4. L'instruction conditionnelle : *if ... : else : ...*

▶ Cours

• Un programme doit parfois réagir de manière différente suivante les valeurs entrées au clavier par l'utilisateur, ou suivant le résultat d'un calcul ou d'un test.

• Tous les langages de programmation permettent d'aiguiller le déroulement d'un programme suivant le résultat de tests (aussi appelés conditions).

Un test est une expression booléenne : c'est une expression qui a pour résultat « *vrai* » (True en python) ou « *faux* » (False). Ce sont les deux seules valeurs possibles pour un test.

```
if condition :
  intruction1
  intruction2
  intruction3
else :
  intruction4
  intruction5
```

• En Python, la syntaxe d'une instruction conditionnelle est montrée dans le script à gauche. Remarquez les points suivants :

1°) la ligne de test commence par *if*, et la condition est suivie d'un deux points (indispensable, sinon Python arrêtera le script avec une erreur de syntaxe).

2°) les instructions 1, 2 et 3 sont celles qui seront exécutées si la condition est vraie (True). Il y en a 3 dans ce script, mais on peut n'en mettre qu'une ou en mettre 15 si necéssaire.

3°) Ces instructions doivent être indentées (décalées vers la droite) du même nombre d'espace pour indiquer qu'elles font partie du même *if*. Ce sont ces indentations qui permettent à Python de choisir les instructions à exécuter lorsque la condition est vérifiée.

Sur papier, quand on écrit un script, on utilise parfois le symbole ␣ qui représente un espace, afin de bien faire voir l'indentation des lignes.

4°) La partie « *else instruction4 instruction5* » est facultative. Ces instructions seront exécutées seulement si la condition qui suit le *if* est fausse (False).

5°) Le mot *else* doit être suivi d'un deux-points ; il doit être aligné avec le *if* auquel il correspond.

Après le *else*, les instructions qui font partie de ce cas doivent être décalées, et alignées les unes avec les autres.

6°) Lorsqu'on veut enchaîner plusieurs tests, on peut utiliser l'instruction *elif*, qui est la contraction de *else if*. Cette instruction doit être alignée avec le *if* correspondant, et elle doit être suivie d'une condition, puis d'un deux-points et après d'instructions indentées, comme pour un *if*. Les instructions seront exécutées si la condition qui suit le *if* est fausse, mais que la condition qui suit le *elif* est vraie.

▶ Exercices

1 Complétez logiquement la ligne qui est dans la partie *else* de ce test :

```
if x<= 4 :
␣␣print("x est inférieur ou égal à 4")
else :
␣␣print("..........................................
```

2 Julien souhaite s'inscrire dans un club d'équitation. Le club lui propose les deux tarifs suivants :

• Tarif A : une cotisation annuelle de 80 € et chaque séance coûte ensuite 10,50 €.

• Tarif B : pas de cotisation annuelle, mais chaque séance coûte 17 €.

Pour prévoir quel tarif serait préférable suivant le nombre de séances qu'il choisira de faire, il a commencé à un script qui demande le nombre (entier) de séances, puis calcule le prix par chacun des tarifs. Terminez son script pour qu' affiche enfin quel est le tarif le plus avantageux.

```
n=int(input("Entrez le nombre de séances :")
tarifA = 80+10.5*n
tarifB = 17*n
if ..........................  :
␣␣print("..........................................
else :
␣␣print("..........................................
```

3 Dans une école de rugby, il y a quatre groupes :

• U8, pour les joueurs entre 8 ans inclus et 10 ans exclus ;

• U10, pour les joueurs entre 10 ans inclus et 12 ans exclus ;

• U12, pour les joueurs entre 12 ans inclus et 14 ans exclus ;

• U14, pour les joueurs entre 14 ans inclus et 16 ans exclus.

Complétez ce script, qui demande l'âge d'un joueur (nombre entier), et qui affiche ensuite à quel groupe ce joueur appartiendra.

```
age = int(input("Donner l'âge du joueur :"))
if age < 8 :
␣␣print("Trop jeune")
elif 8 <= age < 10 :
␣␣print("Groupe U8")
elif ..............................␣ :
␣␣print("..........................................
elif ..........................................
␣␣..........................................
elif ..........................................
␣␣..........................................
else :
␣␣print("Trop âgé")
```

5. Les conditions

▶ Cours

• Lorsqu'on veut tester l'égalité de deux quantités, il faut utiliser un double signe égal : `==`

Le signe égal seul ne peut être utilisé que pour affecter une valeur à une variable.
Mettre un seul signe égal dans une condition (après un `if`) donnera une erreur.

• Pour tester qu'une quantité n'est pas égale à une autre, on utilise un point d'exclamation suivie d'un signe égal : `!=`.

Cela ressemble à un égal barré, qui signifie « *n'est pas égal à* ».

• On peut mettre plusieurs conditions ensembles, en utilisant les mots `and` (et) ou `or` (ou).

Par exemple, pour tester si un nombre x est inférieur à 3 ou supérieur à 5, on mettra la condition `x<3 or x>5`.

• On peut aussi utiliser le mot `not` devant une condition, afin d'en prendre la négation.

Par exemple, pour tester si un nombre x est inférieur à 3 ou supérieur à 5, on peut mettrela condition `not 3<=x<=5` : ce test renverra True si la condtion `3<=x<=5` est False, donc dans le cas où x n'est pas entre 3 et 5.

6. La boucle bornée : *for ... :*

▶ Cours

• Très souvent, un programme doit réaliser une même chose plusieurs fois, qui peut être très grand. Plutôt que de devoir copier-coller de nombreuses fois les instructions à répéter, on utilise une **boucle**.

```
for variable in liste :
␣␣instruction1
␣␣instruction2
␣␣etc.
```

• La syntaxe d'une boucle *for* est montrée à gauche.

Comme pour une instruction conditionnelle *if*, la ligne qui commence par un *for* doit se finir par un deux-points.

Il faut aussi que les instructions que la boucle *for* va répéter plusieurs fois soient indentées (toutes avec le même décalage) par rapport au mot *for*.

• Après le mot *for* vient un nom de variable. Cette variable peut ne pas exister avant le *for*, auquel cas elle sera automatiquement créée.

• Après le nom de variable vient le mot *in*, qui est suivi d'une liste. Comme son nom l'indique, c'est une liste de valeurs, qui sont délimitées par des crochets et séparées par des virgues.

Sur le clavier, on obtient les crochets avec les touches Alt Gr 5 pour le crochet ouvrant, et Alt Gr) pour le crochet fermant.

Un exemple de liste pouvant apparaître après le mot *in* serait `[1,2,3,4,5,6,7,8,9,10]`. Ou bien, par exemple, `[2,3,5,7,11,13,17]`.

Une liste peut aussi contenir des mots, à condition de les mettre entre guillemets simples ou doubles (par exemple, le début de l'alphabet est représenté par la liste `['a', 'b', 'c', 'd', 'e', 'f']`.

Si on met dans une liste un mot non entouré de guillemets, Python considère que c'est un nom de variable, et met dans la liste la valeur de la variable à la place de son nom.

• Quand Python rencontre l'instruction `for variable in liste`, il fait prendre à la variable chacune des valeurs de la liste, et il exécute à chaque fois les intructions qui sont indentées par rapport au mot *for*.

• Python propose une fonction `range` qui génère de manière simple une liste de nombres.

1°) avec l'instruction `range(8)`, Python crée la liste `[0,1,2,3,4,5,6,7]`.

Une liste créée par `range(n)` commence à 0 et va jusqu'à $n-1$ de 1 en 1. Il y a alors n nombres entiers dans cette liste.

2°) avec `range(3,9)`, Python crée la liste `[3,4,5,6,7,8]` : de 3 à 8 par pas de 1 (le 9 n'est pas inclus dans la liste créée).

3°) avec `range(2,14,3)`, Python crée la liste allant de 2 à 14 par pas de 3. Il ne mettra de toute façon pas 14 dans la liste, même si on l'atteint bien en allant de 3 en 3 à partir de 2. On obtient donc la liste `[2,5,8,11]`

4°) avec `range(14,2,-3)`, Python créera la liste `[14,11,8,5]` : de 14 à 2 en allant de -3 en -3, mais 2 sera de toute façon exclu de la liste.

• On peut utiliser n'importe laquelle de ces formes de l'instruction `range` dans une boucle `for`.

Par exemple, `for i in range(4) :` fait que, dans les instructions du bloc qui suit le `for`, i prendra les valeurs 0, 1, 2 et 3.

▶ Exercices

1 Compléter le script ci-dessous pour qu'il affiche la table de multiplication de 13 :

```
for i in range(.....) :
␣␣print("13 x",i,"=", .....)
```

2 Le script ci-dessous calcule et affiche un nombre.

À quelle opération ce nombre correspond-il ?

```
s=0
for i in range(101) :
␣␣s = i + s
print(s)
```

3 Jean a placé 5 000 € sur un compte bancaire rémunéré à 2,5 % par an.

1. Compléter le script suivant pour qu'il calcule et affiche la somme qu'il aura sur son compte après 10 ans d'augmentation.

```
c = .....
for i in range(.....) :
␣␣c = ..........................
print(round(c,2))
```

Remarque : la fonction `round(c,2)` renvoie l'arrondi à 2 chiffres de c.

2. Modifier le script pour qu'il demande le nombre n d'années d'augmentation, puis calcule et affiche la somme sur le compte de Jean après ces n années.

7. La boucle non bornée : *while ... :*

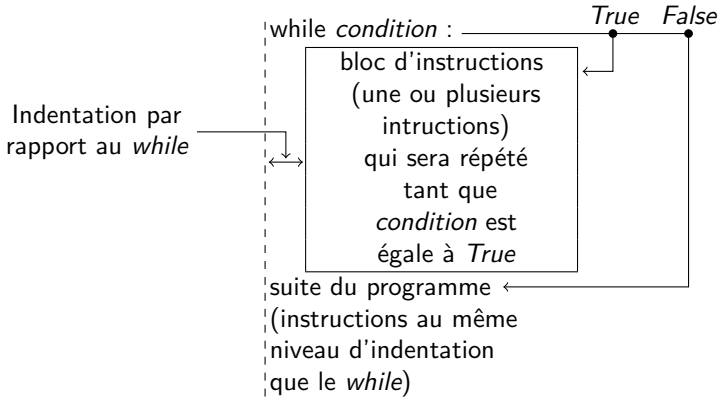
▶ Cours

• La boucle *while* est non bornée dans le sens où on ne sait pas a priori combien de fois elle va être exécutée.

• Après la boucle *while*, on met une condition (un test booléen, qui renvoie *True* ou *False*).

Si le test renvoie *True*, alors les instructions qui sont indentées après le *while* seront exécutées.

Sinon, ces instructions sont passées, et le programme se poursuit après le bloc qui est contenu dans la boucle *while* :



• On peut faire une boucle infinie avec *while True :* car dans ce cas la condition est *True*, dont la valeur est toujours *True*...

Pour sortir d'une telle boucle, on utilise l'instruction *break*.

▶ Exercices

1 Un jardinier souhaite creuser un puits dans son jardin. Le prix du forage dépend de la profondeur atteinte. Le premier mètre coûte 100 €, puis chaque mètre supplémentaire coûte 20 € (120 € pour aller à 2 m, 140 € pour aller à 3 m, etc.)

Pour ce forage, le budget du jardinier est de 700 €. Compléter le script suivant pour qu'il calcule et affiche la profondeur maximale atteinte.

```
profondeur = 1
prix = 700
while prix <= ..... :
    prix = prix + .....
    profondeur = profondeur + .....
print(".....
```

2 On peut générer des nombres aléatories à l'aide du module `random` et de la fonction `randint`. En mettant au début du script l'instruction `import random`, on peut plus loin donner l'instruction `a = random.randint(1,6)` qui affectera à la variable a un nombre aléatoire entre 1 inclus et 6 inclus.

Compléter ce script pour qu'il compte et affiche combien de fois il faut lancer deux dés pour obtenir 12 en faisant la somme des points des dés.

```
import random
somme = 0
nombre = 0
while somme ..... :
    de1 = randint(.....)
    de2 = .....
    somme = .....
    nombre = .....
print(".....
```

3 Réaliser le programme du « plus grand–plus petit » : le script commence par tirer un nombre entier au hasard entre 1 et 1000.

Puis, il demande à l'utilisateur d'entrer un nombre. Le script affiche « *trop grand* » si le nombre de l'utilisateur dépasse celui à deviner, « *trop petit* » sinon.

Ce processus doit se répéter jusqu'à ce que l'utilisateur devine le bon nombre.

Dans ce cas, le script affiche le nombre de propositions que le joueur a faites.

Variante : on peut imposer un nombre maximum de propositions, au-delà duquel le joueur perd le jeu.