Avignon Université - Culture numérique et code

CM - Contrôle du flux d'exécution

14/02/2020

Le "chemin" suivi par Python à travers un programme est appelé un *flux d'exécution*, et les constructions qui le **modifient** sont appelées des *instructions de contrôle de flux*.

Séquence d'instructions

Sauf mention explicite, 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.

In [1]:

```
# Exemple
x = 10
y = 15
z = x + y
print("Le résultat de x+y cest :", z)
```

```
Le résultat de x+y cest : 25
```

In [2]:

```
# Exemple
u = 10
w = u + v
# ATTENTION !!!!
# La variable "v" est crée que dans l'instruction v=15,
#  donc elle n'existe pas pendant l'assignation "w = u + v"
v = 15
print(w)
```

```
---------------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
 last)
<ipython-input-2-f2c6c6c4b529> in <module>
      1 # Exemple
      2 u = 10
----> 3 w = u + v
      4 # ATTENTION !!!!
      5 # La variable "v" est crée que dans l'instruction v=15,

NameError: name 'v' is not defined
```

In [3]:

```
# Exemple
x, y = 10, 15
y = x
x = y
print(x, y)
```

```
10 10
```

In [4]:

```
# Exemple
x, y = 10, 15
x = y
y = x
print(x, y)
```

```
15 15
```

Exécution conditionnelle

Cependant, il est nécessaire d'être capable d'aiguiller le déroulement du programme dans différentes directions à partir 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 : instruction `if` .

```
if CONDITION :
    exécuter si CONDITION est vrai
```

Si la condition est vraie, alors l'instruction que nous avons *indentée* après le `:` est exécutée. Par contre, **si la condition est fausee, rien ne se passe.**

In [5]:

```
print("Bonjour")
if 30 > 10 :
    print("C'est vrai !")
print("Au revoir")
```

```
Bonjour
C'est vrai !
Au revoir
```

In [6]:

```
print("Bonjour")
if 10 > 30 :
    print("C'est vrai !")
print("Au revoir")
```

```
Bonjour
Au revoir
```

In [7]:

```
print("Bonjour")
x = 10
y = 10
if x == y :
    print("C'est vrai, x est y sont égales !")
print("Au revoir")
```

```
Bonjour
C'est vrai, x est y sont égales !
Au revoir
```

Opérateurs de comparaison

Opérateurs	Explication
x == y	x est égal à y
x != y	x est différent de y
x > y	x plus grand que y
x < y	x plus petit que y
x >= y	x plus grand que, ou égal à y
x <= y	x plus petit que, ou égal à y

> **Exercice 1 :** Tester les opérataurs de comparaison en utilisant l'instruction **if**

In [8]:

```
num = 11

if num % 2 == 0 :
    print("Le numéro ", num, "est PAIR.")

if num % 2 != 0 :
    print("Le numéro ", num, "est IMPAIR.")
```

```
Le numéro  11 est IMPAIR.
```

Et dans le cas contraire ?

```
if CONDITION :
    exécuter si CONDITION est vrai
else :
    exécuter si CONDITION est faux
```

Si la condition est vraie, alors l'instruction que nous avons *indentée* après `CONDITION :` est exécutée. Par contre, si la condition est fausee, l'instruction que nous avons *indentée* après `else :` est exécutée.

In [10]:

```
if 30 > 10 :
    print("C'est vrai !")
else:
    print("C'est faux !")
```

```
C'est vrai !
```

In [11]:

```
if 10 > 30 :
    print("C'est vrai !")
else:
    print("C'est faux !")
```

```
C'est faux !
```

In [12]:

```
num = 12

if num % 2 == 0 :
    print("Le numéro ", num, "est PAIR.")
else :
    print("Le numéro ", num, "est IMPAIR.")
```

```
Le numéro  12 est PAIR.
```

> **Exercice 2 :** Écrire un programe qui demande une année de naissance et qui affiche "*Desolée : (tu ne peux pas boire, tu as moins de 18 ans*" si l'age de l'utilisateur est plus petit que 18 ans ; dans le cas contraire le programe doit afficher "*Ok, tu as quand même 18 ans !*".

> **Exercice 3 :** Modifier l'**Exercice 2** à partir des trois conditions suivantes:

1. age < 18 : "*Desolée :(tu ne peux pas boire, tu as moins de 18 ans*"
2. age = 18 : "*Tu est dans la limite, mais ça va*"
3. age > 18 : "*Ok, tu as plus de 18 ans !*"

Instructions composées - Blocs d'instructions

Sous Python, les instructions composées ont toujours la même structure : une ligne d'en-tête terminée par `:` , suivie d'une ou de plusieurs instructions indentées sous cette ligne d'en-tête.

In [13]:

```
prix = 200

if prix >= 200 :
    print("Remise égale à 20% !")
    remise = 0.20
    prix_remise = prix * remise
    prix_final = prix - prix_remise
else :
    print("Pas de remise :(")
    prix_final = prix

print("Prix de base :", prix)
print("Prix final :", prix_final)
```

```
Remise égale à 20% !
Prix de base : 200
Prix final : 160.0
```

Instructions imbriquées

Il est possible d'imbriquer les unes dans les autres plusieurs instructions composées, de manière à réaliser des structures de décision complexes.

In [14]:

```
embranchement = "vertébrés"
classe = "mammifères"
#classe = "oiseaux"
ordre = "carnivores"
famille = "félins"
#famille = "canidés"


if embranchement == "vertébrés":
    if classe == "mammifères":
        if ordre == "carnivores":
            if famille == "félins":
                print("C'est peut-être un chat.")
                print("Ou un lion.")

            else :
                print("Bon, ce n'est pas un félin main en tous cas il est carnivore.

    elif classe == "oiseaux":
        print("C'est peut-être un canari.")
else :
    print("Un escargot peut-être ?")

print("Hmmm... la classification des animaux est complexe.")
```

```
C'est peut-être un chat.
Ou un lion.
Hmmm... la classification des animaux est complexe.
```

Instructions répétitives

L'une des tâches que les machines font le mieux est la répétition de tâches identiques.

In [15]:

```
annee = 1991
annee_min = 1993

if annee < annee_min :
    print("La valeur de la variable 'annee' doit être égale ou plus grand que", anne
    print("Actuallment la valeur de la variable 'annee' est :", annee)
    annee += 1
else:
    print("La valeur de la variable 'annee' est", annee, ", qui est plus grand que o

if annee < annee_min :
    print("La valeur de la variable 'annee' doit être égale ou plus grand que", anne
    print("Actuallment la valeur de la variable 'annee' est :", annee)
    annee += 1
else:
    print("La valeur de la variable 'annee' est", annee, ", qui est plus grand que o

if annee < annee_min :
    print("La valeur de la variable 'annee' doit être égale ou plus grand que", anne
    print("Actuallment la valeur de la variable 'annee' est :", annee)
    annee += 1
else:
    print("La valeur de la variable 'annee' est", annee, ", qui est plus grand que o
```

```
La valeur de la variable 'annee' doit être égale ou plus grand que 199
3
Actuallment la valeur de la variable 'annee' est : 1991
La valeur de la variable 'annee' doit être égale ou plus grand que 199
3
Actuallment la valeur de la variable 'annee' est : 1992
La valeur de la variable 'annee' est 1993 , qui est plus grand que ou
égale à 1993 !!!
```

Boucle while

Le mot `while` signifie "tant que". Cette instruction indique à Python qu'il lui faut *répéter continuellement le bloc d'instructions qui suit, tant que* la condition reste vrai.

```
while CONDITION :
    exécuter pendant que CONDITION est vrai
```

Si la condition est d'abord fausee, rien ne se passe.

In [16]:

```
a = 0
table = 7

print("Bonjour")
print("La table de", table)

while a <= 10 :
    resultat = table * a
    print(table, "x", a, "=", resultat)
    a = a + 1
    # a += 1

print("Au revoir")
```

```
Bonjour
La table de 7
7 x 0 = 0
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70
Au revoir
```

> **Exercice 4 :** Modifier le premier exemple d'**"Instructions répétitives"** pour avoir la même sortie en utlisant un boucle while.

Boucle for - in

Le boucle `for - in` travaille différent au boucle `while` . Dans ce cas, le bloc d'instructions se répète "pour chaque" élément d'une liste.

```
for VARIABLE in LISTE :
    exécuter pour chaque élément dans LISTE
```

In [17]:

```
ma_liste = [5, 6, 7, 8, 9, 10]

print("Premier élément de ma liste :", ma_liste[0])
print("Deuxième élément de ma liste :", ma_liste[1])
print("Dernier élément de ma liste :", ma_liste[-1])
```

```
Premier élément de ma liste : 5
Deuxième élément de ma liste : 6
Dernier élément de ma liste : 10
```

In [18]:

```
ma_liste = [5, 6, 7, 8, 9, 10]

print("Bonjour")

for x in ma_liste :
    print("Valeur de la variable 'x' :", x)

print("Au revoir")
```

```
Bonjour
Valeur de la variable 'x' : 5
Valeur de la variable 'x' : 6
Valeur de la variable 'x' : 7
Valeur de la variable 'x' : 8
Valeur de la variable 'x' : 9
Valeur de la variable 'x' : 10
Au revoir
```

In [19]:

```
ma_liste = range(10)

print("Premier élément de ma liste :", ma_liste[0])
print("Deuxième élément de ma liste :", ma_liste[1])
print("Dernier élément de ma liste :", ma_liste[-1])
```

```
Premier élément de ma liste : 0
Deuxième élément de ma liste : 1
Dernier élément de ma liste : 9
```

In [20]:

```
print("Bonjour")

for x in range(10) :
    print("Valeur de la variable 'x' :", x)

print("Au revoir")
```

```
Bonjour
Valeur de la variable 'x' : 0
Valeur de la variable 'x' : 1
Valeur de la variable 'x' : 2
Valeur de la variable 'x' : 3
Valeur de la variable 'x' : 4
Valeur de la variable 'x' : 5
Valeur de la variable 'x' : 6
Valeur de la variable 'x' : 7
Valeur de la variable 'x' : 8
Valeur de la variable 'x' : 9
Au revoir
```

Exercice 5 : Écrire un programe qui calcule la table de multiplication du 7 en utilisant le boucle `for - in`.