



Chapitre : Types et Instructions de Bases

- *Author* : Ibrahima SY
- *Email* : syibrahima31@gmail.com
- *Github* : [Cliquez](#)
- *Linkdin* : [Cliquez](#)
- *School* : Institut Supérieur Informatique (ISI)
- *Spécialité* : Licence 2 GL

PLAN

1. Introduction au Typage Dynamiques

2. Type Numérique

3. Chaînes de Caractères

4. Les Séquences

5. Listes

6. Introduction aux fonctions

7. Introduction aux modules

Introduction au Typage Dynamique

- En python tout est un objet
- Un objet est un morceau de code qui va contenir des données.
- En plus il contient un ensemble de mécanisme qui permettent de manipuler ces données qu'on appelle méthodes
- Les objets ont tous un type
- Le type est le comportement par défaut qui va être défini pour les objets

Introduction Typage Dynamiques

Nommer / Référence les objets

Pour référence un objet, il faut affecter un objet à un `nom` de variable avec une notation qu'on appelle la notation d'affectation

```
moyenne = 12
```

Le nom des variables en Python peut être constitué de lettres minuscules (`a` à `z`), de lettres majuscules (`A` à `Z`), de nombres (`0` à `9`) du caractères souligné (`_`)

```
age = 15    # bon
Age = 20    # bon
_moy = 98   # bon
age1 = 30   # bon
age_1 = 32  # bon
1age = 32   # c'est pas bon
```

Introduction Typage Dynamiques

Nommer / Référence les objets

Il est impossible d'utiliser les mots clés du langage (ci-dessous) pour nommer des variables.

Introduction Typage Dynamiques

Typage dynamique

L'espace de variable, s'appelle un espace de nommage .

Pour `a = 3` , Python va réaliser 3 opérations :

1. Python crée l'entier 3 dans l'espace des objets
2. Python crée une variable a dans l'espace des variables
3. Python crée une référence entre la variable a et l'entier 3

- si `a = 'data science'` python déréférence l'objet `3` pour que la variable `a` référence désormais l'objet `'data science'`
- Le `typage dynamique` veut dire qu'en Python, le `type` n'est pas lié à la variable qui référence l'objet mais à l'objet .
- `del a` permet de supprimer la variable `a` de l'espace des variables.
- Si l'objet n'a plus de référence, un mécanisme qui s'appelle mécanisme de `garbage collector` libère la mémoire de l'ordinateur.

Type Numérique

En python il existe 4 types numériques :

- `int` -> entier
- `float` -> nombre décimaux / réels
- `complex` -> nombres complexes
- `bool` -> les booléens (qui sont un sous ensemble des int)

Le type `int`

On peut vérifier le type avec la fonction built-in `type()`

```
i = 1
i * 2
i + 2
i - 3
i = i + 5
```


Les entiers en Python sont des objets de précision illimitée. Python n'a aucun problème à faire des opérations sur des entiers extrêmement grands.

Le type `float`

Ce type est utilisé pour stocker des nombres à virgule flottante, désignés en anglais par l'expression `floating point numbers`. Pour cette raison, on appelle ce type : `float`. En français, on parle de flottant.

```
f = 4.6
```

Le type `Booléen`

Les booléens sont le résultat d'opérations logiques et ont deux valeurs possibles : `True` ou `False`. Il est d'ailleurs codé comme un entier. `False` étant l'entier `0` et `True` étant l'entier `1`.

Le type complex

La constante complexe que nous notons i en français, se note j en Python.

Un nombre complexe, c'est 2 nombres float mis l'un à côté de l'autre

```
c = 3 + 4j
```

On peut additionner n'importe quel type numérique. Python se charge de faire la conversion pour nous. Lors de la conversion, on peut avoir une perte de précision.

On peut convertir des nombres à l'aide de fonctions built-in :

1. `int()` convertit en entier (l'opération s'appelle la troncature)
2. `float()` convertit en float
3. `complex()` convertit en complex
4. `str()` convertir en chaîne de caractère

Les Opérateurs

Operator	Name	Example
+	Addition	$x + y$
-	Subtraction	$x - y$
*	Multiplication	$x * y$
/	Division	x / y
%	Modulus	$x \% y$
**	Exponentiation	$x ** y$
//	Floor division	$x // y$

Les Chaînes de caractères

Pour créer une chaîne de caractère , il faut l'entourer soit par des apostrophes ' ', soit par des guillemets " "

Les chaînes de caractères sont des objets immuables

```
# Premier méthode  
'cours de python'  
  
# Deuxième méthode  
"cours de python"
```

Les chaînes de caractères contiennent également de nombreuses méthodes
les fonctions qui manipulent les chaînes de caractères retournent un nouvel objet chaîne de caractères.

Pour accéder à l'intégralité des méthodes qui existent sur un objet particulier, on utilise la fonction built-in `dir()` .

La méthode `str.title()`

Cette méthode met la première lettre de chaque mot en majuscule.

```
# Premier méthode  
'je suis ingénieur en génie logiciel'.title()
```

La méthode `str.replace()`

Cette méthode permet de remplacer un mot de la chaîne

```
'je suis ingénieur en génie logiciel'.replace('logiciel','civil')
```

La méthode `str.isdecimal()`

Un certain nombre de méthodes de chaînes de caractères permettent également de faire des comparaisons ou de faire des tests. Ainsi, si on prend la chaîne '123' et que l'on souhaite la convertir en entier, on pourrait vouloir s'assurer que cette chaîne de caractères représente bien un nombre décimal.

```
'123'.isdecimal()
```

La méthode `str.format`

Cette méthode permet d'insérer des objets à l'intérieur d'emplacements spécifiés dans la chaîne de caractères par des accolades :

```
age = 25  
'Ibou a {} ans '.format(age)
```

Les séquences

Les séquences sont un ensemble de types regroupant les `list`, les `tuple`, les `str` et les `bytes`.

Définition de séquence

Une séquence en Python est un ensemble fini et ordonné d'éléments indicés de `0` à `n-1` si on a `n` éléments.

Opérations de base sur une séquence "chaîne de caractère"

- L'opération `s[0]` permet d'accéder au premier élément de la séquence, c'est-à-dire 'e'
- Pour connaître le nombre d'éléments d'une séquence, il faut utiliser la fonction built-in `len()`

- Toutes les séquences supportent également le test d'appartenance, qui est une opération très puissante en Python. `'egg' in s` veut dire est-ce que 'egg' est dans s. L'opération renvoie `True`. De même on peut faire `'egg' not in s` qui est le test de non-appartenance et qui retourne ici `False`.
- Ensuite on peut faire de la concaténation de séquence. `s + ', and beans'` rajoute `', and beans'` à notre séquence et renvoie donc `'egg, bacon, and, beans'`. Cette concaténation produit une nouvelle chaîne de caractères puisqu'une chaîne de caractère n'est pas `mutable`.
- L'opération `index()` permet de trouver la première occurrence par exemple de la lettre g : `s.index('g')` renvoie `1` puisque la première occurrence de la lettre `g` c'est la deuxième lettre à la place `1`.

Opérations de base sur une séquence "chaîne de caractère"

- L'opération `count()` permet de compter le nombre d'éléments, ici le nombre de `'g'` que l'on a dans
- On peut également appliquer les fonctions built-in `min()` et `max()` qui retournent le minimum et le maximum d'une séquence. `min(s)` renvoie `' '` et `max(s)` renvoie `'o'`. Comme notre séquence est une chaîne de caractère, le `min` et le `max` utilisent l'ordre lexicographique : l'espace pour le minimum et la lettre `o` pour le maximum.
- L'opération de Shallow copy permet de copier un certain nombre de fois un élément : `'x'*30` produit 30 fois `'x'`. Nous verrons par la suite que cette opération produit en fait une shallow copy qui peut avoir des effets de bord lorsque la séquence multipliée n'est pas un `immutable` mais un objet de type mutable.

L'opération de slicing

L'opération de slicing est valable pour toutes les séquences (chaînes de caractères aussi bien que liste). Pour commencer nous créons une chaîne de caractère `s = 'egg, bacon'`

Liste

- La liste représente un type extrêmement souple et puissant. Une liste est une séquence d'objets hétérogènes.
- Elle ne stocke que des références vers les objets. Par conséquent, la taille de l'objet liste est indépendante du type d'objets qui sont référencés.
- La liste c'est un objet mutable . Cela veut dire qu'on peut le modifier là où il est stocké.
- L'avantage de cette mutabilité c'est qu'on n'a pas besoin de faire une copie de l'objet pour le modifier.

L'opération de slicing

- Toutes les opérations applicables aux séquences sont applicables aux listes .
- Prenons une liste mêlant une variable `i = 4` , une chaîne de caractère `'spam'` , un flottant `3.2` et un booléen `True` .

```
liste = ["spam", 1, 3.2, True]
# accede au premier element de notre liste
liste[0]

#On change la premiere valeur de objet liste
liste[0] = liste[0] + 1

#le nombre d' element de la liste
len(liste)

# test appartenance
"spam" in liste
```

- On peut également réaliser des opérations de slicing sur une liste. `liste[1:3]` prend tous les éléments de 1 inclus à 3 exclu .
- On peut aussi faire des opérations d'affectation sur des slice `liste[1:3] = [1,2,3]`
- Cette opération d'affectation sur un slice est un moyen très simple d'effacer des éléments dans une liste. Pour `a=[16,1,2,3,True]` , si on entre `a[1:3] = []` , les éléments entre 1 inclus et 3 exclu sont effacés.
- On peut également utiliser l'instruction `del` pour enlever des éléments dans un slice. Avec `del a[1:2]` , l'élément à l'indice 1 est effacé.

```
i = 4 # creation de la variable
```

```
# creation de la liste
```

```
liste = [i, "spam", 3.2, True]
```

```
#prend tous les elements de 1 inclu a 3 exclu
```

```
liste[1:3]
```

```
#les éléments de 1 inclu à 3 exclu sont rempalcés par la séquence [1,2,3]
```

```
# pour enlever des éléments dans une liste
```

```
del liste[1:2]
```

Opérations `list.append()` et `list.extend()`

- L'opération `list.append()` ajoute un objet à la fin d'une liste.
- L'opération `list.extend()` prend une séquence et ajoute chaque élément de cette séquence à la fin de la liste. C'est l'équivalent de `list.append()` sur a chaque élément de la séquence.

```
L = [1,2,True ]
```

```
# utilisation de la méthode append  
L.append(3)
```

```
# utilisation de la méthode extend  
L.extend([2,3])
```

Opération de tri `list.sort()`

Pour une liste `a = [1,5,3,1,7,9,2]` , on peut appeler la méthode `list.sort()` qui va trier les éléments de la liste.

ATTENTION! `list.sort()` fonctionne en place

```
a = [1,5, 3, 1, 7 , 9, 2 ]  
a.sort()
```


Transformer une chaîne de caractère en liste et vice versa

- Il existe des opérations permettant de passer d'une chaîne de caractère à une liste et d'une liste à une chaîne de caractère. C'est une opération fréquemment utilisée pour accéder à des fichiers que l'on veut traiter avec python.

Pour commencer, on crée une chaîne de caractères `s = 'spam egg beans'`.

L'objectif est de séparer cette chaîne de caractère en colonnes. Pour cela, on utilise la fonction built-in `split()` qui est une fonction des chaînes de caractères

- On peut retransformer la liste en chaîne de caractères avec la syntaxe : `"`
`.join(list)`. L'espace entre guillemets avant `.join()` désigne le séparateur que `.join()` va mettre entre chaque élément.

```
"spam, egg bacon".split() # separe une chaine de caractere en liste  
"spam, egg bacon".split() # on peut aussi specifier un séparateur (" ", "/", ...)  
" ".join(["spam", "egg bacon"]) # on tranforme une liste en chaine de caractère
```

Introduction aux fonctions

Définition d'une fonction

La déclaration d'une fonction en python est très simple on utilise la syntaxe suivante :

```
def nom_fonction(parameters):  
    # corps de la fonction
```

- `def` : pour spécifier qu'on définit une fonction
- `Nom_de_la_fonction` : porte le nom de la fonction
- `Corps de la fonction` : contient les blocs instructions
- `return` : permet de retourner une valeur de la fonction

Exemple de fonction qui calcul le carré d'un nombre passé en argument

```
def carre(x):  
    return x**2
```

Introduction aux modules

Un module est un fichier python qui finit en .py. Lorsqu'on importe ce fichier, cela crée un objet module.

L'idée des modules, c'est de mettre des opérations similaires dans le même fichier.

Un module est une sorte de boîte à outil que l'on importe quand on a besoin de l'ouvrir.

Pour importer un module, on utilise l'instruction `import` suivie du nom du module. Exemple avec le module `random` :

```
import random
```

On peut accéder à tous les objets d'un module avec la fonction `dir()`.

```
dir(random)
```

vec `help(random)`, Python affiche toute l'aide liée à `random`. `help()` est très pratique pour regarder le fonctionnement d'une méthode particulière issue d'un module :

`help(random.randint)` nous indique que la méthode `randint()` renvoie un entier au hasard dans `[a, b]` en incluant `a` et `b`.

```
help(random)  
help(random.randint)
```

Introduction aux modules

Avec la librairie standard on peut :

- Faire de la programmation parallèle ou de la programmation asynchrone
- Faire de la persistance de données, une opération de sérialisation
- pour garder une copie des objets sur le disque dur
- Faire de la communication sur Internet. Python supporte quasiment tous les protocoles classiques sur Internet
- Formater les données et lire des formats spécifiques de données sur Internet
- Manipuler des fractions ou des nombres décimaux
- Écrire des expressions régulières
- Gérer des dates et des calendriers

- Interagir avec le système de fichiers, créer des fichiers, des répertoires, parcourir des répertoires
- Faire de la compression de fichiers
- Écrire des interfaces graphiques

En plus de sa librairie, Python a également des centaines de milliers de modules écrits par des groupes de développement ou par des individus que l'on peut charger et importer notre programme comme **Django** pour faire du développement Web