I11: Chapitre 2
Structures de controle

Nicolas Méloni

Licence 1: 1er semestre
(2018/2019)

Séquence d'instructions

- Les instructions d'un script s'exéctutent les unes après les autres
- L'ordre d'écriture détermine l'ordre d'exécution
- Modifier l'ordre relatif de deux instructions peut complètement changer la signification d'un programme

```
#Script 1
a,b = 1,2
a = b
b = a
print(a,b)

>>>
2 2
```

```
#Script 2
a,b = 1,2
b = a
a = b
print(a,b)

>>>
1 1
```

Séquence d'instructions

- Il est nécessaire de pouvoir perturber l'ordre d'exécution pour écrire des '*vrais*' programmes
- Il existe deux grandes catégories de controleurs de flux :
  - les opérateurs de sélection (**if-elif-else** en Python)
  - les opérateurs de répétition (**while-for** en Python)

Opérateur de sélection : l'instruction if

Syntaxe

- ```
  if condition:
      instruction1
      ...
      instructionN
  ```
- `condition` est une donnée de type **booléen**
- si `condition` est vraie, les instructions 1 à N sont exécutées, sinon elles sont ignorées

```
#Script 1
a = 10
if a > 5:
  a = a+5
  a = a*2
print(a)

>>>
30
```

```
#Script 2
a = 4
if a > 5:
  a = a+5
  a = a*2
print(a)

>>>
4
```
```

La structure if-elif-else

Syntaxe

```
if condition1:
   sequence-d-instructions1
elif condition2:
   sequence-d-instructions2
elif condition3:
   sequence-d-instructions3
...
elif conditionN:
   sequence-d-instructionsN
else:
   sequence-d-instructionsM
```

- Une condition est testées seulement si toutes les précédentes valent `faux`
- Une seule des séquences d'instructions sera exécutée !

La structure if-elif-else

Exemples

```
#Script 1
a = 10
if a >= 10:
   print(">= 10")
elif a<10 and a>0:
   print("0< <10")
else:
   print("<0"

>>>
>= 10
```

```
#Script 2
a = 5
if a >= 10:
   print(">= 10")
elif a<10 and a>0:
   print("0< <10")
else:
   print("<0"

>>>
0< <10
```

```
#Script 3
a = -2
if a >= 10:
   print(">= 10")
elif a<10 and a>0:
   print("0< <10")
else:
   print("<0"

>>>
<0
```

La structure if-elif-else

Instructions composées

Une instruction composée est constituée :

- d'une ligne d'entête terminée par :
- d'un bloc d'instructions identé par rapport à la ligne d'entête

Toutes les instructions de même niveau d'indentation font partie du même bloc.

```
a = float(input()) # bloc 1
b = float(input()) # bloc 1
if a >= b:         # bloc 1
    c = a-b          # bloc 2
    if c == 0:       # bloc 2
        print(c)       # bloc 3
else:              # bloc 1
    c = b-a          # bloc 4
    print(c)         # bloc 4
```

La structure if-elif-else

- Le niveau d'indentation d'une instruction peut changer la sens d'un programme.

```
#Script 1
a,b = 1,2
if a > b:
    a = a+b
    b = b+1
print(a,b)

>>>
1 2
```

```
#Script 2
a,b = 1,2
if a > b:
    a = a+b
b = b+1
print(a,b)

>>>
1 3
```

Exemples

1. Écrire un script qui affiche le plus petit de deux entiers saisis au clavier.
2. Écrire un script qui permet de résoudre l'équation

$$x \in \mathbb{R}, ax + b = 0$$

où a et b sont deux réels saisis au clavier.

Opérateur de répétition : l'instruction while

Syntaxe

- ```
  while condition:
      instruction1
      ...
      instructionN
  ```
- `condition` est une donnée de type **booléen**
- si `condition` est vraie, les instructions 1 à N sont exécutées,
- une fois l'instruction N terminée, si `condition` est toujours vraie, les instructions 1 à N sont à nouveau exécutées etc
  ```

Opérateur de répétition : l'instruction **while**

Remarques

- `condition` s'appelle le **critère de continuation** de la boucle
- les variables composant `condition` doivent toutes avoir une valeur lors de sa première évaluation
- il faut s'assurer que la boucle se termine ! ! !

Opérateur de répétition : l'instruction while

Exemples

```
#Script 1
a = 5
while a > 0:
  a = a-1
  print(a)

>>>
4
3
2
1
0
```

```
#Script 2
a = 5
while a < 3:
  a = a+1
  print(a)

>>>
```

Table de valeurs

- La table de valeurs permet de suivre les modifications des variables ou expression au cours de l'exécution d'un script.

Exemple

```
#Script 1
a = 3
while a > 0:
  a = a-1
  print(a)
```

a	a > 0	Écran
3	True	
2		2
	True	
1		1
	True	
0		0
	False	

Exemples d'itérations

- Écrire un script qui affiche tous les entiers entre 1 et 100.
- Écrire un script qui affiche la somme des entiers entre 1 et 100.
- Écrire un script qui calcule la somme des entiers saisis par l'utilisateur, la saisie s'arrête quand l'utilisateur saisit 0.

Exemples d'itérations

- Écrire un script qui affiche tous les entiers entre 1 et 100.

```
# entiers de 1 a 100
n = 1
while n <= 100:
  print(n)
  n = n+1
```

Exemples d'itérations

- Écrire un script qui affiche la somme des entiers entre 1 et 100.

```
# entiers de 1 a 100
n = 1
somme = 0
while n <= 100:
  somme = somme + n
  n = n+1
print(somme)
```

Exemples d'itérations

- Écrire un script qui calcule la somme des entiers saisis par l'utilisateur, la saisie s'arrête quand l'utilisateur saisit 0.

```
# saisie a la volee (1)
a = 1
somme = 0
while a != 0:
  a = int(input())
  somme = somme+a
print(somme)
```

```
# saisie a la volee (2)
continuer = True
somme = 0
while continuer:
  a = int(input())
  somme = somme+a
  continuer = (a!=0)
print(somme)
```

Boucles imbriquées

- Répéter un bloc d'instructions contenant lui même une boucle `while`
  - Écrire un script qui affiche la table de multiplication d'un entier n saisit au clavier
  - Modifier le script pour qu'il affiche toutes les tables des entiers entre 1 et 10

Boucles imbriquées

```
# table de n
n = int(input())
i = 1
while i <= 10:
    print(n,'*',i,'=',n*i)
    i=i+1
```

```
# table des nombres
# entre 1 et 10
n = 1
while n <= 10:
    i = 1
    while i <= 10:
        print(n,'*',i,'=',n*i)
        i=i+1
    n = n+1
```

Liste des nombres premiers

- On dit que entier $n \geq 2$ est premier s'il n'est : que par 1 et par lui-même.
- On veut afficher tous les nombres premiers inférieurs à un entier n saisit au clavier :
  - Écrire un script permettant de savoir si un entier n est premier
  - Répéter le bloc de code précédent pour tous les entiers entre 2 et n.

Liste des nombres premiers

```
# n est-il premier ?
div = 2
est_premier == True
while div < n:
    if n % 2 == 0:
        est_premier = False
    div = div+1
```

```
# Liste des nombres
# premiers
n_max = int(input())
n = 2
while n <= n_max:
    div = 2
    est_premier == True
    while div < n:
        if n % div == 0:
            est_premier = False
        div = div+1
    if est_premier == True:
        print(n)
    n = n+1
```