

Python

Actions conditionnelles Actions itératives ou répétitives

Valérie Bellynck (2009), d'après un diaporama d'Alain Bozzi (2008), lui-même venant de celui de Bob Cordeau (2008)



Actions conditionnelles (if)

 Utilisées pour effectuer des tests sur une expression ou le contenu d'une variable.

```
if (condition):
   code exécuté si condition est vraie
instruction suivante (après les conditions)
```





Actions conditionnelles (else)

if (condition):

code exécuté si condition est vraie else:

code exécuté si condition est fausse instruction suivante



Observez l'indentation !!!
Pas de condition pour l'instruction else



Actions conditionnelles (elif)

```
if (condition_1):
   code exécuté si condition_1 est vraie
elif(condition_2):
   code exécuté si
   condition_1 est fausse
   et condition_2 est vraie
else:
   code exécuté si
   condition_1 et condition_2 sont fausses
instruction suivante (après les conditions)
```

Opérateurs de comparaison

La condition évaluée dans une action conditionnelle peut contenir les opérateurs de comparaison suivants :

- x == y # x est égal à y
- x != y # x est différent de y
- x > y # x est plus grand que y
- x < y # x est plus petit que y
- x >= y # x est plus grand que, ou égal à y
- x <= y # x est plus petit que, ou égal à y



Opérateurs de comparaison

- Notez bien que l'opérateur de comparaison pour l'égalité de deux valeurs est constitué de deux signes « égale » et non d'un seul
- Le signe « égale » utilisé seul est un opérateur d'affectation, et non un opérateur de comparaison.
- l'opérateur % est l'opérateur modulo : il calcule le reste d'une division entière.
- Ainsi par exemple, a % 2 fournit le reste de la division de a par 2

Attention à la différence entre = et ==

- = pour une affectation
- == pour une comparaison



Exemple

Que fait ce programme?

```
fumeur = raw input("patient fumeur (oui ou non) ?")
if fumeur == "oui":
 niveau de risque = 3
else:
 niveau de risque = 0
print niveau de risque
```



Actions conditionnelles avec opérateurs logiques (and / or / not)

```
if (condition_1) and (condition_2):
    code exécuté si
    condition_1 et condition_2 sont vraies
else:
    code exécuté si
    condition_1 ou condition_2 est fausse
instruction suivante (après les conditions)
```



Observez l'indentation !!!



Exemple avec ET (and), OU (or) et non (not)

Que fait ce programme?

```
if (fumeur == "oui") and (age > 60):
   print "le patient est une personne âgée qui fume !"

if (fumeur == "oui") or (age > 60):
   print "le patient est une personne âgée ou un fumeur!"

if not(fumeur == "oui"):
   # l'expression fumeur == "oui" à pour valeur false
   # donc not (false) à pour valeur true
   print "le patient est non fumeur!"
```



Que fait ce programme?

Exemple

```
if (fumeur == "oui"):
   facteur f = 2
else:
   facteur f = 0
if (age > 60):
   facteur a = 1
else:
   facteur a = 0
niveau de risque = facteur f + facteur a
if niveau de risque == 0:
   print "Le risque est nul !"
if niveau de risque != 0:
   print "Il y a un risque !"
if niveau de risque >= 3:
   print "Risque élevé!"
```



Actions itératives (= répétitives)

Il s'agit de répéter plusieurs fois la ou les mêmes actions

(cette séquence d'actions est appelée un bloc ou boucle)

- 1. soit on connait le nombre de fois d'avance, par exemple n fois
- 2. soit le nombre de fois correspond exactement au nombre d'éléments d'une liste (ou de lignes lues dans un fichier, ou d'une ressource), et les actions à effectuer sont liées tour à tour à chacun des éléments de la liste
- 3. soit le nombre de fois est inconnu au départ,

et dépend d'une condition calculable à chaque itération

(le nombre de fois peut ne pas être identique d'une éxécution à l'autre : sa valeur peut être modifiée dans la boucle)



Actions itératives (= répétitives)

En Python, il n'y a pas d'instruction for avec directement le nombre d'occurrence. Les instructions permettant de commander et contrôler le nombre d'itération sont

- for lié au parcours d'une liste
- while avec une condition explicite d'arrêt

La mise en place de ces actions nécessite généralement :

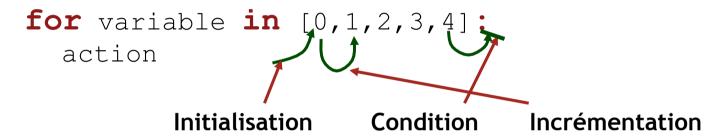
- une initialisation
- une condition
- une incrémentation



Instruction for

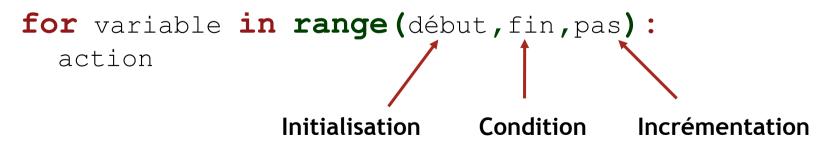
L'instruction for est utilisée pour

parcourir un tableau de valeurs :



La fonction range () est utilisée pour

construire un tableau de valeurs :





Utilisation de for et de range ()

Exemple 3:

```
for i in range(5,55,5):
    # i prend successivement les valeurs 5, 10 ... 55
    print i
```

Exemple 4:

```
for i in range(3, 'a', 3.14) :
    # i prend successivement les valeurs 3, a et 3.14
    print i
```

Exemple 4 bis:

```
for i in [3, 'a', 3.14] :
    # i prend successivement les valeurs 3, a et 3.14
    print i
```



Actions itératives : for

L'action for peut être utilisée de plusieurs manières :

```
# i prend pour contenu de 0 à valeur - 1
# avec une incrémentation de 1
for v in range(valeur):
   action
```

```
# i prend pour contenu de début à fin - 1
# avec une incrémentation de 1
for v in range(debut, fin):
   action
```

```
# i prend pour contenu de début à fin - 1
# avec une incrémentation de pas
for v in range(debut, fin, pas):
   action
```



Actions itératives - exemples

```
for variable in range(début, fin, pas):
    action
```

Exemple 1:

```
for i in range (10):
    # i prend successivement les valeurs 0, 1 ... 9
    print i # affichage de la valeur
```

Exemple 1 bis:

```
print "***** Affichage d'une chaine ******"
ch = raw_input("Saisir une chaine : ")
lg = len(ch) # lg contient le nombre de caractères de la chaîne
for i in range(lg) :
    print ch[i] # affichage du caractère
```



Actions itératives

for variable in range(début, fin, pas):
 action

Exemple 2:

```
print "****** Table de multiplication ******"
mult = raw_input("Saisir la table à afficher : ")
for i in range(1,11) :
    # i prend successivement les valeurs 1, 2 ... 10
    print mult , "x", i,"=", mult * i
    # affiche successivement les multiples : mult * I
```

```
Attention : il y a une faute...

mult = int(raw_input("Saisir la table à afficher : "))
```



Instruction while

L'instruction while est utilisée pour répéter un bloc d'actions tant qu'une condition est vérifiée:

```
initialisation
while condition :
    action
```

- → L'initialisation (ou le bloc d'initialisations) définit des variables et leur affecte des valeurs (appelées valeurs initiales)
- → L'action (ou le bloc d'actions) modifie les valeurs des variables pendant l'exécution du bloc d'instructions, ces valeurs sont appelées les valeurs courantes
- → La condition prend en compte les valeurs courantes des variables

Instruction while

L'instruction while peut-être utilisée de manière plus sûre, répéter un bloc d'actions tant qu'une condition est vérifiée :

```
V = V0
autres_initialisations
while condition_sur_V:
    actions_ne_modifiant_pas_V
    incrément_de_V
```

- → L'initialisation (ou le bloc d'initialisations) définit la variable contrôlée par la boucle en lui affectant une valeur (appelée valeur initiale), et effectue éventuellement des initialisations d'autres variables sur lesquelles ne portent pas la condition
- → La condition ne prend en compte la valeur courante de la variable contrôlée
- → L'action (ou le bloc d'actions) modifie éventuellement les valeurs des variables pendant l'exécution du bloc d'instructions, mais pas celle de la variable contrôlée
- → L'incrément modifie la valeur de la variable contrôle . Les valeurs contenue dans cette variable à chaque boucle sont appelées les valeurs courantes 19



Exemple d'utilisation d'un compteur

Que fait le programme suivant ?

```
i = 0 # initilisation
while (i < 5) : # condition
  print i
  i += 1 # incrémentation ou i = i + 1</pre>
```

La dernière ligne est une écriture abrégée pour l'instruction

$$i = i + 1$$

i est appelé "le compteur" ou l'incrément, 1 est "le pas"



Actions répétitives

```
a = raw input ("Choisir un nombre de 1 à 3 (zéro pour quitter) >>> ")
a = int(a) # transtypage (chaîne vers entier)
while a != 0 : # l'opérateur != signifie "différent de"
 if a == 1:
   print "Vous avez choisi un : "
   print "le premier, l'unique, l'unité ... "
    a = 0 # pour quitter la boucle
  elif a == 2:
   print "Vous préférez le deux : "
   print "la paire, le couple, le duo ... "
    a = 0 # pour quitter la boucle
  elif a == 3:
   print "Vous optez pour le plus grand des trois : "
   print "le trio, la trinité, le triplet ... "
   a = 0 # pour quitter la boucle
  else:
    print "Un nombre entre UN et TROIS, s.v.p. "
  a = raw input ("Choisir un nombre de 1 à 3 >>> ")
  a = int(a) # transtypage (chaîne vers entier)
```