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 aproche de Python.

1 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 gérer les saisies et affichages.
Chaque exercice devra être enregistré dans un fichier.

Exercice 1 (Fonction, structure conditionnelle)**.**
Un magasin de reprographie propose un tarif dégressif. Les 20 premières photocopies sont facturées à 10 centimes et les suivantes à 8 centimes.

On donne ci-dessous à gauche 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 0,1×n
    sinon
        retourner 2+0,08×(n-20)
Fin fonction
```

Exemples d'execution en console :

```
>>> tarif(15)
1.5
>>> tarif(40)
5.2
```

1. Implanter la fonction et la tester en mode console comme montré ci-dessus à droite.

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

```
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
```

Exemples d'execution en console :

```
>>> mention(8)
recalé
>>> mention(11.5)
passable
>>> mention(13)
assez bien
>>> mention(14)
bien
>>> mention(17)
très bien
```

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

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 permet de calculer et d'afficher les N premiers termes de la suite.

```
fonction termesSuite(N)
    u ← 0
    L ← [u]
    Pour i de 1 à N − 1 faire
        u ← u + 2(i − 1)
        L ← L + [u]
    Fin Pour
    retourner L
Fin fonction
```

Exemples d'execution en console :

```
>>> termesSuite(7)
[0, 0, 2, 6, 12, 20, 30]
```

1. Implanter l'algorithme et tester le programme.

Exercice 4 (affectation, boucle non bornée, conditionnelle, fonction et retour de plusieurs valeurs, chargement module)**.**
On considère la fonction f définie sur R par $f(x) = \ln(x) + x$.
L'algorithme ci-dessous permet la résolution de l'équation $f(x) = 0$ par dichotomie avec une précision à 10^{-k} près, les nombres a, b et k étant saisis par l'utilisateur.

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

Exemples d'execution en console :

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

1. Implanter l'algorithme et tester le programme.
indication : La fonction logarithme ($\ln$) nécessite le chargement du module *math* ce qui se fait avec l'instruction "from math import *".

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

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

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

Exemples d'execution en console :

```
>>> depasseSeuil(1200)
6
>>> depasseSeuil(1330)
Traceback (most recent call last):

  File "<console>", line 1, in <module>
  File "/home/seuil.py", line 5, in depasseSeui
    u = 3/4*u+330
KeyboardInterrupt
```

1. Implanter l'algorithme et tester le programme.

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(1330)` *(la suite tend vers* 1320*)*

2 Chaines de caractères, listes et fonctions

Les exercices suivants ont pour but la manipulation de chaines de caractère, de listes et de fonctions. Les exercices seront à faire dans un script ou en console.

Exercice 6 (Manipulations de chaines de caractère)**.**
Tester en console les instructions suivantes :

1.

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

2.

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

3.

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

Interpréter l'erreur indiquée

Exercice 7 (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

- des listes de 2 nombres pour représenter les points ;
- plusieurs fonctions pour bien distinguer les différentes étapes et clarifier le code

La fin de l'exercice concerne un problème la représentation des nombres en informatique. Il est optionnel.

1. Implémenter une fonction qui renvoie la distance AB entre deux points A et B du plan, avec l'appel : *distance(xA,yA,xB,yB)*
   indication : la fonction racine carrée (sqrt) nécessite le module math.

   Exemple :

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

2. On souhaite maintenant saisir les coordonnées de A et de B sous la forme d'une liste de deux nombres.
   Modifier la fonction de la première question pour répondre à cette exigence.

   Exemple :

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

3. Implanter et tester la fonction *RectangleEnPremierSommet(A,B,C)* suivante *(on utilise la fonction distance implantée précédemment)* :

```
fonction RectangleEnPremierSommet(A,B,C) :
    Si distance(A,B)²+distance(A,C)²=distance(B,C)² alors
        retourner True
    sinon
        retourner False
```

   Exemple :

```
>>> A = [2,-1]
>>> B = [5,5]
>>> C = [-2;1]
>>> RectangleEnPremierSommet(A,
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 *ConjectureEgalite(a,b)* qui renvoie True si l'écart entre les deux float a et b est inférieur à 10^{-10}. Ainsi l'appel *ConjectureEgalite(1.7,1.3+0.4)* renvoie *True*

(c) Proposer une modification de la fonction *RectangleEnPremierSommet* pour que le résultat obtenu soit celui attendu.

(d) Implémenter une fonction *TriangleRectangle(A,B,C)* qui utilise la fonction *RectangleEnPremierSommet* et qui renvoie True si le triangle ABC est rectangle (quel que soit le sommet). Un algorithme possible est le suivant :

```
fonction RectangleEnPremierSommet(A,B,C) :
    Si RectangleEnPremierSommet(A,B,C) ou
       RectangleEnPremierSommet(B,A,C) ou
        RectangleEnPremierSommet(C,B,A) alors
        retourner True
    sinon
        retourner False
```

Exercice 8 (complémentaire)**.**
Même esprit que l'exercice précédent.

1. Implémenter une fonction *milieu(A,B)* qui renvoie le couple de coordonnées du milieu de $[AB]$. Ainsi l'appel :

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

doit renvoyer *[1.5,6.0]*

2. Implanter une fonction dont l'appel serait *TestParallelogramme(A,B,C,D)* qui indique si le quadrilatère $ABCD$ est un parallélogramme.
Un algorithme est proposé ci-dessous :

```
fonction TestParallelogramme(A,B,C,D) :
    I←milieu(A,C)
    J←milieu(B,D)
    Si I=J alors
        retourner True
    sinon
        retourner False
```

3 Exercices supplémentaires

Exercice 9 (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 *SommeDiv(n)* qui retourne la somme des diviseurs stricts d'un entier naturel.

Un algorithme possible est :

Fonction
| SommeDiv(n : entier)
:
somme ← 0
pour *i entre 1 et n-1* **faire**
| **si** *i divise n* **alors**
| | somme ← somme+i
| **fin**
fin
retourner somme

2. Implémenter une fonction *EstParfait(n)* retournant un booléen True ou False. Ainsi l'appel *EstParfait(28)* renvoie *True*

3. Un entier est premier si la somme de ses diviseurs stricts est égal à 1. Implémenter une fonction *EstPremier(n)*, utilisant la fonction *SommeDiv(n)*, qui retourne un booléen True ou False.

4. Implémenter une fonction *EstChanceux(n)*, retournant un booléen True ou False. . Ainsi l'appel *EstChanceux(5)* renvoie *True*

Un algorithme possible est :

Fonction
| EstChanceux(n : entier)
:
chanceux ← True
pour *x compris entre 1 et n-2* **faire**
| chanceux←chanceux **et** EstPremier(x^2+x+n)
fin
retourner *chanceux*

5. Utiliser ces trois fonctions pour afficher tous les nombres parfaits et tous les nombres chanceux d'Euler compris entre 2 et 100.

Pour vérification, on obtient :

> *Nombres parfaits entre 2 et 100 : 6 28*
> *Nombres chanceux entre 2 et 100 : 2 3 5 11 17 41*

Exercice 10 (Simulation)**.**
Une puce se déplace sur un axe gradué, en partant de la position 0. Elle fait 4 sauts, tantôt à droite, tantôt à gauche et ceci de façon équiprobable. L'ensemble des 4 sauts s'appelle une marche.

1. Implémenter une fonction *saut()* 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.