COURS : OPERATEURS BOOLEENS et Applications.

CONDITIONS AVANCEES , OPERATEURS BOOLEENS :

Les opérateurs booléens : le « et »

La carte 12-25 n'est disponible que **si** vous avez moins de 25 ans **et si** vous avez plus de 12 ans. Écrivons un programme qui nous dise si on a droit à la carte ou pas :

```
age = int(input())
if age <= 25:
   if age >= 12:
      print("Carte possible")
   else:
      print("Carte impossible")
else:
   print("Carte impossible")
```

On voit que le programme n'est pas très efficace, car on a deux fois l'instruction

```
print("Carte impossible")
```

De plus, si on regarde la phrase ci-dessus on voit qu'on a utilisé le mot "**et**" pour dire qu'on voulait les deux conditions vraies en même temps. En Python, il est aussi possible de combiner des conditions :

```
age = int(input())
if (age <= 25) and (age >= 12):
   print("Carte possible")
else:
   print("Carte impossible")
```

On a donc utilisé l'opérateur `and`, la traduction en anglais du "**et**", qui permet de combiner les deux conditions `(age <= 25)` et `(age >= 12)`.

Le programme est tout de suite plus court, plus clair, et ressemble plus à la phrase en langage naturel.

Remarque :

Une condition est toujours soit vraie, soit fausse. Une valeur qui ne peut être que vraie ou fausse est appelée valeur *booléenne*. Cela donne donc son nom aux opérateurs *booléens* qui permettent de manipuler ces valeurs, de la même manière que les opérateurs numériques permettent de manipuler les nombres. L'opérateur `and` est donc un opérateur *booléen*.

Comme pour les opérateurs numériques (`+`, `-`, `/`, `*`), il existe un ordre de priorité entre les opérateurs numériques et booléens de Python mais nous ne rentrerons pas dans ces détails. Il est, dans tous les cas, préférable de faire comme ci-dessus : utiliser des parenthèses pour avoir un code bien clair, facilement compréhensible.

Les opérateurs booléens : le « ou »

Vous avez vu comment combiner deux conditions lorsqu'on veut que les deux soient vraies en même temps. On veut parfois en avoir au moins une des deux, c'est-à-dire soit l'une, soit l'autre, soit les deux. Par exemple on peut avoir une réduction **si** on a moins de 25 ans **ou** si on a plus de 60 ans. On a ici utilisé le mot "**ou**", qui est une autre manière de combiner des conditions et qui se traduit en Python par l'opérateur booléen `or` :

```
age = int(input())
if (age <= 25) or (age >= 60):
   print("Réduction possible")
else:
   print("Pas de réduction")
```

Dans la réalité il faut en fait avoir entre 12 et 25 ans et non pas simplement moins de 25 ans. On peut donc combiner les conditions en utilisant à la fois un "**et**" et un "**ou**", en n'oubliant pas de mettre les bonnes parenthèses :

```
age = int(input())
if ( (12 <= age) and (age <= 25) ) or (age >= 60):
   print("Réduction possible")
else:
   print("Pas de réduction")
```

On peut donc combiner facilement les opérateurs booléens pour construire des conditions complexes à partir de conditions simples.

Les booléens

En Python le programme suivant :

```
if prix < 10:
   print("Pas cher")
```

peut aussi s'écrire

```
estPasCher = (prix < 10)
if estPasCher:
   print("Pas cher")
```

On a donc stocké dans la variable `estPasCher` la valeur de la comparaison `prix < 10` pour la réutiliser plus tard dans un `if`.

La variable `estPasCher` est appelée une *variable booléenne* ou un *booléen* car elle ne peut être que vraie ou fausse, ce qui correspond en Python aux valeurs `True` (pour vrai) et `False` (pour faux).

Ainsi en Python, `(3 < 10)` vaut `True` et `(11 == 7)` vaut `False` . `True` et `False` sont donc des constantes du langage, au même titre que 0, 1 ou encore 42.

On peut donc affecter directement la valeur `True` ou `False` à une variable :

```
toujoursVraie = True
if toujoursVraie:
   print("La variable toujoursVraie vaut 'True'")
```

On peut également utiliser les opérateurs `and` et `or` comme le montre l'exemple suivant :

```
estSenior = (age >= 60)
estJeune = (age <= 25) and (age >= 12)
reductionPossible = (estSenior or estJeune)
if reductionPossible:
   print("Réduction!")
```

Les opérateurs booléens : la négation

Imaginons la situation suivante : vous avez le droit à une réduction si vous avez entre 12 et 25 ans ou si vous avez plus de 60 ans mais, si vous n'avez pas de réduction et que vous faites plus de 5 000km par an, alors vous avez le droit à un cadeau.

Un programme Python traduisant cela pourrait être :

```
age = int(input())
nbKm = int(input())
if ( (12 <= age) and (age <= 25) ) or (age >= 60):
   print("Réduction possible")
else:
   print("Pas de réduction")
if ( ( (age < 12) or (age > 25) ) and (age < 60) ) and (nbKm >= 5000):
   print("Cadeau")
else:
   print("Pas de cadeau")
```

Ce programme est complexe et on sent qu'il y a des répétitions. En effet, si on définit la variable

```
reductionPossible = ( (12 <= age) and (age <= 25) ) or (age >= 60)
```

alors on a un cadeau si `réductionPossible` n'est pas vraie et si la longueur du trajet est plus grande que 5000km.

Mais, une condition n'est pas vraie signifie que la condition contraire est vraie !

Il est possible de calculer le contraire d'une condition en Python, en utilisant l'opérateur booléen `not` qui renvoie le contraire de la valeur qu'on lui donne :

```
age = int(input())
nbKm = int(input())
reductionPossible = ( (12 <= age) and (age <= 25) ) or (age >= 60)
if reductionPossible:
   print("Réduction possible")
else:
   print("Pas de réduction")
if ( not (reductionPossible) ) and (nbKm >= 5000):
   print("Cadeau")
else:
   print("Pas de cadeau")
```

Le programme est tout de suite beaucoup plus clair !

Lorsqu'on utilise l'opérateur booléen `not` pour avoir le contraire d'une condition, on dit qu'on a pris la *négation* de la condition.

BOOLEENS : choses à ne pas faire

Une maladresse classique avec les booléens est de faire quelque chose comme ceci

```
prix = int(input())
estCher = (prix > 100)
if estCher == True:
    print("C'est cher !")
```

Le code est correct mais on n'a pas besoin de tester si quelque chose est égal à `True` ou `False` , si ce quelque chose est lui même déjà `True` ou `False` !

On remplacera donc

```
if estCher == True:
    print("Cher")
```

par

```
if estCher:
    print("Cher")
```

et

```
if estCher == False:
    print("Pas Cher")
```

par

```
if not(estCher):
    print("Pas Cher")
```

Un programme ne doit jamais contenir de `== True` ou `== False` .