

Mise à Niveau

Sébastien Gagné, Université d'Orléans

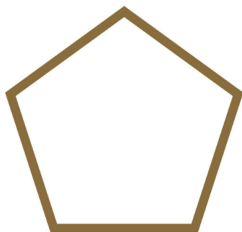
Cours 01

nombres, chaînes, variables

Plan du cours 1

- Symboles, écritures
 - Concept abstrait
 - Représentations concrètes
- Valeurs, types, expressions
 - Variété de valeurs
 - Types classiques
 - Opérateurs et fonctions classiques
 - Expressions
- Variables
 - Expérience mentale
 - Représentation graphique
 - Affectation

Symboles, écritures



3 représentations "humaines" d'un concept abstrait.
La chaîne de caractères "**cinq**" en est une quatrième.
Le caractère "**5**" en est une ... cinquième.

Symboles, écritures

Pour l'ordinateur, qui utilise aussi ce concept **cinq**, une représentation est ... **l'écriture binaire** $(101)_2$.

Par **l'écriture binaire**, l'ordinateur peut

- représenter des symboles (chiffres, caractères, ...)
- effectuer des opérations
- évaluer des expressions
- exécuter des instructions

Symboles, écritures

Pourquoi utiliser le binaire (des **0** et des **1**) ?

→ cohérent avec le fonctionnement des composants électroniques d'un ordinateur !

En mesurant la tension aux bornes d'un transistor, on constate deux états possibles :

→ absence de tension **valeur 0**

→ présence d'une tension **valeur 1**

Il est donc simple, en fonction de l'état d'un circuit, de représenter une valeur 0 ou 1, ou d'autres valeurs plus compliquées, par des combinaisons de 0 et de 1.

Valeurs

L'humain utilise en permanence des valeurs diverses.

→ 0, 1, 2, 3, 4, 5, ...

→ -1, -12, ...

→ 2.5 €, 3.15 kg, 37.2 °C, ...

→ lundi 2 septembre 2024 à 11h20

→ 'a', 'b', '?', "Bonjour", "Hello", ...

→ OUI (ou NON), VRAI (ou FAUX)

L'ordinateur doit savoir ce qu'il peut en faire.

Types de Valeurs

La notion de **Type** permet de regrouper, par famille, des valeurs de même "nature".

Par exemple (liste non exhaustive) :

→ 0, 1, 2, -5, -8, ... sont de type **nombre entier**.

→ 'a', 'b', '?', ... sont de type **caractère**.

→ "ab", "H4", "hello world!", ... sont de type **chaîne de caractères**.

→ 2.5, 3.0, ... sont de type **nombre décimal**.

Types en Python

En python, la **fonction type** donne le type de la valeur passée en paramètre.

```
>>> type(5)
<class 'int'>
>>> type(2.57)
<class 'float'>
>>> type('a')
<class 'str'>
>>> type("bonjour")
<class 'str'>
>>> type(True)
<class 'bool'>
```

Remarque : pour les caractères ou les chaînes de caractères, python ne propose que le type **str**.

Type int python

- Le type **int** python regroupe les valeurs entières relatives (maths : entiers négatifs ou positifs), comme 5, -7, 0, 215423658714.
- python n'impose pas de limite à la taille des **int**, contrairement à la plupart des autres langages.
- python propose pour les int :
 - La représentation en base 10 classique : 1298
 - La représentation en base 2 : 0b10100010010
 - La représentation en base 8 : 0o2422
 - La représentation en base 16 : 0x512

Type float python

Le type **float** correspond aux nombres "réels".

→ 2.5, -7.4, 2.718

→ 3.0, 6. (équivalent de 6.0)

→ 1e-33, 1.5e12

ATTENTION, cette représentation des nombres réels est imparfaite et parfois approximative !

```
>>> 0.1 + 0.2
0.30000000000000004
>>> 0.3 - 0.1
0.19999999999999998
```

Opérateurs sur les nombres

- Des opérateurs sont prévus pour agir sur des valeurs numériques, de types identiques ou non.
- Ces opérateurs produisent une valeur résultat.
- Un opérateur utilise un symbole et une ou des valeurs placées autour du symbole.
- Ex d'opérateurs binaires : $2 + 6$ ou $8.3 * 5.1$
- Ex d'opérateur unaire : $- 8$

Opérateurs sur les nombres

Opérateur unaire : agit sur **un** nombre

→ opérateur de changement de signe : l'opérateur **-** est appliqué à l'entier **8** pour donner l'entier **-8**.

Opérateur binaire : agit sur **deux** nombres

- **+** addition
- **-** soustraction
- ***** multiplication
- **/** division
- **//** division entière
- **%** modulo
- ****** puissance

Promotion de type

Certains opérateurs binaires (+, *, ...) sont prévus pour agir sur des valeurs de **même** type.

→ $2 + 3$ donne 5 : $\text{int} + \text{int} \Rightarrow \text{int}$

→ $2.5 + 4.2$ donne 6.7 : $\text{float} + \text{float} \Rightarrow \text{float}$

→ $2.7 + 4.3$ donne 7.0 (attention : pas 7 !)

On peut pourtant demander $2 + 3.6$

C'est là qu'intervient la **promotion de type** :

$2 + 3.6$ est interprété en $2.0 + 3.6$, qui donne 5.6

Opérateurs // et %

Ces opérateurs ont pour cadre naturel les int, mais sont adaptés aux float.

- // désigne l'opérateur **division entière**
- % désigne l'opérateur **modulo**

exemple :

- $139 // 8 = 17$
- $139 \% 8 = 3$

$$\begin{array}{r|l} 139 & 8 \\ 3 & 17 \end{array}$$

$$139 = 17 * 8 + 3$$

Opérateurs // et %

Plus généralement :

- $a // b = q$
- $a \% b = r$

revient à

$$a = b * q + r \quad \text{et} \quad 0 \leq r < b$$

exemples :

- $27 // 6 = 4$ et $27 \% 6 = 3$
- $-31 // 4 = -8$ et $-31 \% 4 = 1$

Le type chaîne de caractères (str)

Python ne définit pas le type **caractère**, mais le type **str** dont les valeurs sont des séquences finies de caractères parmi

- les lettres
- les chiffres
- la ponctuation
- espace, tabulation, retour à la ligne
- tout autre caractère *unicode* (é, &, ù, £, @, ...)

Le type chaîne de caractères (str)

Les chaînes de caractères sont définies par des suites de caractères entourées de **délimiteurs**. Il y a quatre délimiteurs possibles en python :

- ' et " pour les cas classiques :

```
>>> message1 = 'coucou'  
>>> message2 = "bonjour"
```

```
>>> message1  
'coucou'  
>>> message2  
'bonjour'
```

Le type chaîne de caractères (str)

- `'''` et `"""` (les mêmes répétés trois fois)
pour les chaînes écrites sur plusieurs lignes :

```
>>> message3 = '''Ceci est une chaîne  
sur plusieurs lignes'''  
>>> message4 = """Ceci en est une autre.  
Elle est écrite sur  
trois lignes."""
```

```
>>> message3  
'Ceci est une chaîne\nsur plusieurs lignes'  
>>> message4  
'Ceci en est une autre.\nElle est écrite sur\ntrois lignes.'
```

```
>>> print(message3)  
Ceci est une chaîne  
sur plusieurs lignes  
>>> print(message4)  
Ceci en est une autre.  
Elle est écrite sur  
trois lignes.
```

Le type chaîne de caractères (str)

Pour écrire un caractère ' ou " dans une chaîne :

- On peut utiliser un délimiteur différent :

```
>>> message5 = "Attention à l'apostrophe !"
>>> message5
"Attention à l'apostrophe !"
>>> message6 = 'il a dit "non !"'
>>> message6
'il a dit "non !"'
```

- On peut échapper le caractère :

```
>>> message7 = 'Attention à l\'apostrophe !'
>>> message7
"Attention à l'apostrophe !"
```

Opérations sur les chaînes

Python propose des opérateurs / fonctions agissant sur les chaînes de caractères.

- concaténation : `'ab' + 'cde'` donne `'abcde'`
- produit par un int : `'ab' * 3` donne `'ababab'`
- longueur : `len("abcde")` donne 5

Attention, certaines opérations n'ont aucun sens.

Par exemple `"35" + 2` :

- Quel sens aurait le `+` ?
- Quel résultat pourrait-on attendre ?

Fonctions de conversion

Python propose des fonctions permettant de convertir les valeurs en changeant leur type. Par exemple :

- `int('17')` retourne 17
- `int(2.5)` retourne 2
- `float(3)` retourne 3.0
- `float("3.21")` retourne 3.21
- `str(2.5)` retourne '2.5'
- ...

Fonctions classiques

Python propose un catalogue de fonctions directement utilisables. Exemples :

- la fonction *min* (resp. *max*) retourne la valeur minimum (resp. maximum) de valeurs données.
- la fonction *abs* retourne la valeur absolue de la valeur donnée.

```
>>> min(2.5, -1.7, 4.2, 6)
-1.7
>>> max(5, 4, 6, 8, 3, 10)
10
>>> abs(-2.5)
2.5
```

Fonctions disponibles

Python propose, via un import, d'autres fonctions.

Exemples :

- la fonction *sqrt* du module *math*.
- la fonction *randint* du module *random*.

Divers imports possibles

- *from math import sqrt* : fonction accessible.
- *from math import ** : tout est accessible.
- *import math* : le module est accessible.

Expressions

- Une **expression** est une suite syntaxiquement correcte de valeurs, d'opérateurs, de parenthèses et d'appels de fonctions qui produit une **valeur**.
- Python évalue une expression en respectant des **priorités** (opérateurs, parenthèses, fonctions)
- A priorité égale (+ et - par exemple), on évalue par **associativité** de gauche à droite

moins prioritaire



plus prioritaire

+ - (soustraction)	* / // %	- (opposé)	parenthèses, fonctions
--------------------	----------	------------	------------------------

Mécanique d'évaluation

- Une seule opération à la fois
- Remplacement progressif
- penser aux conversions
- Pour les parenthèses, évaluer la sous-expression entre parenthèses et remplacer

$$\begin{array}{r} 5 * (2 * (7 + 8)) \\ \hline 15 \\ \hline 30 \\ \hline 150 \end{array}$$

Variables - Expérience mentale

Quelle est la plus grande des valeurs ?

4	7	2	3	5
---	---	---	---	---

Variables - Expérience mentale

Quelle est la plus grande des valeurs ?

49	27	67	46	17	42	48	24
13	43	46	71	55	47	54	32
59	22	20	16	14	74	58	21
72	87	64	78	21	85	17	29
81	74	65	49	21	47	68	77

Variables - Expérience mentale

Quelle est la plus grande des valeurs ?

1316	1534	1295	1611	1505	1406	1022	1715	1394	1203	1125	1890	1723	1377	1788
1298	1842	1648	1429	1594	1206	1462	1693	1000	1789	1136	1798	1186	1224	1017
1548	1793	1483	1734	1858	1691	1119	1282	1439	1367	1128	1202	1604	1700	1111
1752	1742	1575	1037	1306	1034	1712	1565	1335	1609	1865	1897	1683	1294	1714
1459	1599	1767	1446	1032	1446	1433	1755	1298	1176	1433	1710	1133	1628	1132
1560	1104	1866	1618	1049	1071	1697	1083	1724	1248	1321	1719	1755	1828	1035
1846	1765	1920	1154	1866	1803	1355	1192	1636	1689	1508	1513	1422	1340	1191
1273	1250	1089	1153	1332	1043	1009	1094	1360	1324	1653	1071	1598	1671	1093
1487	1523	1187	1043	1254	1371	1207	1250	1834	1811	1206	1118	1634	1055	1027
1339	1518	1463	1212	1312	1519	1914	1891	1785	1454	1341	1727	1391	1170	1905
1168	1293	1476	1000	1224	1096	1721	1866	1331	1066	1520	1860	1331	1451	1613
1835	1846	1295	1168	1804	1172	1160	1770	1249	1772	1136	1197	1173	1393	1827
1509	1132	1056	1162	1193	1255	1629	1623	1434	1905	1504	1305	1622	1693	1907

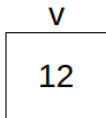
Variables

Pour diverses raisons, on peut avoir besoin de conserver en mémoire une valeur.

On utilise pour cela une **variable**. Elle correspond à un endroit de stockage en mémoire.

Le **nom** de la variable permet d'accéder à l'espace correspondant en mémoire.

Il est aussi utilisé à la **création** de la variable, avec **affectation** de valeur : $v = 12$



Variables

L'évocation du nom de la variable permet de récupérer la valeur.

On peut créer des expressions complexes à partir des noms de variables : si $v = 12$, $2 * v + 1$ retourne 25

La variable est détruite à la fin de l'exécution

Entre sa création et sa destruction, une variable peut voir sa valeur évoluer.

```
>>> v = 12
>>> a = 2 * v + 1
>>> b = 15 - v
>>> v = a + b
>>> v
28
```

Trace

Il est essentiel de pouvoir produire la **trace**, qui montre l'évolution des valeurs

Code Python

```
1  n = 2
2  p = 7
3  n = p + 1  # n change de valeur
4  p = n * 3  # p change de valeur
5  r = 102
6  r = 33 * r  # r change de valeur
```

Trace (de l'exécution du code)

	n	p	r	
1	2			avant l'exécution de la ligne 1
2	2	7		après l'exécution de la ligne 1
3	8	7		
4	8	24		
5	8	24	102	
6	8	24	3366	
fin				

n° de ligne de l'instruction exécutée

- une ligne par unité de temps / instruction effectuée
- une colonne par variable
- une variable n'apparaît qu'à sa création
- à la fin toutes les variables sont détruites (ligne fin)

Choix du nom d'une variable

Un nom de variable commence par une lettre (minuscule ou majuscule), ou un `_`.

Ce premier caractère est suivi (ou non) de lettres, de chiffres, et/ou du caractère `_`.

Le choix du nom de la variable est important. Il peut rendre le programme explicite (ou non).

- ```
TVA = 20
prix_unitaire = 6
qtt = 28
prix = qtt * prix_unitaire * (1 + TVA / 100)
```

- ```
en_stock = 49
vendues = 18
en_stock = en_stock - vendues
```