
TRAVAUX PRATIQUES n° 4

Boucle while et programmation fonctionnelle

Pour bien démarrer ce TP, commencez par lire la fiche méthode suivante :

- ★ **I6** - Utiliser une boucle `while`

Exercice 1 (*Échauffement boucle while*) :

- 1) Recopier les scripts suivants et les exécuter un par un :

```
1 ## Script 1
2 i = 0
3 while i < 10:
4     print(i)
5     i = i + 1
6
7 ## Script 2
8 i = 0
9 while i < 10:
10    i = i + 1
11    print(i)
12
13 ## Script 3
14 x = 0
15 while x**2 < 100:
16    print(x)
17    x = x + 1
```

- 2) Que se passera-t-il si on exécute les scripts suivants ?

```
1 ## Script 1
2 i = 0
3 while i < 10:
4     print(i)
5
6 ## Script 2
7 i = 0
8 while i > 10:
9     print(i)
```

- 3) Recopier le script suivant, le tester et expliquer à quoi il sert :

```
1 from math import pi
2
3 theta = float(input("Entrer un angle : "))
4 while not 0 <= theta < 2*pi:
5     theta = theta - 2*pi
6 print(theta)
```

Ce script n'est pas parfait : pourquoi ?

Exercice 2 (Échauffement fonctions) :

- 1) a) Recopier le programme suivant, l'exécuter puis tester la fonction dans le shell :

```
1 def f(x):  
2     return x**2 + 3*x - 1
```

Pour tester une fonction, on doit taper dans le shell `f(x)` avec différentes valeurs pour `x`. Par exemple `f(2)`.

- b) Recopier le programme suivant, l'exécuter puis tester la fonction dans le shell :

```
1 def g(x):  
2     print(x**2 + 3*x - 1)
```

- c) Taper ensuite dans le shell les instructions suivantes :

```
>>> f(4)  
>>> g(4)  
>>> x = f(4)  
>>> x  
>>> y = g(4)  
>>> y  
>>> x + y
```

- d) A quoi sert l'instruction `return` dans la fonction `f` ? Quelle est la différence avec le `print` dans la fonction `g` ? Quel est le meilleur choix ?

- 2) Recopier le programme suivant, l'exécuter puis tester les fonctions dans le shell :

```
1 from math import sqrt  
2  
3 def discriminant(a, b, c):  
4     delta = b**2 - 4 * a * c  
5     return delta  
6  
7 def solutions(a, b, c):  
8     delta = discriminant(a, b, c)  
9     if delta > 0:  
10         x1 = (-b - sqrt(delta)) / (2 * a)  
11         x2 = (-b + sqrt(delta)) / (2 * a)  
12         return [x1, x2]  
13     elif delta == 0:  
14         x0 = -b / (2 * a)  
15         return [x0]  
16     else:  
17         return []
```

On remarquera les points suivants :

- ★ Les paramètres d'une fonction ne sont pas définis au moment de la création de la fonction : ils doivent rester quelconque, c'est même un principe fondamental des fonctions.
- ★ On n'utilise plus la fonction `input` pour attribuer des valeurs aux paramètres : on utilise la syntaxe d'appel d'une fonction en écrivant son nom puis, entre parenthèses, les valeurs des paramètres séparés par des virgules.
- ★ Une fonction peut en utiliser une autre : c'est aussi un principe fondamental des fonctions, elles sont réutilisables.

Exercice 3 (Script ou fonction ?) :

- 1) Écrire un script calculant le plus petit entier n tel que $1 \times 2 \times 3 \times \dots \times n > 10^6$.
- 2) Écrire une fonction sans paramètre `plus_petit_entier()` qui calcule et renvoie le plus petit entier n tel que $1 \times 2 \times 3 \times \dots \times n > 10^6$.

Exercice 4 (Origami) : Une feuille de papier a une épaisseur de 0,1 mm. La distance Terre-Lune est d'environ 384 400 km.

- 1) Estimez, sans calcul, combien de fois il faut plier la feuille sur elle-même pour que l'épaisseur de la feuille dépasse la distance Terre-Lune.

Attention : quand on plie la feuille 2 fois sur elle-même, son épaisseur n'est pas de 0,3 mm !

- 2) Déterminez la valeur exacte de ce nombre à l'aide d'une boucle **while**.

Exercice 5 (Suites et fonctions) :

- 1) Écrire une fonction `suite_u(n)` qui calcule et renvoie le n^{e} terme de la suite $(u_n)_{n \geq 0}$ défini par

$$u_0 = 2 \quad \text{et} \quad \forall n \in \mathbb{N}, \quad u_{n+1} = (n+1)u_n - n$$

- 2) Écrire une fonction `suite_v(n)` qui calcule et renvoie le n^{e} terme de la suite $(v_n)_{n \geq 0}$ défini par

$$v_0 = 3, \quad v_1 = 2 \quad \text{et} \quad \forall n \in \mathbb{N}, \quad v_{n+2} = nv_n - v_{n+1}$$

Exercice 6 (Sommes et fonctions) :

- 1) Écrire une fonction `gauss(n)` qui calcule et renvoie la somme $\sum_{k=0}^n k = 0 + 1 + \dots + n$ sans utiliser la formule de Gauss.

- 2) Écrire une fonction `euler(n)` qui calcule et renvoie la somme

$$\sum_{k=1}^n \frac{1}{k^2} = \frac{1}{1^2} + \frac{1}{2^2} + \dots + \frac{1}{n^2}$$

Exécuter votre fonction pour un nombre `n` assez grand et vérifier que `euler(n)  $\approx \frac{\pi^2}{6}$` .

Exercice 7 (Suivre un algorithme) : Écrire une fonction `nombreDeZeros(n)` qui calcule le nombre de zéros qui sont à la fin du nombre entier `n`. Pour cela, on pourra utiliser l'algorithme suivant :

- ★ Si `n = 0`, alors on renvoie la valeur 1.
- ★ Sinon, on définit un variable `R = 0`. Puis, tant que `n` n'est pas nul :
 - Si `n` est divisible par 10, on le divise par 10 avec l'instruction `n = n // 10` puis on ajoute 1 à la variable `R`.
 - Sinon, on renvoie la valeur de `R`.

Exercice 8 (Factorielle) :

- 1) Écrire une fonction `factorielle(n)` qui calcule le nombre $n! = 1 \times 2 \times 3 \times \dots \times n$.
- 2) Nous verrons en dénombrement que le nombre de façons de mélanger un paquet de n cartes est égal à $n!$.
- a) Quel est le nombre de façons de mélanger un paquet de 52 cartes ?
- b) Que pensez-vous alors de l'affirmation :

Chaque mélange de paquet de cartes est unique : il n'a jamais été obtenu avant et ne sera plus jamais obtenu dans le futur.

Exercice 9 (Fonction puissance) : Écrire une fonction `puissance(x, n)` qui calcule le nombre x^n où $x \in \mathbb{R}$ et $n \in \mathbb{N}$.

Il est bien entendu interdit d'utiliser la syntaxe `x**n` ou la fonction **pow** : le but étant de trouver l'algorithme permettant de calculer une puissance.