Feuille de cours 2 (informatique) : Booléens et test if

Le test `if` (“si”) permet de demander à Python de n'exécuter une partie du code que si une certaine condition est vérifiée. Cela s'avère très utile dans des situations où plusieurs alternatives se dégagent, mais aussi pour gérer des exceptions.

1 Booléens

La ou les conditions sous lesquelles une partie du code va être exécutée s'expriment par des booléens. Il s'agit d'un type de variable Python qui ne peut prendre que deux valeurs : `True` (“vrai”) et `False` (“faux”). On construit des booléens grâce aux opérateurs suivants :

Opérateur	Description	Exemple
`==`	est égal à	`1 == 1` renvoie `True`
`!=`	est différent de	`1 != 1` renvoie
`>=`	est supérieur (ou égal) à	`1 >= 1` renvoie
`>`	est supérieur strict à	`1 > 1` renvoie
`<=`	est inférieur (ou égal) à	`1 <= 1` renvoie `True`
`<`	est inférieur strict à	`1 < 1` renvoie `False`

Table 1 – Les opérateurs de comparaison en Python

Attention, il ne faudra pas confondre le symbole = de l'affectation, et le symbole == qui permet de tester si deux valeurs sont égales. L'expression `1=2` renvoie un message d'erreur, et l'expression `1==2` renvoie le booléen `False`.

Par ailleurs, on peut les combiner grâce aux connecteurs logiques suivants :

Connecteur	Description	Exemple
`and`	ET	`1 == 1` **`and`** `1 == 2` renvoie
`or`	OU (inclusif)	`1 == 1` **`or`** `1 == 2` renvoie
`not`	NON (négation)	**`not`** `1 == 1` renvoie

Table 2 – Les connecteurs logiques en Python

Exercice 1
On suppose qu'une variable réelle x a été définie préalablement. Construire les booléens exprimant les assertions suivantes :

1. $x \in [2, +\infty[$
2. $x \in [2, 3[$
3. $x \in]-\infty, 1[\cup [2, +\infty[$
4. $x^2 + x + 1 = 0$

Exercice 2
Soient Soient `a` et `b` des variables entières. Donner un booléen exprimant les conditions suivantes :

- `a` vaut 3 et `b` ne vaut pas 5 :
- `a` est supérieur à `b` ou inférieur à -`b` :
- `a` et `b` sont de même signe :

On aura enfin souvent besoin de tester la parité d'une variable entière (c'est-à-dire si elle est paire ou impaire). Plus généralement, il peut être utile de savoir distinguer le cas où `n` est un multiple de `d`. La commande `n % d` permet d'obtenir le reste de la division euclidienne de `n` par `d`.

Cela signifie que `n % d` renvoie l'unique entier `r` compris entre 0 et $d-1$ tel que $n = dq+r$ pour un certain entier `q`.

Par exemple, `10 % 3` vaut , `(-5) % 4` vaut .

Important : la condition "`n` est pair" s'écrit donc :
et la condition "`n` est impair" :

Exercice 3
Soient `a` et `b` des variables entières. Donner un booléen exprimant les conditions suivantes :

- `a` et `b` sont impairs :
- `a` est multiple de 3 :
- `b` n'est pas multiple de 3 :
- `a` et `b` ont le même chiffre des unités quand ils sont écrits en base 10 :

2 Instructions `if` et `if/else`

2.1 `If`

La syntaxe d'un test est la suivante :

```
1 if condition :
2     # Vos instructions si la condition est satisfaite
3 # Vos instructions en dehors du test
```

Si la condition `condition` est vérifiée, Python "rentre dans le `if`" et exécute les instructions. Si elle ne l'est pas, Python, ne fait rien et passe directement aux instructions en dehors du test. Faisons quelques remarques :

- Les deux points à la fin de la condition sont indispensables.
- C'est l'indentation qui délimite les instructions dépendant du `if` de celles en dehors du test.
- La condition booléenne `condition` doit être écrite sur une seule ligne.
- On peut faire suivre plusieurs tests à la suite, ou imbriqués les uns dans les autres.

2.2 If...else

On peut utiliser un test `if` seul, mais bien souvent, on veut traiter une alternative de deux cas : "si telle condition est vérifiée alors faire ..., et sinon faire ...".

Cela peut se faire avec deux tests `if` à la suite : l'un testant la condtion `condition` et l'autre testant **`not`**`(condition)`. Complétez par exemple la fonction suivante pour qu'elle renvoie x^2 si $x \geq 1$ et 0 sinon :

```
def fun(x):
    if x >= 1:
        return x**2
    if      :
        return 0
```

Pour alléger le code[1] dans ce cas de figure, il existe la commande `else` ("sinon") qui suit la syntaxe suivante :

```
if condition :
    # Vos instructions si la condition est satisfaite
else :
    # Vos instructions si la condition n'est pas satisfaite
# Vos instructions en dehors du test
```

Si la condition est satisfaite, Python exécute les instructions du bloc `if` puis ignore celles du bloc `else`. Inversement si la condition n'est pas satisfaite, Python ignore les instructions du bloc `if` et exécute celle du bloc `else`.

Faisons quelques remarques :

- La condition `condition` ne s'écrit qu'une seule fois : après le `if` **et pas après le `else`**.
- Le bloc `else` est au même niveau d'indentation que le bloc `if`.
- Les deux points et l'indentation sont toujours indispensables.

On renvoie aux exemples du TP2 : fonction valeur absolue, maximum, etc.

1. et surtout pour ne pas demander à Python d'effectuer un calcul dont on connaît déjà le résultat : si on vient d'évaluer la condition `condition`, il est inutile d'évaluer juste ensuite la condition **`not`**`(condition)` !

2.3 Entraînement

Exercice 4
Écrire une fonction `fun1` prenant en argument un entier `n` et renvoie la chaîne de caractères `"pair"` si `n` est pair, et la chaîne `"impair"` sinon. On proposera 3 écritures :

1. une avec `else`, `if` et deux `return`,
2. une avec un seul `return`,
3. une sans `else`.

Exercice 5
Écrire une fonction `fun2` prenant en argument un entier `n` et renvoyant le booléen `True` si `n` est pair, et `False` si `n` est impair.

3 Instruction `if...elif`

Si l'on souhaite considérer une situation à plus de deux alternatives, on peut utiliser la commande `elif`.

Quoiqu'un peu subtil, le sens de cette commande est relativement transparent : il s'agit de la contraction de `else if` ("sinon, si"). Il s'agit donc de donner des instructions dans le cas où **une première condition n'est pas vérifiée et une deuxième est vérifiée**. La syntaxe est la suivante :

```
if condition1 :
    # Vos instructions si condition1 est satisfaite
elif condtion2 :
    # Vos instructions si condition1 n'est pas satisfaite mais que
       condition2 l'est

# Vos instructions en dehors du test
```

Si la condition `condition1` est satisfaite, Python exécute le premier bloc d'instruction et ignore le deuxième (même si `condition2` est satisfaite !). Si la condition `condition1` n'est pas saisfaite, Python ignore le premier bloc d'instructions, et n'exécute le deuxième que si `condition2` est satisfaite.

Faisons quelques remarques :

- À la différence de `else`, **la commande `elif` est toujours suivie d'une condition.**
- Tout comme pour `else`, les deux points et l'indentation sont indispensables.
- Le bloc `elif` est au même niveau d'indentation que le bloc `if`.

Remarque : La convention d'indentation de la commande `elif` peut sembler étrange car elle est entièrement équivalente aux instructions suivantes, où l'indentation est différente :

```
if condition1 :
    # Instructions du bloc 1
else :
    if condition 2 :
        # Instructions du bloc 2

# Instructions en dehors du test
```

Exercice 6
Que contient la variable `a` après exécution des scripts suivants ?

```
a = 1
if a > 2 :
    a += 1
elif a > 0 :
    a -= 1
```

```
a = 0
if a < 2 :
    a += 1
elif a < 3 :
    a -= 1
```

On peut enchaîner les commandes `elif`. Dans ce cas, tout nouveau bloc d'instructions suivant une commande **`elif`** `condition_k` ne sera traité qu'à condition : qu'aucune des conditions précédentes n'ait été satisfaites, et que `condition_k` soit satisfaite. Il est enfin possible – et assez habituel – de conclure une telle distinction de cas par une commande `else` donnant des instructions dans le cas où aucune condition testée précédemment n'ait été satisfaite.

Dans ce cas, il faut faire particulièrement attention à la façon dont les conditions demandées s'excluent les unes par rapport aux autres. En particulier l'ordre des conditions a son importance.

Exercice 7
Que contient la variable `a` après exécution des scripts suivants dans le cas où `a` est initialisée à 12 ? à 7 ? à 3 ? à -1 ?

```
1 if a >= 10 :
2     a = a + 1
3 elif a >= 5 :
4     a = a +1
5 elif a >= 0 :
6     a = 2*a
7 else :
8     a = 3*a
```

```
1 if a >= 10 :
2     a = a + 1
3 elif a >= 0 :
4     a = 2*a
5 elif a >= 5 :
6     a = a - 1
7 else :
8     a = 3*a
```

```
1 if a >= 10 :
2     a = a + 1
3 elif a >= 0 :
4     a = 2*a-2
5 if a >= 5 :
6     a = a - 1
7 else :
8     a = 3*a
```

Exercice 8
Écrire une fonction `fun3` qui prend en argument une variable réelle `x` et qui renvoie la valeur de $f(\texttt{x})$ où f est la fonction définie par morceaux par :

$$f(x) = \begin{cases} x+1 & \text{si } x \leq 0 \\ 1 & \text{si } 0 < x \leq 2 \\ x-1 & \text{si } 2 < x \end{cases}$$

Exercice 9
Écrire une fonction `fun4` qui prend en argument une variable réelle `x` et qui renvoie la valeur de $f(\texttt{x})$ où f est la fonction définie par morceaux par :

$$f(x) = \begin{cases} x^2 & \text{si } x < 1 \\ \dfrac{x^2}{2} & \text{si } 1 \leq x < 3 \\ \left(\dfrac{x}{2}\right)^2 & \text{si } 3 \leq x \end{cases}$$

Exercice 10
Écrire une fonction `fun5` prenant en argument une variable réelle `x` et renvoyant la valeur de $g(x)$ où g est la fonction "en escalier" dont le graphe est le suivant :