

Chapitre 0 : Constructions Élémentaires

1 TYPES DE BASE

Les langages de programmation permettent de manipuler des **données** de différents **types**. Un type définit l'ensemble des valeurs possibles pour les données qu'il admet. On distingue les **types de base**, décrits dans ce chapitre, des **types structurés** décrits dans les chapitres suivants.

Les principaux types de base du langage Python sont **les nombres, les textes et les booléens**.

Python est un langage de programmation objet, **multi-paradigme et multi-plateformes**. Il est placé sous une licence libre et fonctionne sur la plupart des plates-formes informatiques, des supercalculateurs aux ordinateurs centraux, de Windows à Unix en passant par GNU/Linux, Mac OS, ou encore Android.

Il existe deux manières de coder en Python. La première, est d'écrire des fichiers de code source (dont l'extension est .py), puis d'exécuter ces fichiers. La seconde, est d'utiliser l'interpréteur de commandes.

1.1 LES NOMBRES

Python distingue les nombres entiers (**int** pour « integer » en anglais) des nombres à virgule flottante (**float**). Les nombres entiers s'écrivent de façon usuelle.

Les nombres à virgule flottante sont un sous-ensemble de nombres réels. En effet, un ordinateur ne peut pas représenter exactement tous les nombres réels. Python utilise les normes d'écritures anglaises, les décimales sont donc séparées de la partie entière par un point plutôt que par une virgule. Enfin, on peut utiliser la notation scientifique avec la lettre e ou E, 5000 peut s'écrire 5e3.

1.2 LES CHAINES DE CARACTERES

Les **chaines de caractères** (**string** en anglais) sont des textes entre guillemets simples ou doubles : 'Bonjour' ou « Bonjour ». L'utilisation d'un type de guillemets ou l'autre ne change pas la façon dont le texte va être traité ; cela permet par contre de mettre l'autre type de guillemets dans le texte.

« Il dit qu'il fait beau et s'en va. »

'Il dit : « il fait beau » et part.'

1.3 LES BOOLEENS

Les données de type **booléen** (**bool** pour « boolean » en anglais) ne peuvent prendre que deux valeurs Vrai et Faux, notées **True** et **False** en Python. Elles sont utilisées pour la prise de décision. Par exemple, si l'on compare deux entiers avec l'expression $2 < 3$, le résultat est **True** car 2 est inférieur à 3, et le résultat de $3 < 2$ est donc **False**.

2 VARIABLES ET AFFECTATION

Les données d'un programme sont rangées dans des **variables**. Chaque variable Python a un **nom**, qui ne doit pas contenir d'espace ni de caractères autres que des lettres, des chiffres et l'underscore `_`.

Un nom de variable ne peut pas commencer par un nombre : `2pi` sera rejeté par Python comme incorrect. Les majuscules sont différentes des minuscules : la variable `Taille` est différente de `taille` et de `TAILLE`.

Ainsi une variable est constituée :

- d'une valeur : par exemple le nombre qu'elle stocke.
- d'un nom : c'est ce qui permet de la manipuler.

```
>>> a = 5
>>> b = 32
>>> a,b = b,a # permutation
>>> a
32
>>> b
5
>>>
```

2.1 AFFICHER LE TYPE D'UNE VARIABLE

La fonction **type** renvoie le type de la variable passée en paramètre.

```
>>> type(3.4)
<class 'float'>
>>> type("un essai")
<class 'str'>
>>>
```

2.2 RECUPERER UNE SAISIE

La fonction **input()** interagit avec l'utilisateur : cette instruction interrompt le programme et attend la saisie de l'utilisateur. `input()` accepte un paramètre facultatif : le message à afficher à l'utilisateur.

```
>>> # Test de la fonction input
>>> annee = input("Saisissez une année : ")
Saisissez une année : 2009
>>> print(annee)
'2009'
>>>
```

Le type de la variable `annee` après l'appel à `input()` est une chaîne de caractères (apostrophes qui encadrent la valeur de la variable). Pour convertir une variable vers un autre type, il faut utiliser le nom du type comme une fonction.

Exemple :

```
# Programme testant si une année, saisie par l'utilisateur, est bissextile ou non

annee = input("Saisissez une année : ")      # saisie de l'année à tester

annee = int(annee)                            # Risque d'erreur si l'utilisateur n'a pas saisi un nombre

if annee % 400 == 0 or (annee % 4 == 0 and annee % 100 != 0):
    print("L'année saisie est bissextile.")
else:
    print("L'année saisie n'est pas bissextile.")
```

Remarques :

- La fonction `input()` renvoie toujours une chaîne de caractères. Vous pouvez ensuite convertir cette chaîne en nombre à l'aide de `int()` ou de `float()`.

3 EXPRESSIONS

Opération	Signe	Remarque
Addition	+	
Soustraction	-	
Multiplication	*	
Division	/	Division entière
Modulo	%	Reste de la division entière
Division euclidienne	//	Donne le quotient de la division entière
Egale	==	Test l'égalité
Différent de	!=	
and	&	Opération bit à bit
or		
xor	^	
Partie réelle	real	Méthode appliquée à un nombre complexe
Partie imaginaire	imag	Méthode appliquée à un nombre complexe

Attention à respecter les types lors des opérations : diviser deux entiers donnera un entier même si la variable qui reçoit le résultat est un flottant.

```
a, b = 5, 2 + 3j
c = float(a) / float(b.real) # au moins un des deux nombres doit être converti
print(c)
```

Comme pour les entiers, les calculs sur les flottants sont, naturellement, réalisés par le processeur. Cependant contrairement au cas des entiers où les calculs sont toujours exacts, les flottants posent un problème de précision. Cela n'est pas propre au langage python, mais est dû à la technique de codage des nombres flottants sous forme binaire.

Si le problème se pose bien en termes de nombres rationnels, il est alors tout à fait possible de le résoudre avec exactitude avec le module `decimal` qui permet de résoudre le problème.

```
from decimal import Decimal
```

```
Decimal('0.3') - Decimal('0.1') == Decimal('0.2')
```

4 POUR ALLER PLUS LOIN

4.1 RACCOURCIS D'ECRITURE

Il existe des techniques permettant de raccourcir l'écriture des opérations.

Au lieu d'écrire : `nombre = nombre + 1`

on écrira : `nombre += 1`

De même pour : `nombre = nombre - 1`

on pourra écrire : `nombre -= 1`

Il existe d'autres raccourcis qui fonctionnent sur le même principe qui fonctionnent pour toutes les opérations de base :

```
nombre = 9
```

```
nombre *= 5;           // ... nombre vaut 45
```

```
nombre /= 3;           // ... nombre vaut 15
```

```
nombre %= 2;           // ... nombre vaut 1 (car 15 = 2 x 7 + 1)
```

4.2 LA BIBLIOTHEQUE MATHEMATIQUE

En langage Python, il existe ce qu'on appelle des bibliothèques « standard », c'est-à-dire des bibliothèques toujours utilisables. Ce sont en quelque sorte des bibliothèques « de base » qu'on utilise très souvent.

Les fonctions intégrées au langage sont relativement peu nombreuses : ce sont seulement celles qui sont susceptibles d'être utilisées très fréquemment. Les autres sont regroupées dans des fichiers séparés que l'on appelle des modules.

Il existe un grand nombre de modules pré-programmés qui sont fournis d'office avec Python. Vous pouvez en trouver d'autres chez divers fournisseurs. Souvent on essaie de regrouper dans un même module des ensembles de fonctions apparentées que l'on appelle des bibliothèques.

Le module `math`, par exemple, contient les définitions de nombreuses fonctions mathématiques telles que sinus, cosinus, tangente, racine carrée, etc. Pour pouvoir utiliser ces fonctions, il vous suffit d'incorporer la ligne suivante au début de votre script :

```
from math import *
```

Cette ligne indique à Python qu'il lui faut inclure dans le programme courant toutes les fonctions (c'est là la signification du symbole `*`) du module `math`, lequel contient une bibliothèque de fonctions mathématiques pré-programmées.

Voici quelques fonctions principales :

fonction	description
<code>abs</code>	Cette fonction retourne la valeur absolue d'un nombre flottant, c'est-à-dire $ x $.
<code>ceil</code>	Cette fonction renvoie le premier nombre entier après le nombre décimal qu'on lui donne.
<code>floor</code>	C'est l'inverse de la fonction précédente : elle renvoie le nombre directement en dessous.
<code>pow</code>	Cette fonction permet de calculer la puissance d'un nombre.
<code>sqrt</code>	Cette fonction calcule la racine carrée d'un nombre.
<code>sin, cos, tan</code>	Ce sont les trois fonctions sinus, cosinus et tangente utilisées en trigonométrie. Ces fonctions attendent une valeur en radians.
<code>asin, acos, atan</code>	Ce sont les fonctions arc sinus, arc cosinus et arc tangente utilisées en trigonométrie.
<code>exp</code>	Cette fonction calcule l'exponentielle d'un nombre.
<code>log</code>	Cette fonction calcule le logarithme népérien d'un nombre.
<code>log10</code>	Cette fonction calcule le logarithme base 10 d'un nombre.

4.3 LES COMMENTAIRES

Un commentaire permet d'ajouter des annotations dans le codes source et de mieux vous expliquer à quoi peut servir telle ou telle ligne de code.

Il y a plusieurs manières d'insérer un commentaire selon la longueur du commentaire.

`# Ceci est un commentaire sur une ligne`

ou

`" Ceci est un commentaire sur une ligne "`

ou

`""" Ceci est`

une documentation

sur plusieurs lignes `"""`

4.4 TESTER POUR VALIDER

Aucun programmeur n'est capable d'écrire à coup sûr une fonction de plus de quelques lignes sans erreur. Pour s'assurer qu'un programme fonctionne comme prévu, il faut le **tester**. Pour cela, il ne suffit pas de l'exécuter et d'essayer un peu au hasard plusieurs données d'entrée et voir si le résultat est correct. Il faut tester chaque fonction avec un jeu de tests qui couvre le mieux possible les cas d'usage. Pour choisir les tests pertinents, il est utile de consulter la spécification et la réalisation de la fonction pour identifier :

- Les **cas typiques**, qui représentent des données d'entrée « normales » pour la fonction, comme un tableau de nombre tirés au hasard pour un algorithme de tri ;
- Les **cas limites**, qui correspondent à des données d'entrée qui satisfont les pré-conditions mais de manière limite, comme trier un tableau vide ;
- Les **cas interdits**, qui sont des données pour lesquelles la fonction n'est pas prévue et qui doit les rejeter via les pré-conditions, par exemple une recherche dichotomique dans un tableau non trié.

Une bonne approche du test est d'écrire *d'abord* le jeu de tests, *puis* la fonction à tester. Tant que les tests ne passent pas avec succès, on identifie les erreurs et on les corrige.