Cours_boucle_for

January 12, 2021

[1]:
```
#!pip install nbtutor
#!jupyter nbextension install --overwrite --py nbtutor
#!jupyter nbextension enable nbtutor --py
%load_ext nbtutor
```

1 Notion de séquence

Une séquence est une **suite d'objets**

1.1 Une séquence ordonnée de caractères : le type `str`

Tout objet de type `str` peut être considéré comme une suite ordonnée de caractères

1.1.1 Exemple

- `"python"` est la suite des caractères `'p'`, `'y'`, `'t'`, `'h'`, `'o'`, `'n'`

1.2 Une séquence ordonnée d'entiers : la fonction `range`

(Voir documentation officielle pour plus d'informations)

`range` permet de créer une **séquence ordonnée d'entier** . Les paramètres de la fonction `range` sont toujours des entiers.

- Pour créer une séquence ordonnée d'entier de 0 **inclus** à x **exclu** :

  `range(x)`

- Pour créer une séquence ordonnée d'entier de a **inclus** à b **exclu** :

  `range(a,b)`

- Pour créer une séquence ordonnée d'entier de a **inclus** à b **exclu** par pas de p :

  `range(a,b,p)`

1.2.1 Exemple

- `range(10)` est donc la suite `0, 1, 2, 3, 4, 5, 6, 7, 8, 9`
- `range(5,10)` est donc la suite `5, 6, 7, 8, 9`
- `range(5,10,2)` est donc la suite `5, 7, 9`

Remarque 1 : Si on tappe `range(10)` dans l'interpréteur python, il n'affiche pas la suite `0, 1, 2, 3, 4, 5, 6, 7, 8, 9`. Ceci est dû au fait que `range` est un itérateur (notion avancée de programmation). En niveau pré-bac, rappelez-vous que c'est équivalent à la suite `0, 1, 2, 3, 4, 5, 6, 7, 8, 9`...

Remarque 2 : Les **listes** et les **dictionnaires** (qui sont d'autres types d'objets en python que l'on verra plus tard) sont aussi des séquences

2 Qu'est-ce qu'une boucle ?

Une boucle consiste à répéter l'exécution d'un bloc d'instructions "un certain nombre" de fois

2.1 Un peu de vocabulaire

- **Incrémentation** : C'est **ajouter 1 à une variable**. C'est une action tellement courante en programmation qu'il existe un opérateur particulier pour ça. En python `+=`

[2]:
```
i = 5
print ("la variable est égale à", i)
print("\nIncrémentation de la variable\n")
i = i + 1
print ("la variable est égale à", i)
```

```
la variable est égale à 5

Incrémentation de la variable

la variable est égale à 6
```

[3]:
```
# Utilisation de l'opérateur +=
i = 5
print ("la variable est égale à", i)
print("\nIncrémentation de la variable\n")
i += 1
print ("la variable est égale à", i)
```

```
la variable est égale à 5

Incrémentation de la variable

la variable est égale à 6
```

- **Décrémentation**

[4]:
```
i = 5
print ("la variable est égale à", i)
print("\nDécrémentation de la variable\n")
i = i - 1
print ("la variable est égale à", i)
```

```
la variable est égale à 5

Décrémentation de la variable

la variable est égale à 4
```

[5]:
```
# Utilisation de l'opérateur -=
i = 5
print ("la variable est égale à", i)
print("\nDécrémentation de la variable\n")
i -= 1
print ("la variable est égale à", i)
```

```
la variable est égale à 5

Décrémentation de la variable

la variable est égale à 4
```

- **Itération**

La notion d'itération est une notion avancée de Python (*objet itérateur*). A votre niveau, retenez juste : * **Itérer** veut dire parcourir une boucle * **Faire une itération** veut dire faire un tour de boucle

3 La boucle `for`

Rôle de la boucle `for` : * faire parcourir une séquence à une variable appelée "variable de boucle". * exécuter un bloc d'instructions à chaque fois que la variable de boucle "avance dans la séquence"

La syntaxe de la boucle `for` est la suivante :

[ ]:
```
du code
du code      # Ce code ne fait pas parti de la boucle : il est exécuté␣
 ↪normalement
du code

for variable_de_boucle in sequence :
    du code
```

```
    du code     # bloc d'instructions de la boucle (délimité par l'indentation)␣
 ↪: ici 3 lignes de code
    du code

du code
du code      # Ce code ne fait pas parti de la boucle : il sera exécuté␣
 ↪normalement après la sortie de la boucle
du code
```

La boucle `for` réalise **automatiquement** ces actions dans cet ordre :

1. **"variable_de_boucle"** va prendre la **première** valeur de la **sequence**
2. Exécution du bloc d'instruction
3. **"variable_de_boucle"** va prendre la valeur **suivante** de la **sequence**
4. Et ainsi de suite...

$\Longrightarrow$ Lorsque **"variable_de_boucle"** a fini avec la **dernière** valeur de la séquence, python sort de la boucle et continue "normalement" la suite du programme

3.1 Exemples

- la variable de boucle `lettre` parcourt la séquence `'b'`, `'o'`, `'n'`, `'j'`, `'o'`, `'u'`, `'r'`

[7]:
```
%%nbtutor -r -f
print("ici, on n'est pas encore entré dans la boucle\n")

for lettre in "bonjour":
    print("On affiche la lettre",lettre)

print("\nici, on est sorti de la boucle")
```

```
ici, on n'est pas encore entré dans la boucle

On affiche la lettre b
On affiche la lettre o
On affiche la lettre n
On affiche la lettre j
On affiche la lettre o
On affiche la lettre u
On affiche la lettre r

ici, on est sorti de la boucle
```

- la variable de boucle `i` parcourt la séquence `0,1,2,3,4,5,6`

[8]:
```
%%nbtutor -r -f
for i in range(7):
    print(i)
```

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

- la variable de boucle `i` parcourt la séquence `5,6,7,8,9`

[9]:
```
%%nbtutor -r -f
for i in range(5,10):
    print(i)
```

```
5
6
7
8
9
```

- la variable de boucle `i` parcourt la séquence `1,3,5,7,9`

[10]:
```
%%nbtutor -r -f
for i in range(1,10,2):
    print(i)
```

```
1
3
5
7
9
```

3.2 La variable de boucle

Comme son nom l'indique la **"variable de boucle"** est une variable. En effet, elle **varie** car elle prend successivement toutes les valeurs de la séquence à chaque tour de boucle

- Une erreur fréquente quand on débute est d'écrire quelque chose comme ça (Bien sûr cela ne peut pas fonctionner car `1` et `"b"` ne sont pas des variables) :

[11]:
```
%%nbtutor -r -f
for 1 in range(7):
    print(i)
```

```
  File "<ipython-input-11-24384507e9df>", line 1
    for 1 in range(7):
        ^
SyntaxError: can't assign to literal
```

[12]:
```
%%nbtutor -r -f
for "b" in "bonjour":
    print("On affiche la lettre",lettre)
```

```
  File "<ipython-input-12-9ce0d2d78403>", line 1
    for "b" in "bonjour":
        ^
SyntaxError: can't assign to literal
```

- Un autre problème fréquent est que les débutants ne savent pas quoi mettre comme variable de boucle.

$\Longrightarrow$ comme c'est une variable, il suffit de lui choisir un nom : le choix du nom est entièrement libre, donc choisissez !!

Les bonnes pratiques veulent qu'on choisisse un nom qui traduit ce que représente la variable de boucle. Dans les exemples précédents : * `lettre` est un nom bien choisi car cette variable de boucle va parcourir successivement chaque **lettre** du mot `"bonjour"` (mais on aurait très bien pu écrire `machintruc` à la place de `lettre`) * On choisit souvent `i` ou `j` comme nom de variable de boucle lorsque celle-ci parcourt des entiers (mais là encore rien d'obligatoire !)

3.3 Parcourir la séquence VS parcourir les indices de la séquence

3.3.1 Rappel

Une chaîne de caractère est une **séquence ORDONNEE de caractères**. Ces numéros s'appellent des **INDICES**

Exemple pour la chaîne de caractères `"python"`:

p	y	t	h	o	n
0	1	2	3	4	5

Dans la chaine de caractères `'python'`, le caractère `'p'` est d'indice 0 et le caractère `'h'` est d'indice 3

Plus d'information sans le cours `bloc1/representation_du_texte/Cours_Le_Type_chaine_de_caracteres`

[13]:
```
#Accès à un caractère
ma_chaine="python"
ma_chaine[2]
```

[13]: 't'

Comparons les 2 programmes suivants : * le premier parcourt la séquence composée des caractères 'p', 'y', 't', 'h', 'o', 'n' * le deuxième parcourt la séquence composée des indices 0, 1, 2, 3, 4, 5

[14]:
```
%%nbtutor -r -f
mot = "python"

for lettre in mot :
    print(lettre)
```

```
p
y
t
h
o
n
```

[15]:
```
%%nbtutor -r -f
mot = "python"

for i in range(len(mot)) :
    print (mot[i],"est la lettre n°",i + 1,"du mot",mot)
```

```
p est la lettre n° 1 du mot python
y est la lettre n° 2 du mot python
t est la lettre n° 3 du mot python
h est la lettre n° 4 du mot python
o est la lettre n° 5 du mot python
n est la lettre n° 6 du mot python
```

Arrêtons-nous sur la syntaxe `range(len(mot))` qui est très courante et qui pose parfois problèmes car on a 2 appels de fonctions imbriqués l'une dans l'autre. Quand cela se présente, il faut commencer par regarder l'appel "le plus à l'intérieur" : 1. `len(mot)` = `len("python")` = 6 2. donc `range(len(mot))` = `range(6)` = 0, 1, 2, 3, 4, 5 $\Longrightarrow$ Il s'agit bien de la séquence composée des indices de la chaîne de caractères "python"

Remarque : Une bonne façon d'assimiler cela est d'utiliser Thonny en mode debug : la décomposition en 2 étapes comme ci-dessus est parfaitement visible !

Attention : Prenez le temps de bien regarder cette syntaxe car il arrive souvent que des élèves écrivent quelquechose comme ci dessous et ne comprennent pas leur erreur.

[16]:
```
for i in len(mot) :
    print(i)
```

```
␣
↪----------------------------------------------------------------------
```

```
       TypeError                                 Traceback (most recent call␣
↪last)

       <ipython-input-16-23aa73552899> in <module>
    ----> 1 for i in len(mot) :
          2     print(i)


       TypeError: 'int' object is not iterable
```

[17]:
```
for i in range(mot) :
    print(i)
```

```
      ␣
↪---------------------------------------------------------------------------

       TypeError                                 Traceback (most recent call␣
↪last)

       <ipython-input-17-2b54c3a026d5> in <module>
    ----> 1 for i in range(mot) :
          2     print(i)


       TypeError: 'str' object cannot be interpreted as an integer
```

3.3.2 Parcourir la séquence VS parcourir les indices de la séquence : comment choisir ?

$\Longrightarrow$ Tout dépend du programme : * Si on a besoin des indices dans le bloc d'instructions de la boucle $\Longrightarrow$ Parcourir les indices * Sinon Parcourir la séquence (En effet, cela mobilise souvent moins de ressources machine)

3.4 Une boucle `for` dans une fonction

Le `return` peut parfois interrompre la boucle `for`. En effet, `return` correspondant au résultat renvoyé par la fonction, celle-ci "a fini son travail" et python ne continue donc pas la boucle `for` puisque c'est devenu inutile.

Exemple :

[18]:
```
%%nbtutor -r -f

def maFonction(chaine, lettre):
    for i in range(len(chaine)):
        if chaine[i] == lettre:
            return i + 1

numeroLettre = maFonction("python","t")

print("la lettre t est la ", numeroLettre, "ème lettre dans le mot python",␣
 ↪sep='')
```

```
la lettre t est la 3ème lettre dans le mot python
```

3.5 Les boucles `for` imbriquées

Considérer le code suivant et exécuter le en mode pas à pas

[19]:
```
%%nbtutor -r -f

for i in range(3):
    print("boucle externe numero", i)
    for j in range(4):
        print("    boucle interne numero", j)
    print("_____________________________")
```

```
boucle externe numero 0
    boucle interne numero 0
    boucle interne numero 1
    boucle interne numero 2
    boucle interne numero 3
_____________________________
boucle externe numero 1
    boucle interne numero 0
    boucle interne numero 1
    boucle interne numero 2
    boucle interne numero 3
_____________________________
boucle externe numero 2
    boucle interne numero 0
    boucle interne numero 1
    boucle interne numero 2
    boucle interne numero 3
_____________________________
```

- Les indentations permettent de bien distinguer ce qui fait partie de la boucle externe de ce qui fait partie de la boucle interne
- On termine entièrement la boucle interne avant de passer au tour de boucle externe suivant

4 Pour aller plus loin : Instruction `break` et `continue`

On utilise assez peu ces instructions, mais on ne sait jamais... Il est d'ailleurs recommandé de limiter leur usage...

- `break` permet de sortir brutalement d'une boucle
- `continue` permet de passer brutalement au tour de boucle suivant

Exemples :

[20]:
```
%%nbtutor -r -f

for i in range(10):
    print ("Tour de boucle n°", i)
    if i == 5:
        break
print ("Sortie brutale de boucle ")
```

```
Tour de boucle n° 0
Tour de boucle n° 1
Tour de boucle n° 2
Tour de boucle n° 3
Tour de boucle n° 4
Tour de boucle n° 5
Sortie brutale de boucle
```

[21]:
```
%%nbtutor -r -f

for i in range(10):
    if i == 5:
        continue
    print ("Tour de boucle n°", i)
```

```
Tour de boucle n° 0
Tour de boucle n° 1
Tour de boucle n° 2
Tour de boucle n° 3
Tour de boucle n° 4
Tour de boucle n° 6
Tour de boucle n° 7
Tour de boucle n° 8
Tour de boucle n° 9
```

5 Méthodes de travail pour bien assimiler cette notion de base tout le temps utilisé en programmation

Constructions élémentaires (la base à connaître parfaitement pour programmer) : * Séquences, variables, types et affectations * Structures conditionnelles * Fonctions * **Boucles** $\Longrightarrow$ Un langage de programmation qui possède ces constructions élémentaires est dit Turing-complet La notion de boucle pose parfois des problèmes d'assimilation. Pour vous aider à passer ce cap, il convient d'exécuter les cellules en mode pas à pas soit : * directement avec ce notebook ouvert en mode interactif à la maison avec **binder** (voir lien dans la page d'accueil) * avec le mode debug de Thonny (voir `premiere/Outils_et_environnement_informatiques/Thonny`) * en copiant le code dans pythontutor