

Bases d'algorithmique et de programmation

Sébastien Gagné, Université d'Orléans

L1 Algo - Prog - S1

Cours 03 - Expressions booléennes - Instructions conditionnelles - Instructions répétitives

Plan du cours

- Expressions booléennes
- Instruction conditionnelle
- Instruction répétitive - For
- Instruction répétitive - While

Expressions booléennes

Une *expression booléenne* est une expression dont l'évaluation produit une valeur de vérité : vrai (*True*) ou faux (*False*).

Elle peut notamment contenir des opérateurs de comparaison ou logiques, des opérations arithmétiques, des nombres, des chaînes de caractères, ...

Quelques exemples

expression booléenne	valeur
<code>4 > 5</code>	
<code>2 + 3 < 7 - 1 and 'a' < 'b'</code>	
<code>len("bonjour !") == 9</code>	
<code>not(False or 1 < 2) or 2 != 1 + 1</code>	
<code>"chat"[: 2] != "chien"[-5 : -3]</code>	

Expressions booléennes

Une *expression booléenne* est une expression dont l'évaluation produit une valeur de vérité : vrai (*True*) ou faux (*False*).

Elle peut notamment contenir des opérateurs de comparaison ou logiques, des opérations arithmétiques, des nombres, des chaînes de caractères, ...

Quelques exemples

expression booléenne	valeur
<code>4 > 5</code>	False
<code>2 + 3 < 7 - 1 and 'a' < 'b'</code>	True
<code>len("bonjour !") == 9</code>	True
<code>not(False or 1 < 2) or 2 != 1 + 1</code>	False
<code>"chat"[: 2] != "chien"[-5 : -3]</code>	False

Instructions conditionnelles

Une instruction conditionnelle est de la forme

si	<expression booléenne>
alors	<suite d'instructions 1>
sinon	<suite d'instructions 2>

Le mécanisme de cette instruction :

- l'expression booléenne est évaluée (à True ou à False) ;
- si le résultat est True, la suite d'instructions 1 est exécutée, mais pas la suite d'instructions 2 ;
- dans le cas contraire, la suite d'instructions 2 est exécutée, mais pas la suite d'instructions 1.

Instructions conditionnelles

Exemples d'instructions conditionnelles en python

```
1 n = int(input('entrez un nombre entier : '))
2
3 if n % 3 == 0 :
4     print('ce nombre est un multiple de 3.')
5 else :
6     print('ce nombre n\'est pas un multiple de 3.')
```

```
1 mot = input("entrez un mot : ")
2 voyelles = ('a', 'e', 'i', 'o', 'u', 'y')
3
4 if mot[0].lower() in voyelles :
5     print("le mot " + mot + " commence par une voyelle.")
6     print("cette voyelle est : " + mot[0])
7 else :
8     print("le mot " + mot + " commence par une consonne.")
9     print("cette consonne est : " + mot[0])
10
11 print("Au revoir !")
```

Instructions conditionnelles

- Le mot-clé **if** précède l'expression booléenne ;
- L'expression booléenne est suivie de " : " ;
- La suite d'instructions exécutée si l'expression est évaluée à True est écrite décalée (indentation) ;
- Le mot-clé **else** suivi de " : " annonce les instructions exécutées si l'expression booléenne est évaluée à False ;
- Cette suite d'instructions est aussi décalée (indentation) ;
- L'instruction conditionnelle se termine par un retour à l'indentation précédente.

Imprécision d'écriture

```
1 n = int(input("entrez un nombre : "))
2
3 if n % 2 == 0 :
4     print(n, "est un nombre pair")
5
6 if n % 2 == 1 :
7     print(n, "est un nombre impair")
```

L'algorithme est à revoir car $n \% 2 == 0$ et $n \% 2 == 1$ sont des cas exactement contraires !

Il est donc inutile d'enchaîner systématiquement les deux tests.

C'est l'intérêt du *else* !

```
1 n = int(input("entrez un nombre : "))
2
3 if n % 2 == 0 :
4     print(n, "est un nombre pair")
5 else :
6     print(n, "est un nombre impair")
```

Variante pratique

On peut dans certains cas utiliser la variante pratique suivante

si	<expression booléenne 1>
alors	<suite d'instructions 1>
sinon si	<expression booléenne 2>
alors	<suite d'instructions 2>
sinon	<suite d'instructions 3>

- Si l'expression 1 est vraie, alors la suite 1 est exécutée.
- Sinon (expression 1 fausse) on évalue l'expression 2, et ...
 - + Si elle est vraie, on exécute la suite 2,
 - + Sinon on exécute la suite 3.

Cela se traduit en python par *if*, *elif* et *else*.

Variante pratique - exemples

```
1 n = int(input('entrez un nombre entier : '))
2
3 if n < 0 :
4     print('ce nombre est négatif.')
5 elif n < 1000 :
6     # dans ce cas, n >= 0 et n < 1000
7     print('ce nombre est un petit nombre positif.')
8 else :
9     # dans ce cas, n >= 0 et n >= 1000
10    print('ce nombre est un grand nombre positif.')
```

```
1 voyelles = "aeiouy"
2 consonnes = "bcdfghjklmnpqrstvwxyz"
3 chiffres = "0123456789"
4
5 mot = input("entrez une chaîne de caractères : ")
6
7 if len(mot) == 0 :
8     print("votre chaîne est vide !")
9 else :
10    initiale = mot[0]
11    if initiale in chiffres :
12        print("votre chaîne commence par un chiffre.")
13    elif initiale.lower() in voyelles :
14        print("votre chaîne commence par une voyelle.")
15    elif initiale.lower() in consonnes :
16        print("votre chaîne commence par une consonne.")
17    else :
18        print("votre chaîne commence par un caractère bizarre !")
```

Instructions répétitives

Une instruction répétitive peut être vue sous une des deux formes suivantes

- Parcours dans un itérable (boucle *pour* - *for*)

```
pour <element> dans <iterable> faire  
    <suite d'instructions 1>
```

- Boucle *tant que* - *while*

```
tant que <expression booléenne> faire  
    <suite d'instructions 1>
```

Instructions répétitives – *for*

- type range

```
for i in range(1, 11) :  
    print(i, "^ 2 =", i*i)
```

- type str

```
mot = "algorithmique"  
for lettre in mot :  
    print(lettre, end = "-")
```

- type tuple

```
t = (1, 'a', [4, 7], True)  
for elt in t :  
    print(elt, "est de type", type(elt))
```

- type list

```
for v in [5, 3, 4] :  
    print(v)
```

Instructions répétitives - *for*

- type set

```
s = {5, 'coucou', False, 9}
for elt in s :
    print(elt)
```

- type dict

```
d = {'x' : 3.1, 'y' : 2.1, 'z' : -1.4}
for item in d.items() :
    print(item)

for k in d.keys() :
    print(k)

for v in d.values() :
    print(v)
```

Instructions répétitives - *while*

Ces boucles de type "tant que" (*while*) sont naturellement utilisées quand on ne sait pas à l'avance combien de fois la séquence d'instructions doit être répétée.

```
1 n = -1
2
3 while n < 0 :
4     n = int(input("entrez un nombre entier positif ou nul : "))
5
6 print("merci, vous avez entré le nombre", n)
```

```
entrez un nombre entier positif ou nul : -8
entrez un nombre entier positif ou nul : -1
entrez un nombre entier positif ou nul : 4
merci, vous avez entré le nombre 4
```

Instructions répétitives - *while*

Le même exemple, plus élaboré

```
1 saisie_correcte = False
2 nbEssais = 0
3
4 while not saisie_correcte :
5     n = int(input("entrez un nombre entier positif ou nul : "))
6     saisie_correcte = (n >= 0)
7     nbEssais += 1
8
9 print("merci, vous avez entré le nombre", n, "en", nbEssais, "essais !")
```

```
entrez un nombre entier positif ou nul : -7
entrez un nombre entier positif ou nul : -6
entrez un nombre entier positif ou nul : 15
merci, vous avez entré le nombre 15 en 3 essais !
```

Instructions répétitives - *while*

Dans l'exemple suivant, la boucle `while` utilise une expression booléenne (`not saisie_correcte`), initialement vraie.

```
1 # initialisations
2 saisie_correcte = False
3 nb_essais = 0
4
5 # instruction répétitive - tant que
6 while not saisie_correcte :
7     reponse = input("entrez une voyelle minuscule et sans accent : ")
8     nb_essais += 1
9
10 # affichage du résultat
11 print("Vous avez respecté la consigne en", nb_essais, "essais.")
```

Dans la boucle, on demande une réponse tant que la saisie n'est pas correcte (`while not saisie_correcte`).

Problème : à chaque nouvelle saisie, la variable booléenne `saisie_correcte` n'est pas recalculée $\Rightarrow$ **boucle infinie**.

Instructions répétitives - *while*

Pour corriger cette erreur, il faut recalculer la variable `saisie_correcte` dans la boucle :

```
1 # initialisations
2 saisie_correcte = False
3 nb_essais = 0
4
5 # instruction répétitive - tant que
6 while not saisie_correcte :
7     reponse = input("entrez une voyelle minuscule et sans accent : ")
8     saisie_correcte = reponse in ('a','e','i','o','u','y')
9     nb_essais += 1
10
11 # affichage du résultat
12 print("Vous avez respecté la consigne en", nb_essais, "essais.")
```

```
entrez une voyelle minuscule et sans accent : t
entrez une voyelle minuscule et sans accent : r
entrez une voyelle minuscule et sans accent : e
Vous avez respecté la consigne en 3 essais.
```

Instructions répétitives - *break*

On peut à tout moment sortir d'une boucle *for* ou *while* par un "**break**", en général après validation d'une condition.

```
1 # initialisations
2 nb_essais = 0
3
4 # instruction répétitive - tant que
5 while True :
6     reponse = input("entrez une voyelle minuscule et sans accent : ")
7     saisie_correcte = reponse in ('a','e','i','o','u','y')
8     nb_essais += 1
9     if saisie_correcte :
10         break
11
12 # affichage du résultat
13 print("Vous avez respecté la consigne en", nb_essais, "essais.")
```

Dans l'exemple ci-dessus, on a cette fois utilisé une boucle **a priori** infinie (`while True`) dont on sort par un **break**, à la condition que `saisie_correcte` soit *True*.