Chap. 1 Introduction à la programmation Python Thème 1

Instructions

Sommaire du chapitre 1

Les **instructions** sont les briques de base d'un programme : un programme est une suite d'instructions. Chaque instruction est écrite sur une ligne. L'interprète Python les exécute l'une après l'autre, dans l'ordre du fichier. Nous avons déjà vu trois types d'instructions :

- **Affectation** : `variable = expression`, par exemple `delta = b**2 - 4*a*c`
- **Appel de fonction** : `fonction(paramètres)`, par exemple `print("bonjour")`
- **Importation d'une bibliothèque** : `from bibliotheque import *`, par exemple `from math import *`

Le langage Python a trois autres types d'instructions : les **conditionnelles**, les **boucles** et les **définitions de fonctions**.

- **Conditionnelle** : Les instructions conditionnelles permettent, en fonction du résultat de l'évaluation d'une expression booléenne, d'exécuter une certaine portion de code plutôt qu'une autre
- **Boucle** : Pour répéter plusieurs fois la même opération on utilise souvent une boucle
- **Définition de fonction** : L'écriture de fonctions rend un programme plus simple à comprendre, plus facile à vérifier, et plus facile à modifier

I Conditionnelles et blocs

La **conditionnelle** est une instruction qui permet d'exécuter du code selon qu'une condition est remplie ou non. Par exemple, pour faire reculer la tortue de 10 pas et la faire tourner à gauche de 20 degrés si une certaine variable `x` contient une valeur négative :

```
1 if x < 0 :
2     backward(10)
3     left(20)
```

L'instruction `if` est suivie d'une expression booléenne et du caractère `:`. Les instructions à exécuter si la condition est satisfaite sont *indentées*, c'est-à-dire qu'elles sont décalées par rapport au `if` en insérant des espaces. Une telle séquence d'instructions, toutes *indentées* du même

nombre d'espaces, est appelée un **bloc**. C'est ainsi que le langage Python différencie la séquence d'instructions à effectuer quand la condition est remplie, par opposition à la suite du code qui sera exécutée dans tous les cas :

```
if x < 0 :
    backward(10)  # exécutée si x < 0
    left(20)      # exécutée si x < 0
forward(10)       # exécutée dans tous les cas
```

La conditionnelle peut aussi avoir une branche `else:` pour exécuter un bloc si la condition n'est pas remplie, c'est-à-dire si le résultat de l'expression qui suit `if` est `False`. Ici, la tortue recule de 10 et tourne à gauche si `x` est négatif, sinon elle avance de 10 pas.

```
if x < 0 :          # "Si x est négatif ..."
    backward(10)
    left(20)
else :              # "sinon, ..."
    forward(10)
```

Plusieurs tests peuvent être enchaînées grâce au mot-clé `elif`, abbréviation de « else if ». Ici la tortue tourne à gauche si `x` est négatif, elle tourne à droite si `x` est supérieur à 100, et elle avance de 10 dans les autres cas.

```
if x < 0 :          # "Si x est négatif ..."
    left(20)
elif x > 100 :      # "sinon, si x > 100 ..."
    right(20)
else :               # "sinon, ..."
    forward(10)
```

Notons que Python cesse d'évaluer la véracité des conditions dès lors que l'une d'entre elle est `True`

```
if x < 0 :
    print('négatif')
elif x < -100 :   # condition jamais évaluées
    print('très négatif')
```

Enfin, les conditionnelles peuvent être **imbriquées** :

```
if x < 0 :
    if x % 2 == 0 :
        print('pair et négatif')
    else :
        print('impair et négatif')
else :
    print("positif, et c'est tout ce qui m'intéresse")
```

II Boucles

Les boucles permettent de répéter des blocs d'instructions. Python propose deux types de boucles : les **boucles bornées** (`for`) et les **boucles non bornées** (`while`).

II.a Boucles bornées

Une **boucle bornée** (ou boucle `for`, *« pour »* en français) permet d'exécuter un bloc d'instructions, appelé **corps de boucle**, un nombre prédéfini de fois. La forme la plus courante utilise la fonction `range(n)` pour exécuter la boucle `n` fois. Dans l'exemple suivant, la tortue tourne, puis avance aléatoirement 100 fois de suite :

```
for i in range(100) :       # Exécuter 100 fois le bloc suivant
    left(360.0*random())    # Tourner d'un angle aléatoire
    forward(40*random())    # Avancer d'une distance aléatoire
```

A chaque **tour de boucle**, c'est-à-dire à chaque exécution du bloc, la variable indiquée après `for` augmente de 1, en partant de 0. Dans l'exemple suivant, la tortue avance de plus en plus à chaque tour de boucle, en tournant du même angle : elle décrit une spirale.

```
for i in range(100) :       # Exécuter 100 fois le bloc suivant
    left(10)                # Tourner d'un angle fixe
    forward(i)              # Avancer d'une distance croissante
```

Il peut sembler curieux de commencer à compter à partir de 0 plutôt que 1, mais nous verrons que cette norme en programmation est en réalité plus commode en pratique.

Remarques

La fonction `range` peut être appelée avec deux arguments : `range(a, b)` énumère les entiers de `a` à `b-1` inclus.
`range(b)` est donc équivalent à `range(a, b)`.

II.b Boucles non bornées

La **boucle non bornée** (ou boucle `while`, *« tant que »* en français) permet d'exécuter un bloc d'instruction tant qu'une expression booléenne est vraie. Dans l'exemple qui suit, on programme à nouveau une marche aléatoire de la tortue mais au lieu de fixer le nombre de pas, on s'arrête lorsque la position de la tortue sort d'un carré de 400 pas autour du centre de la fenêtre. Les fonctions `xcor()` et `ycor()` donnent la position de la tortue, et la fonction `abs(x)` calcule la valeur absolue de `x`.

```
while abs(xcor()) < 400 and abs(ycor()) < 400 :
    left(360.0*random())
    forward(10 + 40*random())
```

Selon les nombres aléatoires, la tortue peut s'arrêter au bout de quelques pas, ou bien errer très longuement, peut-être pour toujours !

Remarques

L'exécution d'un programme implique souvent la réalisation d'un grand nombre d'instructions. Cependant, un programme dont l'exécution implique la réalisation d'un million d'opérations n'est pas forcément constitué d'un million de lignes. On aura communément un programme de taille modeste, dont certains fragments sont exécutés un grand nombre de fois. Les boucles sont un des moyens de réaliser de telles répétitions.

II.c Lien entre boucle while et boucle for

Ce programme

```
i = 0
while i < n :
    print(i)
    i = i + 1
print("Fin")
```

et ce programme

```
for i in range(n)
    print(i)
print("Fin")
```

ont le même comportement. On privilégie tant que possible la boucle `for`.

Attention : Un programme doit toujours s'arrêter ! !

L'arrêt des itérations d'une boucle `while` étant conditionné à l'échec d'un test, il peut arriver parfois que le programme ne s'arrête jamais. Dans ce cas, il faudra intervenir directement sur l'interprète pour forcer à s'interrompre.

Exemple :

```
while i != 0 :
    print(i)
    i = i - 1
```

Lorsque la valeur initiale de `i` est un entier positif ou nul, ce programme affiche toutes les valeurs entières de `i` jusqu'à 1. Cependant si la valeur initiale est négative, le programme ne cessera d'afficher des nombres négatifs de plus en plus profonds, sans jamais atteindre la valeur 0. Et si la valeur initiale est 2.5, le programme ne s'arrêtera pas à condition qu'on remplace le test `i != 0` par `i > 0`.

```
while i > 0 :
    print(i)
```

La valeur de `i` n'est jamais modifiée, on dit que le programme diverge.

L'instruction `break` est une instruction spéciale qui ne peut être écrite qu'à l'intérieur d'une boucle qui provoque l'interruption immédiate d'une boucle.

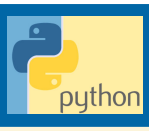
Recapitulatif

SI

— Si la condition est vraie (`True`), la ou les instruction(s) indentée(s) après le test sont exécutées.

— Si la condition est fausse (`False`), elles ne sont pas exécutées.

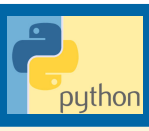
```
if condition :
# ne pas oublier le signe de ponctuation ':'
    bloc_instructions  # attention à l'indentation
# suite du programme
```



SI ALORS

On peut également préciser des instructions à effectuer si la condition est fausse, à l'aide de l'instruction `else` :

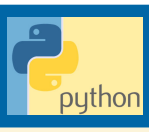
```
if condition :
    bloc_instructions_1    # attention à l'indentation
else :                 # else est au même niveau que if
    bloc_instructions_2    # attention à l'indentation
# suite du programme
```



SI ALORS SINON

L'instruction `elif` est équivalente à `else if`.

```
if condition 1 :
    bloc_instructions_1
elif condition 2 :
    bloc_instructions_2
elif condition 3 :
    bloc_instructions_3# ici deux instructions elif, mais il
   n'y a pas de limitation
...
else :
    bloc_instructions
# suite du programme
```



POUR

Pour exécuter `n` fois la ou les instruction(s) indentée(s) d'un bloc d'instructions, à l'aide d'une variable entière `variable` allant de `0` à `n-1` :

```
for variable in range(n) :
    bloc_instructions
```

Si la variable varie d'un entier `m` à `n-1` (`n>m`), on remplace la première ligne par :

```
for variable in range(m,n) :
    bloc_instructions
```

TANT QUE

Pour exécuter en boucle la ou les instruction(s) indentée(s) du bloc d'instructions tant que la condition est vraie (`True`), on écrit :

```
while condition :
    bloc_instructions
```