Informatique IV
Tests et boucles

Laurent Kaczmarek

PCSI^2 2013-2014
Lycée Louis Le Grand

Jeudi 19 septembre 2013

Le type bool de Python

- Le type `bool` permet de stocker dans une variable la *valeur de vérité* d'une assertion logique.
- Une variable booléenne ne peut prendre que deux valeurs : `True` (*vrai*) ou `False` (*faux*).
- Attention, `True` et `False` sont les valeurs du type `bool`. Ce ne sont donc pas des chaînes de caractères.
- Attention, `True` et `False` sont indistinguables des entiers `1` et `0`.

Opérateurs de comparaison

Ils permettent de construire des expressions boolénnes à partir d'autres expressions, par exemple de type numérique.

Opérateurs de comparaison

Pour des expressions `x` et `y` :

- `x == y` : `x` est égal à `y`;
  En Python, `=` est l'opérateur d'affectation, tandis que `==` est un opérateur de comparaison (qui correspond à la relation d'égalité des mathématiques).
- `x != y` : *`x` est différent de `y`;*
- `x > y` : *`x` est strictement supérieur à `y`;*
- `x < y` : *`x` est strictement inférieur à `y`;*
- `x >= y` : *`x` est supérieur ou égal à `y`;*
- `x <= y` : *`x` est inférieur ou égal à `y`.*

Exemples de comparaisons

```
>>> 15==45-3*10
True
```

```
>>> 15!=45-3*10
False
```

Opérateurs logiques

Pour combiner des variables booléennes, on dispose des trois *opérateurs logiques*.

- Par ordre de priorité : `or` (ou), `and` (et), et `not` (non).
- Ces opérateurs sont *paresseux*. Dans le cadre d'un `and`, si la première assertion est fausse, la deuxième n'est pas évaluée.

Exemples

```
>>> x=12
>>> (x%2==0) and
(x%3!=0)
False
```

```
>>> (x%2==0) or (x%3!=0)
True
>>> not((x%2==0) and
(x%3!=0))
True
```

Les instructions if, elif et else

Dans certains programmes, on vérifie des conditions avant d'exécuter les instructions. On se sert alors de l'instruction `if`.

Le test sans alternative

La forme la plus simple de l'instruction `if` est la suivante :

```
if expression booleenne:
    Bloc d'instructions
```

L'expression booléenne après le mot `if` est appelée la *condition booléenne*. Si sa valeur est `True`, alors le bloc d'instructions est executé. Sinon, Python passe à la suite. Les deux points `:` marquent la fin de la condition et l'indentation du bloc d'instructions est obligatoire, car elle délimite la portée de l'instruction `if`.

Les instructions if, elif et else

Le test avec une alternative

```
if expression booleenne:
    Bloc d'instructions 1
else:
    Bloc d'instructions 2
```

Si la valeur de l'expression booléenne est `True`, alors le bloc d'instructions 1 est exécuté. Sinon, Python exécute le bloc d'instructions 2.

Le test avec plusieurs alternatives

Une troisième forme élaborée de l'instruction `if` permet d'exécuter une ou plusieurs instructions alternatives. On utilise alors l'instruction `elif` (qui est la contraction de *else if*). L'instruction `else` permet de déterminer les autres cas.

Les instructions if, elif et else

Le test avec plusieurs alternatives

```
if expression booleenne 1:
    Bloc d'instructions 1
elif expression booleenne 2:
    Bloc d'instructions 2
elif expression booleenne 3:
    Bloc d'instructions 3

etc.

elif expression booleenne N:
    Bloc d'instructions N
else:
    Bloc d'instructions N+1
```

Les instructions if, elif et else

Le test avec plusieurs alternatives

Dans ce cas, le déroulement de la suite d'instructions est la suivante :

- La `condition booleenne 1` est évaluée. Si elle est vraie, le bloc d'instructions 1 est exécuté.
- Si la `condition booleenne 1` est fausse et la `condition booleenne 2` est vraie, le bloc d'instructions 2 est exécuté.
- etc.
- Si les `condition booleenne k` pour $k \in [\![1, N]\!]$ sont fausses, le bloc d'instructions $N+1$ est exécuté.
- À noter que dès qu'une condition est satisfaite, les instructions associées sont effectuées et le programme sort du test. Dans un test, on peut utiliser plusieurs instructions `elif`. Cependant, il y a au plus une instruction `else`.

Un exemple de test

- Écrire un programme demandant à l'utilisateur d'entrer deux nombres a et b et renvoyant $\max(a, b)$.
- Voici une solution :

```
a=int(input("Entrer a : "))
b=int(input("Entrer b : "))
if a<b:
    print(b)
else:
    print(a)
```

- L'instruction `x=input("Message à l'utilisateur")` a pour effet d'afficher le `message à l'utilisateur` dans l'interpréteur, d'attendre que l'utilisateur entre une réponse au clavier, d'enregistrer celle-ci dans la variable `x` (sous la forme d'une chaîne de caractères, d'où le passage par une conversion en entier au moyen de la fonction `int`) dès l'activation de la touche *Return* par l'utilisateur.

La boucle for

Les boucles `for` permettent de réaliser successivement une même action un nombre déterminé de fois.

La commande range

- La commande `range` permet de créer un objet de type `range` listant une suite arithmétique d'entiers.
- Par exemple :

```
>>> range(2,9)
2, 3, 4, 5, 6, 7, 8
>>> range(9)
[0,1, 2, 3, 4, 5, 6, 7, 8]
```

- Plus précisément, la commande `range(debut,fin,pas)` permet de parcourir les entiers compris entre `debut` et `fin-1` avec un pas égal à `pas`.
- On prendra garde aux indices de début et de fin ! L'argument `fin` est par défaut égal à `0` et l'argument `pas` est par défaut égal à `1`.

La boucle for

Syntaxe

- La boucle `for` permet d'appliquer un même bloc d'instructions, successivement, à chacun des éléments d'une liste.
- La syntaxe d'une boucle for est la suivante :

```
for indice in range(debut,fin,pas):
    Bloc d'instructions
```

- Le `bloc d'instructions` est executé pour chacune des valeurs `indice` de la suite générée par `range`.
- L'utilisateur peut choisir n'importe quel nom valable de variable pour l'`indice`.
- L'indentation du bloc d'instructions est obligatoire, car elle délimite la portée de l'instruction `for`.

La boucle for

Un exemple

- Sommons les entiers pairs compris entre 0 et 6.
- Voici une solution :

```
somme=0
for i in range(0,7,2):
    somme=somme+i
print(somme)
```

- On trouve bien sûr 12.
- Pour comprendre le fonctionnement des boucles `for`, on représentera l'évolution des variables dans un tableau :

i	0	2	4	6
somme	0	2	6	12

La boucle while

Si on souhaite répéter une séquence d'instructions un nombre de fois indéterminé, on utilise une boucle `while`.

Syntaxe

- Voici la syntaxe :

```
while expression booleenne:
    Bloc d'instruction
```

- Le déroulement de la boucle est le suivant : on évalue l'expression booléenne (appelée condition comme dans le cas des tests); si le résultat est `True`, on effectue le bloc d'instructions et on revient au début de la boucle; si le résultat est `False`, on sort de la boucle.
- Il faut veiller à ce que le corps de la boucle contienne une instruction qui change la valeur d'une variable intervenant dans la condition de manière à ce que la boucle se termine.

La boucle while

Un exemple

- Déterminer le plus petit entier n tel que

$$1 + 2 + \cdots + n \geqslant 2013$$

- Voici une solution :

```
somme,k=0,0
while somme<2013:
    k=k+1
    somme=somme+k
print(k)
```

- On trouve 63.