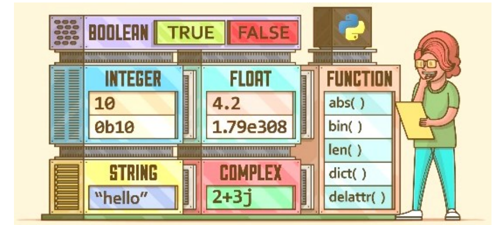
Cours – Les types de base



realpython.com

Objectifs :

- ⇨ Revoir la notion de type et les commandes associées
- ⇨ Revoir les différents types de base
- ⇨ Introduire la notion de type construit et les tuples

I - Objets et types

En python, toute variable fait référence à un *objet*[1]. On l'a vu quand on a décrit le fonctionnement de python avec l'espace des noms et l'espace des objets.

Les objets peuvent être très simples (ce sont les types de base que nous allons décrire ensuite) ou assez complexes (nous verrons plus tard des types d'objets plus avancés comme les dictionnaires). En programmation objet, on peut même définir de nouveaux objets.

A chaque objet est associé un type (ou *classe*) bien défini ce qui permet à l'interpréteur python de savoir comment le traiter.

1) La commande `type`

On peut récupérer le type d'un objet avec la commande `type`.

```
>>> a = 365
>>> type(a)
<class 'int'>
>>> type(12.3)
<class 'float'>
>>> type(False)
<class 'bool'>
>>> type("toto")
<class 'str'>
```

Types de base

```
>>> type([1,2,6])
<class 'list'>
>>> type(range(5))
<class 'range'>
>>> type(type(128))
<class 'type'>
```

Types plus complexes
Même le type d'un objet est un objet (de classe `type`)

[1] Terme informatique qui sera développé l'an prochain. Pour l'instant on peut considérer cela comme un élément d'information plus ou moins complexe.

Licence Creative Commons BY-SA

Application n°1 :

1) En utilisant la console python, déterminer le type d'une comparaison (par exemple a == 13), d'un module (comme turtle), et d'une fonction

2) Que renvoi l'instruction `type(print("toto"))` ? Essayez de devinner avant de l'exécuter et commentez le résultat.

2) Comment tester le type d'un objet ?

On peut tester le type d'un objet avec la commande `type` et le mot-clé `is` qui teste l'identité, ou avec la commande `isinstance(objet, type)` qui renvoi un booléen indiquant si l'objet `objet` est du type `type`. Il y a une légère différence entre ces deux commandes, mais il faudra attendre le cours de terminale pour l'apprécier[2]. En attendant on peut utiliser l'une ou l'autre syntaxe.

```
>>> type(62) is int
True
>>> isinstance(62,float)
False
```

II - Les différents types

1) Types de base

On a déjà vu les types de base dans le chapitre sur les variables :

Type	Nom en python	Description
Booléen (valeur binaire)		Deux valeurs possibles : `True` et `False` Utilisé dans les conditions (`if` / `elif` / `else` et `while`)
Entier relatif		En python 3.x les entiers sont de taille arbitraire (ils peuvent être aussi grands que la mémoire l'autorise, c'est-à-dire vraiment très très très grands)
Flottant (nombres à virgule)		Approximation des nombre réels Valeurs entre environ $2x10^{-308}$ et $2x10^{308}$
Chaîne de caractère		"Joliot-Curie" 'A' On peut utiliser les simples ou doubles cotes (mais il faut démarrer et finir la chaîne avec le même symbole).

Tous ces types sont non mutables (ou immuables), c'est-à-dire qu'on ne peut pas les modifier dans l'espace des objets. Si une variable de type `int` par exemple change de valeur, alors son nom va simplement changer de référence dans l'espace des objets (pointer vers une valeur différente).

[2] `isinstance` tient compte des héritages de classe alors que `type`… `is` ne tient compte que du type immédiat.

Licence Creative Commons BY-SA

2) Types complexes

On a vu quelques types plus complexes :

Type	Nom en python	Description
Module		C'est la représentation en mémoire d'un module (et tous ses objets) déclaré avec la commande `import`.
Fonction		Fonction déclarée avec le mot-clé `def`.

3) Types construits

Il y a enfin les types construits. Ce sont des types qui sont élaborés à partir des types de base. Nous en verrons pour l'instant seulement deux :

Type	Nom en python	Mutable ?	Description
Listes		Mutable	Tableau de taille modifiable pouvant contenir des données ordonnées de types divers.
Tuples ou n-uplets		Non-mutable	Type constitué de l'association ordonnée de plusieurs objets

Les tuples peuvent être vus comme des listes non modifiables (donc de taille et de contenu fixe). On peut les utiliser presque en tout point comme des listes.

Application n°2 :

1) Ecrire un programme qui définit un tuple contenant 3 nombres et 2 chaînes de caractère. Vérifier que sa longueur est bien de 5 (instruction `len`) et écrire le bout de code qui permet d'afficher une à une les valeurs stockées dans le tuple (deux façons possibles).

2) Modifiez le programme précédent afin que le tuple à afficher contienne 2 nombres, un booléen et la liste `[1, 2, 3]`. Faire ensuite en sorte que le programme modifie la liste contenue dans le tuple et affiche ce dernier. Que remarque-t-on ? Vous pouvez utiliser python tutor pour mieux comprendre le fonctionnement.

3) Changer pour que le programme modifie le premier nombre du tuple. Que remarquez-vous ?

Licence Creative Commons BY-SA

4) Mutable or immutable, that is the question !

Revenons un instant sur la notion de type mutable et non mutable qui est assez importante en python.

Un objet **mutable** est un objet dont la valeur peut changer au cours du temps.

Un objet **non-mutable** ou ***immuable*** est un objet dont la valeur ne peut plus changer après sa création.

Les nombres (`int`, `float`, …), les booléens (`bool`), les chaînes (`str`) et les tuples sont non mutables. Pourtant une variable `a` de type `int` peut changer de valeur…
A l'aide de pythontutor, nous allons essayer de comprendre ce qui se passe lorsqu'on modifie et passe en paramètre des variables mutable et immuables.

```
1  a = 5
2  c = a
3  b = "xyz"
4  t = [a, b, 0]
5  a = a + 1
6  def f(x, texte, tab):
7      x = x + 3
8      tab[2] = 'Z'
9      texte = texte + 'w'
10     return x
11 d = f(a,b,t)
12 print(t)
```

Application n°3 :
Copier-coller le programme ci-contre dans pythontutor (attention : pour bien voir ce qui se passe, il faut avoir sélectionné « Render all objects on the heap (Python/Java) » dans les options de visualisation) et visualiser son exécution puis compléter les phrases récapitulatives suivantes.

Modification :

Lorsqu'une instruction modifie une variable de type immuable, en fait la valeur ....,

mais le nom de variable va pointer vers une ....de l'espace des objets (cas de la variable

`a` ligne 5 ou `texte` ligne 9 dans le programme précédent).

Si on modifie une variable de type mutable, c'est .... qui est changé (la case d'indice 2 du tableau `t` change car elle pointe vers une autre valeur après l'exécution de la ligne 8).

Passage en argument :

Lorsqu'on passe une variable de type immuable en argument à une fonction, on peut être sûr que celle-ci ....

.... (cas des variables `a` et `b` à l'exécution de la ligne 11).

Si on passe une variable de type mutable à une fonction, celle-ci .... par le code de la fonction (cas du tableau `t` après l'exécution de la fonction `f`).

Licence Creative Commons BY-SA

III – Sens des opérateurs

Suivant le type de données sur lesquels ils agissent, les différents opérateurs n'ont pas le même sens.

Application n°4 :
En utilisant la console python ou un petit programme, faites des tests pour compléter le tableau suivant. A côté de chaque opérateur on indiquera l'action réalisée par python (ex : addition, concaténation, …) ou une barre oblique « / » si cette opération n'est pas autorisée par python.

	`int`	`float`	`str`	`list` (ou `tuple`)
`int`	`+` : `-` : `*` :	`+` : `-` : `*` :	`+` : `-` : `*` :	`+` : `-` : `*` :
`float`		`+` : `-` : `*` :	`+` : `-` : `*` :	`+` : `-` : `*` :
`str`			`+` : `-` : `*` :	`+` : `-` : `*` :
`list` (ou `tuple`)				`+` : `-` : `*` :

Références :
Pour aller plus loin sur les types mutables et immuables : https://www.mygreatlearning.com/blog/understanding-mutable-and-immutable-in-python/

Licence Creative Commons BY-SA