Chapitre 3 : Boucles et fonctions

Spé ISN

À partir d'une version originale de M.Duclosson

Introduction

Pour répéter la même opération un certain nombre de fois jusqu'à obtenir un certain résultat, on dispose de deux types de boucles.
Pour factoriser du code, on peut aussi étendre le langage avec une fonction qui englobe un bloc d'instructions.
L'objectif de ce chapitre est d'apprendre à utiliser de manière pertinente des boucles et des fonctions.

I Boucle inconditionnelle

Lorsque l'on veut répéter un certain nombre de fois un ensemble d'instructions on utilise la boucle inconditionnelle ou boucle `for` :

Programme 1

```
>>> for k in range(4):
        print("bonjour")

bonjour
bonjour
bonjour
bonjour
```

Programme 2

```
>>> for k in range(4):
        print(k)

0
1
2
3
```

Ici la fonction range renvoie un *itérateur* qui produit consécutivement les valeurs entières de 0 à 5. La boucle `for` ne fait que parcourir les valeurs de cet itérateur.
Plus généralement, la boucle `for` permet de parcourir tout objet *itérable* :

Programme 3

```
>>> for c in 'AB':
        print(c)

A
B
```

Programme 4

```
>>> for x in [2, 6]:
        print(x**2)

4
36
```

La syntaxe générale d'une boucle inconditionnelle `for` est :

```
for element in iterable:
    instruction   #
    instruction   # bloc d'instructions
    instruction   #
```

La fonction `range` possède trois arguments dont deux sont optionnels :

- `range(n)` retourne un itérateur parcourant les entiers consécutifs entre 0 et `n` exclu.
- `range(m,n)` retourne un itérateur parcourant les entiers consécutifs entre `m` compris et `n` exclu.
- `range(m,n,s)` retourne un itérateur parcourant les entiers consécutifs entre `m` compris et `n` exclu avec un pas de `s`.

Exercice 1

1. Faire afficher les entiers de 10 à 0 de manière décroissante.

   .....

   .....

2. Faire afficher les entiers pairs compris entre 0 et 50 dans l'ordre croissant.

   .....

   .....

3. Faire afficher les entiers pairs compris entre 0 et 50 dans l'ordre décroissants.

   .....

   .....

On peux imbriquer deux boucles :

Programme 5

```
for i in range(10):
    for j in range(10):
        print(i, 'dizaines et ', j, ' unités font ', 10*i + j)
```

Exercice 2

1. Quel affichage obtient-on en exécutant le code ci-dessous ?

.....

.....

.....

.....

```
for i in range(5):
    for j in range(i, 5):
        print(j, end="")
    print()
```

2. Pour les questions suivantes, il faut utiliser obligatoirement deux boucles imbriquées et au maximum deux fois la fonction `print`.

a. Écrire un script Python qui produit l'affichage 1 ci-dessous.

.....

.....

.....

.....

b. Écrire un script Python qui produit l'affichage 2 ci-dessous.

.....

.....

.....

.....

Affichage 1

```
0
01
012
0123
01234
```

Affichage 2

```
01234
12345
23456
34567
45678
```

Entraînement 1

En 2020, le mois de juin commence un lundi. À l'aide de deux boucles imbriquées, faire afficher les jours des quatre premières semaines du mois sous la forme :

```
lundi 1 mai
mardi 2 mai
.
.
```

II Fonctions

II.1 Définir une fonction

Lorsqu'on a besoin de réutiliser tout un bloc d'instructions, on peut l'encapsuler dans une **fonction**. On étend ainsi le langage avec une nouvelle instruction. Une fonction sert à factoriser du code, à rendre le programme plus lisible et plus facile à maintenir et à partager. C'est un outil de modularité.
Pour déclarer une fonction, on définit son **en-tête** avec son **nom** et des **paramètres formels** d'entrée. Viennent ensuite le bloc d'instructions et une valeur de retour. En Python on définit une fonction ainsi :

Programme 6

```
def mafonction(parametre1, parametre2):
    #bloc d'instructions (optionnel)
    return valeur
```

Par exemple une fonction `carre` qui prend en paramètre un nombre `x` et qui retourne son carré, s'écrira :

Programme 7

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

Une fonction peut prendre plusieurs paramètres. Par exemple une fonction `carre_distance_origine(x,y)` qui prend en paramètres deux nombres `x` et `y` et qui retourne le carré de la distance d'un point de coordonnées `(x, y)` à l'origine d'un repère orthonormal, s'écrira :

Programme 8

```
def carre_distance_origine(x, y):
    return x ** 2 + y ** 2
```

Une fonction peut retourner un tuple de valeurs. Par exemple une fonction `coord_vecteur` qui prend en paramètres quatre nombres `xA, yA, xB, yB` et qui retourne les coordonnées du vecteur lié dont les extrémités ont pour coordonnées `(xA, yA)` et `(xB, yB)`, s'écrira :

Programme 9

```
def coord_vecteur(xA, yA, xB, yB):
    return (xB - xA, yB - yA)
```

Nous venons de voir la définition d'une fonction. On exécute la fonction en substituant aux **paramètres formels** des valeurs particulières appelées **arguments**. Cette instruction se nomme un **appel de fonction** :

```
In [4]: carre(3)
Out[4]: 9

In [5]: carre_distance_origine(1,2)
Out[5]: 5

In [6]: coord_vecteur(1,5,4,2)
Out[6]: (3, -3)
```

Il existe aussi des fonctions sans paramètres d'entrée ou sans valeur de retour. Par exemple, une fonction qui affiche le nom complet d'un acronyme peut s'écrire :

Programme 10

```
def isn():
    print("Informatique et Science du Numérique")
```

Attention, `return valeur` retourne `valeur` qu'on peut capturer dans une variable alors que `print(valeur)` affiche valeur sur la sortie standard (l'écran par défaut) mais valeur ne peut alors être capturée dans une variable. On donne ci-dessous un extrait de console Python, où on a défini maladroitement une fonction cube avec un `print` à la place d'un `return`. On ne récupère pas la valeur de retour souhaitée mais `None` lorsqu'on appelle la fonction.

```
In [10]: def cube(x):
    ...:     print(x ** 3)
    ...:

In [11]: cube(4)
64

In [12]: b = cube(5)
125
```

```
In [13]: b

In [14]: print(type(b))
<class 'NoneType'>

In [15]: print(b + 1)

TypeError: unsupported operand type(s) for +: 'NoneType' and 'int'
```

Exercice 3

1. L'Indice de Masse Corporelle se calcule par la formule IMC = $\frac{\text{masse}}{\text{taille}^2}$ où la masse est en kilogrammes et la taille en mètres. Un IMC est considéré comme normal s'il est compris entre 18,5 et 25. En dessous de 18,5, la personne est en sous-poids et au-dessus de 25 elle est en sur-poids.

Écrire une fonction d'en-tête `imc(m, t)` qui retourne la classification de l'IMC correspondant à une masse de `m` kilogrammes et une taille de `t` mètres : classe 0 pour sous-poids, 1 pour normal et 2 pour surpoids.

.....

.....

.....

.....

.....

2. **a.** Écrire une fonction `max2(a, b)` qui retourne le maximum de deux nombre `a` et `b`.

.....

.....

.....

.....

.....

b. Écrire une fonction `max3(a, b, c)` qui retourne le maximum de trois nombres `a`, `b` et `c`.

.....

.....

.....

.....

.....

Principaux opérateurs de comparaison de variables

`x == y`	`x est égal à y`
`x != y`	`x est différent de y`
`x > y`	`x est strictement supérieur à y`
`x < y`	`x est strictement inférieur à y`
`x >= y`	`x est supérieur ou égal à y`
`x <= y`	`x est inférieur ou égal à y`

Principaux opérateurs sur des expressions booleenes

`E and F`	`Vraie si E est Vraie ET F est Vraie`
`E or F`	`Vraie si E est Vraie OU F est Vraie`
`not E`	`Vraie si E est Fausse`

Exercice 4 *Fonctions de tests*

1. Écrire une fonction `aumoinsun(a,b,c)` qui retourne un booléen indiquant si l'un au moins des nombres `a, b` ou `c` est positif.

   .....

   .....

   .....

2. Écrire une fonction `tous(a,b,c)` qui retourne un booléen indiquant si tous les nombres `a, b, c` sont positifs.

   .....

   .....

   .....

3. Écrire une fonction `croissant(a,b,c)` qui retourne un booléen indiquant si `a, b, c` sont dans l'ordre croissant.

   .....

   .....

   .....

4. Une année est bissextile si elle est divisible par 400 ou si elle n'est pas divisible par 100 et qu'elle est divisible par 4. Écrire une fonction `bissextile(a)` qui retourne un booléen indiquant si l'année `a` est bissextile.

   .....

   .....

   .....

Entraînement 2

Écrire une fonction `nombreRacinesTrinome(a,b,c)` qui retourne le nombre de racines distinctes d'un trinôme $T : x \mapsto ax^2 + bx + c$.

II.2 Utiliser des bibliothèques de fonctions

Méthode

On a parfois besoin d'utiliser des fonctions de Python qui ne sont pas chargées pas défaut. Ces fonctions sont stockées dans des programmes Python appelées **modules** ou **bibliothèques**. Par exemple le module `math` contient les fonctions mathématiques usuelles et le module `random` contient plusieurs types de générateurs de nombres pseudo-aléatoires.
Pour importer une fonction d'un module on peut procéder de deux façons :

```
#import du module de mathématique (création d'un point d'accès)
import math

#pour utiliser la fonction sqrt, on la préfixe du nom du module et d'un
    point
racine = math.sqrt(2)
```

Première façon

```
#import de la fonction sqrt du module math
from math import sqrt

racine = sqrt(2)

#Pour importer toutes les fonctions de math, ecrire
#from math import *
```

Deuxième façon

Pour obtenir de l'aide sur le module math dans la console Python, il faut d'abord l'importer avec `import math` puis taper `help(math)`, mais le mieux est encore de consulter la documentation en ligne https://docs.python.org/3/. Sans connexion internet, on peut lancer en local le serveur web de documentation avec la commande `python3 -m pydoc -b`.

Principales fonctions du modules `random` :

Fonction	Effet
`randrange(a,b)`	retourne un entier aléatoire dans [a;b[
`randint(a,b)`	retourne un entier aléatoire dans [a;b]
`random()`	retourne un décimal aléatoire dans [0;1[
`uniform(a,b)`	retourne un décimal aléatoire dans [a;b]

Exercice 5

1. Écrire une fonction `sommeDe(n)` qui retourne la somme des résultats obtenus en lançant `n` dé à 6 faces.

.....

.....

.....

.....

2. Écrire une fonction `urne()` qui retourne le numéro de la boule tirée dans une urne qui contient cinq boules numérotées 1, trois boules numérotées 2 et deux boules numérotées 3.

.....

.....

.....

.....

.....

.....

Méthode

Le module `turtle` est une implémentation en `Python` du langage `Logo` créé dans les années 1970 pour l'enseignement de l'informatique à l'école. Il est disponible dans la distribution standard de `Python`.
En déplaçant une pointe de stylo qui peut être matérialisée par une tortue, on peut tracer des figures géométriques dans un repère cartésien dont l'origine est au centre de la fenêtre et dont l'unité par défaut est le pixel. Lorsqu'on déplace le crayon, il laisse une trace s'il est baissé ou pas de trace s'il est levé.
Nous utiliserons les fonctions suivantes de `turtle`.

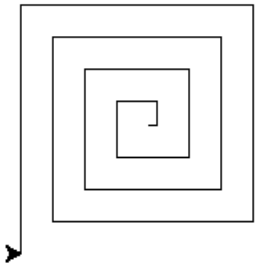
Syntaxe	Sémantique
`goto(x,y)`	déplace la tortue jusqu'au point de coordonnées (x, y)
`penup()`	lever le crayon
`pendown()`	baisser le crayon
`setheading(angle)`	choisir l'angle d'orientation de la tortue en degrés
`forward(n)`	avancer de `n` pixels selon l'orientation de la tortue
`color("red")`	choisir la couleur rouge (ou "black", "green", "blue" ...)

On donne ci-dessous un programme permettant de tracer une spirale.

Programme 11

```
from turtle import *

penup()
goto(0,0)
pendown()
c = 5
for i in range(4):
    for j in range(4):
        forward(c)
        c = 10 + c
        left(90)
exitonclick()
```



Exercice 6

1. Écrire une fonction `spirale1(n)` qui permet de tracer une spirale constituée de `n` carrés déformés.

.....

.....

.....

.....

.....

2. Écrire une fonction `spirale2(n, m)` qui permet de tracer une spirale constituée de `n` polygones déformés à m côtés.

.....

.....

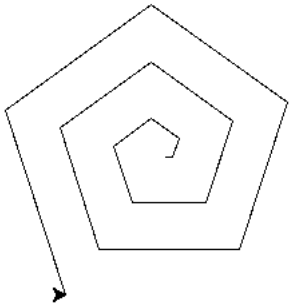
.....

.....

.....

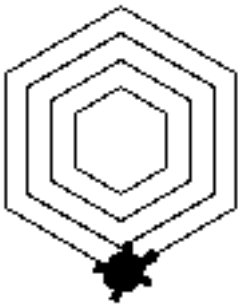
.....

.....



Entraînement 3

Écrire une fonction `spirale3(n, m)` qui permet de tracer une spirale constituée de `n` polygones réguliers concentriques à m côtés.



III Boucle conditionnelle

Dans une boucle `for`, le nombre d'itérations est connue à l'avance (au plus tard lors de la première exécution de l'instruction). Lorsque l'on ne sait pas à l'avance combien de fois la boucle devra être exécutée, on utilise une boucle `while`.
La syntaxe est la suivante :

Programme 12

```
while condition:
    instruction   #
    instruction   #   bloc d'instructions
    instruction   #
```

La `condition` est une expression qui doit pouvoir être évaluée sous forme de booléen. Tant que sa valeur est True, le bloc d'instructions est exécuté.

Lors de l'utilisation de `while`, il faudra s'assurer que la condition finisse par prendre la valeur `False` sans quoi la boucle ne se terminera pas!!
Remarque : Pour faire une opération jusqu'à obtenir une certaine condition il suffit d'ajouter l'opérateur logique `not()`.

Exercice 7

1. Écrire un programme équivalent au programme `1` à l'aide d'une boucle `while`.

.....

.....

.....

.....

.....

2. La suite de Syracuse est une suite d'entiers naturels définie de la manière suivante : on part d'un nombre entier plus grand que zéro ; s'il est pair, on le divise par 2 ; s'il est impair, on le multiplie par 3 et on ajoute 1. Une conjecture non démontrée stipule que la suite atteint 1 quel que soit le premier terme. Écrire une fonction `tempsvol(n)` qui retourne le rang du premier terme égal à 1 de la suite de Syracuse si le terme de rang 0 est `n`

.....

.....

.....

.....

.....

.....

.....

.....

Exercice 8

Les opérateurs `//` et `%` permettent de calculer respectivement le quotient et le reste de la division euclidienne de deux entiers.

```
In [1]: 43 // 5
Out[1]: 8

In [2]: 43 % 5
Out[2]: 3
```

1. Avec l'algorithme d'Euclide déterminer le Plus Grand Commun Diviseur des entiers 1050 et 315.

.....

.....

.....

.....

2. Écrire une fonction `euclide(a, b)` qui retourne le PGCD des entiers `a` et `b` passés en paramètres.

.....

.....

.....

.....

.....

.....

Entraînement 4

On lance un dé équilibré à six faces numérotées de 1 à 6.
Le code Python ci-dessous permet d'afficher la face supérieure du dé lors de dix lancers successifs d'un dé à 6 faces. On commence par importer la fonction `randint` du module `random`. Cette fonction prend deux paramètres : par exemple `randint(1, 6)` retourne un entier aléatoire compris entre 1 et 6, les bornes sont incluses.

```
from random import randint
for k in range(10):
    print(randint(1, 6))
```

1. Écrire une fonction `moyenneDe(n)` qui retourne la valeur moyenne des faces obtenues sur un échantillon de `n` lancers.
2. Écrire une fonction `premier6()` qui retourne le rang du premier 6 obtenu lorsqu'on lance successivement le dé.
3. Écrire une fonction `tempsAttente(n)` qui retourne le temps d'attente moyen du premier 6 sur un échantillon de `n` lancers.