

Mise à Niveau

Sébastien Gagné, Université d'Orléans

Cours 02

Structure classique d'un programme, Type booléen,
Instructions conditionnelles, Notation pointée

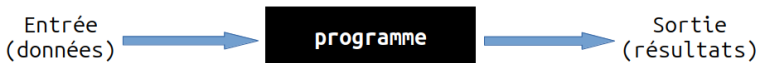
Plan du cours 2

- Entrées et Sorties de base
- Structure classique d'un programme
- Le type booléen
- Exécution conditionnelle
- La notation pointée

Entrées / Sorties

On va interagir avec la plupart des programmes

- fournir des données (Entrée) au programme par le biais du clavier par exemple ;
- recevoir des résultats (Sortie) du programme dans la console par exemple.



Fonction *print*

La fonction *print* permet d'afficher un message composé de valeurs, de valeurs de variables ou d'expressions. Elle affiche un résultat en sortie.

```
>>> a = 7
>>> b = 2
>>> print('la somme de', a, 'et', b, 'est', a + b)
    la somme de 7 et 2 est 9
>>> p = a * b
>>> print('le produit de', a, 'et', b, 'est', p)
    le produit de 7 et 2 est 14
```

Le séparateur par défaut est l'espace.

Le caractère de fin par défaut est le retour à la ligne.

Fonction *input*

La fonction *input* permet de récupérer en entrée une chaîne de caractères, en affichant un message optionnel à l'utilisateur.

```
>>> prenom = input('quel est votre prénom ? ')
quel est votre prénom ? Toto
>>> print('Bonjour', prenom)
Bonjour Toto
```

Fonction *input*

Attention, ce qui est récupéré par la fonction *input* est toujours une chaîne de caractères !

```
>>> n = input('entrez un nombre : ')
entrez un nombre : 12
>>> print('le carré de', n, 'est', n * n)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can't multiply sequence by non-int of type 'str'
>>> n
'12'
```

Fonction *input*

On doit parfois utiliser les fonctions de conversion !

```
>>> ch = input('entrez un nombre : ')
entrez un nombre : 12
>>> ch
'12'
>>> n = int(ch)
>>> n
12
>>> print('le carré de', n, 'est', n * n)
le carré de 12 est 144
```

Fonction *input*

On peut enchaîner (on dit aussi "composer") une fonction de conversion avec la fonction *input*.

Exemple : composer *int* avec *input* récupère sous forme d'un *int* la réponse de l'utilisateur.

```
>>> n = int(input('entrez un nombre : '))
entrez un nombre : 12
>>> n
12
>>> print('le carré de', n, 'est', n * n)
le carré de 12 est 144
```

Structure usuelle d'un programme

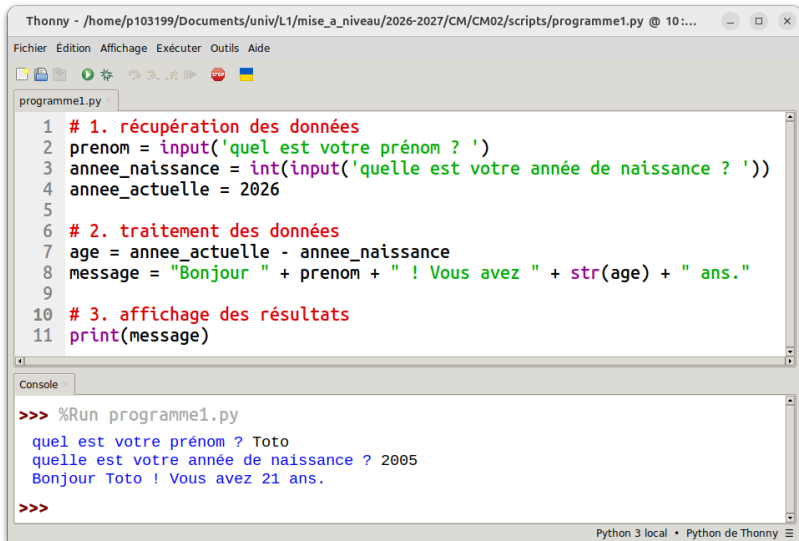
On s'habitue à concevoir nos premiers programmes selon une structure en trois étapes.

- récupération des données (entrée).
Possible grâce à la fonction *input*.
- traitement, calcul
- affichage des résultats (sortie).
Possible grâce à la fonction *print*.

Commentaires (commencent par *#*) bienvenus !

Enfin, on peut sauvegarder un programme python dans un fichier texte d'extension **.py**

Exemple de programme



The screenshot shows the Thonny Python IDE interface. The title bar indicates the file path: `Thonny - /home/p103199/Documents/univ/L1/mise_a_niveau/2026-2027/CM/CM02/scripts/programme1.py @ 10:...`. The menu bar includes `Fichier`, `Édition`, `Affichage`, `Exécuter`, `Outils`, and `Aide`. The toolbar contains icons for file operations and execution. The editor window, titled `programme1.py`, contains the following Python code:

```
1 # 1. récupération des données
2 prenom = input('quel est votre prénom ? ')
3 annee_naissance = int(input('quelle est votre année de naissance ? '))
4 annee_actuelle = 2026
5
6 # 2. traitement des données
7 age = annee_actuelle - annee_naissance
8 message = "Bonjour " + prenom + " ! Vous avez " + str(age) + " ans."
9
10 # 3. affichage des résultats
11 print(message)
```

The console window, titled `Console`, shows the execution output:

```
>>> %Run programme1.py
quel est votre prénom ? Toto
quelle est votre année de naissance ? 2005
Bonjour Toto ! Vous avez 21 ans.
>>>
```

The status bar at the bottom right indicates `Python 3 local • Python de Thonny`.

Le type booléen

- Le type booléen regroupe les variables qui prennent des valeurs booléennes.
- Il y a deux valeurs booléennes : *True* (vrai) et *False* (faux)
- Ces deux valeurs sont la base de la logique et du raisonnement.
- Elles permettent de décider de faire quelque chose ou non (voir instruction conditionnelle).

Expressions booléennes

Une expression booléenne est composée :

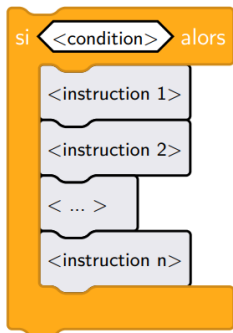
- de valeurs booléennes (True, False),
- de variables de type booléen,
- de valeurs ou variables de types divers, avec des opérateurs de comparaison,
- d'opérateurs booléens,

```
>>> a = 5
>>> (a + 1) * (a + 7) > 60
True
>>> b = 4
>>> a < b
False
>>> c = (a * b > 2 * a + 3 * b)
>>> c
False
```

Exécution conditionnelle

Certaines instructions vont voir leur exécution conditionnée à la réalisation d'une condition.

Cela se traduit par ce schéma et ce code Python :



```
if <condition> :  
    <instruction_1>  
    <instruction_2>  
    ...  
    <instruction_n>
```

Exécution conditionnelle

La syntaxe Python impose que

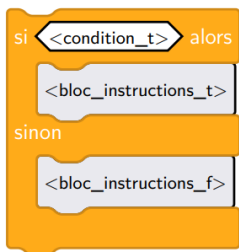
- le mot clé *if* précède la condition
- la condition doit être une valeur ou expression booléenne
- les " : " suivent la condition
- une indentation cohérente (identique) soit respectée pour toutes les instructions à exécuter si la condition vaut *True*

```
1  # données en entrée
2  n = int(input('entrez un nombre positif : '))
3
4  # traitement conditionnel
5  if n >= 0 :
6      print('le nombre', n, 'est positif')
7      print('merci, vous avez respecté la consigne')
8      print('à bientôt !')
9
10 print('fin du programme')
```

Exécution conditionnelle

On peut enrichir l'exécution conditionnelle en indiquant quoi faire si la condition est évaluée à False.

Cela se traduit par ce schéma et ce code Python



```
if <condition> :  
    <instruction_t_1>  
    <instruction_t_2>  
    ...  
    <instruction_t_n>  
else :  
    <instruction_f_1>  
    <instruction_f_2>  
    ...  
    <instruction_f_m>
```

Exécution conditionnelle

Exemple : triplet pythagoricien

Trois entiers a, b, c donnés en ordre croissant forment un triplet pythagoricien si $a^2 + b^2 = c^2$

```
1 a = int(input('a = '))
2 b = int(input('b = '))
3 c = int(input('c = '))
4
5 if a*a + b*b == c*c :
6     print('c\'est un triplet pythagoricien')
7 else :
8     print('ce n\'est un triplet pythagoricien')
```

Plusieurs cas distincts

Parfois on est amené à envisager des cas distincts.

Exemple : on demande un nombre entier.

- S'il est négatif, on signale une erreur ;
- Sinon, s'il est entre 0 et 10, on dit qu'il est "petit" ;
- Sinon, on dit qu'il est "grand".

```
1 a = int(input('entrez un entier positif : '))
2
3 if a < 0 :
4     print('erreur de saisie !')
5 else :
6     if a <= 10 :
7         print('ce nombre est petit')
8     else :
9         print('ce nombre est grand')
```

Plusieurs cas distincts

Le code suivant est exactement équivalent :

```
1 a = int(input('entrez un entier positif : '))
2
3 if a < 0 :
4     print('erreur de saisie !')
5 elif a <= 10 :
6     print('ce nombre est petit')
7 else :
8     print('ce nombre est grand')
```

elif est la contraction de *else if*.

On peut écrire autant de cas *elif* que l'on veut.

Le cas choisi correspond à la première condition satisfaite.

Opérateur unaire de négation

- Cet opérateur "switch" entre les deux valeurs.
- Traduction en Python : *not*
- Notation algo : $\neg$
- Table de vérité

p	not p
True	False
False	True

Opérateur binaire de conjonction

- Cet opérateur traduit le "et" entre deux valeurs booléennes.
- Traduction en Python : *and*
- Notation algo : $\wedge$
- Table de vérité

p	q	p and q
True	True	True
True	False	False
False	True	False
False	False	False

and	True	False
True	True	False
False	False	False

Opérateur binaire de disjonction

- Cet opérateur traduit le "ou" inclusif entre deux valeurs booléennes.
- Traduction en Python : *or*
- Notation algo : $\vee$
- Table de vérité

p	q	p or q
True	True	True
True	False	True
False	True	True
False	False	False

or	True	False
True	True	True
False	True	False

Priorités entre opérateurs

- Du moins prioritaire au plus prioritaire :
or < and < not
- Si priorités égales : associativité de gauche à droite

False or True and not True or True

Priorités entre opérateurs

- Du moins prioritaire au plus prioritaire :
or < and < not
- Si priorités égales : associativité de gauche à droite

False or True and not True or True
 False
 False
 False
 False
 True

Tests d'égalité / différence

Les tests d'égalité (==) et de différence (!=) produisent des valeurs booléennes.

```
>>> 5 == 3
```

```
False
```

```
>>> 3 == 2 + 1
```

```
True
```

```
>>> 4 != 4
```

```
False
```

```
>>> 'ab' != 'AB'
```

```
True
```

Tests de comparaison

Les tests `<`, `<=`, `>`, `>=` produisent également des valeurs booléennes.

```
>>> 1 < 1
False
>>> 1 < 2
True
>>> 1 <= 1
True
>>> 'ab' < 'acd'
True
```

Ces comparaisons s'appliquent à des valeurs comparables (pour les *str*, ordre alphabétique utilisé).

Tests de comparaison

Python tolère les enchaînements de comparaisons

```
>>> -1 < 3 < 6
True
>>> 1 <= 5 < 3
False
```

Ces enchaînements de comparaisons reviennent à des conjonctions ("et").

Ainsi, $-1 < 3 < 6$ revient à : $-1 < 3$ **and** $3 < 6$

Dans la plupart des langages, ce n'est pas toléré.

Mêler nombres et booléens

Comment évaluer l'expression suivante ?

```
58 + 65 < 8 * 8 * 2 and 58 % 7 != 23 // 11
```

1. faire les opérations entre nombres (en respectant les priorités et règles d'associativité)
2. évaluer les comparaisons de nombres
3. faire les opérations entre booléens

Mêler nombres et booléens

Règles

1. priorité des opérations entre nombres sur les comparaisons
2. priorité des comparaisons sur les opérations entre booléens

$$\begin{array}{ccccc} \frac{58 + 65}{123} & < & \frac{8 * 8 * 2}{128} & \text{and} & \frac{58 \% 7}{2} \neq \frac{23 // 11}{2} \\ & \text{True} & & & \text{False} \\ & \text{False} & & & \end{array}$$

moins prioritaire  plus prioritaire

or	and	not	< , ≤ , > , ≥ , == , !=	+ – (binaire)	*, / , // , %	– (unaire)
----	-----	-----	-------------------------	---------------	---------------	------------

Notation pointée

En Python comme dans d'autres langages, on a souvent recours à la "notation pointée".

La syntaxe est de la forme

`<objet> . <methode> (<arguments>)`

ou de la forme

`<objet> . <attribut>`

Notation pointée

Exemple 1

```
>>> import random  
>>> nbr_aleat = random.randint(4, 11)
```

On demande ici au module *random* d'utiliser sa méthode *randint*, avec les *arguments* 4 et 11.

Notation pointée

Exemple 2

```
>>> phrase = 'Bonjour !'  
>>> phrase_minuscule = phrase.lower()  
>>> phrase_minuscule  
'bonjour !'  
  
>>> phrase_majuscule = phrase.upper()  
>>> phrase_majuscule  
'BONJOUR !'
```

On demande ici à la chaîne *phrase* d'utiliser, sans argument, ses méthodes *lower* et *upper*.

Ces deux méthodes retournent une nouvelle chaîne.

Notation pointée

Exemple 3

```
>>> import math
>>> p = math.pi
>>> p
3.141592653589793
```

On demande ici au module *math* de communiquer la valeur de son *attribut* nommé *pi* et accessible par la syntaxe *math.pi*.