

VARIABLES ET TYPES

Un *algorithme* est un ensemble d'étapes permettant d'atteindre un but en réalisant un nombre fini de fois un nombre fini d'instructions.

Cette définition dépasse largement le cadre de l'informatique (algorithme d'Euclide pour le calcul du PGCD, pivot de Gauss, recettes de cuisine...).

On peut décrire un algorithme à l'aide d'un *pseudo-langage* utilisant des instructions élémentaires comme l'affectation, *tant que...*, *Si... alors...*, etc.

Un *programme* est la traduction d'un algorithme en un langage interprétable par un ordinateur. Le langage que nous utiliserons est le langage Python.

I. Variables et affectation

On peut voir une *variable* comme le nom d'une « boîte » pouvant recevoir un contenu. Dans beaucoup de langage, la boîte ne peut recevoir qu'un *type* de valeur (par exemple, entier, ou réel, ou chaîne de caractère, etc.) et doit être préalablement déclarée avant d'être utilisée. En Python, une variable n'a pas un type fixé, et la déclaration se fait automatiquement lors de la première affectation.

Un nom de variable est formé de caractères alphanumériques, doit commencer par une lettre ou « `_` » (« tiret bas » ou « underscore »), et ne doit pas être un mot réservé de Python :

Exemples

`a, b, c, liste, liste2, Liste3_22, _truc` sont des noms valides

`2a, x-y, for, #$a` ne sont pas des noms valides.

On s'efforcera de bien choisir les noms de variables : suffisamment explicites, de manière à bien exprimer ce que contient la variable, tout en restant d'une longueur raisonnable.

Pour placer un contenu dans une variable (*affecter* la variable), on utilise l'opérateur d'affectation `=`, qui fonctionne essentiellement de la manière suivante : dans l'instruction `<var> = <expression>`, `<expression>` est évalué et son résultat est placé dans `<var>`.

Le symbole `=` en Python n'a donc pas la signification habituelle : l'instruction `x = 3` signifie « x reçoit la valeur 3 ».

Exemple

On utilise ici le mode interactif de Python : les commandes sont entrées après l'invite `>>>`, et la machine répond après validation de la commande (touche entrée).

```
>>> x = 3
>>> print(x) #la fonction print permet d'afficher le contenu d'une variable
3
>>> y = 2 + 3.5
>>> print(y)
5.5
>>> print(x+y)
8.5
```

Si l'expression utilisée pour l'affectation fait appel à des variables précédemment affectées, celles-ci sont interprétées au moment de leur affectation : c'est leurs *valeurs* qui servent à l'affectation.

```
>>> a = 42
>>> b = 2*a
>>> print(a,b)
42 84
>>> a = 0
>>> print(a,b)
0 84
```

En Python, il est possible de faire des affectations multiples

```
>>> x, y = -5, 14
>>> print(x)
-5
>>> print(y)
14
```

Exemple

Comment permuter les contenus de deux variables ?

Le programme `a = b ; b = a` ne va pas fonctionner :

```
>>> a = 1789
>>> b = 1515
>>> a = b
>>> b = a #la valeur de a a déjà été modifiée
>>> print(a,b)
1515 1515
```

Première solution : passer par une variable auxiliaire :

```
>>> a = 1789
>>> b = 1515
>>> c = a #On conserve la valeur initiale de a
>>> a = b
>>> b = c
>>> print(a,b)
1515 1789
```

Deuxième solution : utiliser une affectation multiple :

```
>>> a = 1789
>>> b = 1515
>>> a, b = b, a
>>> print(a,b)
1515 1789
```

II. Types

La valeur d'une variable est stockée dans un emplacement mémoire, sous la forme d'une suite de bits. Cette dernière ne représente pas forcément un nombre entier naturel, mais peut représenter un entier signé, un nombre réel, un nombre complexe, une chaîne de caractères, etc.

Pour distinguer les différentes interprétations possibles du mot binaire correspondant à la valeur de la variable, le langage de programmation utilise différents types de variables : le type entier, le type réel, le type chaîne de caractères, etc.

Nous allons passer en revue quelques types courants du langage Python.

1) Les types numériques

a) Type int

Le type `int` est utilisé pour représenter des nombres entiers.

```
>>> x = 2016
>>> type(x)
<class 'int'>
```

En Python 3.0, les entiers représentables à l'aide du type `int` ne sont limités que par la mémoire de la machine.

On dispose de plusieurs opérations sur les entiers :

```
>>> 15 + 17 #Addition
32
>>> 5 - 216 #Soustraction
-211
>>> 15 * 17 #Multiplication
255
>>> 4 / 2 #Division
2.0
>>> type(4 / 2)
<class 'float'>
>>> 19 / 8
2.375
```

```
>>> 19 // 8 #Division entière : quotient de la division euclidienne de 19 par 8.
2
>>> 19 % 8 #Modulo : reste de la division euclidienne de 19 par 8.
3
>>> 2 ** 10 #Exponentiation
1024
>>> abs(-5), abs(6) #Valeur absolue
(5, 6)
```

b) Type float

Ce type sert à représenter des nombres réels (ou en tout cas des valeurs approchées de ces réels).

```
>>> type(6.5656)
<class 'float'>
>>> 6.0221409e23 #Notation scientifique
6.0221409e+23
>>> 1.67265e-27
1.67265e-27
```

Tout comme pour les entiers, on dispose des opérations arithmétiques classiques sur les flottants.

<pre>>>> 1.44 + 0.5, 1.44 - 0.5 (1.94, 0.94) >1.44 * 0.5, 1.44 / 0.5 (0.72, 2.88)</pre>	<pre>>>> 1.44**0.5 1.2 >>> abs(-6.5) 6.5</pre>
---	--

Nous verrons plus tard dans l'année que les nombres décimaux ne peuvent pas tous être représentés de manière exacte avec le type float.

<pre>>>> 0.1 + 0.2 0.30000000000000004 >>> 2.2 * 3 6.6000000000000005</pre>	<pre>>>> 1/49 * 49 0.9999999999999999 >>> 1/2**1000 * 2**1000 1.0</pre>
---	---

Il est possible de convertir un flottant en entier avec la fonction `int`, qui réalise en fait une troncature :

```
>>> int(2.), int(3.8), int(-4.9)
(2, 3, -4)
```

Réciproquement, on peut convertir un entier en flottant avec la fonction `float` :

```
>>> float(1732)
1732.0
```

Remarque : Il existe également un type pour représenter des nombres complexes, mais nous ne l'utiliserons pas.

```
>>> type(2+3j)
<class 'complex'>
>>> 1j**2
(-1+0j)
```

c) Précédence des opérateurs

En l'absence de parenthèse, la règle est la même qu'en mathématiques. Sont calculées en priorité :

- les exponentiations (opérateur `**`)
- puis les multiplications, divisions et modulo (opérateurs `*`, `/`, `//`, `%`)
- puis les additions et soustractions (opérateurs `+`, `-`)

Lorsqu'il y a ambiguïté, dans la plupart des cas, ce sont les opérateurs les plus à gauche qui sont évalués en premier, à l'exception de l'exponentiation.

```
>>> 1 + 2*3
7
>>> 2 * 3**2
18
>>> 98 + 1%3
99
```

```
>>> 2 - 3 + 4
3
>>> 6 * 2 % 4
0
>>> (2**1)**2, 2**1**2
(4, 2)
```

2) Les booléens : type bool

Le type booléen¹ est réservé aux variables pouvant prendre deux valeurs : `True` et `False`, qui sont des mots réservés du langage Python.

Les comparaisons ont pour résultat des valeurs booléennes :

```
>>> 2 < 3
True
>>> 2 < 0
False
>>> 2 <= 1+1
True
>>> 9 > 5
True
>>> 3 >= 6
False
```

```
>>> 0 + 0 == 0
True
>>> 0 + 0 = 0
File "<stdin>", line 1
SyntaxError: can't assign to operator
>>> 3 + 2 != 5
False
>>> 3 != 1
True
```

⚠ ATTENTION ! Ne pas confondre `==` et `=`.

On dispose aussi d'opérateurs logiques : `and`, `or`, `not`.

```
>>> 2 > 3 or 2 < 5
True
>>> 2 == 3 or 2 > 5
False
>>> True or True
True
>>> (2 < 3) and (3 == 4 - 1)
True
```

```
>>> (2 < 3) and (5 == 2 + 2)
False
>>> not (5 > 3)
False
>>> not(8 <= 6)
True
```

3) Les chaînes de caractères : type str

Une chaîne de caractères est une suite de caractères consécutifs notée entre apostrophes (ou entre guillemets) :

```
>>> s = 'anticonstitutionnellement'
>>> type(s)
<class 'str'>
>>> print(s)
anticonstitutionnellement
>>> len(s) #Longueur de la chaîne de caractères
25
>>> s[0],s[1],s[24] #Lettres indexées à partir de 0
('a', 'n', 't')
>>> s[25]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
>>> 'bon' + 'jour' #Concaténation de deux chaînes de caractères
'bonjour'
```

1. L'algèbre de Boole, développée par le mathématicien anglais George Boole (1815-1864), est une structure algébrique qui permet une approche de la logique à deux états.