Exercices d'initiation au langage Python

Les exercices indiqués « complémentaires » sont optionnels. Ils sont là pour ceux qui souhaitent retravailler les notions.
Les questions supplémentaires ne sont pas obligatoires et même déconseillées dans une première approche de Python.

I. Premiers programmes : implantation d'algorithmes

Les exercices de cette première section ont pour objectif la transcription d'un algorithme en un script Python. Cette opération est appelée **implantation**. Il s'agit uniquement d'apprendre la syntaxe Python et l'utilisation de fonctions pour se libérer des saisies et affichages.
Chaque exercice devra être enregistré dans un fichier et exécuté avant de pouvoir l'utiliser en console.

Exercice 1 (fonction, structure conditionnelle)
Un magasin de reprographie propose un tarif dégressif. Les 20 premières photocopies sont facturées 10 centimes, les 20 suivantes 8 centimes, et au-delà 6 centimes.
On donne ci-dessous l'algorithme d'une fonction qui renvoie le montant de la facture en euros pour un nombre de photocopies donné :

```
Fonction tarif(n)
    Si n ⩽ 20 alors
        retourner 0,10 × n
    Sinon si n ⩽ 40
        retourner 2 + 0,08 × (n − 20)
    Sinon
        retourner 3,6 + 0,06 × (n − 40)
    Fin Si
Fin fonction
```

1. Implanter la fonction dans un script et la tester en console comme ci-dessous :

```
>>> tarif(15)
1.5
>>> tarif(30)
2.8
>>> tarif(45)
3.9
```

2. Modifier le programme pour qu'il comporte une saisie du nombre de photocopies effectuées et affiche le prix à payer.

Exercice 2 (complémentaire : fonction, structure conditionnelle)
L'algorithme ci-dessous propose une fonction qui renvoie, selon la moyenne obtenue au baccalauréat, la mention correspondante :

```
Fonction mention(note)
    Si note < 10 alors
        retourner « recalé »
    Sinon si note < 12
        retourner « mention passable »
    Sinon si note < 14
        retourner « mention assez bien »
    Sinon si note < 16
        retourner « mention bien »
    Sinon
        retourner « mention très bien »
    Fin Si
Fin fonction
```

1. Implanter la fonction dans un script et la tester en console comme ci-dessous :

```
>>> mention(8)
recalé
>>> mention(14)
mention bien
```

2. Modifier le programme pour qu'il comporte une saisie de la moyenne obtenue au bac et affiche la mention correspondante.

Exercice 3 (affectation, liste, boucle bornée)
On considère la suite (u_n) définie par $u_0 = 0$ et, pour tout entier naturel n, par $u_{n+1} = u_n + 2n$.
L'algorithme ci-dessous propose une fonction qui renvoie le N-ième terme de la suite :

```
Fonction terme_suite(N)
    u ← 0
    Pour i allant de 1 à N − 1
        u ← u + 2(i − 1)
    Fin Pour
    retourner u
Fin fonction
```

1. Implanter la fonction dans un script et la tester en console comme ci-dessous :

```
>>> terme_suite(6)
20
```

2. Modifier la fonction pour qu'elle **affiche** les N premiers termes de la suite :

```
>>> terme_suite(6)
0
0
2
6
12
20
```

3. **Question supplémentaire**
   Modifier la fonction pour qu'elle **renvoie** les N premiers termes de la suite, sous forme d'une liste :

```
>>> terme_suite(6)
[0, 0, 2, 6, 12, 20]
```

Exercice 4 (complémentaire : affectation, liste, boucle bornée)
On considère un entier s compris entre 3 et 18.
L'algorithme ci-dessous propose une fonction `somme(s)` qui renvoie le nombre de façons d'obtenir une somme égale à s en lançant trois dés.

```
Fonction somme(s)
    compteur ← 0
    Pour i allant de 1 à 6
        Pour j allant de 1 à 6
            Pour k allant de 1 à 6
                Si i + j + k = s alors
                    compteur ← compteur + 1
                Fin Si
            Fin Pour
        Fin Pour
    Fin Pour
    retourner compteur
Fin fonction
```

1. Implanter la fonction dans un script et la tester en console :

```
>>> somme(7)
15
```

2. **Question supplémentaire**
   Modifier la fonction pour qu'elle retourne aussi toutes les façons possibles d'obtenir la somme s :

```
>>> somme(7)
(15, ['1+1+5', '1+2+4', '1+3+3', '1+4+2', '1+5+1', '2+1+4', '2+2+3', '2+3+2',
'2+4+1', '3+1+3', '3+2+2', '3+3+1', '4+1+2', '4+2+1', '5+1+1'])
```

Exercice 5 (boucle non bornée)
On considère la suite définie par :

$$u_0 = 800 \quad \text{et} \quad u_{n+1} = \frac{3}{4}u_n + 330 \quad \text{pour tout entier naturel } n.$$

L'algorithme ci-dessous propose une fonction qui renvoie la plus petite valeur de n à partir de laquelle u_n dépasse le seuil donné en paramètre :

```
Fonction depasse_seuil(seuil)
    n ← 0
    u ← 800
    Tant que u < seuil
        u ← 3/4 u + 330
        n ← n + 1
    Fin Tant que
    retourner n
Fin fonction
```

1. Implanter la fonction dans un script et la tester en console :

```
>>> depasse_seuil(1200)
6
```

2. **Question supplémentaire**
   On pourra vérifier que l'on sait stopper un script Python lorsque la boucle ne s'arrête pas, par exemple dans le cas de l'appel `depasse_seuil(1330)` (la suite est majorée par 1 320) :

```
>>> depasse_seuil(1330)
Traceback (most recent call last):
  File "<console>", line 1, in <module>
  File "C:\seuil.py", line 5, in depasse_seuil
    u = 0.75*u+330
KeyboardInterrupt
```

Exercice 6 (boucle non bornée, import de module, conditions multiples avec "ET", fonction sans paramètre)
Un concours entre un lièvre et une tortue se déroule de la façon suivante :

- on lance un dé à quatre faces.
  - si le dé tombe sur 4, le lièvre atteint directement l'arrivée et a gagné.
  - si le dé tombe sur 1, 2 ou 3, la tortue avance d'une case et on relance le dé. La tortue a gagné lorsqu'elle a avancé de quatre cases.
- la partie continue jusqu'à ce qu'il y ait un gagnant.

L'algorithme ci-dessous propose une fonction qui réalise une simulation et renvoie le nom du gagnant :

```
Fonction course()
    T ← 0
    L ← 0
    Tant que L ≠ 4 et T ≠ 4
        alea ← entier aléatoire entre 1 et 4
        Si alea = 4 alors
            L ← 4
        Sinon
            T ← T + 1
        Fin Si
    Fin Tant que
    Si L = 4 alors
        retourner « Le lièvre a gagné. »
    Sinon
        retourner « La tortue a gagné. »
    Fin Si
Fin fonction
```

1. Implémenter l'algorithme et tester le programme.
   Indication : pour générer un nombre aléatoire, il faut utiliser le module `random` (voir l'aide-mémoire).
2. **Questions supplémentaires**
   - **a.** Modifier le programme afin qu'il retourne une simulation de 100 courses
   - **b.** Modifier le programme pour qu'il affiche la fréquence de victoire du lièvre pour la simulation des 100 courses effectuées.
   - **c.** La probabilité de victoire du lièvre est $\frac{175}{256}$. Modifier le programme pour qu'il indique si la fréquence de victoire calculée est dans l'intervalle de fluctuation au seuil de 95 % correspondant.

Exercice 7 (boucle non bornée, conditionnelle, fonction et retour de plusieurs valeurs, chargement module)
On considère la fonction f définie sur $\mathbb{R}$ par $f(x) = \ln(x) + x$.
L'algorithme ci-dessous permet la résolution par dichotomie, sur un intervalle $[a\,;b]$, de l'équation $f(x) = 0$ avec une précision de 10^{-k} (les nombres a, b et k sont passés en paramètres) :

```
Fonction f(x)
    retourner ln(x) + x
Fin fonction

Fonction dichotomie(a, b, k)
    Tant que b − a > 10^−k
        m ← (a+b)/2
        Si f(m) < 0 alors
            a ← m
        Sinon
            b ← m
        Fin Si
    Fin Tant que
    retourner [a, b]
Fin fonction
```

Implanter l'algorithme et tester le programme en console :

```
>>> dichotomie(0,1,3)
[0.56640625, 0.5673828125]
```

Indication : la fonction ln est notée `log` en Python. Elle nécessite d'être chargée, en début de script, avec l'instruction « **from** `math` **import** `log` ».

II. Chaines de caractères, listes et types

Les exercices suivants ont pour but la manipulation de chaines de caractère, de listes et de types afin de mieux appréhender ces notions.

Exercice 8 (manipulations de chaines de caractère)

1. Tester en console les instructions suivantes :

```
>>> a = "Hello "
>>> b = "World !"
>>> a + b
```

2. Tester de même :

```
>>> 3 * a
```

3. Et enfin :

```
>>> a + 2017
```

   Interpréter l'erreur indiquée.

Exercice 9 (manipulations de listes)

1. Tester en console les instructions suivantes :

```
>>> L = [8, 3, 5]
>>> len(L)
>>> L[0]
```

2. Comment afficher le dernier élément de la liste ?
3. Tester en console les instructions suivantes :

```
>>> L = L + 2
>>> L = L + [2]
```

   Interpréter l'erreur indiquée.

Exercice 10 (typage)

1. Tester en console les instructions suivantes :

```
>>> 23 / 2
>>> 23 // 2
>>> 23 % 2
```

   À quoi correspondent-elles ?
2. Comparer :

```
>>> type(10 / 2)
>>> type(10 // 2)
```

3. Proposer une instruction permettant de tester si un entier est pair.

III. Compléments sur les fonctions

Dans cette section, on aborde l'utilisation de fonction en tant que sous-partie d'un programme.

Exercice 11 (listes et fonctions)
Le but de cet exercice est de créer un programme qui indique si un triangle est rectangle.
Afin de faciliter la lecture du code, nous utilisons :

- plusieurs fonctions pour bien distinguer les différentes étapes ;
- à partir de la question **2**, des listes de deux nombres pour représenter les points.

La fin de l'exercice concerne un problème concernant la représentation des nombres en informatique. Elle est optionnelle.

1. Implémenter une fonction `distance(xA, yA, xB, yB)`, qui renvoie la distance AB entre deux points A et B du plan.
On aura par exemple :

```
>>> distance(2, 3, 5, 7)
5.0
```

Indication : la fonction racine carrée (`sqrt`) nécessite le module `math`.

2. On souhaite maintenant saisir les coordonnées de A et de B sous la forme d'une liste de deux nombres :

```
>>> A = [2, 3]
>>> B = [5, 7]
>>> distance(A, B)
5.0
```

Modifier la fonction de la première question pour répondre à cette exigence.

3. Implanter et tester la fonction `rectangle_en_premier_sommet(A, B, C)` suivante (on utilisera la fonction `distance(A, B)` implantée précédemment) :

```
Fonction rectangle_en_premier_sommet(A, B, C)
    Si distance(A,B)^2 + distance(A,C)^2 = distance(B,C)^2 alors
        retourner True
    Sinon
        retourner False
    Fin Si
Fin fonction
```

En console, on obtiendra par exemple :

```
>>> A = [2, -1]
>>> B = [5, 5]
>>> C = [-2, 1]
>>> rectangle_en_premier_sommet(A, B, C)
False
```

4. Sachant que l'on a $AB = \sqrt{45}$, $AC = \sqrt{20}$ et $BC = \sqrt{65}$, proposer une explication à la réponse obtenue.

5. Questions supplémentaires

a. Tester en console :

```
>>> 1.7 - 1.3 - 0.4
>>> 1.7 == 1.3 + 0.4
```

b. Implémenter une fonction `conjecture_egalite(a, b)` qui renvoie True si l'écart entre les deux réels `a` et `b` est inférieur à 10^{-10}.
On obtiendra ainsi :

```
>>> conjecture_egalite(1.7, 1.3 + 0.4)
True
```

c. Proposer une modification de la fonction `rectangle_en_premier_sommet(A, B, C)`, pour que le résultat obtenu soit celui attendu.

d. Implémenter une fonction `triangle_rectangle(A, B, C)` qui utilise la fonction `rectangle_en_premier_sommet(A, B, C)`, et qui renvoie **True** si le triangle ABC est rectangle (quel que soit le sommet).

Un algorithme possible est le suivant :

```
Fonction triangle_rectangle(A, B, C)
    Si rectangle_en_premier_sommet(A, B, C) ou ... ou ... alors
        retourner True
    Sinon
        retourner False
    Fin Si
Fin fonction
```

Exercice 12 (complémentaire)
Même esprit que l'exercice précédent.

1. Implémenter une fonction `milieu(A, B)`, qui renvoie le couple de coordonnées du milieu de $[AB]$.
Exemple d'utilisation :

```
>>> A = [1, 5]
>>> B = [2, 7]
>>> milieu(A, B)
[1.5, 6.0]
```

2. Implanter une fonction `test_parallelogramme(A,B,C,D)`, qui indique si le quadrilatère $ABCD$ est un parallélogramme.
Un algorithme est proposé ci-dessous :

```
Fonction test_parallélogramme(A, B, C, D)
    I ← milieu(A,C)
    J ← milieu(B,D)
    Si I = J alors
        retourner True
    Sinon
        retourner False
    Fin Si
Fin fonction
```

IV. Exercices supplémentaires

Exercice 13 (arithmétique)

- Un nombre premier est un entier naturel qui admet exactement deux diviseurs distincts, entiers et positifs (qui sont alors 1 et lui-même).
- Un diviseur strict d'un entier naturel n est un entier naturel diviseur de n, mais distinct de n.
- Un nombre parfait est un entier naturel égal à la somme de tous ses diviseurs stricts.
  Par exemple, 28 est parfait car ses diviseurs stricts sont 1, 2, 4, 7 et 14, et $1+2+4+7+14=28$.
- Un nombre chanceux d'Euler est un entier naturel n tel que x^2+x+n est premier, pour tout entier x de l'intervalle $[0\,;\,n-2]$.
  Par exemple, 5 est un nombre chanceux d'Euler car :

$$0^2+0+5=5,\quad 1^2+1+5=7,\quad 2^2+2+5=11,\quad 3^2+3+5=17$$

  et ces nombres sont tous premiers.

1. Implémenter et tester une fonction `somme_div(n)`, qui retourne la somme des diviseurs stricts d'un entier naturel n.
Un algorithme possible est :

```
Fonction somme_div(n)
    somme ← 0
    Pour i allant de 1 à n − 1
        Si i divise n alors
            somme ← somme + 1
        Fin Si
    Fin Pour
    retourner somme
Fin fonction
```

2. Implémenter une fonction `est_parfait(n)`, retournant un booléen True ou False.
On pourra tester en console :

```
>>> est_parfait(28)
True
```

3. Implémenter une fonction `est_premier(n)`, utilisant la fonction `somme_div(n)`, et retournant un booléen True ou False (un entier est premier si la somme de ses diviseurs stricts est 1).
4. Implémenter une fonction `est_chanceux(n)` retournant un booléen True ou False.
Un algorithme possible est :

```
Fonction est_chanceux(n)
    chanceux ← Vrai
    Pour x allant de 1 à n−2
        chanceux ← chanceux et est_premier(x² + x + n)
    Fin Pour
    retourner chanceux
Fin fonction
```

On pourra tester en console :

```
>>> est_chanceux(5)
True
```

5. Implémenter deux fonctions `liste_parfaits(n)` et `liste_chanceux(n)` qui retournent respectivement la liste des nombres parfaits et la liste des nombres chanceux inférieurs à n.
Pour vérification, on obtient :

```
>>> liste_parfaits(100)
[6, 28]
>>> liste_chanceux(100)
[2, 3, 5, 11, 17, 41]
```

Exercice 14 (simulation)
Une puce se déplace sur un axe gradué, en partant de la position 0. Elle fait quatre sauts, tantôt à droite, tantôt à gauche et ceci de façon équiprobable.
L'ensemble des quatre sauts s'appelle une marche.

1. Implémenter une fonction `marche()` qui retourne la position de la puce après sa marche.
2. Implémenter une fonction `simulation(n)` qui retourne une liste avec les n positions finales de la puce après n simulations de marches.
3. Implémenter une fonction `histogramme(n)` qui affiche l'histogramme des effectifs des différentes positions finales après n simulations.
4. Implémenter une fonction `frequence(n)` qui affiche, pour chacune des positions finales possibles, sa fréquence pour la simulation de n marches effectuées.