# Boucles et fonctions en Python

Olivier Baudon, Marc Zeitoun

3 octobre 2023

Les exercices marqués ♠ ne seront pas corrigés en TD : ils sont à faire en autonomie.

Pour **écrire, sauvegarder et exécuter** du code, nous utiliserons le logiciel **Thonny**. Il est conseillé au début de **visualiser** l'exécution **pas à pas** de vos programmes. Le mode pas-à-pas peut être activé directement dans Thonny par la touche **Control-F5** (icône « déboguer le script courant »). Puis :

— la touche `F6` (*Step over*) permet d'exécuter l'instruction courante. Si c'est un appel de fonction, cette appel sera exécuté entièrement.

— la touche `F7` (*Step into*) permet d'exécuter la l'instruction courante, mais s'il y a un appel de fonction sur cette instruction, on ***entrera*** dans cette fonction pour l'exécuter, elle aussi, pas-à-pas.

Une autre façon d'exécuter du code Python est d'utiliser le site http://www.pythontutor.com/visualize.html. Pour cela, copiez-collez le code écrit sous Thonny dans la fenêtre de saisie de code (en sélectionnant Python 3 ou Python 3.6 comme version du langage), et cliquez sur *Visualize Execution*.

## 1 Les boucles

Nous déjà avons vu qu'il est possible de demander la répétition d'une ou plusieurs instructions grâce à l'instruction `while`.

```
while <condition>:
  instruction1
  instruction2
  ...
```

L'effet d'une telle boucle est le suivant :

1. La condition est d'abord évaluée.
2. Si elle est fausse (c'est-à-dire qu'elle a la valeur `False`), on saute le bloc d'instructions indentées par rapport au mot-clé `while`, et on continue le programme à l'instruction qui suit ce bloc.
3. Si elle est vraie (c'est-à-dire qu'elle a la valeur `True`), les instructions se trouvant dans le bloc d'instructions indentées par rapport au mot-clé `while` sont exécutées, et on revient à l'étape 1.

**Exercice 1.1 — Boucle while.** Quel effet l'exécution du code suivant produit-elle ?

```
i = 1
while i <= 10:
  print(i)
  i = i + 1
```

Il existe une autre instruction permettant de répéter certaines instructions : la boucle `for`. Un exemple d'utilisation est le suivant :

```
for i in range(1, 11):
  print(i)
```

L'effet est de répéter les instructions à l'intérieur de la boucle (ici, il n'y en a qu'une : `print(i)`) pour chaque valeur de la variable `i` allant de `1` (inclus) jusqu'à `11` (exclu). La variable `i` est **automatiquement** incrémentée d'une unité à chaque « tour de boucle », il **ne faut pas** la modifier soi-même.

Par défaut dans la boucle `for` ci-dessus, la variable `i` est augmentée de 1 à chaque nouveau exécution de `print(i)`. On peut aussi préciser, dans l'instruction `for`, une valeur par laquelle incrémenter la variable de la boucle. Par exemple,

```
for i in range(0, 100, 20):
  print(i)
```

affiche les entiers 0, 20, 40, 60, 80 (mais pas 100, car la borne d'arrivée est exclue), et

```
for zzz in range(5, 0, -1):
  print(zzz)
```

affiche les entiers 5, 4, 3, 2, 1 (chacun sur une ligne, puisque l'instruction `print` revient à la ligne après chaque affichage).

**Exercice 1.2 — Boucle for.** Quel effet l'exécution du code suivant produit-elle ?

```
for i in range(5, 20, 5):
  print(i)
```

**Exercice 1.3 — Boucles while et for.** 1. Quel est l'effet du code suivant ?

```
somme = 0
for i in range(1, 10):
  somme = somme + i
print(somme)
```

2. Ré-écrire le code précédent en utilisant la boucle `while` au lieu de la boucle `for`.
3. Quel est l'effet du code suivant ?

```
x = 1
resultat = 1
while x <= 10:
  resultat = resultat * x
  x = x + 2
```

```
print(resultat)
```

4. Ré-écrire le code précédent en utilisant la boucle `for` au lieu de la boucle `while`.

## 2 Introduction aux fonctions

Les fonctions permettent d'encapsuler des morceaux de code pour les répéter. Elles peuvent :

— dépendre de paramètres,

— calculer des valeurs, qui peuvent être utilisées par d'autres fonctions.

Nous n'aborderons pas le second point dans cette feuille.

Le code suivant **définit** une fonction dont le nom est `imprime_5`, et qui n'a pas de paramètres :

```
def imprime_5():
    for i in range(0,5):
        print(i, "Bonjour")
```

Le nom de la fonction (ici, `imprime_5`) est choisi par la personne qui programme. Après exécution de ce code, rien de visible ne se produit. En particulier, on ne voit aucun affichage malgré l'utilisation de la fonction `print`, mais la fonction `imprime_5` est maintenant enregistrée et peut servir autant de fois que nécessaire. Pour **utiliser** une fonction, on donne son nom, suivi d'une liste de valeurs pour chacun de ses paramètres. Ces valeurs sont écrites entre parenthèses, et séparés par une virgule. Dans notre cas, comme il n'y a pas de paramètres, il suffit d'exécuter :

```
imprime_5()
```

Notez que les parenthèses sont obligatoires. Toutes les instructions que l'on a pré-enregistrées dans la fonction sont alors exécutées.

**Exercice 2.1** Quel est l'effet du programme suivant ? Testez-le.

```
def imprime_5():
    for i in range(0,5):
        print(i, "Bonjour")

imprime_5()
```

On peut faire dépendre une fonction d'un ou plusieurs paramètres. Lors de la **définition** de la fonction, on donne aux paramètres, dont on ne connaît pas encore la valeur exacte, des noms symboliques, entre les parenthèses qui suivent le nom de la fonction. S'il y a plusieurs paramètres, ils sont séparés les uns des autres par une virgule. Par exemple, la fonction suivante a un seul paramètre, `n`.

```
def imprime(n):
    for i in range(0,n):
        print(i, "Bonjour")
```

Lorsqu'on veut **utiliser** la fonction, on doit donner une valeur concrète à chaque paramètre. Pour cela, on appelle la fonction en donnant explicitement ces valeurs (dans l'ordre où apparaissent les paramètres, s'il y en a plusieurs). Par exemple :

```
imprime(10)
```

exécutera le code de la fonction `imprime` avec `10` comme valeur du paramètre `n`.

On peut aussi utiliser le nom des paramètres lors de l'appel, par exemple :

```
imprime(n=10)
```

Dans ce cas, s'il y a plusieurs paramètres nommés, il n'est pas nécessaire de respecter leur ordre.

**Exercice 2.2** Quel est l'effet du programme suivant ? Testez-le.

```
def imprime(n):
    for i in range(0,n):
        print(i, "Bonjour")

imprime(10)
```

**Exercice 2.3** 1. Écrire une fonction qui affiche la table de multiplication de 7, de 0 jusqu'à 9. La fonction, lorsqu'on l'appelle, doit donc afficher :

```
0 * 7 = 0
1 * 7 = 7
2 * 7 = 14
3 * 7 = 21
4 * 7 = 28
5 * 7 = 35
6 * 7 = 42
7 * 7 = 49
8 * 7 = 56
9 * 7 = 63
```

2. En modifiant cette fonction, écrire une fonction qui prend un paramètre `n` et qui affiche la table de multiplication de `n`, de 0 à 9.

**Exercice 2.4 — ♠.** 1. Écrire une fonction qui prend maintenant trois paramètres, `n`, `start` et `stop`, et qui affiche la table de multiplication de `n`, de `start` à `stop`.

2. En **utilisant** la fonction précédente, écrire une fonction qui prend 4 paramètres, `n`, `m`, `start` et `stop`, et qui affiche toutes les tables de multiplication de chaque entier entre `n` et `m`, de `start` à `stop`, avec un titre pour chacune d'elles.