Épisode III : Les boucles while et for

EXERCICE 1

On considère la fonction suivante :

```
def mystere(n):
    s = 0
    i = 0
    while i <= n :
        s = s + i**2
        i=i+1
    return s
```

1. Quelle est la valeur de `mystere(20)` ?
2. De façon générale, que calcule la fonction `mystere` ?

EXERCICE 2

Dans la fonction `mystere2` suivante, `a` et `n` sont deux entiers positifs supérieurs ou égaux à 0.

```
def mystere2(a, n):
    i = 1
    x = a
    y = n
    while y > 0:
        if y % 2 == 1:
            i = i * x
        x = x * x
        y = y //2
    return i
```

1. Simuler l'exécution de `mystere2(a,n)` lorsque `a` est égale à `2` et `n` est égale à `8` en donnant les valeurs successives des variables `i`, `x` et `y` dans le tableau suivant :

   `mystere2(2,8)`

i	
x	
y	

   Quelle est la valeur retournée par la fonction dans ce cas ?

2. Que retourne la fonction lorsque `a` et `n` prennent les valeurs suivantes ?

a	2	3
n	6	4
valeur retournée		

3. Comment interpréter la valeur retournée par la fonction `mystere2` en général, pour deux valeurs `a` et `n` quelconques ?

EXERCICE 3

Pour simuler le lancé d'un dé à 6 faces on peut utiliser l'appel `randrange(1,7)`. Pour utiliser cette fonction il faudra que votre code commence par la phrase magique :

```
from random import *
```

Écrire le code des fonctions suivantes :

1. `obtenirUn6()` qui retourne le nombre de lancés effectués avant d'obtenir le nombre 6.
2. `obtenirUnDouble()` qui retourne le nombre de lancés effectués pour obtenir deux fois consécutivement le même nombre.

EXERCICE 4

Pour tout entier $n \geq 0$, le *nombre de Cullen d'indice* n est le nombre $n \times 2^n + 1$.

1. Écrire une fonction `cullen(n)` prenant en paramètre un entier `n` et retournant le nombre de Cullen d'indice `n`. On rappelle que Python peut calculer une puissance a^k avec la notation `a**k`.
2. Écrire une fonction `indiceCullen(x)` prenant en paramètre un entier `x`, et retournant le plus grand entier n tel que `x` soit supérieur au nombre de Cullen d'indice n.

EXERCICE 5

On considère la fonction suivante :

```
def mystere3(n):
    s = 0
    while n > 0 :
        s = s + n % 10
        n = n // 10
    return s
```

1. Quelle est la valeur de `mystere3(2705)`. De façon générale, que calcule la fonction `mystere3` ?
2. Écrire une fonction `nombreDeChiffres(n)` qui retourne le nombre de chiffres contenus dans le nombre entier `n`. Par exemple, la valeur de `nombreDeChiffres(2705)` est 4.
3. Écrire une fonction `plusGrandChiffre(n)` qui retourne le plus grand chiffre contenu dans le nombre entier `n`. Par exemple, la valeur de `plusGrandChiffre(2705)` est 7.

EXERCICE 6

On considère la définition suivante :

```
def mystere4(n):
    s = 0
    for i in range(1, n+1):
        s = s + i
    return s
```

Que retournent les appels `mystere4(2)`, `mystere4(3)`, et `mystere4(n)` en général ?
Connaissez-vous une formule qui permet de calculer le même résultat ?

EXERCICE 7

Entrer les instructions suivantes et analyser les réponses de python :

```
for j in range(10):
    print(j, j * j)
```

```
for k in range(3, 8):
    print(k, 2 * k + 1)
```

```
for k in range(3, 9, 2):
    print(k)
```

EXERCICE 8

Écrire une fonction `sommeCarres(n)` qui calcule et retourne $\sum_{i=1}^{n} i^2$.

EXERCICE 9

1. Écrire une fonction `nombreDiviseurs(n)` qui affiche le nombre de diviseurs d'un entier naturel n.
2. Écrire une fonction `estPremier(n)` qui renvoie `True` si le nombre est premier `False` sinon.
   Rappel : un nombre entier est premier s'il est strictement supérieur à 1 et si le nombre de ses diviseurs est égal à deux, 1 et lui-même.

EXERCICE 10

Écrire une fonction `approxPi(nbEtapes)` qui calcule en nbEtapes une approximation de π à partir de la formule de Leibniz :

$$\pi = 4\left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots\right)$$

EXERCICE 11

Écrire une fonction `liste3chiffres()` qui permet d'afficher par ordre croissant tous les nombres à 3 chiffres dont la somme des chiffres est multiple de 5.

EXERCICE 12

1. La fonction `factorielle(n)` (notée mathématiquement « $n!$ ») peut être définie de la manière suivante (on suppose que $n \geq 1$) :

   $$n! = 1 \times 2 \times \cdots \times n$$

   Écrire une fonction `factorielle(n)` qui utilise cette définition pour calculer et retourner $n!$. Tester votre fonction en affichant les factorielles des nombres de 0 à 100.
2. Écrire une fonction `coeff_binomial(n,k)` qui prendre en argument deux entiers naturels n et k (avec $k \leq n$) et qui renvoie la valeur du coefficient binomial correspondant "k parmi n" : $\binom{n}{k} = \frac{n!}{(n-k)!\,k!}$.

L'exercice qui suit est à faire à la maison.

EXERCICE 13

Écrire une fonction `triangle_pascal(nb_lignes)` qui prend en argument un entier nb_lignes et qui affiche le triangle de Pascal (voir définition ci-dessous) en s'arrêtant au bout du nombre de lignes indiqué par l'argument. Voici un exemple du triangle de Pascal avec 6 lignes :

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1

En numérotant les lignes et les colonnes à partir de zéro, le nombre sur la ligne numéro n et la colonne numéro k est le coefficient binomial $\binom{n}{k}$. Par exemple, pour la colonne numéro 0, on a toujours $\binom{n}{0} = 1$ et pour la colonne 1 on a toujours $\binom{n}{1} = n$. De même, lorsque $k = n$, on a $\binom{n}{n} = 1$ donc chaque ligne se termine par un 1.