#4 Introduction au langage Python

Les structures itératives (itérations ou boucles)

Édition 2021-2022

Plan

L'idée

- Dans un programme, il est souvent nécessaire d'exécuter un groupe d'actions (action composée) quand une variable dont dépendent les actions du groupe évolue
- L'« itération » (ou « boucle », ou « répétition ») répond à ce besoin

Définitions

- Une boucle est une structure de contrôle qui permet la répétition d'actions un certain nombre de fois sous l'effet d'une variable, appelée « indice », qui évolue de manière contrôlée
- Le nombre de répétitions peut ne pas être connu
  - C'est le cas de la boucle « tant que » ou « `while` » qui utilise une condition ;
  - Suivant sa valeur de vérité, la condition permet les itérations dans la boucle ou les achève. C'est ce qui conduit aussi à l'appeler « boucle non bornée ».
- Le nombre de répétitions est clairement défini
  - Cas de la la boucle « pour » ou « `for` » pour laquelle un groupe d'indices est fourni
  - Ce groupe d'indices peut être une plage de valeurs numériques contiguës ou des valeurs prises dans un ensemble. C'est ce qui conduit aussi à l'appeler « boucle bornée ».

Ces définitions sont directement issues de l'algorithmique (Cf. cours correspondant)

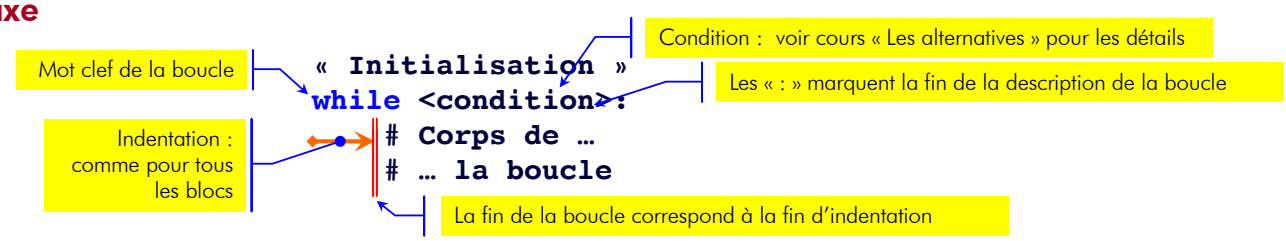
- Les boucles « pour » et « tant que » sont reprises au sein de Python ;
- La boucle « répéter jusqu'à » (`repeat … until`) ne l'est pas.

Structure itérative conditionnelle « `while` »

Définition, mode de fonctionnement

- La boucle « `while` » permet de répéter un bloc d'instructions tant qu'une condition est vraie.
- Pour « sortir » de la boucle, il faut que la condition devienne fausse.
  - Par conséquent, la condition doit utiliser une variable qui change dans la boucle
- Pour évaluer la condition la première fois, les variables dont elle dépend doivent être connues : il faut donc définir ces variables au préalable. Cette action se nomme « initialiser la boucle ».
- Il faut prendre garde aux boucles perpétuelles pour lesquelles Python ne s'arrête pas.

Syntaxe



Exemple : table de multiplication par 7

```
>>> i = 1 # initialiser indice de boucle
>>> while i <= 10:
...     print(i, '* 7 =', i*7, '\t')
...     i += 1
...
1 * 7 = 7    2 * 7 = 14 ... 10 * 7 = 70
```

Initialiser la boucle en fixant l'indice i de départ
Condition vraie, on effectue…
… l'affichage avec print() et …
… l'incrémentation de l'indice de boucle
Les résultats apparaissent…
En les espaçant d'une tabulation à chaque itération.

Quelques informations, exemples et astuces avec les boucles « `while` »

Initialiser est impératif ! Sinon c'est l'erreur assurée

```
>>> while i<4: # i n'est pas connue
...     print('Et de', i, end='\t\t')
...     i += 1
...
Traceback ... <module>
   while i < 5:
NameError: name 'i' is not defined
```

```
>>> i = 0
... while i < 4: # i existe
...     print('Et de', i, end='\t\t')
...     i += 1
...
Et de 0    Et de 1    Et de 2    Et de 3
```

Boucle infinie d'attente

- La boucle qui suit ne s'arrête jamais :

```
>>> while True:
...     print('Encore et encore !',end='\t')
Encore et encore !    Encore et encore !    Encore et encore !    Encore et encore ! (etc.)
```

- Pour interrompre, dans l'EDI, il faut presser Ctrl & C. (ou cliquer le bouton d'arrêt ■)

La variété des conditions

- Tout ce qui a une valeur de vérité, vrai (`True`) ou faux (`False`)
- Des combinaisons diverses sont possibles : elles sont exposées dans le cours 3 « alternatives »

Recherche de logarithmes entiers de nombres entiers (travail personnel)

- **Analyser ce script et annoter ce qui est réalisé**

```
# Script de recherche du logarithme entier
nombreDepart = 1256 # à modifier suivant les besoins
                    # ou faire un "input(int(…))"
nombre = nombreDepart
Logarithme = 0
while nombre > 1: # si nombre <= 1, arret
     nombre = nombre//2
     logarithme = logarithme + 1
print('Le logarithme entier de', nombreDepart, 'est',
logarithme)
```

Annotations personnelles

Bases du langage Python

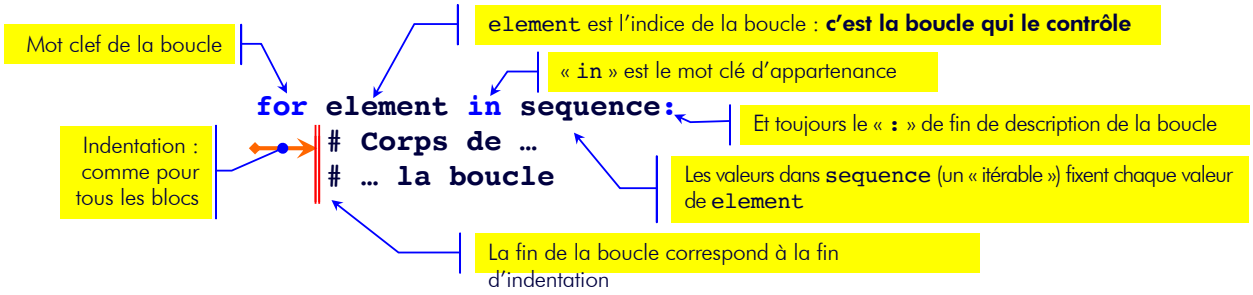
Structure itérative inconditionnelle « for »

Structure itérative inconditionnelle « `for` »

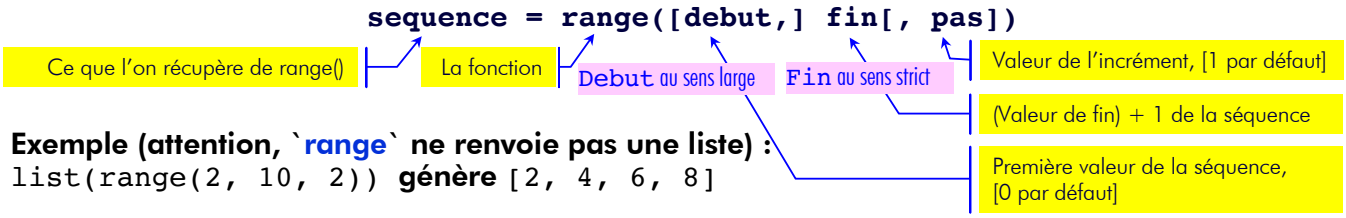
Définition, mode de fonctionnement

- La boucle « `for` » permet de répéter un bloc d'instructions d'après une séquence de valeurs.
- Ces variables de « séquence » (ou d'« itérables ») sont des suites d'entiers, des chaînes, des n-uplets (tuples) ou des listes, voire des tableaux NumPy, etc. (ces derniers seront vus plus tard).
- À la différence de la boucle « `while` » la boucle « `for` » s'arrête toujours.

Syntaxe



Une séquence très répandue est fournie par la fonction « `range()` » à trois arguments :



Exemple (attention, `range` ne renvoie pas une liste) :
`list(range(2, 10, 2))` **génère** `[2, 4, 6, 8]`

Quelques exemples (et astuces) pour la boucle bornée « `for` »

Calcul de la somme des entiers de 1 à 100

```
>>>s = 0
>>>for i in range(1, 101):
...     S = s+i
...
>>>s
5050
>>>s == 100*101/2
True
```

Et avec une chaîne ?

```
>>>str = "C'est cool l'info"
>>>for c in str:
...     print(c, end='-')
...
C-'-e-s-t- -c-o-o-l- -l-'-i-n-f-o-
```

Listes définies en extension et en compréhension

```
>>>liste1 = [1, 2, 3, 4, 5]
>>>liste2 = [i for i in range(1,6)]
>>>liste1 == liste2
True
```

Les ruptures dans les boucles : `break`, `continue` et `pass`

On parle de rupture quand le processus normal de sortie d'une boucle est perturbé

L'instruction « `break` » permet d'arrêter le déroulement d'une boucle

```
>>>while 1: # 1 est toujours vrai donc la boucle est infinie
...     Lettre = input("Taper 'Q' pour quitter.")
...     if lettre == 'Q':
...         print("Fin de la boucle.")
...         break
...
taper 'Q' pour quitter. Q
Fin de la boucle
```

L'instruction « `continue` » reprend une boucle à son début

- L'instruction « `pass` » ne fait rien : c'est l'instruction « `NOP` » des microprocesseurs.

```
>>>for i in range(-2, 2):
...     if i == 1 or i == -1:
...         continue
...     else:
...         pass
...     print(i)
...
-2
0
```

Pour garder le contrôle de ses scripts, éviter autant que possible ces instructions

Les étendues de valeurs au travers de quelques exemples

La séquence qui suit le `in` peut prendre différentes formes

- L' étendue au travers de la fonction `range` a été décrite un peu avant (D9)

Mais il y a aussi :

- Avec une liste d'entiers

```
for val in [1, 4, 12]:
    # Corps de la boucle

lst = [1, 4, 12]
for index in lst:
    # Corps de la boucle
```

- Un n-uplet (tuple)

```
for val in (2, 4, 6, 8):
    # Corps de la boucle

quartet = (2, 4, 6, 8)
for index in quartet:
    # Corps de la boucle
```

- Mais aussi avec des flottants

```
import math as m
for val in (.01, .1, 1, 10):
    print(m.log10(val))

vals = (.01, .1, 1, 10)
for v in vals:
    print(m.log10(v), end=' ')
```

Renvoie `-2.0 / -1.0 / 0.0 / 1.0 /`

- Objets hétérogènes (`list`/`tuple`)

```
str0 = 'enfant'
a = 10
lstr = ['Un', str0, 'de', a, 'ans.']
for e in lstr:
     print(e, end=' ')
```

Renvoie : `Un enfant de 10 ans.`

Références

Sur le net et ailleurs (si si, c'est possible…)

Références

Livres

- **Apprenez à programmer en Python, Vincent Le Goff, Openclassrooms, 2ème édition, 2014**

Wikipédia

- https://fr.wikipedia.org/wiki/Python_(langage)

Cours et références en ligne

- https://www.python.org
- https://www.developpez.com et https://python.developpez.com

Autres cours et références

- **Documents de cours d'informatique de Sophie Dion, lycée Châtelet, Douai**
- **Cours d'informatique de Xavier PESSOLES et Damien ICETA, UPSTI**

Fin de « Introduction au langage Python

4 – Les itérations »

Avez-vous des (d'autres) questions ?