TD 9 — Programmation moldulaire en Python

Comme presque tous les langages de programamtion, Python permet de programmer de manière modulaire : il est possible de décomposer son code en fonctions, réparties dans différents fichiers.

1 Un peu de théorie

En Python, on définit les fonctions avec le mot-clé `def`. Considérons le code source suivant :

Python

```python
# Renvoie la plus grandes des trois valeurs passées en paramètres
def maximum(a, b, c):
    if a>b and a>c:
        return a
    elif b>a and b>c:
        return b
    else:
        return c

# Procédure (ne renvoie rien) qui affiche bonjour à l'écran
def bonjour(nom):
    print("Bonjour " + nom + " !") # L'opérateur + pour les chaines de caractères
                                   # réalise une concaténation.

# Le programme principal est situé sous les fonctions.
print(maximum(7.8, -9.6, 10.5))
print(maximum('a', 's', 'q))
bonjour("Alice")
```

Une remarque sur les types

Comme on le voit ici, les types ne sont pas précisés. *Cela ne signifie pas que les valeurs sont non typées.* On dit que le typage est *implicite*. En Python, les expressions sont typées, pas les variables. Par conséquent, l'instruction `maximum('a', 's', 'q)` est valide et la fonction renvoie `'s'`. On peut également appeler la fonction `maximum` avec des données de type flottant ou même des chaines de caractères. À vous de tester ce qui se passe…

Exercice 1. Récupérer le code des deux fonctions données en exemple et les intégrer dans un fichier `exo1.py`. Tester les fonctions `maximum` et `bonjour` avec différentes valeurs et différents types pour les paramètres.

Pour plus de modularité, on peut réunir les fonctions dans des fichiers séparés. Chaque fichier forme alors un *module* ou une *bibliothèque*[1]. Ainsi, pour importer les fonctions du fichier `exo1.py`, on ajoute au début du code source la ligne `import exo1`. L'appel aux fonctions sera alors `exo1.maximum(7.8, -9.6, 1`

1. Bibliothèque : *library* en anglais.

2 Exercices d'application

À rendre

Vous devez rendez *trois fichiers* : `td9_exercices_<NOMDEFAMILLE>.py` et `td9_main_<NOMDEFAMILLE>.py`. Le premier fichier constituera un module contenant les diverses fonctions programmées, et le deuxième fichier le programme principal, qui importe le module et contient les appels aux fonctions. Vous rendrez également le fichier associé à l'exercice 1.

Exercice 2 (Tables de multiplication). **1)** Écrire une fonction `table_de_7()` qui affiche la table de multiplication par 7.

2) Écrire une fonction `table_de_mult(n)` qui affiche la table de mulitplication par `n`.

Exercice 3. Écrire une fonction qui reçoit deux paramètres `nom` et `prenom` et qui renvoie la chaine de caractères formée du prénom, d'une espace et du nom (en lettres capitales). Par exemple, si `prenom = "Bob"` et `nom = "Razowski"`, la fonction renvoie `"Bob RAZOWSKI"`.

Mettre une chaines de caractères en majuscules

Si `s` est une chaine de caractères, alors `s.upper()` renvoie la chaine `s` en majuscules.

Exercice 4 (Polynômes du second degré). **1)** Écrire une fonction qui reçoit un paramètre flottant x et renvoie la valeur du polynôme $P(x) = 3x^2 - 7x + 4$ évalué en x.

2) Écrire une fonction qui reçoit quatre paramètres a, b, c, x et renvoie la valeur de $ax^2 + bx + c$.

Exercice 5 (Récursivité — Fonction factorielle). On dit qu'une fonction est *récursive* lorsqu'elle s'appelle elle-même. C'est le concept analogue, en informatique, de la *récurrence* en mathématiques. Pour tout $n \in \mathbb{N}^*$, on appelle *factorielle de* n, et on note $n! = n \cdot (n-1) \cdots 2 \cdot 1$, le produit de tous les entiers strictement positifs inférieurs ou égaux à n.

1) Programmer une fonction `factorielle_iterative(n)` qui calcule la factorielle de l'entier `n` reçu en paramètre, avec une boucle itérative.

2) Programmer une fonction `factorielle_recursive(n)` qui calcule la factorielle de l'entier `n` de manière récursive. Notez que pour tout $n \geq 2$, $n! = n \cdot (n-1)!$.

Exercice 6 (Somme des premiers entiers). Pour tout $n \in \mathbb{N}^*$, on note S_n la somme des entiers naturels inférieurs ou égaux à n, i.e.

$$S_n = \sum_{k=0}^{n} k$$

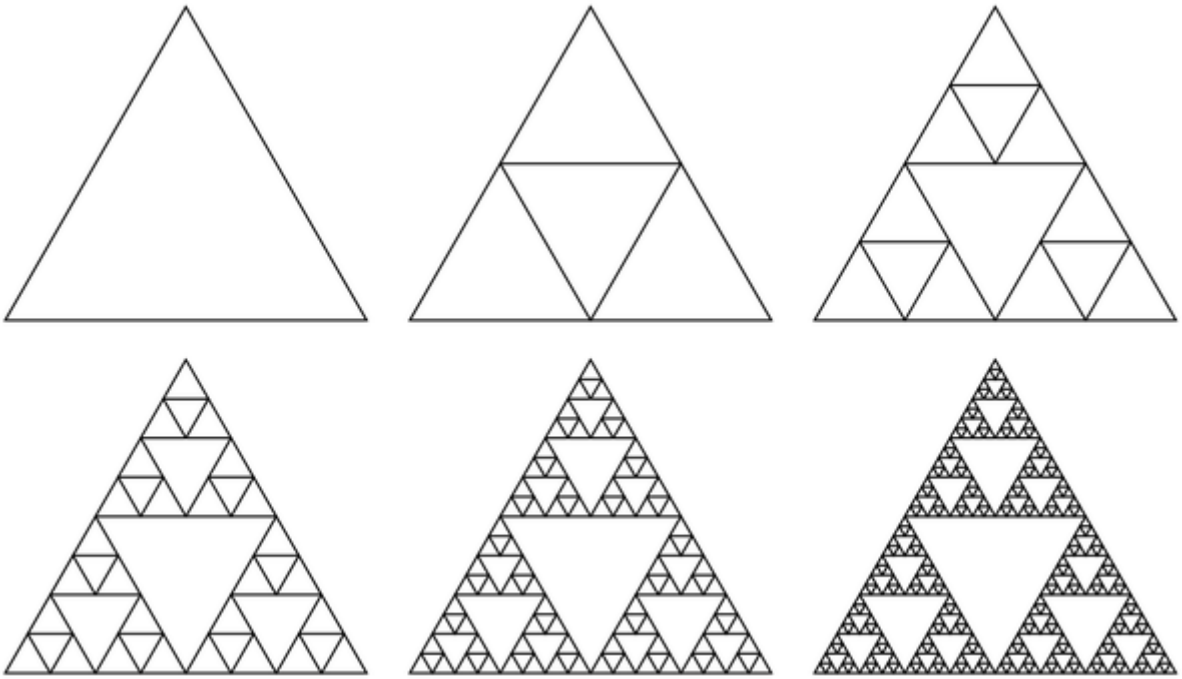
1) Écrire une fonction calcule en revoie cette somme de manière itérative.

2) Faire de même avec une fonction récursive.

3) Écrire maintenant une fonction récursive qui prend deux paramètres : un entier n et un entier p. Elle doit renvoyer

$$\sum_{k=1}^{n} k^p$$

Exercice 7 (Triangle de Sierpiński)**.** Le triangle de Sierpiński est une figure fractale que l'on peut obtenir [2] au moyen d'une fonction récursive. Importer le module Turtle pour faire des dessins. Écrire une fonction récursive `sierpinski(L, n)` où `L` est un paramètre de longueur et `n` un paramètre de profondeur. Pour `n = 1, 2, 3, 4, 5, 6`, on obtient les figures suivantes :



Le principe est le suivant si `n>0`, alors répéter trois fois :

- Appeler récursivement la fonction `sierpinski(L/2, n-1)`.
- Avancer de `L` pas.
- Tourner de 120°.

2. De manière approchée…