Boucles_Fonctions_2019_2020

September 30, 2019

1 Chapitre 3 : Boucles et fonctions

2 Boucle inconditionnelle

2.0.1 Définition

Une boucle inconditionnelle permet de répéter un bloc d'instructions lorsqu'on connaît à l'avance le nombre de répétitions.

La situation la plus courante est :

```
Pour k allant de 1 à 10 répéter
    Bloc d'instructions
FinPour
```

Sa traduction en `Python` est :

```
# flux parent
for k in range(1, 11):
    # bloc d'instructions
# retour au flux parent
```

On remarque l'utilisation `range(1, 11)` alors qu'on attendrait `range(1, 10)`.

En fait `range(1, 11)` retourne un *itérateur*, qui va parcourir avec un incrément de 1 tous les entiers n tels que $1 \leqslant n < 11$.

Il faut bien se souvenir que la borne supérieure de `range(n, m)` est exclue mais il est facile de retenir que le nombre de tours de boucles est $m - n$.

Par exemple, pour calculer la somme des tous les entiers consécutifs de 100 à 200, on peut écrire :

```
somme = 0
for k in range(100, 201):
    somme = somme + k
print("La somme est ", somme)
```

S'il s'agit juste de répéter `n` fois un bloc d'instructions on utilise le raccourci `range(n)` au lieu de `range(0, n)` ou de `range(1, n + 1)`.

Par exemple pour dire 100 fois "Merci", on peut écrire :

```
for k in range(100):
    print("Merci !")
print("Ouf!")
```

La fonction range offre un troisième paramètre optionnel qui permet de changer l'incrément par défaut qui est 1.

Par exemple, pour calculer la somme des tous les entiers pairs consécutifs entre 100 et 200, on peut écrire :

```
sommePair = 0
for k in range(100, 201, 2):
    sommePair = sommePair + k
print("La somme est ", sommePair)
```

Enfin, la boucle `for` permet aussi de parcourir des objets itérables comme les chaînes de caractères, les listes, les fichiers etc ...

Par exemple pour afficher les caractères consécutifs de la chaîne "Bonjour" avec un caractère par ligne, on peut écrire :

```
chaine = "Bonjour"
for c in chaine:
    print(c)
```

Autre exemple, pour afficher des messages d'invitation personnalisés :

```
for nom in ["Jean-Luc", "Emmanuel", "François", "Marine"]:
    print("Salut", nom, "je t'invite à mon anniversaire samedi !")
```

2.1 Programme 1

```
In [32]: for k in range(4):
             print(k)

0
1
2
3
```

2.2 Exercice 1

Faire afficher les entiers de 10 à 0 de manière décroissante

```
In [5]: for n in range(10, -1, -1):
            print(n)

10
9
8
7
```

```
6
5
4
3
2
1
0
```

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

```
In [1]: for n in range(0, 51, 2):
            print(n)

0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40
42
44
46
48
50
```

Faire afficher les entiers pairs compris entre 0 et 50 dans l'ordre décroissant

```
In [3]: for n in range(50, -1, -2):
            print(n)

50
48
```

```
46
44
42
40
38
36
34
32
30
28
26
24
22
20
18
16
14
12
10
8
6
4
2
0
```

Un exemple de boucles imbriquées

```
In [9]: for i in range (10) :
            for j in range (10) :
                print(i , 'dizaines et ', j, ' unités font ', 10 * i + j)
```

```
0 dizaines et  0  unités font  0
0 dizaines et  1  unités font  1
0 dizaines et  2  unités font  2
0 dizaines et  3  unités font  3
0 dizaines et  4  unités font  4
0 dizaines et  5  unités font  5
0 dizaines et  6  unités font  6
0 dizaines et  7  unités font  7
0 dizaines et  8  unités font  8
0 dizaines et  9  unités font  9
1 dizaines et  0  unités font  10
1 dizaines et  1  unités font  11
1 dizaines et  2  unités font  12
1 dizaines et  3  unités font  13
1 dizaines et  4  unités font  14
1 dizaines et  5  unités font  15
```

```
1 dizaines et  6  unités font  16
1 dizaines et  7  unités font  17
1 dizaines et  8  unités font  18
1 dizaines et  9  unités font  19
2 dizaines et  0  unités font  20
2 dizaines et  1  unités font  21
2 dizaines et  2  unités font  22
2 dizaines et  3  unités font  23
2 dizaines et  4  unités font  24
2 dizaines et  5  unités font  25
2 dizaines et  6  unités font  26
2 dizaines et  7  unités font  27
2 dizaines et  8  unités font  28
2 dizaines et  9  unités font  29
3 dizaines et  0  unités font  30
3 dizaines et  1  unités font  31
3 dizaines et  2  unités font  32
3 dizaines et  3  unités font  33
3 dizaines et  4  unités font  34
3 dizaines et  5  unités font  35
3 dizaines et  6  unités font  36
3 dizaines et  7  unités font  37
3 dizaines et  8  unités font  38
3 dizaines et  9  unités font  39
4 dizaines et  0  unités font  40
4 dizaines et  1  unités font  41
4 dizaines et  2  unités font  42
4 dizaines et  3  unités font  43
4 dizaines et  4  unités font  44
4 dizaines et  5  unités font  45
4 dizaines et  6  unités font  46
4 dizaines et  7  unités font  47
4 dizaines et  8  unités font  48
4 dizaines et  9  unités font  49
5 dizaines et  0  unités font  50
5 dizaines et  1  unités font  51
5 dizaines et  2  unités font  52
5 dizaines et  3  unités font  53
5 dizaines et  4  unités font  54
5 dizaines et  5  unités font  55
5 dizaines et  6  unités font  56
5 dizaines et  7  unités font  57
5 dizaines et  8  unités font  58
5 dizaines et  9  unités font  59
6 dizaines et  0  unités font  60
6 dizaines et  1  unités font  61
6 dizaines et  2  unités font  62
6 dizaines et  3  unités font  63
```

```
6 dizaines et  4  unités font  64
6 dizaines et  5  unités font  65
6 dizaines et  6  unités font  66
6 dizaines et  7  unités font  67
6 dizaines et  8  unités font  68
6 dizaines et  9  unités font  69
7 dizaines et  0  unités font  70
7 dizaines et  1  unités font  71
7 dizaines et  2  unités font  72
7 dizaines et  3  unités font  73
7 dizaines et  4  unités font  74
7 dizaines et  5  unités font  75
7 dizaines et  6  unités font  76
7 dizaines et  7  unités font  77
7 dizaines et  8  unités font  78
7 dizaines et  9  unités font  79
8 dizaines et  0  unités font  80
8 dizaines et  1  unités font  81
8 dizaines et  2  unités font  82
8 dizaines et  3  unités font  83
8 dizaines et  4  unités font  84
8 dizaines et  5  unités font  85
8 dizaines et  6  unités font  86
8 dizaines et  7  unités font  87
8 dizaines et  8  unités font  88
8 dizaines et  9  unités font  89
9 dizaines et  0  unités font  90
9 dizaines et  1  unités font  91
9 dizaines et  2  unités font  92
9 dizaines et  3  unités font  93
9 dizaines et  4  unités font  94
9 dizaines et  5  unités font  95
9 dizaines et  6  unités font  96
9 dizaines et  7  unités font  97
9 dizaines et  8  unités font  98
9 dizaines et  9  unités font  99
```

2.3 Exercice 2 Boucles imbriquées

2.3.1 Question 1

```
In [4]: for i in range(5):
            for j in range(i, 5):
                print(j, end="")
            print()
```

```
01234
1234
```

```
234
34
4
```

2.3.2 Question 2

```
In [8]: for i in range(1,6):
            for j in range(0,i):
                print(j, end="")
            print()
```

```
0
01
012
0123
01234
```

2.3.3 Question 3

```
In [9]: for i in range(0,5):
            for j in range(i, i +5):
                print(j, end="")
            print()
```

```
01234
12345
23456
34567
45678
```

2.4 Entrainement 1

En 2017, le mois de mai 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
```

```
In [1]: jour = ['lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi', 'samedi', 'dimanche']
        for s in range(4):
            for j in range(7):
                print(jour[j], j + 1 + s * 7, 'juillet')
```

```
lundi 1 juillet
mardi 2 juillet
```

```
mercredi 3 juillet
jeudi 4 juillet
vendredi 5 juillet
samedi 6 juillet
dimanche 7 juillet
lundi 8 juillet
mardi 9 juillet
mercredi 10 juillet
jeudi 11 juillet
vendredi 12 juillet
samedi 13 juillet
dimanche 14 juillet
lundi 15 juillet
mardi 16 juillet
mercredi 17 juillet
jeudi 18 juillet
vendredi 19 juillet
samedi 20 juillet
dimanche 21 juillet
lundi 22 juillet
mardi 23 juillet
mercredi 24 juillet
jeudi 25 juillet
vendredi 26 juillet
samedi 27 juillet
dimanche 28 juillet
```

```
In [ ]: lundi 1 mai
        mardi 2 mai
```

2.5 Fonctions

2.5.1 Définition

Lorsqu'on a besoin d'utiliser plusieurs fois un même bloc d'instructions, on peut l'encapsuler dans une *fonction*. On peut ainsi étendre le langage avec une nouvelle instruction.

Dans la définition d'une fonction, on précise son nom et en général des *paramètres formels* d'entrée et une valeur de retour. En `Python` on définit une fonction ainsi :

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

Par exemple une fonction `maximum` qui prend en *paramètres* deux nombres `x` et `y` et qui retourne leur maximum, s'écrira :

```
def maximum(x, y):
    if x < y:
```

```
        return y
    else:
        return x
```

En fait, l'exécution d'une instruction `return` correspond à une sortie de la fonction, toutes les instructions suivantes ne seront pas exécutées. Ainsi la fonction précédente peut s'écrire plus simplement :

```
def maximum2(x, y):
    if x < y:
        return y
    return x
```

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* :

```
note1 = int(input('Entrez une note'))    #récupération d'un argument
note2 = int(input('Entrez une note'))    #récupération d'un autre argument
maxi = maximum2(note1, note2)            #affectation de la valeur retournée par un appel de fo
```

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 :

```
def icn():
    print("Informatique et Création 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.

2.6 Exercice 3

2.6.1 Question 1

```
In [13]: def imc(m, t):
             """Retourne la classification selon l'IMC :
                 0 => sous-poids
                 1 => normal
                 2 => surpoids
             """
             indice = m / t ** 2
             if indice < 18.5:
                 return 0
             elif indice <= 25:
                 return 1
             else:
                 return 2
```

2.6.2 Question 2

```
In [12]: def max2(a, b):
             """Retourne le maximum de deux nombres"""
             if a > b:
                 return a
             return b
```

```
In [ ]: def max3(a, b, c):
            """Retourne le maximum de trois nombres"""
            return max2(a, max2(b, c))
```

2.7 Exercice 4 : fonctions de tests

```
In [14]: def aumoinsun(a, b, c):
             """Retourne un booléen indiquant si au moins des trois réels
             a,b ou c est positif"""
             return a >= 0 or b >= 0 or c >= 0
```

```
In [ ]: def tous(a, b, c):
            """Retourne un booléen indiquant si tous les réels
            a, b et c sont positifs"""
            return a >= 0 and b >= 0 and c >= 0
```

```
In [ ]: def croissant(a, b, c):
            """Retourne un booléen indiquant si a,b et c
            sont dans l'ordre croissant"""
            return a <= b and b <= c
```

```
In [ ]: def croissant2(a, b, c):
            """Retourne un booléen indiquant si a,b et c
            sont dans l'ordre croissant.
            Python permet de chainer les opérateurs de comparaison"""
            return a <= b <= c
```

```
In [16]: def bissextile(a):
             """Une année est bissextile si elle est divisible par 400
             ou pas divisible par 100 et divisible par 4"""
             return a % 400 == 0 or (a % 100 != 0 and a % 4 == 0)
```

2.8 Entrainement 2

```
In [18]: def nombreRacinesTrinome(a, b, c):
             delta = b ** 2 - 4 * a * c
             if delta > 0:
                 return 2
             elif delta < 0:
                 return 0
             else:
                 return 1
```

2.9 Exercice 5

In [21]:
```
from random import randint

def sommeDe(n):
    """Retourne la somme des résultats obtenus en lançant n
    dés à 6 faces"""
    s = 0
    for k in range(n):
        de = randint(1, 6)
        s = s + de
    return s
```

In [22]:
```
from random import randint

def urne():
    """Retourne le numéro d'une boule choisie dans une urne
    contenant 5 boules de numéro 1, 3 boules de numéro 2,
    et deux boules de numéro3"""
    choix = randint(1, 10)
    if choix <= 5:
        return 1
    elif choix <= 8:
        return 2
    else:
        return 3
```

2.10 Exercice 5

In [22]:
```
from turtle import *

def spirale1(n):
    """Trace une spirale constituée de n carrés déformés"""
    penup()
    goto(0,0)
    pendown()
    c = 5
    for i in range(n):
        for j in range(4):
            forward(c)
            c = c + 10
            left(90)
    exitonclick()

##Permet de capturer l'exception Terminator générée lorsqu'on ferme la fenêtre Turtle
try:
    reset()
except Terminator:
    pass
```

```
spirale1(10)
```

In [28]:
```
from turtle import *

def spirale2(n, m):
    """Trace une spirale constituée de n polygones déformés
    à m cotés"""
    penup()
    goto(0,0)
    pendown()
    c = 5
    for i in range(n):
        for j in range(m):
            forward(c)
            c = c + 10
            left(360/m)
    exitonclick()

##Permet de capturer l'exception Terminator générée lorsqu'on ferme la fenêtre Turtle
try:
    reset()
except Terminator:
    pass

spirale2(4, 6)
```

2.11 Entrainement 3

In [29]:
```
from turtle import *
from math import sin, pi

def spirale3(n, m):
    shape('turtle')
    ecart = 10
    rayon = 20
    cote = 20
    for i in range(n):
        penup()
        goto(0,0)
        setheading(-90)
        forward(rayon)
        pendown()
        setheading(180/m)
        cote = 2 * rayon * sin(pi/m)
        for j in range(m):
            forward(cote)
            left(360/m)
```

```
            rayon = rayon + ecart
        exitonclick()
```

```
In [31]: ##Permet de capturer l'exception Terminator générée lorsqu'on ferme la fenêtre Turtle
         try:
             reset()
         except Terminator:
             pass

         spirale3(4, 6)
```

3 Boucle inconditionnelle

3.0.1 Définition

Généralement, on souhaite répéter un bloc d'instructions tant qu'une condition est réalisée mais on ne sait pas à l'avance combien de répétitions seront nécéssaires.

La situation la plus courante est :

```
Tant Que Condition répéter
    Bloc d'instructions
Fin Tant Que
```

Elle se traduit en `Python` par :

```
# flux parent
while condition:
    # bloc d'instructions indenté
# retour au flux parent
```

Par exemple, pour demander la saisie d'un login jusqu'à une saisie correcte, on peut écrire :

```
login = 'moi'
saisie = input('Login ?')      # initialisation de la condition d'entrée de boucle
while saisie != login:         # test d'entrée de boucle
    saisie = input('Login ?') # mise à jour de la condition de boucle
print('Bienvenue ', login)     # sortie de boucle
```

Évidemment, on peut aboutir à une *boucle infinie* si la condition d'entrée de boucle n'est jamais vérifiée.

On peut prévoir des conditions plus complexes avec des opérateurs `booleens`. Par exemple si on veut limiter à 10 demandes de login, on peut écrire : ~~python login = 'moi' compteur = 1 saisie = input('Login ?') # intialisation de la condition d'entrée de boucle while compteur < 10 and saisie != login: # test d'entrée de boucle saisie = input('Login ?') # mise à jour de la condition de boucle compteur = compteur + 1 # mise à jour du compteur if compteur == 10 and saisie != login: print('10 tentatives de login échouées') else: print('Bienvenue', login) # sortie de boucle~~

Enfin remarquons que toute boucle `for` peut s'écrire avec une boucle `while` :

```
somme = 0
for k in range(100, 201):
    somme = somme + k
print("La somme est ", somme)
```

peut ainsi s'écrire :

```
somme = 0
k = 100
while k < 201:
    somme = somme + k
    k = k + 1
print("La somme est ", somme)
```

3.1 Exercice 7

3.1.1 Question 1

Programme équivalent au programme 1 avec une boucle `while`: ~~python for k in range(4): print(k)~~

```
In [33]: k = 0
         while k < 4:
             print(k)
             k = k + 1
```

```
0
1
2
3
```

3.1.2 Question 2

```
In [36]: def tempsvol(n):
             """Retourne le rang du premier terme de la suite de Syracuse
             égal à 1 si le premier terme est n"""
             syracuse = n
             k = 0
             while syracuse != 1:
                 if syracuse % 2 == 0:
                     syracuse = syracuse // 2
                 else:
                     syracuse = 3 * syracuse + 1
                 k = k + 1
             return k
```

```
In [37]: tempsvol(734)
```

```
Out[37]: 46
```

3.2 Exercice 8 PGCD

```
In [3]: def euclide(a, b):
            """Retourne le PGCD de a et b par l'algorithme d'Euclide"""
            while b != 0:
                (a, b) = (b, a % b)
            return a
```

```
In [4]: euclide(1050, 315)
```

```
Out[4]: 105
```

3.3 Entrainement 4

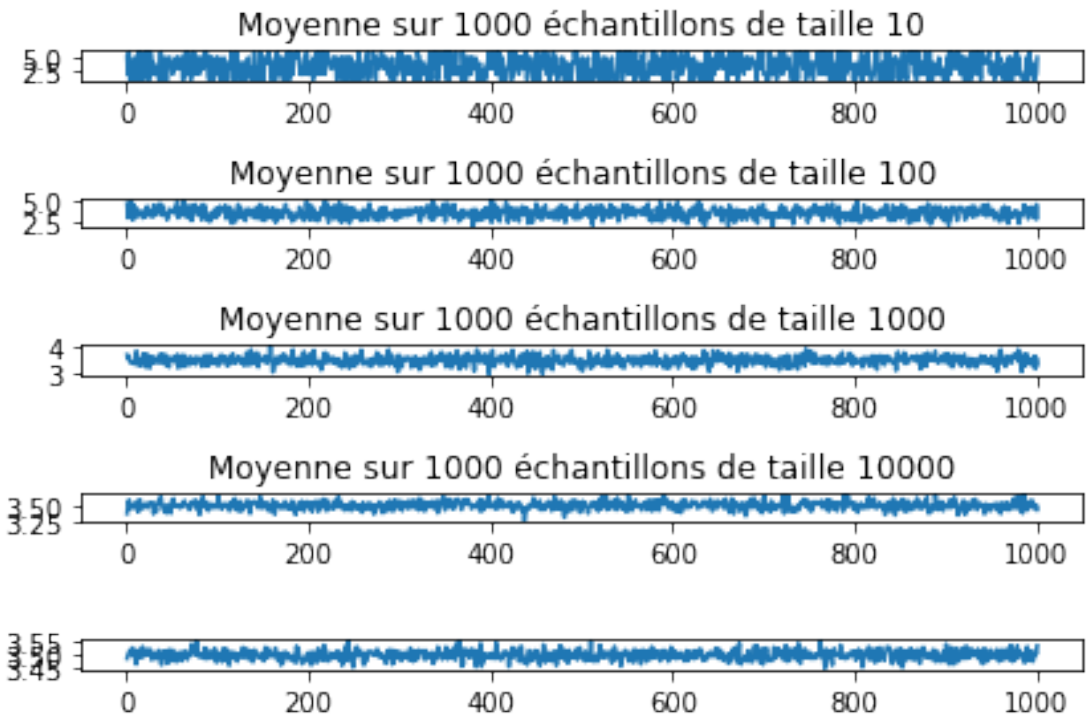
3.3.1 Question 1

```
In [5]: from random import randint

        def moyenneDe(n):
            s = 0
            for k in range(n):
                s = s + randint(1, 6)
            return s / n
```

```
In [6]: # Import de la bibliothèque graphique matplotlib
        import matplotlib.pyplot as plt
```

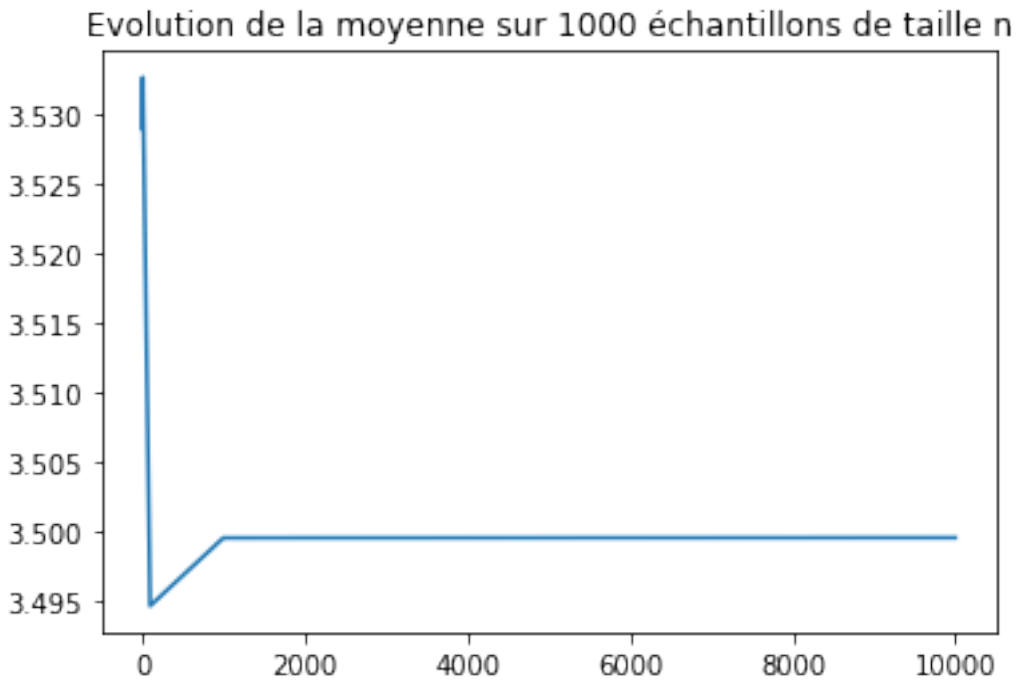
Fluctuation d'échantillonnage

```
In [16]: %matplotlib inline

         for k in range(5):
             n = 10 ** k
             numero_echantillon = [k for k in range(1, 1001)]
             liste_moyenne = [moyenneDe(n) for k in range(1000)]
             plt.title('Moyenne sur 1000 échantillons de taille {}'.format(n))
             plt.subplot(51 * 10 + (k+1))
             plt.plot(numero_echantillon, liste_moyenne)
         plt.tight_layout()
```



Loi faible des grands nombres

```
In [11]: %matplotlib inline

         taille_echantillon = [10 ** k for k in range(5)]
         moyenne_moyenne = [ sum(moyenneDe(n) for k in range(1000))/1000 for n in taille_echant
         plt.title('Evolution de la moyenne sur 1000 échantillons de taille n')
         plt.plot(taille_echantillon, moyenne_moyenne)

Out[11]: [<matplotlib.lines.Line2D at 0x7f473f599198>]
```



3.3.2 Question 2

```
In [18]: def premier6():
             """Retourne le rang du premier 6"""
             k = 1
             while randint(1, 6) != 6:
                 k = k + 1
             return k
```

3.3.3 Question 3

```
In [20]: def tempsAttente(n):
             """Retourne le temps d'attente moyen du premier 6
             sur un échantillon de n lancers"""
             t = 0
             for k in range(n):
                 t = t + premier6()
             return t / n
```