Structures conditionnelles

12 septembre 2021

I Un exemple introductif : le tarif de musée

Les tarif réduits pour l'entrée à un musée sont :

- gratuit pour les moins de 5 ans
- mi-tarif pour les moins de 16 ans
- mi-tarif pour les plus de 65 ans
- mi-tarif pour les étudiants

On désire écrire un script qui donne le prix du billet selon la personne. Une première possibilité pourrait être :

Ce script est écrit en «pseudo-code», le symbole d'affectation est ←.

```
Si age < 5
   prix ← 0
Fin du Si
Si age < 16
   prix ← tarif/2
Fin du Si
Si age > 65
   prix ← tarif/2
Fin du Si
Si statut = "etudiant"
   prix ← tarif/2
Sinon
   prix ← tarif
Fin du Si
```

Mais ce code est incorrect. En effet, supposons que le visiteur du musée soit un petit garçon de 4 ans. Comme `age < 5`, alors la variable prix vaut 0. Mais on a aussi `age < 16`, donc l'affectation `prix ← tarif/2` sera réalisée et `prix` vaudra `tarif` divisé par 2. De même, puisque notre petit garçon n'est pas étudiant [1], c'est l'alternative du sinon qui est exécutée, autrement, la variable `prix` sera à nouveau modifiée et égale à `tarif`. Ainsi le petit garçon paiera plein tarif ! Pour

1. Le petit garçon est pourtant en MatSup (Maternelle Supérieure) mais son statut d'étudiant lui est refusé :-)

indiquer que si une condition est réalisée, on veut ignorer les autres, on utilisera le mot clé `Sinon`.

```
Si age < 5
   prix ← 0
Sinon
   Si age < 16
      prix ← tarif/2
   Sinon
      Si age > 65
         prix ← tarif/2
      Sinon
         Si statut = "etudiant"
            prix ← tarif/2
         Sinon
            prix = tarif
         Fin du Si
      Fin du Si
   Fin du Si
Fin du Si
```

Il faut avouer, que le code est assez lourd. Heureusement, on pourra utiliser la présentation suivante plus claire :

```
Si age < 5
   prix ← 0
Sinon Si age < 16
   prix ← tarif/2
Sinon Si age > 65
   prix ← tarif/2
Sinon Si statut = "etudiant"
   prix ← tarif/2
Sinon
   prix = tarif
Fin du Si
```

Pour terminer signalons une dernière possibilité, plus concise mais peut-être moins lisible qui utilise le connecteur logique «ou» :

```
Si age < 5
   prix ← 0
Sinon Si age < 16 ou age > 65 ou statut = "etudiant"
   prix ← tarif/2
Sinon
   prix = tarif
Fin du Si
```

II La syntaxe Python

C'est une des qualités de Python, son code est très proche du pseudo-code, on retiendra :

- les mots clés `if`, `elif` (contraction de else if), et `else`.
- le symbole : et l'indentation qui délimite les blocs d'instructions à exécuter (il n'y pas de fin de si)

```
<instructions avant le test>
if <condition1> :
   <instructions du bloc "alors">
elif <condition2>: # optionnel
   <instructions du bloc "sinon si">
else:
   <instructions du bloc "sinon">
<instructions après le test>
```

Voici le code pour l'exemple précédent des tarifs de musée :

```
if age < 5:
    prix = 0
elif age < 16:
    prix = tarif/2
elif age > 65:
    prix = tarif/2
elif statut = "etudiant"
    prix = tarif/2
else:
    prix = tarif
```

III Avec la console

Même si cela est moins pratique que dans un fichier, on peut tester son code directement avec la console : par exemple, si a est un nombre, nous souhaitons afficher son double uniquement si $a > 5$. On écrit pour cela avec la console :

```
>>> a = 10
>>> if a > 5:
... print(2*a)
  File "<console>", line 2
    print(2*a)
        ^
 IndentationError: expected an indented block
```

Comprenons le message d'erreur : il s'agit d'une erreur d'indentation. La ligne du prompt principal est suivie de trois points qui représentent un prompt secondaire. Nous avons une erreur car nous n'avons pas indenté l'instruction `print(2*a)`.

Rectifions :

```
>>> a = 10
>>> if a > 5:
...  print(2*a)
...
20
```

Évidemment, si l'on change la valeur de `a` et que la condition `a > 20` est fausse, le double de a n'est pas affiché :

```
>>> a = 2
>>> if a > 5:
...  print(2*a)
...
```

IV Notion de booléen et opérations associées : or, and, not

Pour Python, la condition précédente `a > 20` était un objet de type booléen (se prononce «bouléen», cela vient du mathématicien britannique George Boole). Un booléen ne prend que deux valeurs `True` ou `False`.

1) Opérateurs de comparaison

On fabrique souvent des booléens en comparant des nombres :

```
>>> 9 > 3
False
>>> a = 3 * ( 4 + 5) == 3 * 4 + 5   # test d'égalité
True
>>> type(a)
<class 'bool'>
```

Voici les principaux opérateurs de comparaison entre nombres.

```
>>> 3 < 5  # strictement inférieur
False
>>> 3 <= 3  # inférieur ou égal
True
>>> 3.0 == 3  # égal
True
>>> 3 != 3.1 # différent
False
```

Attention, à ne pas confondre l'opérateur d'égalité `==` avec l'opérateur d'affectation `=`.

2) Quelques cas particuliers

On sera aussi vigilant aux tests d'égalité avec les flottants :

```
>>> 3*0.1 == 0.3
False
>>> import math
>>> a = math.sqrt(3)
>>> a**2 == 3
False
```

```
>>> n = 10**15
>>> 1+ 1/n == 1
False
>>> n = 10**16
>>> 1+ 1/n == 1
True
>>> 1/n == 0
False
```

Il faudra donc être très prudent avec les flottants, se référer au chapitre **??** «Représentation numérique de l'information».

Python peut aussi comparer des chaînes de caractères en utilisant l'ordre naturel du dictionnaire dit ordre lexico-graphique...

```
>>> "football" < "volley"
True
>>> "football" < "Ski"
False
```

Quel est le problème avec la deuxième instruction ? Et bien, la majuscule S n'est pas le même caractère que la minuscule s... En fait un caractère est représenté par un nombre appelé son «code ASCII». Les lettres majuscules A,B,..., Z sont codés par $65, 66, \ldots, 90$, tandis que les minuscules a,b,..., z par $97, 98, \ldots, 122$. Le code ASCII de f étant 102 et celui de S étant 83, on comprend donc que la chaîne `"football"` est plus grande que `"Ski"`.

Rappelons les fonctions `ord()` et `chr()` qui donnent le code ASCII d'un caractère et réciproquement :

```
>>> ord('S'), ord('f')
(83, 102)
>>> chr(83), chr(102)
('S', 'f')
```

3) Connecteurs logiques

On peut enfin combiner des booléens pour fabriquer de nouveaux booléens à l'aide des connecteurs logiques `or`, `and` et `not`.

```
x = 20
x % 5 == 0  # Réponse: True
x % 3 == 0  # Réponse: False
(x % 5 == 0) and (x % 3 == 0) # Réponse: False
(x % 5 == 0) or (x % 3 == 0)  # Réponse: True
```

On rappelle que si P et Q sont des propositions, la proposition (P et Q) est vraie si les deux propositions sont vraies. Elle est fausse sinon. De même, la proposition (P ou Q) est vraie si l'une des deux (ou les deux) propositions est vraie. Elle est fausse sinon. On peut résumer cela à l'aide des tables de vérité suivantes :

A	B	A et B	A ou B
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

V Exemples modèles

Exercice 1 (Fonction valeur absolue) Écrire une fonction nommée `valeur_absolue` qui renvoie la valeur absolue d'un nombre.

Corrigé:

```
def valeur_absolue(x):
   if x>0:
      return x
   else:
      return -x
```

Exercice 2 (Jeu de hasard) On tire au hasard un entier entre 1 et 100.

- Si le nombre est pair, on gagne 1 euro
- s'il se termine par un 7, on gagne 10 euros
- dans les autres cas, on perd 3 euros.

Écrire un script qui affiche le gain obtenu après tirage aléatoire d'un entier. On pourra utiliser la fonction `randint()` du module `random` pour générer un entier aléatoire.

Corrigé: La principale difficulté réside dans la traduction mathématique des deux premières conditions :

- Un entier est pair si et seulement si il est divisible par 2, c'est-à-dire si son reste dans la division par 2 vaut 0 (égal à 0 modulo 2) .
- Un entier se termine par 7 ssi son reste dans la division par 10 vaut 7, (c'est-à-dire s'il est égal à 7 modulo 10).

```
import random # on importe le module random pour le tirage aléatoire
n = random.randint(1,100)  # on tire un entier entre 1 et 100

if n % 2 == 0:
   print( +1)
```

```
elif n % 10 == 7:
   print(+10)
else:
   print(-3)
```

Exercice 3 (Mot de passe) On doit choisir un mot de passe. Il y a deux contraintes :, il doit contenir entre 6 et 10 caractères et le premier caractère doit être une voyelle. Écrire un script qui teste la validité d'un mot de passe.

Corrigé:

```
mot_de_passe  = input('Rentrez un mot de passe: \n')
# le caractère \n est un saut de ligne

if len(mot_de_passe) < 6 or len(mot_de_passe) > 10:
    print("mot de passe non valide, le nombre de caractères doit être compris entre 6 et
else:
    premier_caractere = mot_de_passe[0]
    voyelles = {'a','e','i','o','u','y'}
    if premier_caractere not in voyelles:
         print("mot de passe non valide, la première lettre doit être une voyelle")
    else:
         print("mot de passe valide")
```

VI Quelques pièges à éviter

On initialise une variable `a` à la valeur 10 avec l'instruction `a = 10`.

Vous devez deviner ce qu'affichent les trois scripts suivants :

```
if a < 5:
    print("plus petit que
if a > 2:
    print("plus grand que
if a > 4:
    print("plus grand que
```

```
if a < 5:
5")  print("plus petit que
elif a > 2:
2")  print("plus grand que
elif a > 4:
4")  print("plus grand que
```

```
if a < 5:
5")  print(a)
elif a > 4:
2")  print("plus grand que 4")
elif a > 2:
4")  print("plus grand que 2")
```

Corrigé:

```
"plus grand que 2"
"plus grand que 4"
```

```
"plus grand que 2"
```

```
"plus grand que 4"
```

Explication : la variable a vaut 10, les conditions a $>$ 2 et a $>$ 4 sont vraies. Le script 1 comporte 3 structures conditionnelles (avec une unique alternative) indépendantes, elles sont parcourues les unes après les autres. Au contraire, les scripts 2 et 3 comportent une seule structure conditionnelle mais ayant 3 alternatives. Dès que l'une des alternatives est réalisée, les suivantes sont ignorées. C'est pourquoi dans le script 2 puisque a $>$ 2, `"plus grand que`

2" est affiché mais ensuite on sort de la structure conditionnelle. Ce script 2 est ainsi mal écrit, la dernière alternative est inutile, puisque si $a > 4$, on a en particulier $a > 2$ et c'est donc la deuxième alternative qui sera exécuté. Le script 3 gomme cette abération.

Ceci illustre deux points importants :

- la différence entre if and if et if else
- l'importance de l'ordre dans les elif