LINGE1225 : Programmation en économie et gestion

Cours 3

Boucles simples et multiples

François Fouss & Marco Saerens

Année académique 2020-2021

Livre de référence

- Chapitre 4 : Instructions répétitives

2

Plan

1. Instructions répétitives
2. Boucle while
3. Boucle for
4. Application financière

3

Instructions répétitives

Déclaration répétitive

- Les déclarations répétitives nous permettent d'exécuter une déclaration de nombreuses fois
- Elles sont souvent référées à des *boucles*.
- Comme les déclarations conditionnelles, elles sont contrôlées par les expressions booléennes.
- Python a deux types de déclaration répétitives :

  `while` loop
  `for` loop

- C'est le choix du programmeur d'utiliser le bon type de boucle pour la situation.

5

Boucle `While`

Boucle while

- La déclaration `while` possède la syntaxe suivante :

while est un mot réservé

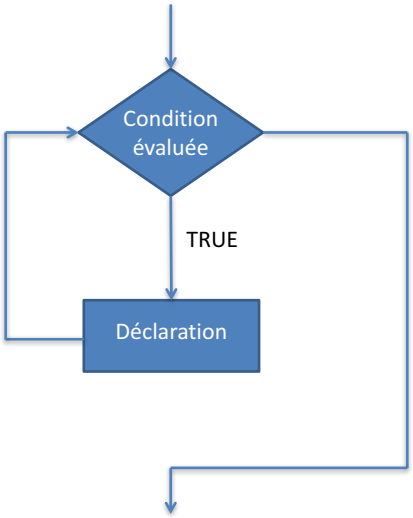
```
while condition:
    #
    # Faire quelque chose
    #
#
# La suite
#
```

- Si la condition est vraie, la déclaration est exécutée. Ensuite la condition est réévaluée.

- La déclaration est évaluée de manière répétitive jusqu'à ce que la condition devienne fausse

7

Logique d'un while



Exemple :

```
1  x = 1
2  while x < 5 :
3      x += 1
4      print (x)
5  print(x)
```

Ce qu'affiche le programme lors de son exécution :

```
2
3
4
5
5
```

8

Logique d'un while

- Remarquez que lorsque la condition est fausse initialement, la déclaration n'est jamais exécutée
- C'est pourquoi, le corps de la boucle `while` s'exécutera zéro ou plusieurs fois

Exemple :

```
1  x = 1
2  while x < 0 :
3      x += 1
4      print (x)
5  print(x)
```

Résultat du programme

```
1
```

- La condition de la boucle while est fausse directement donc, on n'exécute pas les instructions qui sont dans la boucle

9

Boucle infinie

- Le corps d'une boucle `while` peut éventuellement rendre la condition fausse
- Si pas, c'est une **boucle infinie** qui s'exécutera jusqu'à ce que l'utilisateur interrompe le programme
- Il y a une erreur logique courante
- Il faut toujours s'assurer que votre boucle se terminera normalement

Exemple :

```
1  x = 1
2  while x != 2 :
3      x += 2
4      print (x)
5  print(x)
```

Résultat :

```
3
5
7
9
11
13
15
17
19
21
23
...
```

x ne sera jamais égale à 2, donc le programme va rester dans la boucle while indéfiniment et ne jamais se terminer …

10

Boucle while: Indentation

- /!\ Les blocs sont définis par **l'indentation**

- Erreur de syntaxe :

#while

Boucle infinie

```
while n > 0:
↔ sum = sum + n*n
```

Erreur d'indentation

```
while n > 0:
↔ sum = sum + n*n
n = n-1
```

Correct !

```
while n > 0:
↔ sum = sum + n*n
↔ n = n-1
```

11

Boucle `for`

Boucle for

- La boucle `for` possède la syntaxe suivante :

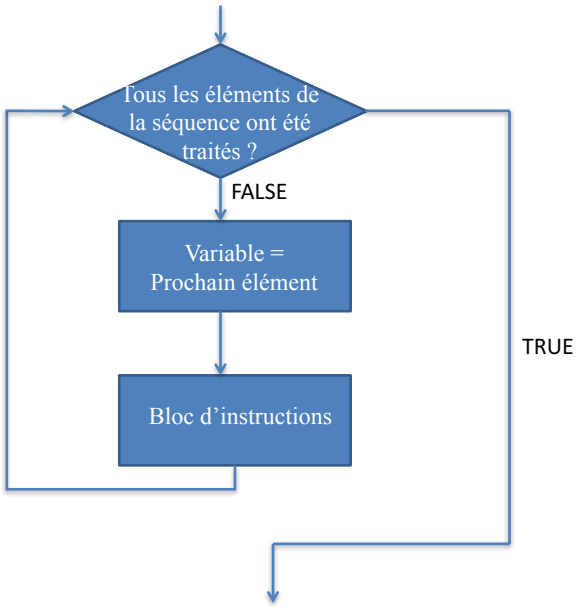
Liste, chaîne de caractère, ..

for est un mot réservé

```
for variable in séquence:
    #
    #   Déclaration
    #
#
# La suite
#
```

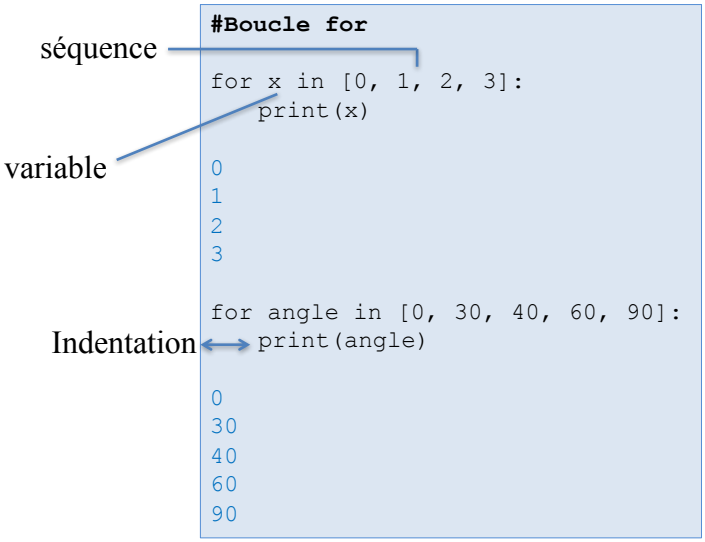
13

Boucle for



14

Boucle for exemple



15

Boucle for exemple avec range()

- range () = fonction prédéfinie qui génère par défaut une séquence de nombres entiers de valeurs croissantes et différent d'une unité.

```
#Boucle for avec range()

for x in range(1,5):
    print(x)

1
2
3
4
```

génère une séquence de nombres entiers de 1 à 4

16

Boucle for

- Comme une boucle `while`, la condition d'un `for` est testée avant d'exécuter le corps de la boucle
- Par conséquent, le corps d'une boucle for s'exécutera zéro fois ou plusieurs fois.
- Les boucles `for` sont adaptées pour l'exécution d'une boucle pour un nombre précis de fois qui peut être déterminé à l'avance

17

Généralités

Choisir une structure de boucle

- Quand ce n'est pas possible de déterminer à l'avance le nombre d'exécution d'une boucle, l'utilisation de la boucle `while` est la plus appropriée.

- S'il est possible de déterminer à l'avance combien de fois le corps de la boucle doit être exécuté, l'utilisation du `for` sera plus appropriée

- Cependant, `while` et `for` sont deux méthodes de boucles qui sont équivalentes

Exemple :

```
1  for i in range (1, 5) :
2      print (i)
3
4  i = 1
5  while i < 5 :
6      print (i)
7      i += 1
```

Ces 2 boucles donnent le même résultat :

```
1
2
3
4
```

19

Imbrication

- Les boucles peuvent être **imbriquées**

Boucle principale

Boucle imbriquée

```
#Boucles while imbriquées

a=1
while a <= 3:
    b=1
    while b <= 3:
        print("x")
        b=b+1
    print()
    a=a+1
```

- La 1ère répétition de la boucle principale est lancée
  - La boucle imbriquée est lancée et s'exécute totalement avant le passage à la répétition suivante de la boucle principale
- La 2ème répétition de la boucle principale est lancée
  - La boucle imbriquée est lancée et s'exécute totalement avant le passage à la répétition suivante de la boucle principale
- etc. (en respectant le nombre de répétitions des 2 boucles)

Résultat du programme :

```
x
x
x

x
x
x

x
x
x
```

20

L'instruction break

- Permet de casser l'exécution d'une boucle
- Elle fait sortir de la boucle et passer à l'instruction suivante

#Boucle avec break: Code

```
for x in range(5):
    print(« Start iteration », x)
    print(« Python »)
    if x == 2:
        break
    print(« End iteration », x)
print(« After iteration »)
```

#Boucle avec break: Exécution

```
Start iteration 0
Python
End iteration 0
Start iteration 1
Python
End iteration 1
Start iteration 2
Python
After iteration
```

21

L'instruction continue

- Permet de passer prématurément au tour de boucle suivant
- Elle fait continuer l'exécution du code sur la prochaine itération de la boucle)

#Boucle avec continue: Code

```
for x in range(4):
    print(« Start iteration », x)
    print(« Python »)
    if x < 2:
        continue
    print(« End iteration », x)
print(« After iteration »)
```

#Boucle avec continue: Exécution

```
Start iteration 0
Python
Start iteration 1
Python
Start iteration 2
Python
End iteration 2
Start iteration 3
Python
End iteration 3
After iteration
```

22

Application financière

Application 3 : intérêts composés

Rappel

- Les **intérêts composés** sont calculés sur le capital ainsi que sur les intérêts cumulés antérieurement

exemple : un placement rémunère 2% d'intérêts simples. La première année il sera égal à celui de l'intérêt simple la deuxième année il sera égale à 1020€ + 2%

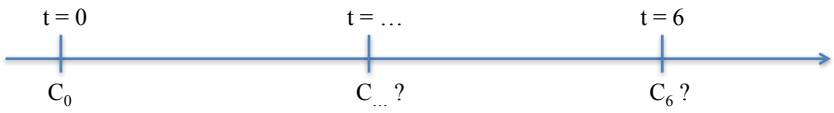
Mathématiquement :

t = 1 $C_1=C_0*(1+i)$
t = 2 $C_2=C_0*(1+i)*(1+i)$
....
t = n $C_n=C_0*(1+i)*(1+i)*\ldots*(1+i)$

24

Application 3 : intérêts composés

Calculez à l'aide d'une boucle `for` le capital acquis sous la théorie des intérêts composés pour 6 années consécutives sachant que le capital de départ et le taux sont respectivement égales à 5000€ et 0.04.



25

Solution

```
#Application financière

c = 5000
taux = 0.04
for i in range(1,7):
    print(c*(1+taux)**i)
```

26