NSI - Première

Python - 3 - Structures de contrôle

qkzk

Les structures de contrôle

Pour l'instant nos programmes s'exécutent de haut en bas sans pouvoir contrôler quoi que ce soit.

Nous allons introduire deux structures essentielles :

les conditions et les boucles

Mais avant ça :

L'indentation

indenter le code c'est le décaler de quelques espaces (généralement 4, parfois 2) depuis la gauche.

En python la structure d'un programme est *imposée* par l'indentation. Ne pas respecter l'indentation conduit à des erreurs du type `IndentationError` qu'il est généralement simple de repérer.

```
code sans indentation

    code mal indenté # provoquera une erreur
```

Strucure d'un bloc de code

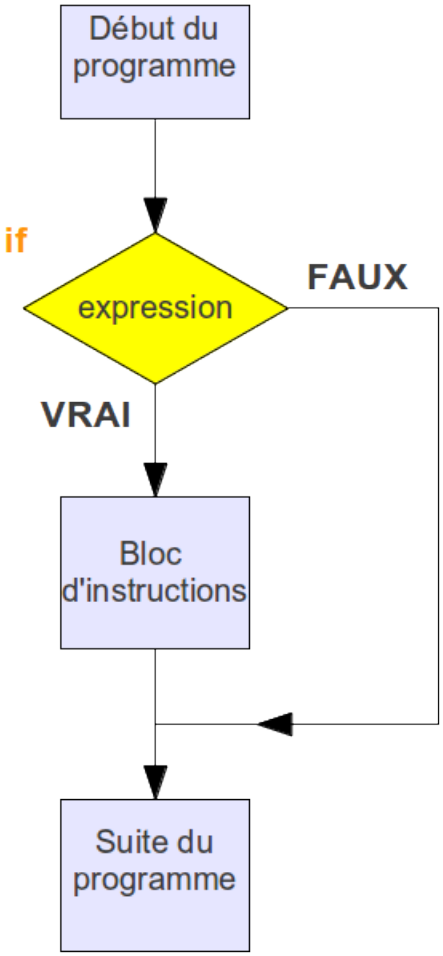
Tous les blocs de code ont la même syntaxe :

```
mot_cle expression:
    expression 1
    expression 2

fin du bloc
```

Conditions `if, elif, else`

Une condition décrit un bloc de code qui n'est exécuté que si le test est vérifié.



Par exemple :

Si j'ai plus de 10 de moyenne au bac, je suis diplômé :

```
moyenne = 3
if moyenne >= 10:                # ne pas oublier les :
    print("je suis diplomé")         # attention à l'indentation

#  suite du programme
```

Lorsqu'on exécute ce code, il ne se passe rien. La condition `moyenne >= 10` n'est pas respectée et la ligne `print("je suis displomé")` n'est jamais atteinte.

Ajoutons une ligne pour mieux comprendre le déroulé des instructions :

```
moyenne = 3
if moyenne >= 10:
    print("je suis diplômé")

print("le programme arrive ici dans tous les cas")
```

Exécutez ce programme, il vous affichera "le code arrive ici dans tous les cas" parce que cette ligne n'est pas indentée. Elle est donc en dehors du bloc `if`.

On peut toujours compléter un bloc `if` à l'aide d'un second bloc `else`, (*sinon*), qui ne sera exécuté que si la condition du `if` est fausse :

```
moyenne = 3
if moyenne >= 10:
    print("je suis diplomé")
else:
    print("j'ai raté le diplome")

print("le programme arrive ici dans tous les cas")
```

Cette fois l'exécution du programme conduit à deux affichages :

- "j'ai raté le diplome", parce que la condition du `if` est fausse, Python exécute le `else`
- "le code arrive arrive ici dans tous les cas" parce que ce bloc n'est pas indenté.

Reprenez cet exemple en changeant la valeur de la moyenne afin d'exécuter le if.

Remarquez que cette fois, le bloc `else` n'est plus exécuté.

C'est l'un ou l'autre.

`elif`

Entre `if` et `else` on peut insérer autant de bloc `elif condition:` que l'on souhaite.

Par exemple :

```
moyenne = 3
if moyenne >= 10:
    print("je suis diplome")
elif moyenne >= 8:
    print("je vais à l'oral de rattrapage")
else:
    print("j'ai raté le diplome")
print("le programme arrive ici dans tous les cas")
```

Lorsqu'on exécute ce code avec différentes valeurs de la moyenne (5, 9, 11 par exemple) on réalise qu'un des blocs `if, elif, else` est exécuté. Jamais deux.

Exercice 0

1. Que vaut la valeur finale de la variable `b` ?

```
a = 7
b = 12
if a > 5:
    b = b - 4
if b >= 10:
    b = b + 1
```

2. Que vaut la valeur finale de la variable `b` ?

```
a = 7
b = 12
if a > 5:
    b = b - 4
elif b >= 10:
    b = b + 1
```

3. Que vaut la valeur finale de la variable `b` ?

```
a = 7
b = 12
if a > 5:
    b = b - 4
else:
    b = b + 1
```

4. Que vaut la valeur finale de la variable `a` ?

```
a = 10
if a < 5:
    a = 20
elif a < 100:
    a = 500
else:
    a = 0
```

Exercice 1

Les codes suivants sont-ils valides ? (On ne demande pas ce qu'ils font) S'ils sont invalides, rectifiez les.

1. Code 1 :

```
a = 10
if a == 5:
a = 2
```

2. Code 2 :

```
a = 10
elif a == 5:
  a = 2
```

3. Code 3 :

```
a = 10
if a = 5:
  a == 2
```

4. Code 4 :

```
a = 10
if a == 5:
a = 2
```

Exercice 2

Complétez le programme "moyenne" afin d'afficher la mention obtenue par le candidat :

- Très bien (>= 16 de moyenne),
- Bien (>= 14 de moyenne),
- Assez Bien (>= 12 de moyenne),

Exercice 3

Un client de boite de nuit est décrit par trois variables : son genre ("Masculin" ou "Féminin"), sa tenue ("Bien sapé", "Mal sapé") et son portefeuille ("Épais", "Mince").

1. Écrire un programme qui affiche si le client peut entrer en respectant les conditions suivantes :

- Un client pauvre mais bien sapé peut entrer.
- Les clientes entrent toujours,
- Un client pauvre et mal sapé n'entre pas.

2. Testez le dans tous les cas.
3. Si ce n'est pas déjà fait, écrire ce programme à l'aide d'un bloc `if` et d'une bloc `else`

Exercice 4

Jean-Raoul a un hygiène de vie très stricte. Ils se pèse tous les matins.

- S'il pèse moins de 100 kilos, durant la journée, il ne boit que de la bière.
- S'il pèse entre 100 et 120 kilos, durant la journée, il ne boit que du vin.
- Si son poids dépasse 120 kilos, durant la journée il ne boit que de l'eau.

Quel athlète !

Ecrire un script qui demande sa masse et affiche le régime qu'il doit suivre.

Exemples :

```
Combien pèses-tu ? 140
Eau
Combien pèses-tu ? 95
Bière
Combien pèses-tu ? 105
Vin
```

Attention, `input` retourne toujours *une chaîne de caractères* du type `str`.
Pour en faire un entier (du type `int`) :

```
poids = int(input("Combien...")
```

Exercice 4

L'impôt sur le revenu de Groland est assez simple à calculer :

- un particulier qui gagne moins de 20.000€ de revenus par an doit verser 50% en impôt,
- entre 50.000 et 100.000€ on doit verser 30% aux impôts,
- au dela de 100.000€ de revenus, on doit verser 1€ d'impôt symbolique.

Ecrire un script qui demande à un utilisateur le montant de ses revenus annuels et affiche le montant des impots qu'il doit verser.

je précise aux plus naïfs que Groland n'existe pas...

Exercice 4 - Niveau première en mathématiques

Ecrire un script qui résout l'équation du second degré : $ax^2 + bx + c = 0$
Par exemple :

```
Résolution de l'équation du second degré : ax² + bx + c = 0
Coefficient a ? 1
Coefficient b ? -0.9
Coefficient c ? 0.056
Discriminant :  0.586
Deux solutions :
0.0672468158199
0.83275318418


Résolution de l'équation du second degré : ax² + bx + c = 0
```

```
Coefficient a ? 2
Coefficient b ? 1.5
Coefficient c ? 4
Discriminant :  -29.75
Il n'y a pas de solution.
```

Boucles

Une *boucle* est un bloc de code qui est répété plusieurs fois.

On distingue deux types de boucles, les boucles *bornées* (`for`) et les boucles *non bornées* (`while`).

Lorsqu'on sait à l'avance combien de fois on souhaite répéter un bloc on emploie une boucle bornée sinon une boucle non bornée.

Boucle bornée

En Python la boucle bornée s'écrit ainsi :

```
for variable in collection:
    bloc_de_la_boucle

fin_de_la_boucle
```

Notez bien la syntaxe : `for ... in ... :`

`variable` est une variable qui va *parcourir* `collection`, c'est à dire qu'elle prendra toutes les valeurs de la collection.

Les collections peuvent être des `list`, `tuple`, `str` etc.

Voyons deux exemples :

Afficher toutes les lettres d'un mot :

```
mot = "bonjour"
for lettre in mot:
  print(lettre)

print("fini !")
```

L'exécution de ce programme affiche :

```
b
o
n
j
o
u
r
fini !
```

On réalise bien que `lettre` a pri pour valeurs successives les caractères composant `mot` : `"b"`, `"o"` etc.

La fonction `range`

Autre exemple : afficher les entiers de 0 à 9.

Pour créer une collection d'entiers, Python dispose de la fonction `range`

C'est une fonction avancée que nous allons étudier pas à pas :

```
for nombre in range(10):
    print(nombre)

print("j'ai affiché les entiers de 0 à 9")
```

Dont l'exécution produit l'affichage :

```
0
1
2
...
9
j'ai affiché les entiers de 0 à 9
```

L'instruction `print(nombre)` a donc été exécutée 10 fois (il y a 10 entiers de 0 à 9)

Lorsqu'elle est utilisée avec un paramètre `n` (ici `n = 10`), `range` produit la collection des entiers jusqu'à `n - 1` inclus. La borne n'est jamais atteinte.

On n'est pas tenu d'utiliser ce paramètre, par exemple pour afficher 20 fois "Allez les bleus !", je peux l'écrire en 20 lignes :

```
print("Allez les bleus !")
print("Allez les bleus !")
print("Allez les bleus !")
print("Allez les bleus !")
... # encore 16 lignes identiques...
```

ou en deux lignes :

```
for k in range(20):
    print("Allez les bleus !")
```

C'est beaucoup plus lisible et on sait immédiatement combien de fois l'instruction sera répétée.

Calculons la somme des entiers entre 0 et 100.

Bien sûr on pourrait l'écrire à la main :

```
somme = 1 + 2 + 3 + ... + 100
```

Une très longue ligne serait nécessaire mais rien d'impossible.

On peut en écrire moins en utilisant une variable `somme`.

On veut parcourir les entiers jusque 100 inclus, il faudra utiliser `range(101)` !

```
somme = 0
for entier in range(101):
    somme = somme + entier
```

Vous apprendrez cette année en mathématiques à calculer ces sommes directement avec une formule très efficace.

`range` avec deux paramètres

Lorsqu'on donne deux paramètres entiers : `range(debut, fin)` parcourt les entiers de `debut` inclu jusque `fin` exclu.

Par exemple `range(5, 10)` va parcourir les entiers : 5, 6, 7, 8, et 9. Mais pas 10.

Exercice 5

Calculer la somme des entiers à trois chiffres (de 100 à 999).

`range` avec trois paramètres

Lorsqu'on utilise trois paramètres : `range(debut, fin, pas)`, on parcourt les entiers de `debut` inclu à `fin` eclu en avançant de `pas` en `pas` :

Collection des entiers pairs entre 100 et 199 : (100, 102, 104, ..., 198) : `range(100, 199, 2)`

Exercice 6

1. Calculer la somme des entiers impairs entre 1000 et 10000.
2. Calculer la somme des entiers divisibles par 3 entre 1000 et 10000.
3. Calculer la somme des entiers qui se terminent par 7 entre 654 et 1 million.
4. Compter le nombre d'entiers se terminant par deux zéros plus petits qu'un milliard à l'aide d'une boucle.

Combiner des blocs

Puisque l'indentation indique la strucutre d'un programme, on doit faire attention à la position qu'on donne à nos strucutres.

Les deux programmes suivants ne font pas la même chose :

```
nombre = 5
couleur = "bleu"

if nombre > 10:
    couleur = "rouge"

for lettre in couleur:
    print(lettre)
```

et

```
nombre = 5
couleur = "bleu"

if nombre > 10:
    couleur = "rouge"

    for lettre in couleur:
        print(lettre)
```

1. Dans le premier cas, la structure `for` est au niveau 0 d'indentation, elle sera exécutée dans tous les cas.

Il affiche ligne par ligne les lettres du mot "bleu".

2. Dans le second cas, la structure `for` est *dans le if*, elle ne sera exécutée que si la condition est vraie.

La condition est fausse et le bloc `if` n'est pas exécuté donc il ne se passe rien.

Exercice 7

1. La variable `lettre` est du type `str` et est de longueur égale à 1.

Proposer un booléen qui soit vrai si `lettre` est une voyelle (pensez à `in` !)

On va négliger les accents dans cet exercice.

2. En utilisant `for` et une condition bien placée, comptez les voyelles dans un mot.

Testez avec les mots "table", "exercice", "bonjour"

Exercice 8 - table de multiplication

1. `nombre = 5`. En une ligne supplémentaire écrire :

```
4 * 5 = 20
```

Le nombre 20 doit être obtenu par opération.

2. Utiliser ce principe pour écrire la table de 4.

```
4 * 0 = 0
4 * 1 = 4
4 * 2 = 8
4 * 3 = 12
4 * 4 = 16
4 * 5 = 20
4 * 6 = 24
4 * 7 = 28
4 * 8 = 32
4 * 9 = 36
4 * 10 = 40
```

3. En utilisant deux boucles l'une dans l'autre (on dit qu'elles sont imbriquées), afficher les tables de multiplications des entiers de 0 à 10 inclu.

Boucles non bornées : `while`

`while` qui signifie "tant que" permet de répéter un bloc tant qu'une condition est respectée :

```
while condition:
    bloc_repete_tant_que_condition_est_vrai

bloc_execute_apres
```

On emploie `while` lorsqu'on ne sait pas à l'avance combien de fois il sera nécessaire de répéter un bloc d'instruction.

Par exemple : "tant que je n'ai pas bon à l'exercice, je recommence".

Autre exemple, le casino :

Jean-Martin a 10€, il joue aux machines à sous (un vrai pigeon). Chaque mise lui coûte 1€ et il a une chance sur 1000 de gagner 10€ (quand je vous disais)...

Le programme suivant simule cette situation dramatique :

```
import random                       # fonctions simulant le hasard

capital = 10
while capital > 0:
    capital = capital - 1           # il mise 1 €
    if random.random() < 0.001:     # une chance sur mille
        capital = capital + 10      # il récupère 10 €
        print("waouh...")
    else:
        print("encore perdu...")

print("Jean-Martin a perdu tout son argent.")
```

Le prof de maths qui sommeille en moi vous signale que :

1. Ce programme termine toujours.
2. L'issue est toujours la même : Jean-Martin perd tout son argent.
3. Le seul moyen de gagner régulièrement aux jeux d'argent est de les organiser. Ce privilège est reservé à l'État.

Exercice 9

Modifier le programme précédent :

1. Créer une variable `compteur` valant 0 avant la boucle,
2. Augmenter `compteur` de 1 à chaque tour de la boucle,
3. Affichez la valeur de `compteur` une fois la boucle terminée.

Boucle infinie

On emploie régulièrement des boucles infinies. Il suffit d'écrire la condition `while True` pour qu'une boucle soit répétée indéfiniment.

Afin de ne pas saturer complètement votre processeur on va intrduire un délai d'une seconde entre deux affichages :

```
from time import sleep

while True:
  print("NSI")
  sleep(1)
```

- On importe la fonction `sleep` qui va mettre l'exécution en pause une seconde avec `sleep(1)`
- On répète indéfiniment l'affichage "NSI" toutes les secondes.

Pour arrêter le programme vous pouvez utiliser le raccourci clavier "Ctrl + C"

À quoi cela peut-il bien servir ?

Et bien c'est ce qu'on fait dans TOUS les logiciels... Pensez un instant à un jeu vidéo.

Lorsque vous arrêtez de jouer, ne serait-ce qu'une seconde, le programme ne s'arrête pas de tourner.

On peut résumer un jeu vidéo à ce schéma :

```
while True:
  lire_les_actions_du_joueur()
  mettre_le_jeu_a_jour()
  afficher_le_jeu_a_l_ecran()
```

L'instruction `break`

L'instruction `break` provoque une sortie immédiate d'une boucle `while` ou d'une boucle `for`.

Dans l'exemple suivant, l'expression `True` est toujours ... vraie : on a une boucle sans fin.

L'instruction `break` est donc le seul moyen de sortir de la boucle.

Affichage de l'heure courante

```
import time     # importation du module time
while True:
    # strftime() est une fonction du module time
    print('Heure courante ', time.strftime('%H:%M:%S'))
    quitter = input('Voulez-vous quitter le programme (o/n) ? ')
    if quitter == 'o':
        break
print("A bientôt")
```

```
>>>
Heure courante  14:25:12
Voulez-vous quitter le programme (o/n) ? n
Heure courante  14:25:20
```

```
Voulez-vous quitter le programme (o/n) ? o
A bientôt
```

Exercice 10

1. Se documenter sur la fonction `randint` du module `random`

```
import random
help(random.randint)
```

2. Écrire une boucle qui affiche 10 nombres aléatoires entre 1 et 100
3. La fonction `input(message)` affiche un message à l'écran pour l'utilisateur, celui-ci saisit au clavier une valeur et cette valeur est renvoyée par `input`.

`input` renvoie toujours une chaîne de caractère.

Pour convertir une chaîne de caractère en un entier on peut utiliser :

```
nombre = int(input("votre nombre : "))
```

Si l'utilisateur tape une valeur pouvant être transformée en entier comme "123", `nombre` vaudra 123.

Si l'utilsateur tape une valeur *ne pouvant pas être transformée en entier* comme "Marcel", Python va générer une erreur.

Écrire le script du jeu du plus ou moins suivant :

```
>>>
Le jeu consiste à deviner un nombre entre 1 et 100 :

--->   50
trop petit !
--->   75
trop petit !
--->   87
trop grand !
--->   81
trop petit !
--->   84
trop petit !
--->   85
Gagné !
```

4. Améliorer le jeu pour indiquer combien de coups ont été necessaires pour gagner.