Structures conditionnelles et itératives

Introduction à la programmation en Python

Sommaire

1. Structures conditionnelles.
2. Structures itératives.

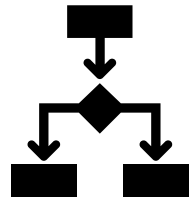
1. Structures conditionnelles.

1. Structures conditionnelles.

Structures conditionnelles : motivation

- Pouvoir réaliser des actions différentes selon la vérification d'une ou plusieurs conditions.

- Cela correspond au SI … ALORS … SINON SI … ALORS … SINON du langage courant ou des mathématiques.



1. Structures conditionnelles.

Syntaxe générale d'un test avec alternatives

```
if condition1:
    bloc d'instructions 1
    ...
elif condition2:
    bloc d'instructions 2
    ...
else:
    bloc d'instructions 3
    ...
```

1. Structures conditionnelles.

Délimitation des blocs d'instructions

1. La ligne les précédant se termine par un double point :

2. L'intégralité du bloc est indentée par rapport aux instructions qui le précèdent et le suivent.

1. Structures conditionnelles.

Exemple 1 : un test simple (*i.e.* sans alternatives) pour calculer une valeur absolue

```
x = eval(input("Saisir une valeur de x :"))
if x<0:
    x = -x
print("La valeur absolue de x est", x)
```

1. Structures conditionnelles.

Exemple 2 : un test avec une alternative pour indiquer la parité d'un entier

```
n = eval(input("Saisir un entier : "))
if n%2 == 0:
    print("Le nombre saisi est pair")
else:
    print("Le nombre saisi est impair")
```

1. Structures conditionnelles.

Exemple 3 : un test avec deux alternatives pour déterminer une réussite au BAC

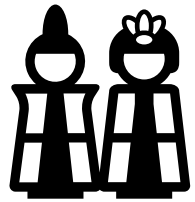
```
note = eval(input("Saisir la moyenne générale : "))
if note < 8:
    print("Lycéen recalé")
elif note < 10:
    print("Lycéen au rattrapage")
else:
    print("Lycéen admis")
```

1. Structures conditionnelles.

Imbrication de tests : principe

- L'imbrication de tests, *i.e.* que l'une des alternatives d'un test contienne un autre test, est possible et souvent fort utile.

- Cela permet dans certains cas le recours à une succession de tests simples et donc rend les codes plus lisibles.



1. Structures conditionnelles.

Imbrication de tests : exemple de résolution d'une équation du premier degré

```
a = eval(input("Saisir la valeur de a : "))
b = eval(input("Saisir la valeur de b : "))
if a == 0:
    if b == 0:
        print("infinité de solutions")
    else:
        print("pas de solution")
else:
    print("unique solution :", -b/a)
```

1. Structures conditionnelles.

Opérateur ternaire : principe

- Permet dans les cas simples d'avoir une écriture synthétique d'un test avec alternative.

```
expression1 if condition else expression2
```

- Les "expressions" ne doivent cependant comporter qu'une seule instruction.

1. Structures conditionnelles.

Opérateur ternaire : exemples

- Affectation d'une variable :

```
statut = "mineur" if age < 18 else "majeur"
```

- Affichage sur la console :

```
print("mineur") if age < 18 else print("majeur")
```

1. Structures conditionnelles.

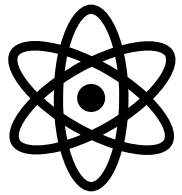
2. Structures itératives.

2. Structures itératives.

Structures itératives : motivation

- Pouvoir exécuter plusieurs fois des blocs d'instructions identiques ou du moins de même nature.

- Selon que le nombre de répétitions soit connu à l'écriture du programme ou pas, on utilisera une structure "for" ou une structure "while".



2. Structures itératives.

La boucle “for“ : principe général

- La boucle “for“ réalise un nombre d’itérations fixe et connu.
- Elle utilise une variable dont la valeur va parcourir une certaine plage au fil des itérations. C’est cette variable qui contrôlera le nombre d’itérations.
- Cette plage de valeurs va être construite grâce à la fonction “range“.

2. Structures itératives.

La fonction “range“ : syntaxe

```
range(begin, end, step)
```

- Produit une plage de valeurs allant de ‘begin’ (inclus) à ‘end’ (non inclus) avec un pas égal à ‘step’.
- Par défaut ‘begin’ et ‘step’ valent respectivement 0 et 1.

2. Structures itératives.

La fonction “range“ : exemple

- range(6) produira la plage de valeurs 0, 1, 2, 3, 4, 5.
- range(3,6) produira la plage de valeurs 3, 4, 5.
- range(3,10,2) produira la plage de valeurs 3, 5, 7, 9.
- range(6,0,-1) produira la plage de valeurs 6, 5, 4, 3, 2, 1.

2. Structures itératives.

La boucle “for“ : syntaxe et fonctionnement

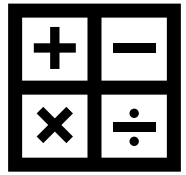
```
for var in range(begin, end, step):
    bloc d'instructions à répéter
    …
```

- Le nombre d’itérations est égal au nombre de valeurs de la plage créée par “range“.
- En pratique les valeurs de la variable ‘var’ seront également utilisées.

2. Structures itératives.

La boucle “for“ : exemple, affichage d’une table de multiplication

```
n = eval(input("Table de : "))
for i in range(1, 11):
    print(i, '*', n, '=', i*n)
```



2. Structures itératives.

La boucle “for“ : détail du fonctionnement de l’exemple précédent

- Admettons que l’utilisateur saisisse la valeur 7.
- À la première itération la variable i vaut 1, il est donc affiché $1 * 7 = 7$.
- À la seconde itération la variable i vaut 2, il est donc affiché $2 * 7 = 14$.
- Etc.

```
Table de : 7
1 * 7 = 7
2 * 7 = 14
3 * 7 = 21
4 * 7 = 28
5 * 7 = 35
6 * 7 = 42
7 * 7 = 49
8 * 7 = 56
9 * 7 = 63
10 * 7 = 70
```

2. Structures itératives.

La boucle “for“ : exemple, affichage des nombres pairs entre 20 et 0

```
for i in range(20, -1, -2):
    print(i)
```

```
20
18
16
14
12
10
8
6
4
2
0
```

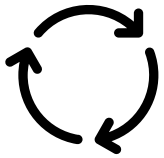
2. Structures itératives.

La boucle “while“ : principe général

- La boucle “while“ réalise un nombre d’itérations non nécessairement connu à l’écriture du programme.

- Ce nombre dépend d’une conditionnelle qui sera évaluée avant chaque itération.

- On la qualifie parfois d’itération conditionnelle.



2. Structures itératives.

La boucle “while“ : syntaxe et fonctionnement

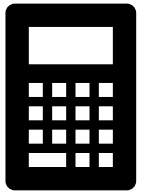
```
while condition:
    bloc d'instructions à répéter
    ...
```

- On exécute le bloc d’instructions tant que la condition est vérifiée.
- Le bloc doit donc nécessairement modifier la valeur de la condition (boucle infinie sinon).

2. Structures itératives.

La boucle “while“ : exemple, calcul de la partie entière d’un réel

```
x = eval(input("Réel à saisir : "))
n = 0
while n+1 <= x:
    n += 1
print("La partie entière de", x, "est", n)
```



2. Structures itératives.

La boucle “while“ : détail du fonctionnement de l’exemple précédent

- Admettons que l’utilisateur saisisse la valeur 2.57.
- Avant les itérations, la variable n est initialisée à 0.

```
Réel à saisir : 2.57
La partie entière de
2.57 est 2
```

- La condition est alors vraie donc on réalise la première itération, et n vaut maintenant 1.
- La condition est de nouveau évaluée est encore vraie donc on réalise la seconde itération, et n vaut maintenant 2.
- Cette fois-ci la condition est fausse et l’on stoppe les itérations.

2. Structures itératives.

La boucle “while“ : exemple à ne pas suivre

- La condition d’entrée dans la boucle est vraie et n’est jamais modifiée, cela conduit à une boucle infinie :

```
i = 2
while i < 3:
    print("Mouvement perpétuel")
```

2. Structures itératives.

Imbrication de structures itératives : principe

- L'imbrication de boucles, *i.e.* qu'une boucle en contienne une autre, est possible et souvent fort utile.

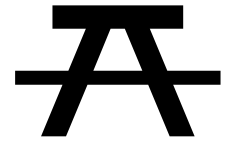
- Et ce indépendamment de leur type, "for" ou "while".

- À noter que l'on peut également imbriquer structures itératives et structures conditionnelles.

2. Structures itératives.

Imbrication de structures itératives : exemple

```
for i in range(1,11):
    print("table de", i)
    for j in range(1,11):
        print("\t", j, "*", i, "=", j*i)
```



2. Structures itératives.

Imbrication de structures itératives : exemple

- À la première itération de la boucle externe la variable i vaut 1, la variable j prend alors toutes les valeurs de 1 à 10 et la première table est affichée.

- À la seconde itération de la boucle externe la variable i vaut 2, la variable j prend alors de nouveau toutes les valeurs de 1 à 10 et la deuxième table est affichée.

- Etc.

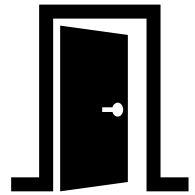
```
table de 1
        1 * 1 = 1
        2 * 1 = 2
        3 * 1 = 3
        4 * 1 = 4
        5 * 1 = 5
        6 * 1 = 6
        7 * 1 = 7
        8 * 1 = 8
        9 * 1 = 9
        10 * 1 = 10
table de 2
        1 * 2 = 2
        2 * 2 = 4
        3 * 2 = 6
        4 * 2 = 8
        5 * 2 = 10
```

2. Structures itératives.

Sorties de boucles : deux possibilités

- La commande “break” permet la sortie définitive de la structure itérative “for” ou “while” qui la contient.

- La commande “continue” permet de passer directement à l’itération suivante dans la structure “for” ou “while” qui la contient.



2. Structures itératives.

Exemple 1 : sortie définitive de boucle avec la commande “break“

```
for i in range(666):
    if input() == 'x':
        print("Au revoir")
        break
```

```
777
Charlie
Watts
x
Au revoir
```

2. Structures itératives.

Exemple 2 : saut d'une itération avec la commande "continue"

```
for i in range(10, 21):
    if i == 13:
        continue
    print(i)
```

```
10
11
12
14
15
16
17
18
19
20
```

2. Structures itératives.