03-cours_python_boucle_while

January 11, 2020

1 Les boucles *while*

1.1 Généralités

En informatique, les boucles sont des instructions qui permettent de répéter des séries d'opérations un certain nombre de fois (voire même un nombre infini de fois). Les boucles sont donc très utiles pour effectuer des opérations répétitives. En Python, il existe deux types de boucles: ***while*** et ***for***. Au cours de cette séance, nous allons étudier le premier type: les boucles ***while***.

L'instruction *while* s'utilise de la façon suivante :

```
while condition_booleene:
    # Instructions si la condition est vérifiée
    # ...
```

Exemple : l'instruction suivante permet d'afficher les nombres entiers compris entre 0 et 6 inclus :

[1]:
```
compteur = 0
# Ce qui suit l'instruction while est une *condition booléenne*
while compteur <= 6:
    print(compteur)
    compteur += 1  # équivalent à compteur = compteur + 1
print("Fin boucle !")
print(compteur)
```

```
0
1
2
3
4
5
6
Fin boucle !
7
```

1.2 L'instruction *break*

On peut également créer une boucle dont la condition est toujours vraie et sortir "brutalement" avec l'instruction ***break*** :

[2]:
```
compteur = 0
while True:  # Ceci est toujours vrai
    if compteur > 6:
        break  # On sort de la boucle
    compteur += 1
print(compteur)
```

```
7
```

Il n'est pas rare de voir cette instruction *while True* dans des programmes Python. Attention cependant aux boucles infinies, c'est à dire qui font que l'instruction *break* n'est jamais exécutée et donc qu'on ne sort jamais de la boucle…

1.3 Exercices

1.3.1 Exercice : valeurs décroissantes

Demander à l'utilisateur une valeur entière. Afficher la liste des valeurs entières qui suivent dans l'ordre décroissant, tant qu'on n'a pas atteint la valeur 0.

[3]:
```
valeur = int(input("valeur : "))
i = valeur
while i > 0:
    print(i)
    i = i - 1  # ou i-= 1
```

```
valeur : 5
5
4
3
2
1
```

1.3.2 Exercice : affichage d'une suite de nombres

Ecrire un programme utilisant une boucle *while* de telle sorte qu'on ait les valeurs : `1, 2, 4, 8, 16, 32, 64, 128`. On affichera les nombre les uns en dessous des autres.

[4]:
```
n = 7
i = 0
while i <= n:
    print(2**i)
    i += 1
```

```
1
2
4
8
16
32
```

```
64
128
```

1.3.3 Exercice : que fait le programme suivant ?

```
i = 1
while True:
    if i > 7:
        print("Je sors !")
```

Euh... il fait chaud...

1.3.4 Exercice : concaténation de chaînes de caractères dans une boucle

- Demander à l'utilisateur un nombre
- Afficher dans une chaîne toutes les valeurs de 1 jusqu'à ce nombre

Exemple :

```
n = 4
# Résultat
"1 2 3 4"
```

[5]:
```
# Version 1
n = int(input("n : "))
i = 1
resultat = ""
while i <= n:
    resultat = resultat + " " + str(i)
    i = i+1
print(resultat)
```

```
n : 5
 1 2 3 4 5
```

[6]:
```
# Version 2
n = int(input("n : "))
i = 1
resultat = ""
while i <= n:
    resultat = f"{resultat} {i}"
    i = i+1
print(resultat)
```

```
n : 5
 1 2 3 4 5
```

Que remarque t-on sur l'affichage ?

1.4 Exercice : boucler tant qu'on n'a pas la bonne réponse

- Demander à l'utilisateur de répondre *oui* ou *non*.
- Si la réponse n'est pas correcte afficher un message d'erreur et reposer la question.
- S'il a répondu *oui* ou *non*, afficher *Bravo !*.

[1]:
```
# 1ère version

reponse = input("Répondre par oui ou par non : ")
while reponse != "oui" and reponse != "non":
    print("Reponse incorrecte... merci d'être attentif !")
    reponse = input("Répondre par oui ou par non : ")
print("Bravo !")
```

```
Répondre par oui ou par non : hello
Reponse incorrecte… merci d'être attentif !
Répondre par oui ou par non : oui
Bravo !
```

[ ]:
```
# 2ème version

while True:
    reponse = input("Répondre par oui ou par non : ")
    if reponse == "oui" or reponse == "non":
        break
    else:  # else est superflu ici, pourquoi ?
        print("Reponse incorrecte... merci d'être attentif !")
print("Bravo !")
```

1.4.1 Exercice : boucle while avec condition d'arrêt - deviner un nombre aléatoire :

- Le programme tire un nombre aléatoire
- On demande à l'utilisateur de saisir un nombre
- On lui indique si il est trop petit, trop grand ou s'il a gagné
- Lorsqu'il a gagné on indique à l'utilisateur en combien de fois

[ ]:
```
import random

# On utilise ci-dessous les majuscules pour indiquer que ce sont des constantes
# Ce n'est qu'une convention... (PEP8)
MIN = 1
MAX = 20

nb_saisies = 0
valeur_a_trouver = random.randint(MIN, MAX)
while True:
    valeur_saisie = int(input(f"Saisissez une valeur entre {MIN} et {MAX} "))
    nb_saisies += 1
```

```
    if valeur_saisie < valeur_a_trouver:
        print("Valeur trop petite")
    elif valeur_saisie > valeur_a_trouver:
        print("Valeur trop grande")
    else:
        print("Bravo, c'est gagné !")
        break

print(f"Vous avez deviné en {nb_saisies} fois.")
```

1.4.2 Exercice : tracer un trait avec l'utilisation de la bibliothèque turtle

Tracer un trait de 100 pixels de façon sacadée, de 10 pixels en 10 pixels.

- On utilise le module turtle en écrivant: import turtle (as tu).
- Documentation (non exhaustive) et exemples.
- Pour marquer des pauses entre le traçage des traits, on utilisera la fonction **sleep**(nb_secondes) *du module* **time***.

```
import time
import turtle as tu

turtle1 = tu.Turtle()
while ...:
    ...
tu.done()
```

[1]:
```
# Solution 1

import time
import turtle as tu

my_turtle = tu.Turtle()
my_turtle.goto(0, 0)
position = 0
while position < 100:
    my_turtle.forward(10)
    time.sleep(0.5)
    position += 10
tu.done()
```

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

Terminator                                Traceback (most recent call␣
↪last)

<ipython-input-1-0469a4164872> in <module>
```

```
     11     time.sleep(0.5)
     12     position += 10
---> 13 tu.done()


/usr/lib/python3.7/turtle.py in mainloop()


Terminator:
```

[ ]:
```python
# Solution 2 (plus élégante)

import time
import turtle as tu

my_turtle = tu.Turtle()
my_turtle.goto(0, 0)
while my_turtle.position()[0] < 100:
    my_turtle.forward(10)
    time.sleep(0.5)

tu.done()
```