainsi :

> ***SI*** *a et b sont deux nombres entiers relatifs et b non nul* ***ALORS****, il existe un unique couple d'entiers q et r vérifiant :*
>
> $$\begin{cases} a = b \times q + r \\ 0 \leqslant r < |b| \end{cases}$$

Correction de l'application 4

1. À la fin c'est `x` qui vaut 32 et `y` qui vaut 17.
2. Pour toutes valeurs de `x` et `y` de départ, le programme inverse ces deux valeurs. Cela peut se démontrer, en effet, notons x_0 et y_0 les valeurs de départ de `x` et `y` :

Ligne	Instruction	x	y
3	`x = x + y`	$x_0 + y_0$	y_0
4	`y = x - y`	$x_0 + y_0$	$x_0 + y_0 - y_0 = x_0$
5	`x = x - y`	$x_0 + y_0 - x_0 = y_0$	x_0

3. Si on essaie le code
```
x = y
y = x
```
cela ne fonctionne pas car à la première ligne on écrase la valeur de `x`, il faut donc utiliser une variable temporaire ici nommée `t` pour conserver cette valeur en mémoire :
```
t = x
x = y
y = t
```

Correction de l'application 5 À la fin a vaut 3 et b vaut 2 :

Ligne	a	b
`a, b = 3, 2`	3	2
`a = a - b`	1	2
`b, a = a + 1, 2 * b`	4	2
`b = b % 3`	4	2
`a = a - b // 2`	3	2

Pour la seconde suite d'instructions, à la fin x, y et z valent tous 2 :

Ligne	x	y	z
`x, y, z = 1, 2, 3`	1	2	3
`x, y = y, z`	2	3	3
`y, z = z, x`	2	3	2
`y = z`	2	2	2
`z = y`	2	2	2

Correction de l'application 6 On sait que le volume d'un cylindre se calcule avec la formule $\mathcal{V} = \pi \times r^2 \times h$ et que l'aire vaut $\mathcal{A} = \underbrace{2\pi \times r \times h}_{\text{aire latérale}} + \underbrace{2\pi \times r^2}_{\text{aire disques}}$.

Ainsi on peut écrire les 3 fonctions demandées ainsi :

```
pi = 3.14

def volume(r, h):
    return pi * r**2 * h

def aire(r, h):
    return 2 * pi * r * h + 2 * pi * r**2

def cylindre(r, h):
    return aire(r, h), volume(r, h)
```

Correction de l'application 7 Il suffit de taper le code demandé :

1. `arrondi(0.3)` = `arrondi(1.9)` = −1.
   De même `arrondi(-3.1)` = `arrondi(-3.9)` = −1.

2. On peut légitimement conjecturer que $\forall x \in \mathbb{R}$, `arrondi(x)` = −1.

3. `arrondi(0.5)` = −2 et `arrondi(1.5)` = 0 !

Pour comprendre ce comportement surprenant, regardons ce que renvoie la fonction `round` sur quelques demi-entiers :

`round(-0.5)` $\longmapsto$ 0	`round(0.5)` $\longmapsto$ 0
`round(1.5)` $\longmapsto$ 2	`round(2.5)` $\longmapsto$ 2
`round(3.5)` $\longmapsto$ 4	`round(4.5)` $\longmapsto$ 4
`round(5.5)` $\longmapsto$ 6	`round(6.5)` $\longmapsto$ 6
`round(7.5)` $\longmapsto$ 8	`round(8.5)` $\longmapsto$ 8

L'arrondi de 3,5 est bien 4, mais l'arrondi de 4,5 est 4 aussi ! Alors qu'avec la définition mathématique, il devrait être de 5.... et cela est dû.... à la finance ! Si on arrondit toujours avec la définition mathématique, tous les montants se terminant à 0,5 sont arrondis à l'unité supérieure entrainant ainsi des cumuls d'erreurs importants à la fin. Avec cette méthode les montants sont arrondis parfois au dessus, parfois en dessous, limitant ainsi les effets de seuil. En résumé, si le chiffre avant le 5 est impair on arrondit à l'entier supérieur et s'il est pair on arrondit à l'entier inférieur et inversement pour les négatifs ! Et c'est pareil pour les autres chiffres par exemple `round(2.65,1)`=`2.6` et `round(2.75,1)` = `2.8`

Correction de l'application 8 Il est plus simple d'écrire u_n en fonction de u_{n-1} pour rédiger la fonction demandée : $\forall n \in \mathbb{N}^*$, $u_n = 2u_{n-1} - 1$. Ainsi on a la fonction :

```
def u(n):
    if n == 0:
        return 2
    else:
        return 2 * u(n-1) - 1
```

Correction de l'application 9 Voici une fonction répondant au problème :

```
def majeur(age) :
    if age >= 18:
        return True
    return False
```

À noter qu'il n'y a pas besoin ici d'instruction `else` : si la personne est majeure l'interpréteur Python rencontre le `return True` et quitte la fonction, sinon il continue et renvoie `False`. Voici une seconde version plus courte (mais peut-être moins claire dans un premier temps) :

```
def majeur(age) :
    return age >= 18
```

En effet dans cette dernière version, on renvoie directement le résultat du test `age >= 18` qui est déjà un booléen !

Correction de l'application 10

1. Voici une première fonction : les mois qui comportent 31 jours sont : Janvier(1), Mars(3), Mai(5), Juillet(7), Août(8), Octobre(10) et Décembre(12).

```
def jours(n) :
    if n == 1 or n == 3 or n == 5 or n == 7 or n == 8 or n == 10 ↙
        ↳ or n == 12:
        return True
    else:
        return False
```

2. Essayons de simplifier le précédent test : on remarque que les mois cherchés sont les mois impairs jusqu'à 7 puis les mois pairs après 8, on pourrait donc faire ainsi :

```
def jours(n) :
    if (n < 8 and n % 2 == 1) or (n >= 8 and n % 2 == 0):
        return True
    else:
        return False
```

3. On peut encore améliorer ce test. L'idée est d'ajouter 1 au numéro de mois après 8. Ainsi les mois qui nous intéresseront seront les mois impairs. Pour cela on va ajouter 1 au numéro du mois lorsque ce numéro est supérieur ou égal à 8. Comme on n'a droit qu'à un seul `if`, on va réaliser une division du numéro du mois par 8, le quotient vaut 0 ou 1 selon que l'on est avant ou après le mois d'août. Finalement, on obtient le code :

```
def jours(n) :
    return (n + n // 8) % 2 == 1:
```

Correction de l'application 11 n prend successivement les valeurs :
1 - 2 - 4 - 8 - 16 - 32 - 64 - 128 (qui est la première à dépasser 100).

Ainsi le programme affiche 128.
En Python, on obtient :

```
n = 1
while n <= 100:
    n = 2 * n
print(n)
```

Correction de l'application 12 Comme souvent, on peut présenter le déroulement de l'algorithme sous forme d'un tableau :

1.

u	L	n	$u \leqslant$ L ?
2	10^4	0	OUI
4	10^4	1	OUI
16	10^4	2	OUI
256	10^4	3	OUI
65536	10^4	4	NON

`mystere(2,10**4)` renvoie 4.

2.

u	L	n	$u \leqslant$ L ?
3	81	0	OUI
9	81	1	OUI
81	81	2	OUI
6561	81	3	NON

`mystere(3,81)` renvoie 3.

3.

u	L	n	$u \leqslant$ L ?
1	0	0	NON

`mystere(3,81)` renvoie 0.

4.

u	L	n	$u \leqslant$ L ?
0	1	0	OUI
0	1	0	OUI
…	…	…	…

... Aucune des variables ne change, et la condition du `while` est toujours vérifiée, le programme ne s'arrête pas.

AU DÉTOUR DE LA BALADE...

À noter que cet exercice peut être donné dès la seconde pour permettre de comprendre les boucles et la différence entre $<$ et $\leqslant$.

Correction de l'application 13 Comme lorsqu'on calcule à la main, on va calculer les termes 1 à 1 :

```
def u(n):
    terme = 2
    for k in range(n): # k va de 0 à n-1 (n tours)
        terme = 2 * terme - 1
    return terme
```

Correction de l'application 14

1. La traduction de l'algorithme (attention au `n+1`) :

```
def calcule(n):
    s = 0
    for i in range(1, n+1) :
        s = s + i
    return s
```

2. Analysons pas à pas l'algorithme pour $n = 8$ à la fin de chaque tour de boucle :

i	s
-	0
1	1
2	3
3	6
4	10
5	15
6	21
7	28

La fonction renvoie $28 = 1 + 2 + 3 + 4 + 5 + 6 + 7$. À chaque tour de boucle on ajoute la nouvelle valeur de i. La valeur de s est donc $1 + 2 + ... + n = \sum_{i=1}^{n} i$.

En cours de spécialité NSI au lycée, on pourra démontrer la véracité de notre conjecture en utilisant un "**invariant de boucle**"... Et oui, l'informatique est une science, les résultats se démontrent !

3. Pour répondre à la question, il suffit de modifier la ligne où `s` est actualisée : la ligne `s = s + i` devient alors

```
s = s + 1 / (i ** 2).
```

Pour $n = 1000$ par exemple, on trouve $s = 1.6439345666815615$. Cette valeur est une bonne valeur approchée de $\frac{\pi^2}{6}$. En effet, on peut montrer après le bac que $\lim_{n\to+\infty} \sum_{i=1}^{n} \frac{1}{i^2} = \frac{\pi^2}{6}$.

Correction de l'application 15

1. Pour la première question, on effectue la conversion avant d'effectuer le calcul. Voici les 3 fonctions :

```
from math import sin, cos, tan, pi

def sinD(angle_degres):
    angle_radians = pi*angle_degres/180
    return sin(angle_radians)

def cosD(angle_degres):
    angle_radians = pi*angle_degres/180
    return cos(angle_radians)

def tanD(angle_degres):
    angle_radians = pi*angle_degres/180
    return tan(angle_radians)
```

on peut aussi utiliser les fonctions de conversion existantes :

```
from math import sin, radians

def sinD(angle_degres):
    angle_radians = radians(angle_degres)
    return sin(angle_radians)
```

2. Pour les réciproques, on travaille en faisant le contraire car on calcule puis on convertit :

```
from math import asin, acos, atan, pi

def asinD(x):
    angle_radians = asin(x)
    return angle_radians/pi*180

def acosD(x):
    angle_radians = acos(x)
    return angle_radians/pi*180

def atanD(x):
    angle_radians = atan(x)
    return angle_radians/pi*180
```

ou en utilisant les fonctions de conversion d'unités :

```
from math import acos, degrees

def acosD(x):
    angle_radians = acos(x)
    return degrees(angle_radians)
```

Correction de l'application 16 Il suffit d'appliquer la formule :

```
from math import factorial

def binom(n, p):
    if p <= n:
        return factorial(n) // (factorial(p) * factorial(n-p))
    else:
        return 0
```

Que l'on peut écrire en une ligne (sûrement moins lisible !) :

```
def binom(n, p):
    return factorial(n)//(factorial(p)*factorial(n-p)) if p <= n else 0
```

Correction de l'application 17 Pas de difficulté spéciale pour cet exercice, on crée un accumulateur `somme` que l'on initialise à 0, puis on ajoute tour à tour les faces des dés :

```
from random import randint

def sommeDes(n):
    somme = 0
    for _ in range(n):
        de = randint(1,6)    # Tirage d'un dé
        somme = somme + de   # Ajout à la somme
    return somme
```

AU DÉTOUR DE LA BALADE...

le nom de variable _ (appelé underscore à savoir tiret bas) dans le `for` est un nom de variable comme les autres, mais il indique par convention au lecteur du programme que cette variable n'est pas utilisée.

Correction de l'application 18 On choisit aléatoirement un nombre dans l'ensemble $\{1;2\}$ par exemple. Arbitrairement, on décide que le nombre 1 représentera Pile et 2 représentera Face.

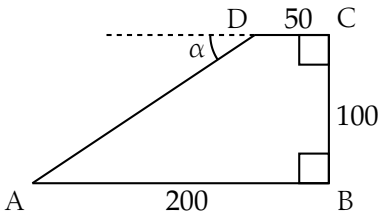
```
from random import randint
def nbpile(n):
    nb = 0
    for i in range(n):
        if randint(1,2) == 1 : # Pile par choix
            print("Pile")
            nb = nb + 1
        else :
            print("Face")
    return nb
```

AU DÉTOUR DE LA BALADE...

On peut aussi remplacer la condition `randint(1,2) == 1` *par* `rand() < 0.5`*. Cette seconde méthode permet d'ailleurs facilement d'adapter la modélisation pour des pièces non équilibrées.*

Correction de l'application 19

En utilisant la figure donnée en aide :

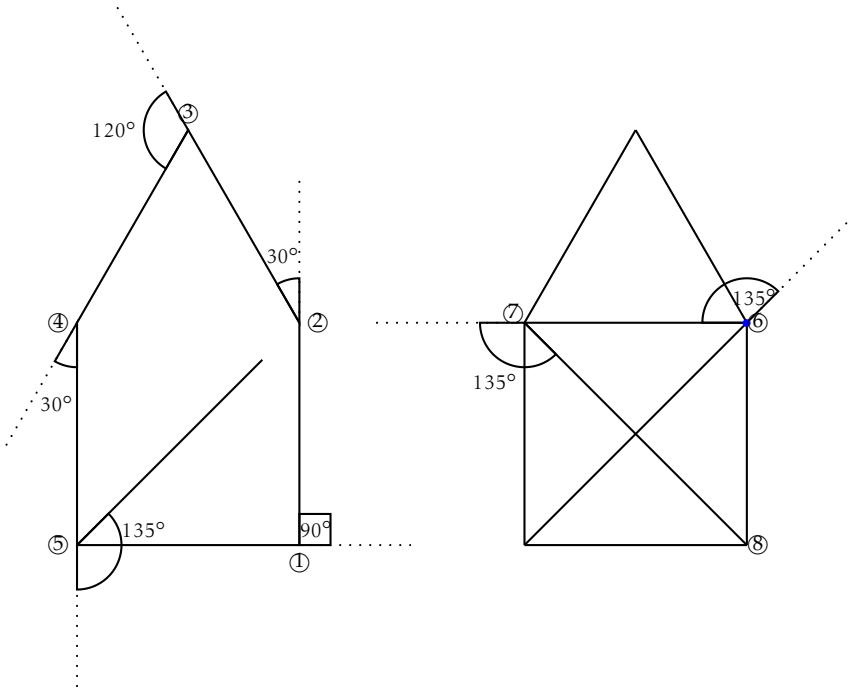


- Avec le théorème de Pythagore, $AD = \sqrt{150^2 + 100^2} = 50\sqrt{13}$.
- Les formules de trigonométrie donnent $\tan\alpha = \dfrac{100}{150} = \dfrac{2}{3}$.

On peut obtenir une valeur approchée de α avec la calculatrice ($\alpha \approx 33,69°$) ou avec Python :

```
from turtle import *
from math import *

AB = 50 * sqrt(13)
alpha = degrees(atan(2/3))
forward(200)
left(90)
forward(100)
left(90)
forward(50)
left(alpha)
forward(AB)

mainloop()
```

Correction de l'application 20 Parfois, une figure vaut mieux que de longs discours...



Voici un programme qui répond à la question :

```
from turtle import *
from math import *

l = 100
forward(l)
left(90)
forward(l)
left(30)
forward(l)
left(120)
forward(l)
left(30)
forward(l)
left(135)
forward(l*sqrt(2))
left(135)
forward(l)
left(135)
forward(l*sqrt(2))
hideturtle()

mainloop()
```

Correction de l'application 21

1. Pour la fonction `present`, on parcourt la liste, dès que l'on trouve l'élément cherché on renvoie `True`. Si à la fin du parcours on est toujours à l'intérieur de la fonction, c'est que l'on n'a pas trouvé l'élément et on renvoie `False` :

```
def present(L, e):
    for elem in L:
        if elem == e:
            return True
    return False
```

2. Pour la fonction `minimum`, c'est la même idée : on parcourt la liste, on compare chaque élément avec le plus petit trouvé jusque là. Si cet élément est encore plus petit, on met à jour la valeur du plus petit trouvé :

```
def minimum(L) :
    petit = L[0]
    for e in L :
        if e < petit:
            petit = e
    return petit
```

   Remarquer qu'il est important d'initialiser `petit` à `L[0]` (premier élément de la liste) plutôt qu'à la valeur `0`. Regardez ce qu'il se passerait en initialisant à `0` avec la liste `[-1,-3]`.

   Pour le maximum, c'est exactement la même chose, il y a juste un < à changer en > (ici on a aussi changé le nom de la variable pour plus de cohérence) :

```
def maximum(L) :
    grand = L[0]
    for e in L :
        if e > grand:
            grand = e
    return grand
```

   Enfin pour la fonction `choice`, on va choisir un rang aléatoirement entre le premier (rang `0`) et le dernier élément (rang `len(L)-1`) :

```
def choix(L):
    n = len(L)
    rang = randint(0, n-1)
    return L[rang]
```

ou en une ligne :

```
def choix(L):
    return L[randint(0, len(L)-1)]
```

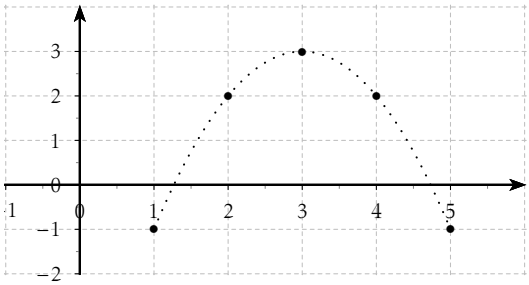
Correction de l'application 22

1.

Instruction	Affichage	P
`P = [2, 4, 6, 8]`		[2,4,6,8]
`print(P[2])`	6	[2,4,⑥,8]
`P.append(3)`		[2,4,6,8,2]
`print(P[2])`	6	[2,4,⑥,8,3]
`P.remove(3)`		[2,4,6,8]
`print(P)`	[2,4,6,8]	[2,4,6,8]
`print(len(P))`	4	[2,4,6,8]

2.

x	f
	[]
1	[-1]
2	[-1,2]
3	[-1,2,3]
4	[-1,2,3,2]
5	[-1,2,3,2,-1]



Finalement, le programme affiche -1 (pour le minimum) et 3 pour le maximum. Remarquons qu'ici ces 2 valeurs sont bien le maximum et le minimum de f sur l'intervalle $[1,5]$ car les extrémums sont atteints pour des valeurs entières ce qui n'est évidemment pas toujours le cas ...

3.

i	L	$i \in$ L ?	L
0	[0]	OUI	[]
1	[1]	OUI	[]
2	[4]	NON	[4]
3	[4, 9]	NON	[4,9]
4	[4, 9, 16]	OUI	[9, 16]
5	[9, 16, 25]	NON	[9, 16, 25]

À la fin de la boucle, le programme affiche $[9, 16, 25]$.

4.

i	print...	L.append...	L.pop...
0	1	[①,1,2]	[1,2]
1	1	[①,2,3]	[2,3]
2	2	[②,3,5]	[3,5]
3	3	[③,5,8]	[5,8]
4	5	[⑤,8,13]	[8,13]
5	8	[⑧,13,21]	[13,21]

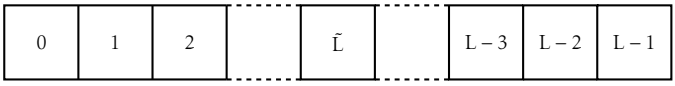
Le programme affiche "1 ; 1 ; 2 ; 3 ; 5 ; 8 ;" qui sont en réalité les premiers termes de la suite de Fibonacci définie par $u_0 = u_1 = 1$ et pour $n \in \mathbb{N}$, $u_{n+2} = u_{n+1} + u_n$ que l'on retrouve dans de nombreux problèmes d'évolution.

Correction de l'application 23 La solution la plus simple, est d'utiliser le morceau de code du programme inversant les lettres d'un mot. Ensuite, il suffit de renvoyer le résultat du test de comparaison du mot de départ avec celui obtenu.

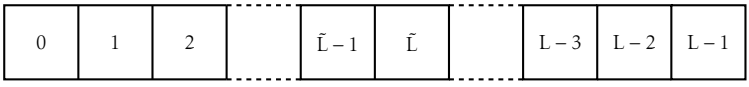
```
def palindrome(mot):
    mot1 = ""
    for lettre in mot:
        mot1 = lettre + mot1
    return mot == mot1
```

Cette méthode fonctionne mais pourrait être améliorée car on doit retourner tout le mot avant de répondre à la question, ce qui peut prendre du temps. Une autre technique serait de comparer les lettres 2 à 2 : la première et la dernière, la seconde et l'avant-dernière jusqu'au milieu du mot. Dès que 2 lettres diffèrent, on arrête le test grâce au `return`. Il faut réfléchir en distinguant le cas où le mot comporte un nombre pair de lettres et le cas où ce nombre est impair. On note L la longueur du mot et $\tilde{L}$ le quotient (entier) de L par 2, on obtient :

Cas où L est impair :



Cas où L est pair :



Dans les deux cas, on peut arrêter le test à la lettre $\tilde{L} - 1$. Ce qui pourrait donner la fonction de la page suivante.

```
def palindrome(mot):
    L = len(mot)
    for i in range(L//2) :
        if mot[i] != mot[L-1-i]:
            return False
    return True
```

Correction de l'application 24 Cette première version parcourt le mot à partir de la lettre d'indice 1 et teste à chaque fois si le caractère précédent est un espace ou un trait d'union (c'est pour cela que l'on commence à 1).

```
def initiales(nom):
    reponse = nom[0]          # Il faut au moins prendre la première lettre
    for i in range(1, len(nom)):
        if nom[i-1] == ' ' or nom[i-1] == '-':
            reponse = reponse + nom[i]
    return reponse
```

Une autre idée peut être d'utiliser une variable `suivant` qui indique si on doit garder ou non la lettre suivante :

```
def initiales(nom):
    reponse = ""
    suivant = True
    for lettre in nom:
        if suivant:
            reponse = reponse + lettre
            suivant = False
        if lettre == ' ' or lettre == '-':
            suivant = True
    return reponse
```

Correction de l'application 25 Comme indiqué dans l'aide, on s'aperçoit que l'écart entre le code ASCII d'une majuscule et de la minuscule correspondante est de 32. On peut donc réaliser le programme suivant :

```
def inverse(phrase):
    reponse = ""
    for lettre in phrase:
        a = ord(lettre)
        if 65 <= a <= 90:         # C'est une majuscule
            a = a + 32
        elif 97 <= a <= 122:      # C'est une minuscule
            a = a - 32
        reponse = reponse + chr(a)
    return reponse
```

Pour information, il existe déjà des méthodes en Python pour convertir une chaîne en majuscule ou en minuscule, il s'agit des méthodes **`upper`** et **`lower`**. On pouvait donc faire comme dans le programme qui suit,

mais c'est d'une part pédagogiquement bien moins intéressant à programmer, d'autre part, l'utilisation de ces fonctions s'appelle de la **programmation objet** sur l'objet `phrase`. Cette notion ne s'inscrit pas dans les programmes du lycée.

```
def inverse(phrase):
    reponse = ""
    for lettre in phrase:
        if lettre == lettre.upper():    # C'est une majuscule
            reponse = reponse + lettre.lower()
        else:
            reponse = reponse + lettre.upper()
    return reponse
```

Correction de l'application 26 Une solution est donc de compter les espaces :

```
def nbMots(phrase):
    nbMots = 1
    for lettre in phrase:
        if lettre == ' ':
            nbMots = nbMots + 1
    return nbMots
```

Correction de l'application 27 En s'appuyant sur la remarque donnée en indication, on peut dans un premier temps modifier la fonction de l'exercice précédent pour compter le nombre d'occurrences d'un caractère quelconque :

```
def nb_car(phrase, car):
    nbCar = 0
    for lettre in phrase:
        if lettre == car:
            nbCar = nbCar + 1
    return nbCar
```

On peut alors parcourir la phrase et compter le nombre d'occurrences de chaque lettre pour trouver celle qui apparait le plus souvent :

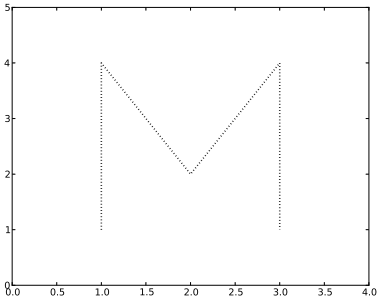
```
def lettre_maxi(phrase):
    nb_max, lettre_max = 0, "?"
    for lettre in phrase:
        nb = nb_car(phrase, lettre)
        if nb > nb_max:
            nb_max = nb
            lettre_max = lettre
    return lettre_max
```

Remarque : vous risquez d'obtenir que c'est l'espace le caractère le plus fréquent ! Vous pouvez donc modifier le `if` ainsi :

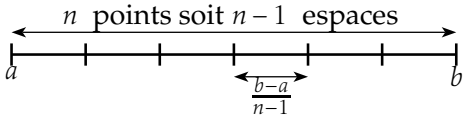
```
if nb > nb_max and lettre != " ":
```

Correction de l'application 28

Le programme permet de construire une ligne brisée reliant les points $A_1(1;1)$, $A_2(1;4)$, $A_3(2;2)$, $A_4(3;4)$ et $A_5(3;1)$ en rouge et en pointillés. On obtient un "M".



Correction de l'application 29 Une petite figure peut aider à réfléchir :



Partant de a, on ajoute à chaque point $p = \dfrac{b-a}{n-1}$:

```
def decoupe(a, b, n):
    p = (b-a) / (n-1)
    L = []
    for k in range(n) :
        L.append(a+k*p)
    return L
```

Ou en une ligne, en utilisant une liste en compréhension :

```
def decoupe(a, b, n):
    return [a+k*(b-a)/(n-1) for k in range(n)]
```

Correction de l'application 30 Pour créer le polygone à n côtés, on génère $n+1$ points (pour fermer le polygone), $A_k\left(5\cos\dfrac{2k\pi}{n};5\sin\dfrac{2k\pi}{n}\right)$ avec $k \in [\![0;n]\!]$ qu'il faut relier. Pour le cercle, c'est le même travail avec 100 points. On peut, soit faire un copier / coller comme dans le code de gauche, soit écrire une fonction qui trouve ici toute son utilité comme à droite !

```
import matplotlib.pyplot as plt
from math import *

def dessin(n):
    X, Y = [], []
    for k in range(n+1):
        X.append(5*cos(2*k*pi/n))
        Y.append(5*sin(2*k*pi/n))

    plt.plot(X, Y, 'b-')

    n = 100
    X, Y = [], []
    for k in range(n+1):
        X.append(5*cos(2*k*pi/n))
        Y.append(5*sin(2*k*pi/n))

    plt.plot(X, Y, 'k:')
    plt.axis('equal')
    plt.show()
```

```
import matplotlib.pyplot as plt
from math import *

def polygone(n, coul) :
    X, Y = [], []
    for k in range(n+1):
        X.append(5*cos(2*k*pi/n))
        Y.append(5*sin(2*k*pi/n))
    plt.plot(X, Y, coul)

def dessin(n):
    polygone(n, 'b-')
    polygone(100, 'k:')
    plt.axis('equal')
    plt.show()
```

Correction de l'application 31 Si $z = a + ib$ est un complexe non nul sous forme algébrique et s'écrivant $z = r(\cos\theta + i\sin\theta)$ sous forme trigonométrique alors on sait que :

$$\begin{cases} \cos\theta = \dfrac{a}{|z|} \\ \sin\theta = \dfrac{b}{|z|} \end{cases}$$

En calculant $\texttt{acos}\left(\dfrac{a}{|z|}\right)$, on obtient soit θ si $\text{Im}(z) \geqslant 0$, soit $-\theta$. Ce qui donne la fonction :

```
from math import acos

def arg(z):
    r = abs(z)
    angle = acos(z.real/r)
    if z.imag >= 0:
        return angle
    else:
        return -angle
```

Correction de l'application 32 Pas de difficulté pour cette question, il suffit d'appliquer la définition : si $z = a + ib$,
$e^z = e^{a+ib} = e^a \times e^{ib} = e^a(\cos b + i \sin b)$, donc :

```
from math import exp, sin, cos

def expC(z):
    a, b = z.real, z.imag
    return exp(a)*(cos(b) + 1j * sin(b))
```

Correction de l'application 33

1. La ligne `A = [[0]*3]*2` est équivalente aux 2 lignes

```
L = [0] * 3
A = [L] * 2
```

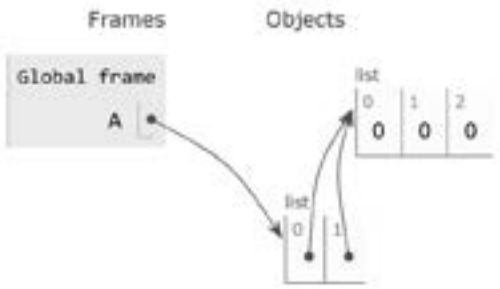
Ainsi, si on tape `A` dans la console, tout semble fonctionner :

```
>>> A
[[0, 0, 0], [0, 0, 0]]
```

Pourtant en creusant un peu :

```
>>> id(A[0])
2872149602504
>>> id(A[1])
2872149602504
```

Comme l'illustre Pythontutor sur son site, les deux lignes pointent sur le même tableau



Ainsi, en modifiant `A[0][0]`, on modifie `L[0]` et donc les deux lignes ....

```
>>> A[0][0] = 1
>>> A
[[1, 0, 0], [1, 0, 0]]
```

Il faut donc générer de nouvelles listes indépendantes pour chaque ligne de la matrice. Par exemple :

```
A = []
for i in range(2):
        A.append([0]*3)
```

2. On peut donc adapter l'exemple précédent :

```
def nulle(n, p):
    A = []
    for i in range(n):
        A.append([0]*p)
    return A
```

ou plus court avec une liste en compréhension :

```
def nulle(n, p):
    return [ [0]*p for i in range(n)]
```

Correction de l'application 34 Dans les 2 cas, on ne vérifie pas la taille des matrices, en imaginant que les fonctions seront correctement utilisées.

1. Pour la somme, on applique la formule :

```
def somme(A, B):
    n = len(A)     # Nombre de lignes
    p = len(A[0]) # Nombre de colonnes
    C = nulle(n, p)
    for i in range(n):
        for j in range(p):
            C[i][j] = A[i][j] + B[i][j]
    return C
```

2. De la même manière, on applique la formule de calcul des coefficients d'un produit :

```
def produit(A, B):
    n = len(A)     # Nombre de lignes du résultat
    p = len(B[0]) # Nombre de colonnes du résultat
    C = nulle(n, p)
    for i in range(n):
        for j in range(p):
            c = 0
            for k in range(len(B)) :
                c = c + A[i][k] * B[k][j]
            C[i][j] = c
    return C
```

Annexes

CONNAISSANCES ET COMPÉTENCES TRAVAILLÉES

CAPACITÉS ALGORITHMIQUES

Les tableaux qui suivent reprennent les capacités algorithmiques des programmes de mathématiques en fonction des niveaux scolaires. Toutes ne sont pas présentes, puisqu'un certain nombre d'illustrations de ces capacités sont disponibles sur le site EduScol qui propose des documents d'accompagnement de grande qualité.

Cette liste peut donc vous aider à chercher un exercice sur une capacité donnée.

Classe de seconde

A11 A13 THÈMES	Déterminer par balayage un encadrement de $\sqrt{2}$ d'amplitude inférieure ou égale à 10^{-n}.
A7	Déterminer si un entier naturel a est multiple d'un entier naturel b.
A7	Pour des entiers a et b donnés, déterminer le plus grand multiple de a inférieur ou égal à b.
B4	Déterminer si un entier naturel est premier.
C6	Déterminer une équation de droite passant par deux points donnés.
A13	Pour une fonction dont le tableau de variations est donné, algorithmes d'approximation numérique d'un extremum (balayage, dichotomie).
D13	Algorithme de calcul approché de longueur d'une portion de courbe représentative de fonction.

Classe de première spécialité mathématiques

A16 E7 E8	Calcul de termes d'une suite, de sommes de termes, de seuil.
F15	Calcul de factorielle.
B2 E6	Liste des premiers termes d'une suite : suites de Syracuse, suite de Fibonacci.
C11	Approximation de π par la méthode d'Archimède.

F11	Algorithme renvoyant l'espérance, la variance ou l'écart type d'une variable aléatoire.
THÈMES	Fréquence d'apparition des lettres d'un texte donné, en français, en anglais.

Classe de premières technologiques

A16 E6	Calculer un terme de rang donné d'une suite, une somme finie de termes.
D1	Déterminer une liste de termes d'une suite et les représenter.
E3 E8	Déterminer le rang à partir duquel les termes d'une suite sont supérieurs ou inférieurs à un seuil donné, ou aux termes de même rang d'une autre suite.
A11 A13 THÈMES	Calculer une valeur approchée d'une solution d'une équation par balayage.
F5	À partir de deux listes représentant deux caractères d'individus, déterminer un sous-ensemble d'individus répondant à un critère (filtre, utilisation des ET, OU, NON).
F4	Dresser le tableau croisé de deux variables catégorielles à partir du fichier des individus et calculer des fréquences conditionnelles ou marginales.
F10	Représenter par un histogramme ou par un nuage de points les fréquences observées des 1 dans N échantillons de taille n d'une loi de Bernoulli.
F10	Compter le nombre de valeurs situées dans un intervalle de la forme $[p-ks;p+ks]$ pour $k \in \{1;2;3\}$.

Classe de terminale spécialité mathématiques

A11 E8	Recherche de seuils. Recherche de valeurs approchées de π, e, $\sqrt{2}$, $\frac{1+\sqrt{5}}{2}$, $\ln(2)$, etc.
D12 THÈMES	Méthode de dichotomie, méthode de Newton, méthode de la sécante.

D7	Méthodes des rectangles, des milieux, des trapèzes.
F16	Problème de la surréservation. Étant donné une variable aléatoire binomiale X et un réel strictement positif α, détermination du plus petit entier k tel que $P(X > k) \leqslant \alpha$.
F9 F11	Simulation d'un échantillon d'une variable aléatoire.
F10	Simuler N échantillons de taille n d'une variable aléatoire d'espérance μ et d'écart type σ. Calculer l'écart type s de la série des moyennes des échantillons observés, à comparer à $\sigma/\sqrt{n}$. Calculer la proportion des échantillons pour lesquels l'écart entre la moyenne et μ est inférieur ou égal à ks, ou à $k\sigma/\sqrt{n}$, pour $k = 1, 2, 3$.

Classe de terminale - mathématiques expertes

B10	Algorithme d'Euclide de calcul du PGCD de deux nombres et calcul d'un couple de Bézout.
B13	Crible d'Ératosthène.
B14 B17	Décomposition en facteurs premiers.

Classe de terminale - mathématiques complémentaires

D12	Résolution d'équations par balayage, par dichotomie.
E9 E10	Calcul d'un terme de rang donné d'une suite.
F14 F15	Dans le cadre de la loi binomiale : calcul de coefficients binomiaux (triangle de Pascal), de probabilités ; détermination d'un intervalle I pour lequel la probabilité $P(X \in I)$ est inférieure à une valeur donnée α, ou supérieure à $1 - \alpha$.

F6 F11	Simulation avec Python d'une variable aléatoire (de la loi de Bernoulli, d'une loi uniforme discrète, etc.) d'un échantillon de taille n d'une variable aléatoire. Fonction Python renvoyant une moyenne pour un échantillon. Série des moyennes pour N échantillons de taille n d'une variable aléatoire d'espérance μ et d'écart type σ. Calcul de l'écart type s de la série des moyennes des échantillons observés, à comparer à $\sigma/\sqrt{n}$. Calcul de la proportion des cas où l'écart entre la moyenne m et μ est inférieur ou égal à $k\sigma/\sqrt{n}$ ou à ks, pour $k = 2$ ou $k = 3$.
E8	Recherche de seuils.
E7 E11	Pour une suite récurrente $u_{n+1} = f(u_n)$, calcul des termes successifs.
THÈMES	Recherche de valeurs approchées de constantes mathématiques, par exemple π, $\ln 2$, $\sqrt{2}$.
THÈMES	Méthodes de recherche de valeurs approchées d'une solution d'équation du type $f(x) = k$: balayage, dichotomie, méthode de Newton.
D7	Méthode des rectangles, des trapèzes.
THÈMES	Méthode de Monte-Carlo pour un calcul d'aire.
F6	Simulation d'une variable de Bernoulli ou d'un lancer de dé (ou d'une variable uniforme sur un ensemble fini) à partir d'une variable aléatoire de loi uniforme sur [0,1].

Classe de terminales technologiques

F14	Générer un triangle de Pascal de taille n donnée.
F11	Calculer l'espérance d'une variable aléatoire suivant une loi de probabilité donnée ; cas particulier d'une variable aléatoire suivant la loi binomiale $\mathcal{B}(n, p)$.

Enseignement scientifique

E5	Valeur au bout d'une fraction d'annuité d'un capital placé à intérêts composés à taux annuel constant.

RÉPARTITION PAR NIVEAU SCOLAIRE

Ce tableau vous permettra de choisir vos exercices en fonction du niveau scolaire. Nous avons fait le choix de recenser les exercices à partir d'un niveau qui est nécessairement à titre indicatif : soit à partir de la seconde, de la première (spécialité ou technologique), de la terminale ou option maths expertes. En effet, certaines notions peuvent être abordées en adaptant légèrement les énoncés en fonction du profil des élèves.

A - Les nombres				
A1	✓			
A2	✓			
A3	✓			
A4	✓			
A5	✓			
A6	✓			
A7	✓			
A8	✓			
A9	✓			
A10	✓			
A11	✓			
A12	✓			
A13		✓		
A14		✓		
A15	✓			
A16	✓			
A17	✓			
A18	✓			
A19	✓			
A20	✓			
A21	✓			
A22	✓			
A23	✓			
A24	✓			

B - Arithmétique				
B1	✓			
B2	✓			
B3	✓			
B4	✓			
B5	✓			
B6	✓			
B7	✓			
B8	✓			
B9	✓			
B10	✓			
B11				✓
B12				✓
B13				✓
B14				✓
B15			✓	
B16			✓	
B17			✓	
C - Géométrie				
C1	✓			
C2		✓		
C3		✓		
C4				✓
C5	✓			
C6	✓			
C7			✓	
C8			✓	
C9	✓			
C10	✓			
C11	✓			
C12	✓			
C13		✓		
C14	✓			

D - Analyse				
D1		✓		
D2	✓			
D3		✓		
D4	✓			
D5		✓		
D6			✓	
D7			✓	
D8		✓		
D9		✓		
D10			✓	
D11		✓		
D12			✓	
D13			✓	
D14			✓	
E - Suites				
E1	✓			
E2	✓			
E3	✓			
E4		✓		
E5			✓	
E6		✓		
E7		✓		
E8		✓		
E9		✓		
E10			✓	
E11			✓	
E12			✓	
E13			✓	
E14				✓
E15		✓		
E16			✓	

Probabilités et statistiques				
F1	✓			
F2	✓			
F3	✓			
F4	✓			
F5	✓			
F6		✓		
F7	✓			
F8		✓		
F9		✓		
F10	✓			
F11		✓		
F12			✓	
F13	✓			
F14			✓	
F15		✓		
F16		✓		
F17		✓		
F18		✓		

RÉPARTITION PAR ENTRÉE INFORMATIQUE

Ce dernier tableau vous aidera à choisir des exercices en fonction des notions Python que vous souhaitez travailler. Voici les pictogrammes utilisés, les autres notions plus classiques comme les tests et les boucles n'ont pas été identifiées :

Exercice utilisant les listes.

Exercice utilisant les chaînes de caractères.


Exercice utilisant des sorties graphiques.


Exercice utilisant les nombres complexes.


Exercice utilisant les matrices.


Exercice utilisant le module `random`.

Seuls les exercices mettant en œuvre ces capacités ont été listés.

A - Calcul numérique						
A8		✓				
A14	✓					
A18		✓				
A24	✓					
B - Arithmétique						
B6	✓					
B7		✓				
B8	✓	✓				✓
B12		✓				
B13			✓			

C - Géométrie						
C3	✓					
C4				✓		
C5			✓			
C7			✓			
D - Fonctions						
D1			✓			
D2			✓			
D4			✓			
D6			✓			
D7						✓
D10			✓			
D11			✓			
E - Modèles évolutifs et suites						
E6	✓					
E11			✓			
E12			✓			
E15	✓					
E16					✓	
Statistiques et probabilités						
F1	✓					
F2	✓					
F3	✓		✓			
F4			✓		✓	
F5					✓	
F6						✓
F7	✓		✓			✓
F9	✓					✓
F10	✓		✓			✓
F11						
F12	✓					✓
F14					✓	
F15						✓
F16	✓					✓
F17	✓		✓			✓
F18	✓					✓

INDEX

CRÉATIONS NUMÉRIQUES

Ce livre propose un **grand nombre d'activités** mathématiques souvent **concrètes** de niveau lycée pour lesquelles l'utilisation de la programmation est pertinente.

Il pourra intéresser les élèves, les étudiants en mathématiques et particulièrement ceux qui préparent le CAPES ou l'agrégation de mathématiques, ainsi que les enseignants plus expérimentés souhaitant **enrichir leur pratique algorithmique** via la programmation en Python.

Ces **problèmes variés et historiques** pourront d'ailleurs introduire ou illustrer leur cours.

- **Plus de 100 exercices avec aides et corrections détaillées** autour de 6 grandes parties : le calcul numérique, l'arithmétique, la géométrie, l'analyse, les suites et les probabilités.
- **Des problèmes historiques et culturels** regroupés en 4 thèmes : les nombres remarquables, les nombres premiers, l'art et la cryptographie.
- **Un cours synthétique mais complet** pour découvrir ou redécouvrir le langage Python accompagné d'une trentaine d'applications directes facilitant la prise en main de ce langage.
- **Différents tableaux de synthèse** permettant de cheminer dans les exercices de mathématiques ou de personnaliser sa propre expérience.

Dans la même collection :

www.editions-ellipses.fr
9 782340 089594