Cours_Les_structures_conditionnelles

August 24, 2019

Enseignement de spécialité > 1ère NSI > PYTHON > Les structures conditionnelles
David LATOUCHE - Cédric GERLAND - Lycée Saint-Exupéry

1 Notion de test

Les tests sont un élément essentiel à tout programme informatique si on veut lui donner un peu de complexité car ils permettent à l'ordinateur de faire des choix. Pour cela, Python utilise l'instruction *if* ainsi qu'une comparaison. Voici deux exemples :

[1]:
```
age = 20
if age >= 18 :
    print("Vous êtes majeur !")
print("Fin du programme")
```

```
Vous êtes majeur !
Fin du programme
```

[2]:
```
age = 16
if age >= 18 :
    print("Vous êtes majeur !")
print("Fin du programme")
```

```
Fin du programme
```

ll y a plusieurs remarques à faire concernant les deux exemples ci-dessus.

- Dans le premier exemple, le test étant vrai, l'instruction *print("Vous êtes majeur !")* est exécutée.
- Dans le second exemple, le test étant faux, l'instruction *print("Vous êtes majeur !")* n'est pas exécutée.
- La ligne qui contient l'instruction *if* se termine par le caractère deux-points (:) .
- Les blocs d'instructions dans les tests doivent forcément être indentés. L'indentation indique la portée des instructions à exécuter si le test est vrai. L'indentation se manifeste par une tabulation (décalage à droite) généralement générée automatiquement par l'IDE Python. Attention néanmoins à vérifier les indentations dans vos codes car c'est une source d'erreur importante.

Les opérateurs de comparaison que l'on peut utiliser dans les structures conditionnelles sont les suivants : < inférieur à ; > supérieur à ; <= inférieur ou égal à ; >= supérieur ou égal ; == égal à ; != différent de.

2 Tester plusieurs cas

2.1 La structure if else

Parfois, il peut être pratique de tester si la condition est vraie ou si elle est fausse dans une structure unique. Plutôt que d'utiliser deux instructions *if* successives, on peut utiliser une structure *if else*.

[3]:
```
age = 20
if age >= 18 :
    print("Vous êtes majeur !")
else:
    print("Vous êtes mineur !")
print("Fin du programme")
```

```
Vous êtes majeur !
Fin du programme
```

[4]:
```
age = 16
if age >= 18 :
    print("Vous êtes majeur !")
else:
    print("Vous êtes mineur !")
print("Fin du programme")
```

```
Vous êtes mineur !
Fin du programme
```

2.2 La structure if elif else

Il arrive souvent que l'on soit amené dans un programme à tester plus que deux cas possibles. La structure *if elif else* permet de faire cela simplement :

[5]:
```
choix = 3
if choix == 1:
    print("Choix 1")
elif choix == 2:
    print("Choix 2")
elif choix == 3:
    print("Choix 3")
else:
    print("Erreur de saisie")
print("Fin du programme")
```

```
Choix 3
Fin du programme
```

[6]:
```
choix = 4
if choix == 1:
    print("Choix 1")
```

```
elif choix == 2:
    print("Choix 2")
elif choix == 3:
    print("Choix 3")
else:
    print("Erreur de saisie")
print("Fin du programme")
```

```
Erreur de saisie
Fin du programme
```

Dans l'exemple précédent, Python teste la première condition, puis, si et seulement si elle est fausse, teste la deuxième et ainsi de suite... Si aucune des conditions précédentes est remplie, l'instruction contenue dans *else* est exécutée puis le programme sort de la structure *if elif else*.

Le *else* est optionnel : s'il n'est pas présent et qu'aucune des conditions est vérifiée, le programme sort de la structure *if elif* sans réaliser la moindre des instructions contenues dans la structure conditionnelle.

[7]:
```
choix = 4
if choix == 1:
    print("Choix 1")
elif choix == 2:
    print("Choix 2")
elif choix == 3:
    print("Choix 3")
print("Fin du programme")
```

```
Fin du programme
```

ATTENTION : il ne faut surtout pas confondre l'instruction d'affectation *choix = 4* (un seul signe égal ; = opérateur d'affectation) avec les instructions de tests d'égalités *choix == 1*, *choix == 2* ... (double signe égal ; == opérateur d'égalité). Ceci est une source d'erreur courante dans les programmes.

2.3 Tests multiples

Les tests multiples permettent de tester plusieurs conditions en même temps en utilisant des opérateurs booléens. Les deux opérateurs les plus couramment utilisés sont le **OU** et le **ET**. En Python, on utilise le mot réservé *and* pour l'opérateur **ET** et le mot réservé *or* pour l'opérateur **OU**. Respectez bien la casse des opérateurs *and* et *or* qui, en Python, s'écrivent en minuscule.

Exemple 1 : utilisation de *and* dans le test

[8]:
```
x = 1
y = 15
if x == 1 and y > 10:
    print("Le test est vrai")
else:
    print("Le test est faux")
print("Fin du programme")
```

```
Le test est vrai
Fin du programme
```

[9]:
```
x = 1
y = 5
if x == 1 and y > 10:
    print("Le test est vrai")
else:
    print("Le test est faux")
print("Fin du programme")
```

```
Le test est faux
Fin du programme
```

[10]:
```
x = 2
y = 15
if x == 1 and y > 10:
    print("Le test est vrai")
else:
    print("Le test est faux")
print("Fin du programme")
```

```
Le test est faux
Fin du programme
```

On peut remarquer qu'une structure faisant intervenir des conditions liées par l'opérateur booléen *and* est identique à des *if* imbriqués les uns dans les autres.

[11]:
```
x = 1
y = 15
if x == 1:
    if y > 10:
        print("Le test est vrai")
    else:
        print("Le test est faux")
else:
    print("Le test est faux")
print("Fin du programme")
```

```
Le test est vrai
Fin du programme
```

[12]:
```
x = 1
y = 5
if x == 1:
    if y > 10:
        print("Le test est vrai")
```

```
    else:
        print("Le test est faux")
else:
    print("Le test est faux")
print("Fin du programme")
```

```
Le test est faux
Fin du programme
```

[13]:
```
x = 2
y = 15
if x == 1:
    if y > 10:
        print("Le test est vrai")
    else:
        print("Le test est faux")
else:
    print("Le test est faux")
print("Fin du programme")
```

```
Le test est faux
Fin du programme
```

Néanmoins la structure *if condition1 and condition2:* est plus efficace et plus élégante que l'imbrication de deux boucles if l'une dans l'autre.

Exemple 2 : utilisation de *or* dans le test

[14]:
```
x = 1
y = 5
if x == 1 or y > 10:
    print("Le test est vrai")
else:
    print("Le test est faux")
print("Fin du programme")
```

```
Le test est vrai
Fin du programme
```

[15]:
```
x = 0
y = 5
if x == 1 or y > 10:
    print("Le test est vrai")
else:
    print("Le test est faux")
print("Fin du programme")
```

```
Le test est faux
Fin du programme
```

3 Tests de valeurs sur des floats

3.1 Le type float est une approximation des nombres réels

Lorsque l'on souhaite tester la valeur d'une variable de type float, le premier réflexe serait d'utiliser l'opérateur d'égalité comme suit qui doit renvoyer le booléen True si la condition est vérifiée et False dans le cas contraire :

[16]:
```
print(1/10 == 0.1)
```

```
True
```

Toutefois il est fortement déconseillé de procéder de la sorte. Pourquoi ? Pour la simple et bonne raison que Python stocke en mémoire les valeurs numériques des *floats* sous forme de nombres flottants (d'où leur nom), et cela mène à certaines limitations. Observez l'exemple suivant :

[17]:
```
print((3 - 2.7) == 0.3)
```

```
False
```

[18]:
```
print(3 - 2.7)
```

```
0.2999999999999998
```

Nous remarquons que le résultat de l'opération 3 - 2.7 n'est pas exactement égal 0.3 d'où le *False* renvoyé par le test conditionnel. En fait, ce problème ne vient pas de Python lui même, mais plutôt de la manière dont un ordinateur traite les nombres réels (ils doivent être préalablement convertis en nombres binaires pour devenir un *float*). Ainsi certaines valeurs de *float* ne peuvent être qu'approchées en raison de l'espace mémoire limité qui leur est alloué après conversion en binaire. Un *float* est donc une aproximation du nombre réel qu'il code. Une manière de s'en rendre compte est d'utiliser l'écriture formatée (sur laquelle nous reviendrons plus tard) en demandant l'affichage d'un grand nombre de décimales :

[19]:
```
print(0.3)
print("{:.5f}".format(0.3))
print("{:.60f}".format(0.3))
print("{:.60f}".format(3.0 - 2.7))
```

```
0.3
0.30000
0.299999999999999988897769753748434595763683319091796875000000
0.299999999999999822364316059974953532218933105468750000000000
```

On observe que lorsqu'on tape *print(0.3)*, Python affiche une valeur arrondie. En réalité, le nombre réel 0.3 ne peut être qu'approché lorsqu'on le code en nombre flottant. Il est donc essentiel d'avoir cela en tête lorsque l'on effectue un test .

3.2 Bonne pratique avec les floats

Pour les raisons évoquées ci-dessus, il ne faut surtout pas tester si un *float* est égal à une certaine valeur. La bonne pratique est de vérifier si un float est compris dans un intervalle avec une certaine précision. Si on appelle cette précision delta, on peut procéder ainsi :

[20]:
```
delta = 0.0001
nombre = 3.0 - 2.7
print(0.3 - delta < nombre < 0.3 + delta)
```

```
True
```

[21]:
```
print(abs(nombre - 0.3) < delta)
```

```
True
```

4 Commentaires

Vous commencez à connaître suffisament d'éléments de programmation pour réaliser des petits programmes de plus en plus élaborés. Il est conseillé dans ce cas de commenter votre code afin d'en faciliter sa lecture et sa compréhension. En effet, les programmeurs travaillent généralement rarement seuls sur un programme complexe ! En Python, on introduit une ligne de commentaires dans un code en la faisant précéder du symbole #. Les commentaires ne sont pas pris en compte lors de l'exécution du programme. Attention néanmoins à ne pas surcharger votre code de commentaires inutiles : il faut savoir trouver la juste mesure !

[22]:
```
# Ceci est un court commentaire tenant sur une ligne !
```

5 Entrée / sortie

5.1 Entrée clavier : la fonction input()

Certains programmes nécessitent de recueillir une donnée entrée au clavier par l'utilisateur afin d'effectuer des tests conditionnels. La fonction *input()* permet de réaliser cela en recueillant la chaîne de caractères (de type *str*) frappée au clavier. **ATTENTION :** si le test conditionnel porte sur une valeur numérique, il ne faudra pas oublier de convertir le résultat de *input()* en type *int* ou *float* à l'aide des fonctions de conversion *int()* ou *float()*. Ceci est une source d'erreur courante dans les programmes.

[23]:
```
age = input("Quel est votre age ? : ")
age = int(age)
if age >= 18 :
    print("Vous êtes majeur !")
else:
    print("Vous êtes mineur !")
print("Fin du programme")
```

```
Quel est votre age ? : 20
Vous êtes majeur !
Fin du programme
```

En l'absence de conversion str --> int du résultat de la fonction *input()*, le programme précédent ne pourra pas fonctionner correctement :

[24]:
```
age = input("Quel est votre age ? : ")
if age >= 18 :
    print("Vous êtes majeur !")
else:
    print("Vous êtes mineur !")
print("Fin du programme")
```

```
Quel est votre age ? : 20
```

```
␣
↪---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call␣
↪last)

<ipython-input-24-a2e7a70cdbec> in <module>
  1 age = input("Quel est votre age ? : ")
----> 2 if age >= 18 :
  3     print("Vous êtes majeur !")
  4 else:
  5     print("Vous êtes mineur !")


TypeError: '>=' not supported between instances of 'str' and 'int'
```

5.2 Sortie écran console : la fonction print()

Dans le cours précédent "Notion de variable", nous avons rencontré la fonction *print()* qui affiche une chaîne de caractères. En fait, la fonction print() affiche l'argument qu'on lui passe entre parenthèses et un retour à ligne. Ce retour à ligne supplémentaire est ajouté par défaut. Si toutefois, on ne veut pas afficher ce retour à la ligne, on peut utiliser l'argument par mot-clé *end* :

[25]:
```
print("Hello ")
print("John")
```

```
Hello
John
```

[26]:
```
print("Hello ", end="")
print("John")
```

```
Hello John
```

Il est également possible d'afficher le contenu de plusieurs variables (quel que soit leur type) en les séparant par des virgules :

[27]:
```
prenom = "Bruce"
nom = "Banner"
points_vie = 1000
print(prenom,nom,points_vie)
```

```
Bruce Banner 1000
```

Enfin, le caractère d'échappement antislash suivi de la lettre n permet de forcer un passage à la ligne lors de l'affichage d'une chaîne de caractères.

[28]:
```
print("Première ligne.\nDeuxième ligne.\nTroisième ligne")
```

```
Première ligne.
Deuxième ligne.
Troisième ligne
```

Il est à noté que le caractère d'échappement antislash doit être utilisé dans certaines situations. C'est le cas par exemple d'une chaîne de caractères qui comporterait par exemple des guillements (citation). Ces dernières doivent être échappées afin de ne pas interférer avec les guillements spécifiques à la désignation de la chaîne de caractères elle même.

[29]:
```
print("Je cite : \"une citation\"")
```

```
Je cite : "une citation"
```

L'absence du caractère d'échappement devant les guillemets de la citation conduira à un message d'erreur :

[30]:
```
print("Je cite : "une citation"")
```

```
  File "<ipython-input-30-49b8fb7b217e>", line 1
    print("Je cite : "une citation"")
                      ^
SyntaxError: invalid syntax
```

5.3 Ecriture formatée

La méthode *.format()* permet une meilleure organisation de l'affichage des variables. Elle est, par exemple, particulièrement bien adaptée lorsque l'on ne connait pas à l'avance la valeur d'une variable mais que l'on souhaite afficher celle-ci via un *print()* après un appel à la fonction *input()*.

[31]:
```
prenom = input("Entrez votre prénom : ")
nom = input("Entrez votre nom : ")
age = input("Entrez votre age : ")
message = "Bonjour {} {}. Vous avez {} ans !".format(prenom,nom,age)
print(message)
```

```
Entrez votre prénom : Tony
Entrez votre nom : Stark
Entrez votre age : 52
Bonjour Tony Stark. Vous avez 52 ans !
```

Dans la chaîne de caractères *message*, les accolades vides {} précisent l'endroit où le contenu de la variable doit être inséré. Juste après la chaîne de caractères, l'instruction *.format(prenom,nom,age)* fournie dans l'ordre la liste des variables à insérer. La méthode *.format()* agit sur la chaîne de caractères à laquelle elle est attachée par l'intermédiaire du point.

La méthode *.format()* permet également de gérer finement l'affichage des *floats* dans la console en précisant le nombre de décimales souhaité. Les accolades vides {} sont alors remplacées par {:.xf} ou x désigne le nombre de décimales que l'on souhaite conserver.

[32]:
```
vitesse = 100 / 3.6
print("La vitesse de la voiture est : ",vitesse)
```

```
La vitesse de la voiture est :  27.77777777777778
```

[33]:
```
vitesse = 100 / 3.6
print("La vitesse de la voiture est : {:.2f}".format(vitesse))
print("La vitesse de la voiture est : {:.3f}".format(vitesse))
print("La vitesse de la voiture est : {:.4f}".format(vitesse))
```

```
La vitesse de la voiture est : 27.78
La vitesse de la voiture est : 27.778
La vitesse de la voiture est : 27.7778
```

6 Activités pour vous entraîner

Activité 1 : Coder un programme Python simulant une interface d'authentification simplifiée demandant à l'utilisateur un id et un mot de passe. Les deux informations fournies seront comparées à celles stockées en dur dans le programme. Si l'authentification est réussie, le programme devra afficher un message de bienvenue comportant l'id saisie au clavier par l'utilisateur. En cas d'échec d'authentification le programme devra afficher l'un des messages suivants en fonction du type d'erreur rencontrée : - Echec d'authentification ! L'identifiant = xxxxx est incorrect. - Echec d'authentification ! Le mot de passe = xxxxx est incorrect. - Echec d'authentification ! Identifiant et mot de passe incorrects. Dans les messages précédents xxxxx désigne la valeur de l'identifiant ou du mot de passe saisie par l'utilisateur.

Remarque : Il s'agit bien évidemment d'une activité de découverte à but purement pédagogique. En réalité les id et les mots de passe des utilisateurs ne sont jamais stockés en clair dans le programme mais d'une manière cryptée dans une base de données annexe.

Activité 2 : Coder un programme Python qui calcule une vitesse en m/s puis en km/h en demandant à l'utilisateur : - la distance parcourue en m ; - la durée du parcours en s.

Les résultats du calcul de la vitesse seront affichés avec 2 décimales. Rappel : 1 m/s = 3.6 km/h. Le programme devra également s'assurer que la distance et la durée sont des grandeurs strictement positives. Dans le cas contraire un message d'erreur devra être affiché.