

III – Types numériques et variables

1/ Les types numériques

En mathématiques, les nombres utilisés vont appartenir à $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{R}$ ou $\mathbb{C}$. En Python, comme dans d'autres langages, les nombres vont être de trois types :

int ce sont les entiers longs (depuis Python3), qui sont codés sur plusieurs mots et disposent ainsi d'une précision « illimitée ».

float ce sont les nombres à virgule flottante, qui permettent de représenter les nombres réels, quoiqu'avec une perte de chiffres significatifs (cf. chapitre précédent).

complex ce sont les nombres complexes, en fait des couples de nombres flottants.

Il y a d'autres types d'objets sous Python, que nous verrons ultérieurement.

Bon à savoir

- La commande `type()` permet d'accéder au type d'un objet donné.
- On peut changer le type d'un objet avec `int()`, `float()` et `complex()`.
- `int()` tronque la partie décimale d'un flottant.
- `complex(a,b)` crée le complexe $a + ib$ et `complex(a)` crée $a + i0$.

1.1) Les entiers

Voici les opérations usuelles sur les nombres entiers :

+	-	*	/	**	//	%
addition	soustraction	multiplication	division	puissance	quotient	reste

les deux dernières étant le quotient et de la reste de la division euclidienne.

Remarque : notons que la division « normale » renvoie **toujours un flottant**, même si le reste est nul. Au passage, 5.0 ou 5. sont des flottants, et non des entiers comme 5.

1.2) Les flottants

Attention, **ils sont notés à l'aide d'un point** et non d'une virgule : celle-ci a un autre sens en Python. On dispose des opérations de base (comme pour les entiers) ainsi que de :

<code>abs(x)</code>	valeur absolue du réel x
<code>round(x,n)</code>	arrondi du réel x à la n^{e} décimale
$x \text{ E } n$ ou $x \text{ e } n$	$x \times 10^n$

Pour utiliser d'autres fonctions qui ne sont pas chargées en mémoire par défaut, on a besoin d'**importer des bibliothèques** supplémentaires : par exemple, la bibliothèque `math` contient un certain nombre de fonctions et constantes usuelles. Citons les plus communes :

<code>pi</code>	<code>e</code>	<code>cos</code>	<code>sin</code>	<code>tan</code>	<code>sqrt</code>	<code>exp</code>	<code>log</code>
-----------------	----------------	------------------	------------------	------------------	-------------------	------------------	------------------

- On importe cette bibliothèque avec l'instruction `import math`
Les commandes peuvent alors être utilisées en tapant `math.commande`
- Il est possible d'importer simplement certaines fonctions pour simplifier leur usage.
Par exemple, après `from math import exp` on tapera directement `exp(4)`.
- On peut tout charger en tapant `from math import *`. Ce n'est pas recommandé, car le même nom peut désigner différents objets.
- On peut renommer la bibliothèque que l'on importe comme avec `import math as m`
C'est la solution à privilégier car elle évite les conflits de noms entre bibliothèques.

Exemple : voici comment utiliser ce qui précède

<code>import math</code>	<code>from math import *</code>	<code>import math as m</code>
<code>math.exp(2.5)</code>	<code>exp(2.5)</code>	<code>m.e**2.5</code>
<code>math.log(3)</code>	<code>log(3)</code>	<code>m.log(3)</code>
<code>math.sin(math.pi/6)</code>	<code>sin(pi/6)</code>	<code>m.sin(m.pi/6)</code>

La documentation

En tapant `help(commande)`, on obtient des informations détaillées sur l'usage et le fonctionnement d'une commande donnée. De même, `help(bibliotheque)` détaille le contenu d'une bibliothèque donnée. Avec un type ou un objet, on a la liste des fonctions associées.

Exemples : essayer `help(math.log)`, `help(math)` et `help(5.0)`.

Problèmes de précision

Les nombres à virgule flottante sont des approximations des nombres réels, si bien que **beaucoup de calculs sont intrinsèquement faux sous Python**.

Exemples : calculs de $1 - 0.2 - 0.2$ et de $321^{100}/32.1^{100}$.

On peut remédier à ce problème en utilisant des nombres rationnels. Pour cela, importer la fonction `Fraction` de la bibliothèque `fractions`, qui permet de créer le rationnel a/b en tapant `Fraction(a,b)`. On pourra convertir les nombres obtenus avec la commande `float()`.

1.3) Les complexes

Un nombre complexe sous forme algébrique $a + ib$ va se noter `a+bj` en Python. Attention, le nombre i est noté `1j` et non `j` ou `1*j` (ce qui permet d'éviter les confusions avec une variable `j`). On dispose, outre les opérations algébriques, de quelques fonctions de base :

<code>abs(z)</code>	<code>z.real</code>	<code>z.imag</code>	<code>z.conjugate()</code>
$ z $	$\text{Re}(z)$	$\text{Im}(z)$	$\bar{z}$

Exemples : taper dans une console `1+1j` ; `3-2j` ; `x+y*1j`.

2/ Le type booléen

Les booléens sont des objets qui permettent de représenter les résultats des évaluations d'expressions logiques. Ils ne prennent que les valeurs **True** ou **False** et sont de type **bool**. La commande `bool()` convertit tout objet non nul en **True** et le reste en **False**.

Exemple : taper `3>2` puis `5<4` dans une console ... et admirer le résultat.

On dispose sur ces booléens des opérations logiques de base (vues dans le cours de logique) :

or	and	not
ou	et	non

Pour créer des tests logiques, on a besoin des **opérateur de comparaison** :

<code>==</code>	<code>!=</code>	<code><</code> et <code>></code>	<code><=</code> et <code>>=</code>	<code>in</code>	<code>not in</code>
égalité	différence	inég. strictes	inég. larges	appartenance	non appartenance

3/ Les variables

3.1) Notion de variable

L'essentiel du travail d'un ordinateur consiste à manipuler des données. Elles peuvent être de divers types mais se ramènent toujours, dans la mémoire de la machine, à une suite finie de nombres binaires.

Pour y accéder, les langages de programmation utilisent des **variables** : ce sont des noms qui servent à désigner un emplacement précis de la mémoire vive. On peut alors y stocker une valeur, la lire ou la modifier.

Comme Python utilise un **typage dynamique**, il n'y pas besoin de préciser les types des variables ; elles peuvent même en changer en cours de programme (ce qui est à la fois un avantage et un inconvénient).

On essaye en général d'utiliser des noms simples mais suffisamment explicites, afin de faciliter la lecture (et l'écriture !) des programmes.

3.2) Noms de variables

Le nom d'une variable doit impérativement être composé de caractères alphanumériques et d'underscore `_`. De plus, il ne peut pas commencer par un chiffre.

Exemples : `PyThon_34` convient, mais pas `Python-34` ou `34_Python`.

Voici deux fonctions qui peuvent être utiles pour contrôler les types des variables :

<code>help(variable)</code>	affiche des informations détaillées (type de la variable, fonctions associées).
<code>isinstance(variable, type)</code>	regarde si la variable est du type précisé, et renvoie True ou False .

3.3) Affectations de variables

<code>variable = valeur</code>	affecte une valeur à une variable
<code>var1, var2, ... = val₁, val₂, ...</code>	affecte plusieurs valeurs à plusieurs variables.

Pour affecter une valeur à une variable, on tape `variable = valeur`. C'est simple, mais il faut faire attention à l'ordre et aux modifications effectuées en cours de programme. Par contre, **il ne faut pas confondre les affectations de variables et les comparaisons d'objets.**

Exemples : tester `x = 7` puis `x == 3` et `y = 4*x+3`.

On peut assigner une même valeur à plusieurs variables simultanément.

Exemple : `a = b = 5`

On peut aussi effectuer des affectations parallèles avec une seule instruction, ce qui permet par la même occasion d'échanger les contenus de certaines variables.

Exemples :

- ❶ `a,b,c = 5,3,8` affecte 3 valeurs à 3 variables.
- ❷ `a,b = b,a` permet d'échanger les contenus des variables.

3.4) Entrées et sorties

Lors de son déroulement, un programme a parfois besoin de demander à l'utilisateur des données ou d'afficher les résultats des opérations effectuées. Pour les messages, on utilise des **chaînes de caractères** : ce sont des portions de texte délimitées par des guillemets, que l'on étudiera en détail plus tard. Voici les instructions à connaître :

<code>print("Texte",variable,"Texte bis")</code>	permet d'afficher la valeur de variable au sein d'une phrase.
<code>print(a,end="")</code>	affiche a sans retour à la ligne, si bien que le prochain affichage se fera à sa suite.
<code>variable = input("Texte à afficher")</code>	affiche le texte, attend une entrée et l'affecte sous forme de texte à variable.
<code>variable = eval(input("Texte"))</code>	affiche le texte, attend une entrée et l'affecte, en conservant son type, à variable.

Attention, la fonction `input()` **transforme** toujours **ce qui est tapé en chaîne de caractères**. Pour conserver le type de l'entrée, on utilise la fonction `eval`. Si l'on veut être sûrs d'avoir un nombre entier ou réel, on force la conversion en mettant `int` ou `float` au lieu de `eval`.

Exercices

Exercice 1.

Dans cet exercice, il faudra taper Entrée après chaque instruction.

- 1) Taper `5+3` puis `5+3`. Observer et commenter les résultats.
- 2) Taper sur la flèche "haut". Que se passe-t-il ?
- 3) Taper `a=5` puis regarder l'explorateur de variables.
Taper ensuite `a` puis `print(a)`. Quelle est la différence ?

Exercice 2. Donner les types des nombres suivants :

- (a) 5 (b) 5. (c) 6/2 (d) 6//2

Exercice 3. Donner les résultats des instructions suivantes :

- (a) `8.5/2.5` (b) `int(8.5)/int(2.5)` (c) `int(8.5/2.5)`

Exercice 4. Calculer les expressions suivantes :

$$1 + 7 \quad -2 + 5 - 3 \quad 21 \times 17 \quad 2/7 \quad 2(5 + 7) \quad 23 \times 87/23 \quad 3^2 \quad 2^{1000}$$

Exercice 5. Évaluer les expressions suivantes :

$$\sqrt{2} \quad \cos(\pi) \quad \ln(e^2) \quad \sin\left(\frac{\pi}{2}\right) \quad e^4 \quad \log_{10}(732)$$

Exercice 6.

Calculer $1 - 0.2 - 0.2$ et $321^{100}/32.1^{100}$ utilisant des fractions.

Exercice 7. Calculer les expressions suivantes :

$$(1 + 2i)^3 \quad (1 + i)/(1 - i) \quad e^{i\pi/4} \quad |3 + 4i| \quad \operatorname{Re}\left(e^{-i\frac{\pi}{6}}\right)$$

Exercice 8. Tester la véracité des propositions suivantes :

- (a) $\cos^2\left(\frac{\pi}{4}\right) + \sin^2\left(\frac{\pi}{4}\right) = 1$ (b) $2 = 0$ ou $3 > 0$ (c) $2 = 0$ et $3 > 0$ (d) $2 \neq 0$

Exercice 9. Écrire des expressions booléennes traduisant les conditions suivantes :

- 1) L'entier n est divisible par 5.
- 2) L'un des deux entiers p et q est multiple de l'autre.
- 3) Les trois entiers m , n et p sont de même signe.

Exercice 10.

Taper le code ci-contre dans l'éditeur de texte et l'exécuter.

Utiliser le débogueur, ainsi que l'explorateur de variables, pour observer le déroulement du programme.

```
x=10
y=7
x=x+y
y=x
x=5/(x-y)
```

Exercice 11.

Écrire un programme qui demande la hauteur d'un phare (en m) et renvoie la distance à laquelle on voit depuis ce phare, en km et arrondie à un chiffre après la virgule.

On prendra $R_{\text{Terre}} = 6400$ km.

Exercice 12. Écrire un programme qui :

- 1) demande la taille en mètres ;
- 2) demande le poids en kg ;
- 3) affiche l'IMC.

Rappel : l'IMC se calcule en divisant le poids par le carré de la taille.

Exercice 13. Écrire un programme qui :

- 1) demande la distance parcourue (en mètres) ;
- 2) demande le temps de parcours (en secondes) ;
- 3) renvoie la vitesse en $\text{km} \cdot \text{h}^{-1}$, arrondie à un chiffre après la virgule.

Exercice 14. Écrire un programme qui :

- 1) demande à l'utilisateur une variable a et une autre b ;
- 2) affiche les valeurs des deux variables en les nommant ;
- 3) échange les deux valeurs ;
- 4) affiche à nouveau les variables en les nommant.

Exercice 15. Écrire un programme qui :

- 1) demande les coordonnées de deux points du plan ;
- 2) affiche la distance entre ces points.