1 Premiers programmes : chaînes de caractères, fonctions et listes

Exercice 1. Tester en console les instructions suivantes :

1.

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

2.

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

3.

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

Interpréter l'erreur indiquée

Exercice 2. Modifier la fonction `welcome` suivante pour qu'elle ait comme argument un prénom et l'affiche après bonjour. Ainsi l'appel `welcome("Pierre")` renvoie `"Bonjour Pierre"`

```
def welcome():
    return "Bonjour"
```

Exercice 3. 1. Tester en console les instructions suivantes

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

Comment afficher le dernier élément de la liste L ?

2. Implanter la fonction suivante :

Fonction *dernier(L : liste)* **:**
    n ← longueur de la liste L
    retourner élément de la liste L dont le rang est n
Fin

Saisir alors une liste de nombre et afficher son dernier élément.

Exercice 4. 1. Implémenter une fonction qui renvoie la distance AB de 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.

```
from math import *
```

2. On souhaite maintenant saisir les coordonnées de A et de B sous la forme d'une liste de deux nombres de type float. Modifier la fonction de la première question pour que l'appel soit :

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

2 Instructions conditionnelles

Exercice 5. Un magasin de reprographie propose un tarif dégressif. Les 20 premières photocopies sont facturées à 10 centimes et les suivantes à 8 centimes.

Implanter une fonction qui renvoie le montant de la facture en euros pour un nombre de photocopies donné. Un algorithme est proposé ci-dessous :

Fonction *tarif(n : entier)* **:**
    si *n≤20* **alors**
        retourner *0,1×n*
    sinon
        retourner *2+0,08×(n-20)*
    fin
Fin

Exercice 6. Implémenter une fonction qui, en fonction de la moyenne obtenue au baccalauréat, renvoie la mention obtenue.

Ainsi l'appel `mention(16.2)` renvoie `"Très bien"`

Remarque : on pourra utiliser l'instruction `elif`

Exercice 7. 1. Tester les instructions suivantes :

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

A quoi correspondent-elles ?

2. Comparer :

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

3. Implanter et tester la fonction `pair(n)` suivante :

Fonction *pair(n : entier)* **:**
    si *le reste de la division de n par 2 est 0* **alors**
        retourner *True*
    sinon
        retourner *False*
    fin
Fin

4. Implanter et tester la fonction dont l'algorithme est donnée ci-dessous :

Fonction *mediane(L : liste)* **:**
    Trier la liste L par ordre croissant
    n← longueur de L
    si *n est pair* **alors**
        a ← terme de L de rang $\frac{n}{2}$
        b ← terme de L de rang $\frac{n}{2}+1$
        retourner $\frac{a+b}{2}$
    sinon
        retourner *terme de L de rang (n+1)/2*
    fin
Fin

Exercice 8. Un numéro de Sécurité Sociale est constitué de 13 chiffres. On lui ajoute une clé de contrôle, constituée de 2 chiffres et donnée par la formule :

$$\text{clé} = 97 - r$$

où r est le reste dans la division euclidienne du numéro de Sécurité Sociale par 97.

Implémenter une fonction de contrôle : `controle(numero,cle)` vérifiant le numéro de Sécurité Sociale.

Exercice 9. 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) (A,B,C,D : listes à deux éléments)* **:**
    I←milieu(A,C)
    J←milieu(B,D)
    si *I=J* **alors**
        retourner *True*
    sinon
        retourner *False*
    fin
Fin

Exercice 10. 1. Implanter la fonction `RectangleEnPremierSommet(A,B,C)` suivante *(on utilise la fonction* `distance` *implantée précédemment)* :

Fonction *RectangleEnPremierSommet(A,B,C) (A,B,C : listes à deux éléments)* **:**
    si *distance(A,B)²+distance(A,C)²=distance(B,C)²* **alors**
        retourner *True*
    sinon
        retourner *False*
    fin
Fin

2. Tester cette fonction avec $A(2;-1)$, $B(5;5)$ et $C(-2;1)$ ($AB = \sqrt{45}$, $AC = \sqrt{20}$ et $BC = \sqrt{65}$). Expliquer la réponse obtenue.

3. Tester en console :

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

4. 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`

5. Proposer une modification de la fonction `RectangleEnPremierSommet` pour que le résultat obtenu soit celui attendu.

6. Implanter 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 *TriangleRectangle(A,B,C) (A,B,C : listes à deux éléments)* **:**
    si *RectangleEnPremierSommet(A,B,C)* **ou** *RectangleEnPremierSommet(B,A,C)* **ou** *RectangleEnPremierSommet(C,B,A)* **alors**
        retourner *True*
    sinon
        retourner *False*
    fin
Fin

3 Boucle bornée

Exercice 11. 1. Tester en console les instructions suivantes :

```
>>> for in in range(5):
        print(i)
>>> for in in range(1,7):
        print(i)
```

2. Implanter une fonction `SommeGeometrique(q,n)` qui renvoie la somme

$$1 + q + q^2 + \ldots + q^n$$

Ainsi l'appel `SommeGeometrique(0.5,100)` renvoie `2.0` Un algorithme possible est le suivant :

Fonction *SommeGeometrique (q : float, n : entier)* **:**
    s←1
    pour *k allant de 1 à n* **faire**
        s← s+q^k
    fin
    retourner *s*
Fin

Exercice 12. Implanter un script qui calcule la moyenne de plusieurs notes, dont le nombre est saisi au préalable. Un algorithme possible est :

Afficher *Nombre de notes ?*
Saisir *n : entier*
total ← 0
pour *i entre 1 et n* **faire**
    Afficher Note i ?
    Saisir note
    total ← total+note
fin
moyenne ← $\frac{\text{total}}{n}$
Afficher *moyenne*

Exercice 13. 1. Implanter une fonction `moyenne(L)` qui calcule la moyenne de la liste L. Ainsi l'appel

```
L=[10,12.5,16.5]
moyenne(L)
```

renvoie `13.0`.

Un algorithme possible est :

Fonction *moyenne(L : liste)* **:**
    total ← 0
    n ← longueur de la liste L
    pour *i entre 1 et n* **faire**
        total ← total + élément de L de rang i
        moyenne ← $\frac{\text{total}}{n}$
    fin
    retourner *moyenne*
Fin

2. Implanter une fonction `carre(L)` qui retourne une liste constituée des carrés des élément de la liste L. Ainsi l'appel

```
L=[2,3,4]
carre(L)
```

renvoie `[4,9,16]`.

Un algorithme possible est :

Fonction *carre(L : liste)* **:**
    LCarre ← liste vide
    n ← longueur de la liste L
    pour *i entre 1 et n* **faire**
        LCarre ← LCarre + carré de l'élément de L de rang i
    fin
    retourner *LCarre*
Fin

3. En utilisant la formule

$$V(X) = E(X^2) - (E(X))^2$$

implanter une fonction `ecarttype(L)` faisant appel aux fonctions précédentes qui retourne l'écart-type de la liste L. Ainsi l'appel

```
L=[10,12.5,16.5]
ecarttype(L)
```

renvoie `2.6770630673681666`.

Exercice 14 (Méthode des rectangles)**.** On suppose que l'on a défini une fonction f, par exemple :

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

Implanter et tester la fonction `rectangle(a,b,n)` qui donne une valeur approchée de $\int_a^b f(x)\,\mathrm{d}x$ par la méthode des rectangles à gauche avec n subdivisions régulières. L'algorithme est donné ci-dessous :

Fonction *rectangle(a,b : float, n : entier)* **:**
    s$\leftarrow$ 0
    h$\leftarrow$ $\frac{b-a}{n}$
    pour *k allant de 0 à n-1* **faire**
        s$\leftarrow$ $s + hf\left(a + kh\right)$
    fin
    retourner *s*
Fin

Exercice 15. Soit s un entier compris entre 3 et 18. Implémenter une fonction `somme(s)` affichant le nombre de façons d'obtenir une somme égale à s en lançant trois dés. Un algorithme est donné ci-dessous :

Fonction *somme(s : entier)* **:**
    compteur$\leftarrow$ 0
    pour *i allant de 1 à 6* **faire**
        pour *j allant de 1 à 6* **faire**
            pour *k allant de 1 à 6* **faire**
                si *i+j+k est égal à s* **alors**
                    compteur $\leftarrow$ compteur+1
                fin
            fin
        fin
    fin
    retourner *compteur*
Fin

Bonus : La fonction affiche toutes les façons possibles

4 Boucle non bornée

Exercice 16 (Recherche de seuil)**.** On considère la suite définie par

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

Implémenter une fonction `depasse(seuil)` qui renvoie la plus petite valeur de n à partir de laquelle u_n dépasse le seuil. Un algorithme possible est :

Fonction *depasse(seuil : float)* **:**
    n$\leftarrow$ 0
    u$\leftarrow$ 800
    tant que *u<seuil* **faire**
        u $\leftarrow$ $\frac{3}{4}u + 330$
        n $\leftarrow$n+1
    fin
    retourner *n*
Fin

Tester avec l'appel `depasse(1300)` qui renvoie 12.

Remarque : 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*)*

Exercice 17. L'ordinateur choisit un nombre entier au hasard entre 1 et 1000, et le joueur doit le deviner le plus rapidement possible. L'affichage pourra être de la forme :

```
J'ai choisi le nombre secret .
Proposition numéro 1 : 500
C'est moins .
Proposition numéro 2 : 250
C'est plus .
```

Un algorithme possible est :

nombreSecret ← nombre au hasard entre 1 et 1000
pasTrouve ← Vrai
compteur ← 0
tant que *pasTrouve* **faire**
    compteur ← compteur+1
    Afficher *Proposition numéro compteur ?*
    Saisir reponse
    si *reponse=nombreSecret* **alors**
        Afficher *Vous avez gagné en compteur coups*
    sinon
        si *reponse<nombreSecret* **alors**
            Afficher *C'est plus*
        sinon
            Afficher *C'est moins*
        fin
    fin
fin

Bonus : Le nombre de tentatives est limité à 7

5 Exercices supplémentaires

Exercice 18 (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
Fin

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

3. Un entier est parfait 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*
Fin

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 19 (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.