Le contrôle du flux d'instructions

BENMANSOUR Asma

Avril 2018
1.0

Table des matières

Introduction

Dans les chapitres précédents, nous avons vu que l'activité essentielle d'un programmeur est la conception d'un programme permettant la résolution d'un problème spécifique. Or, pour résoudre un problème informatique, il faut effectuer une série d'actions suivant certain ordre précis. Le « chemin suivi par Python à travers un programme est appelé un flux d'exécution. Pour que amener Python a suivre un cheminement précis lors de l'exécution d'un programme il faut obéir aux règles de construction des instructions de contrôle de flux. Les structures de contrôle et les sont les groupes d'instructions qui déterminent l'ordre dans lequel les actions sont effectuées[1].

Un script Python est formé d'une suite d'instructions exécutées en séquence c'est-à-dire les unes après les autres de haut en bas. Chaque ligne d'instructions est formée d'une ou plusieurs lignes physiques. Cette exécution en séquence peut être éditée de façon à choisir tel ou tel partie du programme en fonction de la réalisation d'une ou plusieurs conditions c'est ce qu'on appelle les structures conditionnelles (if ...elif... else...) ou répéter des portions de code tant qu'une ou plusieurs conditions sont réalisées c'est ce qu'on appelle les structures répétitives (while .../for....). Nous allons étudier en détails ces deux structures: conditionnelles et répétitives tout au long ce chapitre.

Objectifs spécifiques du chapitre

A l'issu de ce chapitre l'apprenant sera capable de :

- Mémoriser la syntaxe d'écriture d'une structure conditionnelle sous toute ses forme (if +elif)ou (il+elif+..+else),etc...
- Mettre en pratique cette syntaxe lors de la résolution d'un problème posé en faisant ressortir le nombre de conditions impliqués c'est à dire le nombre de tests à effectuer.
- Comprendre et mémoriser les règles de construction d'une condition (expression logique)
- Évaluer le résultat d'une expression logique (True ou False)
- Mémoriser la syntaxe d'écriture de la boucle for et la boucle while
- Comprendre la différence entre la boucle for et la boucle while
- Sélectionnez et mettre en pratique la boucle adéquate à un problème posé

Notions de base:

II

Dans ce qui suit nous allons parler des principes de base concernant les instructions qui composent un programme :

1. Séquence d'instructions

Comme il a été précisé de nombreuses fois, les instructions d'un programme s'exécutent les unes après les autres, dans l'ordre où elles ont été écrites à l'intérieur du script. Cette affirmation peut vous paraître évidente à première vue sauf que l'expérience montre qu'un grand nombre d'erreurs sémantiques dans les programmes sont la conséquence d'une mauvaise disposition des instructions. Plus vous allez progresserez dans l'art de la programmation, plus vous allez observez ce fait et plus vous allez être extrêmement attentif à l'ordre dans lequel vous placez vos instructions les unes derrière les autres[1]. *p.18*

Exemple : Ordre des instructions dans un programme :

Par exemple, dans la séquence d'instructions suivantes, nous voulons permuter entre les valeurs des variables x et y :

```
1 >>> x,y=1,2
2 >>> c=x
3 >>> x=y
4 >>> y=c
5 >>> print(x,y)
6 2 1
7
```

si on permute entre les les lignes 3 et 4 c'est-à-dire je mets y=c puis y=x je vais obtenir 1 1 pour print(x, y)

Remarque

Python exécute normalement les instructions de la première à la dernière, sauf lorsqu'il rencontre une

instruction conditionnelle comme l'instruction *if* (décrite ci-après) ou une instruction répétitive comme l'instruction *for* ou *while* (décrite plus loin dans le chapitre).

2. Les instructions composées

Syntaxe

Une instruction composée se compose :

d'une ligne d'en-tête terminée par deux-points :

d'un bloc d'instructions indenté par rapport à la ligne d'en-tête

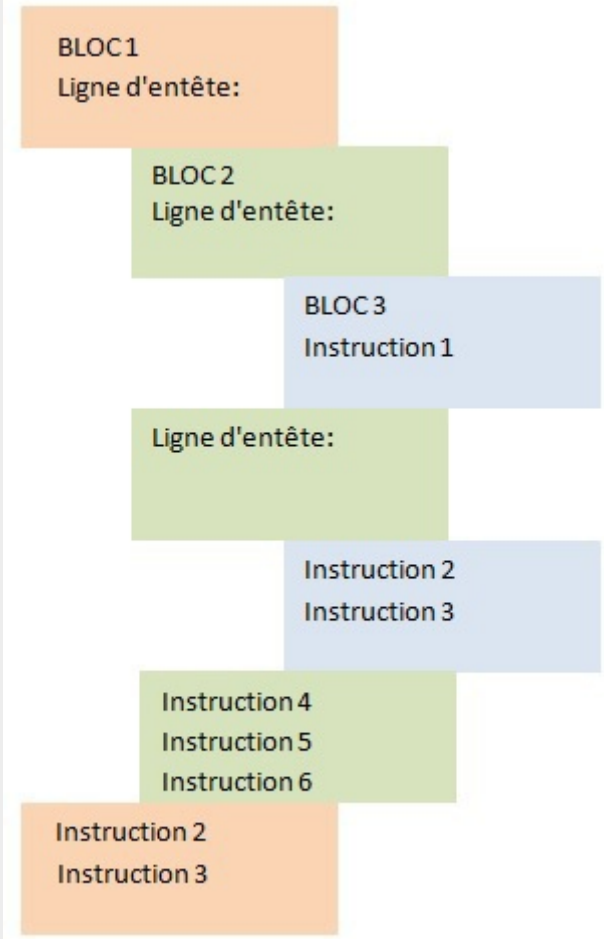
Remarque

Une ligne d'en tête inclut l'instruction conditionnelle plus condition suivi de : ou l'instruction répétitive for condition suivi de : ou while condition suivi de : p.18

Attention

Un bloc d'instructions veut dire une ou plusieurs instructions ayant une indentation décalé par apport à la ligne d'en tête et une indentation égale entre eux.

Toutes les instructions au même niveau d'indentation appartiennent au même bloc[3] p.18 (voir figure ci-dessous).



Indentations de différents blocs d'instructions

3. Les valeurs booléennes et les tests

On appelle « valeur booléenne » une valeur qui représente une évaluation logique et qui représente donc l'une des deux possibilités suivantes : vrai ou faux. Les valeurs booléennes sont le résultat de l'évaluation d'expressions logiques appelées conditions et elles servent à faire des choix dans un programme (effectuer telle action quand telle condition est réalisée). Voici quelques exemples d'expressions logiques à partir de deux variables a et b dont nous supposons qu'elles contiennent des valeurs entières [2].

Exemple : Expressions logiques simples et expressions logiques composées

Voici quelques expression logiques simple ;

```
>>> var1=20
>>> var1 > 50
False
>>> var2=15
>>> var2 <= 30
True
>>> var2< var1
```

```
8 True
9 >>> var1== var2
10 False
```

Voici le résultat d'exécution de quelques expressions logiques composées :

```
1 >>> var1>= var2 and var1 <= 60
2 True
3 >>> not (var1 < 5 or var2 > 10)
4 False
```

- Lorsqu'il y'a un or, Python retourne True si l'une des deux expression est vrai, si les deux retournent False Python va retourner False, puis le not() renvoie la valeur inverse.
- Toutes ces expressions représentent une valeur booléenne, elles sont soient vraies, soit fausses.

Structures conditionnelles

III

1. if...elif...else ?

Si nous voulons pouvoir écrire des applications véritablement utiles, il nous faut des techniques permettant d'orienter le déroulement du programme dans différentes directions, en fonction des circonstances rencontrées. Pour ce faire, nous devons disposer d'instructions capables de tester une certaine condition et de modifier le comportement du programme en conséquence[1].

Syntaxe : Voici la structure d'un bloc if, elif et else:

if expression_logique :

<espace>instruction 1

<espace>instruction 2

....

<espace>Instruction n

elif expression_logique :

<espace>instruction 1

<espace>instruction 2

....

<espace>Instruction n

...

else:

<espace>instruction 1

<espace>instruction 2

. .

<espace>Instruction n

Notez qu'après le if et le elif il y'a une expression logique suivi de deux point (:) ce que nous appellerons désormais une condition. L'instruction if permet de tester la validité de cette condition. Si la condition est vraie, alors l'instruction ou les n instructions que nous avons indentée après le « : » est/sont exécutées. Si la condition est fausse, rien ne se passe.

Le mot réservé elif, permet des instructions de tests successifs, ce qu'on appelle aussi l'alternative multiple. C'est pareil pour elif expression_logique:. l'instruction elif vient toujours après une instruction if expression_logique: suivi d'instructions indentées. Si la condition qui suit le if n'est pas vraie alors python test la validité de la condition(expression_logique) qui vient après elif. Si la condition est vraie, alors l'instruction ou les n instructions que nous avons indentée après le « : » est/sont exécutées. Si la condition est fausse alors python n'exécute pas les instructions et continue le cheminement du programme. Tout dépend du contenue du programme s'il y'a un autre elif expression_logique:. alors python va tester la condition (vrai, fausse). Si aucune des conditions que ce soit celle associé au if ou celles associées aux elif n'a été vrai alors python exécute les instructions indentées par apport au else.

Exemple : Nombre de elif utilisés dans le programme ?

Nous pouvons mettre autant de elif que nécessaire, par exemple pour un nombre positif x saisie au clavier, nous devons tester:

- si x<0 et afficher 'x est négatif' si la condition est vrai
- puis tester x>=0 et x<50 et afficher ' x est inférieur à 50' si la condition est vrai
- puis tester x>=50 et x<100 et afficher 'x est supérieur a 50 et inférieur à 100' si la condition est vrai
- puis tester x>=100 et afficher 'x est supérieur ou égale à 100'

Ci dessous le programme Python pour les différents tests :

```
>>> x=int(input("saisissez un entier"))
>>> if (x<0):
  print('x est négatif')
elif (x>=0 and x<50):
  print('x est un entier positif inférieur à 50')
elif (x>=50 and x<100):
  print('x est supérieur a 50 et  inférieur à 100')
else:
  print('x est supérieur ou égale à 100')

```

Dans cet exemple nous allons associer la première condition à un if. La deuxième condition à un elif. La troisième condition à un autre elif et terminer par un else ou les instructions du dernier cas lui seront associées.

Remarque : Utilisation facultative du elif dans certains cas !

l'utilisation de l'alternative elif dépend du nombre de conditions à traiter dans le programme. Par exemple si pour un entier x saisie au clavier, nous n'avons que deux conditions à tester: x positif et x négatif, la solution ne nécessitera que l'utilisation du if et du else.

Fondamental : Pas forcément de else pour le if !

Dans la structure if, l'alternative else est facultative. Dans ce cas, si l'expression logique est évaluée à faux, le bloc d'instruction sous le if est simplement sauté, l'exécution du programme reprenant à la première instruction après le bloc if (donc en fait la première instruction qui aura le même décalage que le if lui-même)[2].

2. Exercice : Mettre dans l'ordre les instructions d'une structure conditionnelle

[solution n°1 p.16]

Écrire un programme qui demande à l'utilisateur de saisir deux entiers non nul a et b. Le programme doit afficher si a est divisible par b ou si b est divisible par a ou si aucun des deux n'est divisible par l'autre. Les instructions ci dessous constituent la solution a cet exercice, mettez les dans l'ordre !

```
if a%b==0 :
```

```
print(a,"est divisible par",b)
```

```
elif b%a==0:
```

```
a=int(input("entrez un entier")) retour à la ligne b=int(input("entrez un entier"))
```

```
print(b,"est divisible par",a)
```

```
else:
```

```
print(a,"n'est pas divisible par",b,"et réciproquement")
```

Structures répétitives (les boucles)

IV

1. La boucle while

Nous venons de voir comment tester une expression logique avec la structure if. Grâce à la structure while, il est aussi possible d'exécuter un même bloc d'instructions tant qu'une condition est vérifiée, voici la syntaxe :

Syntaxe : Écriture de la boucle while

while condition :

<espace> Instruction 1

<espace> Instruction 2

...

<espace> Instruction n

Comme dans la structure if, la condition est d'abord évaluée et si elle est vraie, le bloc d'instructions est exécuté. Si la condition associée au while s'avère vrai après évaluation, python exécute le bloc d'instruction qui suit la condition ensuite, la condition est évaluée à nouveau et le bloc d'instructions est ré-exécuté tant que la condition est vrai. Quand la condition devient fausse, le contrôle du programme passe à l'instruction suivant le bloc while[2].Le bloc d'instruction est, ici également, défini par son indentation ou décalage.

Exemple : Utiliser le while pour écrire un programme avec contrôle de saisie sur la valeur de la note d'informatique

écrire un programme Python permettant la saisie d'un entier représentant la note d'informatique d'un étudiant , tant que la note n'est pas situé dans l'intervalle [0,20], le programme demande à l'utilisateur

de saisir une autre valeur:

```
note=float(input("saissez la note"))
saissez la note-3
>>> while (note<0 or note>20):
  note=float(input("saissez la note"))
saissez la note-6
saissez la note50
saissez la note13
note=float(input("saissez la note"))

```

Fondamental : Influence sur la condition

Il est très important, dans la structure while, que le bloc d'instructions ait une influence sur la condition, de manière à ce que si la condition est vraie au départ, elle puisse devenir fausse après un certain nombre de tours dans la boucle, sans quoi on se trouverait dans le cas d'une boucle infinie et d'un programme bloqué[2].

En tenant compte de l'exemple ci-dessous si on ne met pas l'instruction note=float(input("saissez la note")) aprés >>> while (note<0 or note>20): et que l'utilisateur saisit une valeur en dehors de l'intervalle [0,20], la condition note<0 or note>20 sera toujours vrai et le programme va boucler indéfiniment. L'instruction note=float(input("saissez la note")) permet l'arrêt de la boucle while une fois que l'utilisateur ai saisit une valeur dans l'intervalle [0,20].

2. Exercice : Contrôle de saisie !

[solution n°2 p.16]

Mettez dans l'ordre le programme permettant de faire la saisie d'un entier positif ou négatif par un utilisateur, la saisie s'arrête lorsque l'utilisateur entre le 0, le programme doit afficher le nombre d'entiers saisies pas l'utilisateur

nb_entier=nb_entier+1

while n !=0 :

n=int(input("entrez un entier non nul, tappez 0 pour sortir du programme"))

print("le nombre d'entiers saisis est", nb_entier)

nb_entier=0 retour à la ligne n=int(input("entrez un entier non nul, tappez 0 pour sortir du programme"))

3. La boucle for

En langage Python la structure for permet de parcourir un nombre fini d'éléments. La structure

générale de la boucle for est la suivante :

Syntaxe : Écriture de la boucle for

for élément in tableau :

<espace>instruction1

<espace>instruction2

...

<espace>instructionn

Les mots for et in sont les deux mots réservés utilisés dans cette structure. Dans le bloc d'instructions, le programme dispose de la variable représentant l'élément courant pris dans le tableau. Ci-dessus cette variable a été nommée élément. Le bloc d'instructions sera exécuté autant de fois qu'il y a de valeur dans le tableau. Chaque valeur étant affectée tour à tour dans la variable élément[2].

Exemple : Différentes possibilités pour le tableau

le tableau dans la définition de la syntaxe peut être géré de différentes façon :

```
1 for l in "bonjour":
2 print(l, end=" ") # b o n j o u r
3 for x in [20, 'b', 5.3]:
4 print(x, end=" ") # 20 b 5.3
5 for j in range(6):
6 print(j, end="_ ") # 0_1_2_3_4_5
7
```

1. Dans l'exemple 1 de l'exemple ci dessous on a utilisé une chaine de caractère "bonjour" à la place de tableau. l va prendre la valeur b de "bonjour" et le bloc d'instructions suivant la condition sera exécuté, puis lettre va prendre la valeur o de "bonjour" et le bloc d'instructions suivant la condition sera exécuté, et ainsi de suite jusqu'au r de "bonjour". Donc au total le bloc d'instructions suivant la condition sera exécuté sept fois puisqu'il y'a sept lettres dans "bonjour".
2. Python procède de la même façon pour l'exemple 2 de l'exemple ci-dessous, x va prendre les valeur du tableau une a une [20, 'b', 5.3]et donc le bloc d'instruction associé à la boucle sera exécuté trois fois.
3. Concernant le troisième exemple il inclut la fonction range(val1, val2) de la bibliothèque standard de Python. Cette fonction renvoie la « collection » des valeurs comprises entre val1 (inclus) et val2 (exclus). Par exemple range(1,10) renvoie la liste des 9 éléments suivants : [1,2,3,4,5,6,7,8,9]. Vous remarquerez que le 10 ne fait pas parti des valeurs du tableau renvoyé par le range, Python s'arrête au niveau de val2-1. Si val1 n'est pas précisé, la valeur par défaut est 0. Donc pour troisième exemple range(6) revoie [0,1,2,3,4,5] et le i va prendre chaque valeur du tableau en partant de 0 à 5, ainsi le bloc associé à la condition sera exécuté 6 fois.

Fondamental : Boucle for ou while ?

- On utilise le for lorsque le nombre d'exécutions du bloc d'instructions est connu à l'avance.

- On utilise le while lorsque le bloc d'instruction doit s'exécuter tant qu'une condition est vrai et donc on ne connaît pas à l'avance le nombre d'exécution du bloc d'instructions.

4. Exercice : Choix de la structure for ou while ?

[solution n°3 p.16]

Si vous deviez écrire un programme permettant la saisie d'un entier positif, la saisie s'arrete lorsque l'entier saisie est égale à zero, que choisiriez vous la boucle while ou la boucle for ou les deux ?

- O while
- O for
- O les deux

5. Exercice : Somme des entiers à l'aide de la boucle for

[solution n°4 p.17]

Mettez dans l'ordre les instructions composant le programme qui calcule et affiche la somme des entier de 1 à n sachant que n est saisie par l'utilisateur, n doit être saisie en premier, commencez par la saisie du n :

```
print("la somme est",somme)
```

```
somme=somme+i
```

```
somme=0
```

```
for i in range(1,n+1) :
```

```
n=int(input("entrer un entier"))
```

Solutions des exercices

> **Solution** n°1

Exercice p. 11

Écrire un programme qui demande à l'utilisateur de saisir deux entiers non nul a et b. Le programme doit afficher si a est divisible par b ou si b est divisible par a ou si aucun des deux n'est divisible par l'autre. Les instructions ci dessous constituent la solution a cet exercice, mettez les dans l'ordre !

```
a=int(input("entrez un entier")) retour à la ligne b=int(input("entrez un entier"))
if a%b==0 :
print(a,"est divisible par",b)
elif b%a==0:
print(b,"est divisible par",a)
else:
print(a,"n'est pas divisible par",b,"et réciproquement")
```

> **Solution** n°2

Exercice p. 13

Mettez dans l'ordre le programme permettant de faire la saisie d'un entier positif ou négatif par un utilisateur, la saisie s'arrête lorsque l'utilisateur entre le 0, le programme doit afficher le nombre d'entiers saisies pas l'utilisateur

```
nb_entier=0 retour à la ligne n=int(input("entrez un entier non nul, tappez 0 pour sortir du programme"))
while n !=0 :
nb_entier=nb_entier+1
n=int(input("entrez un entier non nul, tappez 0 pour sortir du programme"))
print("le nombre d'entiers saisis est", nb_entier)
```

> **Solution** n°3

Exercice p. 15

Si vous deviez écrire un programme permettant la saisie d'un entier positif, la saisie s'arrete lorsque l'entier saisie est égale à zero, que choisiriez vous la boucle while ou la boucle for ou les deux ?

- ◉ while
- ○ for
- ○ les deux

> **Solution** n°4

Exercice p. 15

Mettez dans l'ordre les instructions composant le programme qui calcule et affiche la somme des entier de 1 à n sachant que n est saisie par l'utilisateur, n doit être saisie en premier, commencez par la saisie du n :

```
n=int(input("entrer un entier"))
somme=0
for i in range(1,n+1) :
somme=somme+i
print("la somme est",somme)
```

Bibliographie

IV

[1] Swinnen Gérard, 02/02/2012, "Apprendre à programmer avec python 3", (3e édition), Eyrolles, 435 p, Noire

[2] Jachym, Marc, "Cours de Programmation avec le langage Python: Niveau débutant en programmation",Licence professionnelle – Métrologie dimensionnelle et qualité IUT de St Denis, Université Paris 13.

[3] Cordeau, Bob, Introduction à Python 3, version 2.71828.