Introduction à Python

C. Fonlupt
Inspiré du tutoriel de Shubin Liu, Ph.D, UNC
et du livre Python3 (Dunod)

Sommaire

- Introduction
- Exécuter Python
- Programmation Python
  - Type de données
  - Contrôle du flux
  - Classes, fonctions et modules
- Exercices

Quelques éléments

- Manipuler rapidement python et savoir écrire des scripts pour le cours de Data Science
- Introduction aux modules comme numpy, scikit-learn

Python ?

- Langage orienté-objet comme Java
- Langage interprété ≠ langage compilé
- Typage des données dynamiques (pas de déclaration de type)
- Disponible sur de nombreuses plateformes
- Actuellement python 3 (3.8 en 2020)
- Grammaire relativement simple entre C et Java
- Gestion automatique de la mémoire
- open source https://www.python.org

Bref historique

- Naissance de Python Dec 1989
  - Par Guido van Rossum, maintenant chez GOOGLE
- First public release (USENET) - Feb 1991
- Site python.org - 1996 or 1997
- 2.0 sortie en 2000
- Python Software Foundation - 2001
- 3.0 en 2008/2009
- Les 2 versions coexistent mais la 3 vise à terme le remplacement de la 2

2 vs 3

- D’après wikipedia
  - Fusion des types 'int' et 'long'
  - Les chaînes sont en Unicode par défaut, 'bytes' remplace l'ancien type 'str'
  - Utilise des itérateurs plutôt que des listes là où c'est approprié (ex : dict.keys())
  - a/b est la vraie division par défaut
  - exec et print deviennent des fonctions
  - None, True et False deviennent des mots clé
  - Le fichier __init__.py n'est plus nécessaire pour les modules Python
  - `x` et l'opérateur <> disparaissent
  - De nombreuses fonctions disparaissent : apply(), buffer(), callable()...
  - reduce() disparaît au profit des boucles explicites

Quelques propriétés

- Tout est objet en python
- Modules, classes et fonctions
- Gestion des exceptions
- Typage dynamique et polymorphisme
- Surcharge des opérateurs
- Les blocs sont définis par l'indentation

Types de données

- Nombres : int, float, complex
- bool : True et False
- Chaînes de caractères : immutables
- Listes, dictionnaires et tuples : containers
- Bytes et bytesarray

Utilisation de python?

- Administration système (i.e., scripting)
- Graphic User Interface (GUI)
- Programmation internet
- Programmation de base de données
- Traitement de textes
- Traitement de données (Data Science)
- Opérations numériques
- IA
- ...

Python vs. Perl

- Apprentissage plus aisé
  - Surtout pour les utilisateurs occasionnels
- **Code plus lisible**

```
@P=split//,".URRUU\c8R";@d=split//,"\nrekcah xinU / lreP rehtona tsuJ";sub p{
@p{"r$p","u$p"}=(P,P);pipe"r$p","u$p";++$p;($q*=2)+=$f=!fork;map{$P=$P[$f^ord
($p{$_})&6];$p{$_}=/ ^$P/ix?$P:close$_}keys%p}p;p;p;p;p;map{$p{$_}=~/^[P.]/&&
close$_}%p;wait until$?;map{/^r/&&<$_>}%p;$_=$d[$q];sleep rand(2)if/\S/;print
```

- Plus sécurisé
- Intégration plus poussée avec java
- Moins orienté Unix

Python vs. Java

- Code en général plus concis
- Typage dynamique
- Développement plus rapide
  - Pas de compilation
  - Peut-être moins adapté que Java pour les grands projets
- Plus lent que java

- Python avec Java : Jython!

- Introduction
- Exécuter Python
- Programmation de Python
  - Type de données
  - Contrôle du flot
  - Classes, fonctions et modules
- Les modules

Exécution interactive de python

- Démarrage de par "python3"
  - Python 3.6.9 (default, Jul 17 2020, 12:50:27)
    [GCC 8.4.0] on linux
    Type "help", "copyright", "credits" or "license" for more information.
- ^D (control-D) sortie
  - % python3
  - >>> ^D
  - %
- Les commentaires en python commencent par '#'
  - >>> 2 + 2 # des commentaires
  - 4
  - >>> 7/3 # résultat réel
  - 2.333333
  - >>> 7 // 3
  - 2
  - a = 5
  - c = b = a # assignation multiple

Exécution de programmes python

- En général
  - % python myprogram.py
- Il est possible d'exécuter des scripts python
  - % emacs myprogram.py # Les programmes ont le suffixe .py.
  - Il suffit juste de taper le nom du programme
    - % python myprogram.py
- Encore mieux avec Linux (peut-être avec Windows ??)
  - #! /usr/bin/python
  - Il faut rendre le programme exécutable:
    - % chmod +x ./myprogram.py
  - Et on tape juste le nom du script
    - % ./myprogram.py

Python interactif

Soit le script `script.py` :

```
print("Hello world")
x = list(range(10))
```

- ```
  python3 -i ./script.py
  Hello world
  >>> x
  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
  ```

- ```
  $ python3
  >>> exec(open("./script.py").read()) # bel exemple de POO
  >>> x
  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
  ```

Python interactif

Reprenons `script.py`

- `python3`

  `>>> import script` # ne pas rajouter le suffixe .py suffix. Script est un *module* ici

  `>>> x`

  ```
  Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  NameError: name 'x' is not defined
  ```

  `>>> script.x` # pour utiliser x il faut indiquer
  # son module de provenance

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Python interactif

```
# Pour amener x au top-level
$ python3
>>> from script import x
Hello world
>>> x
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>
# Pour tout amener au top-level
from script import * → non recommandé
>>> from script import *
```

Convention de nommage

- Les fichiers pythons ont l'extension **`.py`**
- Mais en général les fichiers python exécutables n'ont pas l'extension **`.py`**
- Les modules ont toujours l'extension **`.py`**

Commentaires

- Avec # et jusqu'à la fin de la ligne
- Commentaires à la java sur plusieurs lignes ?
  - NON !

Python 3

- Introduction
- Exécuter Python
- Programmation de Python
  - Type de données
  - Contrôle du flot
  - Classes, fonctions et modules
- Les modules

Syntaxe

- Syntaxe très semblable à JAVA
- Exceptions :
  - pas d'opérateurs : `++`, `--`
  - pas de `{}` , pour les blocs utilisation de l'indentation
  - mots-clefs un peu différents
  - pas de déclarations de type !

Simple data types

- Nombres
  - Entier, réel, complexe !
- Chaînes de caractères
  - les caractères sont des chaînes de longueur 1
- Booleans `False ou True`

Numbers

- Notation classique des opérateurs
  - 12, 3.14, 0xFF, (-1+2)*3/4**5, abs(x), 0<x<=5
- Décalage et masque binaire
  - 1<<16, x&0xff, x|1, ~x, x^y
- Division entière
  - 1//2 -> 0 # 1 / 2 -> 0.5
- Précision importante, complexe
  - 2**1000 ->
    10715086071862673209484250490600018105614048117055336074437503
    88370351051124936122493198378815695858127594672917553146825187
    14528569231404359845775746985748039345677748242309854210746050
    62371141877954182153046474983581941267398767559165543946077062
    914571196477686542167660429831652624386837205668069376
  - 1j**2 -> (-1+0j)

Chaînes et formatage

```
i = 10
d = 3.1415926
s = "je suis une chaîne"
print ("%d\t%f\t%s" % (i, d, s))
print ("newline\n")
print ("no newline")
et depuis la version 3.6 les f-strings
>>> x = 5
>>> print(f"x a la valeur {x} ici")
x a la valeur 5 ici
beaucoup d'options de formatage
   https://docs.python.org/fr/3/tutorial/inputoutput.html
```

Chaînes

- "ing"+" 3" "ing 3" # concaténation
- "ing 3"*3 'ing 3ing 3ing 3' # répétition
- "ing 3"[0] "i" # indexation
- "ing 3"[-1] "3" # -1 index du dernier élément -3 → g
- "ing 3"[1:4] "ng " # découpage
- len("ing 3") 5 # taille
- "ing 3" > "ing 1" True # comparaison
- "3" in "ing 3" True # **3 in "ing 3" → erreur**
- Chaîne sur plusieurs lignes '''
- Les chaînes sont immutables
- + comme Java pour les concaténer

Variables

- Pas de déclaration préalable
- Il faut les initialiser avant de les utiliser
  - sinon déclenchement d'une exception
- Les variables ne sont pas **typées**

```
if x == 5:
   greeting = "hello world"
else: greeting = 12**2
print (greeting)
```

- Tout objet python a une valeur
  - fonctions, modules, classes…

```
>>> len

<built-in function len>
```

Référence

- Comme en Java, on ne manipule que des références → danger
  - x = y x et y font référence au même objet
- Exemple :

```
>>> a = list(range(10))
>>> b = a
>>> a.append(10)
>>> b
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Entrées-Sorties de base

- La fonction input() réalise une saisie au clavier
  - Retourne toujours un objet texte

```
a = input()
17.23
>>> type(a)
<class 'str'>
```

  - Conversion de type

```
>>> f = float(a)
>>> type(f)
<class 'float'>
```

  - Utilisation i = input("Entrez une valeur : ")

- Explications sur le " " "

```
>>> s = """ une très longue chaîne
sur plusieurs
lignes"""
>>> s
' une très longue chaîne\nsur
 plusieurs\nlignes''
 >>> len(s)
 44
```

Méthodes pour les chaînes

On est en **OO**

- upper()
- lower()
- capitalize()
- count(s)
- find(s)
- rfind(s)
- index(s)

- strip(), lstrip(), rstrip()
- replace(a, b)
- split()
- join()
- center(), ljust(), rjust()

```
>>> s = "une chaîne de caractères"
>>> s.split(' ')
['une', 'chaîne', 'de', 'caractères']
```

Testez ces méthodes

Containers standards : listes

- Liste :
  - Une liste est une collection mutable ordonnée d'éléments éventuellement hétérogènes
  - Définie entre crochets , séparée par des virgules

```
a = [1, 2, 3, 4, "chien"]
print (a[1])  #  2
une_liste = []
une_liste.append("chat")
une_liste.append(12)
print (len(une_liste))  # 2
```

Liste

- a = [1, 2, 3, 4, "chien"]

- Mêmes opérateurs que pour les chaînes
  - a+b, a*3, a[0], a[-1], a[1:], len(a)
- Affectation par morceau
  - a[0] = 85
  - a[1:2] = [7, 8, 9]

    -> [85, 7, 8, 9, 3, 4, 'chien']
  - del a[-1] # -> [85, 7, 8, 9, 3, 4]

Opérateurs utiles

```
>>> a = range(5)                # [0,1,2,3,4]
>>> a.append(5)                 # [0,1,2,3,4,5]
>>> a.pop()                     # [0,1,2,3,4]
5
>>> a.insert(0, 5.5)            # [5.5,0,1,2,3,4]
>>> a.pop(0)                    # [0,1,2,3,4]
5.5
>>> a.reverse()                       # [4,3,2,1,0]
>>> a.sort()                    # [0,1,2,3,4]
```

Opérations sur les listes

- append
- insert
- index
- count
- sort
- reverse
- remove
- pop
- extend

- indexation e.g., L[i]
- Découpage e.g., L[1:5]
- Concaténation e.g., L + L
- Répétition e.g., L * 5
- Test d'appartenance in L e.g., ‘a’
- Longueur e.g., len(L)

Listes imbriquées

- Liste dans une liste
- E.g.,
  - >>> s = [1,2,3]
  - >>> t = ["a", s, "b"]
  - >>> t

    ['a', [1, 2, 3], 'b']

    >>> t[1]

    [1, 2, 3]

Attention

```
>>> a = [1, 2, 3]
>>> b = a
>>> b[1] = 5
>>> a
[1, 5, 3]
>>> b = a*2
>>> b
[1, 5, 3, 1, 5, 3]
>>> a[1] = 2
>>> a
[1, 2, 3]
>>> b
[1, 5, 3, 1, 5, 3
```

Tuples

- Qu'est-ce qu'un tuple?
  - Un tuple est une collection ordonnée immutable d'éléments éventuellement hétérogènes
  - Exemple :

```
>>> t = ()
>>> type(t)
<class 'tuple'>
>>> t = (1, 2, 3)
>>> t = (1, )
>>> t = 1,
>>> a = (1, 2, 3, 4, 5)
>>> print (a[1])  # 2
```

Opérations sur les tuples

- Indexation T[i]
- Découpage T[1:5]
- Concaténation T + T
- Répétition T * 5
- Test d'appartenance 'a' in T
- Longueur len(T)

Liste vs. Tuple

- Des caractéristiques communes
  - permettent le stockage d'objets hétérogènes
  - Sont une suite ordonnée d'objets
- Mais des différentes
  - Tuple **modification impossible**
  - Tuple pas de méthodes
  - Tuples plus **rapide**

Dictionnaires

- Dictionnaires : { }
  - Permet de stocker des couples (clé : valeur)
  - On parle de tables de hachage
  - Les couples sont séparés par des ",", les valeurs par " :"
  - Tout type est accepté
    - {3:"pommes", 8:"yaourths", ("hello","world"):1, 42:"infini"}

```
>>> d
{3: 'pommes', 8: 'yaourths', ('hello', 'world'): 1, 42:
  'infini'}
>>> type(d)
<class 'dict'>
>>> d[8]
'yaourths'
>>> d.keys()
dict_keys([3, 8, ('hello', 'world'), 42])
```

Quelques méthodes

- keys()
- values()
- items()
- has_key(key)
- clear()
- copy()
- get(key[,x])
- popitem()
- ...

Quelques précisions

- Les clefs sont **immutables** :
  - nombres, chaînes et tuples sont immutables
  - **pas** les listes ou les dictionnaires
    - qui sont des objets mutables
- Attention, pas d'ordre, les clefs ne sont pas **ordonnées**
  - → algorithme de hachage

Ensemble

- Un ensemble est une suite non ordonnée d'éléments **uniques** :
  - Les éléments sont séparés par des virgules et entourés { }
  - De nombreuses méthodes disponibles
  - Exemple :

```
>>> chiffres = set(range(10))
>>> chiffres
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
>>> type(chiffres)
<class 'set'>
>>> chiffres.add(8)
>>> chiffres
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
>>> chiffres.add(10)
>>> chiffres
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

Résumé

- Entiers : 2323, 3234
- Réels : 32.3, 3.1E2
- Complexes : 3 + 2j, 1j
- Listes: l = [1,2,3]
- Tuples : t = (1,2,3)
- Dictionnaires : d = {‘hello’ : ‘there’, 2 : 15}

Résumé

- Listes, Tuples, et dictionnaires peuvent stocker tout type
- Seules les listes et dictionnaires sont mutables
- Attention : les variables sont des références

Quelques éléments sur les fichiers

- Ouverture est fermeture des fichiers en mode texte

  f1 = open("monFichier", "r,", encoding="utf8")

  - r en lecture, w en écriture, a en ajout
  - encoding utf8 à privilégier
  - latin1 disponible pour les pages HTML

- N'oubliez pas de fermer les fichiers
  - f1.close()

Lecture séquentielle

- Lire un fichier en entier :

```
s = f.read() # lit tout le fichier → chaîne
```

```
s = f.readlines() # liste de chaînes
```

- Lecture partielle

```
s = f.read(3) # lit 3 caractères
```

```
s = f.readline() # lit une ligne
```

Travailler avec fichiers et répertoires

- Module dédié
  - Path et PurePath de pathlib
  - Module os très performant https://docs.python.org/fr/3/library/os.html
- Exemple :
  >>> import os
- >>> os.getcwd()
- '/home/fonlupt'
- >>> os.chdir("/home/fonlupt/Téléchargements")
-

  '/home/fonlupt/Téléchargements'

  >>> f1 = open("notes.txt", "r", encoding="utf8")

  >>> s = f1.readline()

  >>> s

  "(31/01/2018). C'est une société ne comportant qu'un salarié jusqu'au\n"

Contrôle du flux (1)

- Pour identifier les instructions composées, Python utilise la notion d'indentation significative
- les ":" à la fin d'une ligne pour une instruction composée
- **if**, **if/else**, **if/elif/else**

```
if a == 0:
    print "zero !"
elif a < 0:
    print "négatif !"
else:
    print "positif !"
```

Contrôle du flux (2)

- Boucles while

```
a = 10
while a > 0:
    print (a)
    a -= 1
```

Contrôle du flux (3)

- Boucles for
- Attention : un peu différent du for classique

```
for a in range(10):
  print (a)
for a in "toto":
  print (a)
for a in [1, 2, 3, 4]:
  print (a)
```

Exemple

- Utilisation pour lire un fichier

```
f = open(fichier, "r", enconding="UTF8")
while True:
    line = f.readline()
    if not line:
        break
    print(line.upper())
```

Contrôle du flux

- `continue` **comme** en C (passe la boucle à son itération suivante)
- `pass` :

```
if a == 0:
    pass  # à écrire ultérieurement
else:
    # autre chose
```

Quelques mots sur la POO

- Il s'agit juste ici de faire quelques rappels → allez voir votre cours
- classe – le moule permettant de créer des objets
- instance – un objet particulier de la classe (une instance de la classe)
- méthode – une fonction faisant partie de la classe et agissant sur une instance de la classe
- constructeur – méthode spéciale permettant de créer l'instance

Création d'une classe

```
class MaClasse:

    """Cette classe crée des  objets."""

    def __init__(self, valeur):

        """Constructeur de la classe."""

        self.valeur = valeur

    def printme(self):

        """Affiche l'objet."""

        print ("valeur = ",self.valeur)
```

constructeur

méthode

Utilisation de la classe

```
t = MaClasse(10)   # appel au constructeur __init__
t.printme()        # affiche "valeur = 10"
```

- **t** est une instance de la classe **MaClasse**
- **printme** est une méthode de la classe
- **__init__** est le constructeur de la classe **MaClasse**
  - La méthode __init__ est appelée automatique à la création
- Vous vous doutez que les méthodes débutant par __ sont "spéciales"

utilisation d'une classe (2)

```
print (t.valeur)  # affiche "10"
```

- **valeur** est une *attribut* de la classe

```
t.valeur = 20   # change la valeur, pas
  d'attribut privé
print (t.valeur)  # affiche "20"
```

Les méthodes spéciales

- Elles débutent et finissent avec __ (deux underscores)
- Un peu en deça du cours
- On peut les utiliser pour la surcharge des opérateurs
  - `__add__, __sub__, __mult__, ...`
  - Voir la documentation de python

Définition de fonctions

```
def calcul(x):
    y = 10 * x + 2
    return y
```

- Toutes les variables de la fonction sont locales sauf présence du mot-clef **global**
- Les arguments sont passés par valeur

Exécution de fonction

```
def calcul(x):
    y = 10 * x + 2
    return y

print (calcul(10)) # 102
```

Les Modules

- Les modules sont des variables, des fonctions et des classes (ie du code python)
- Deux commandes from ou import
  - from module import function
  - Function() → la fonction est au top-level
  - import module
  - module.function() → la fonction doit être préfixée par le nom du module
- Les modules sont des espaces de noms

Exemple

- L'importation de modules est une des briques de construction de python

```
import math

print (math.sqrt(2)) # 1.4142135623730951
```

- Ou

```
from math import sqrt

print (sqrt(2))
```

Modules

- Ou (très déconseillé)

```
from math import *

print (sqrt(2.0))
```

- On peut importer plusieurs modules à la fois

```
import sys, string, math
```

- Un seul "`from x import y`" par ligne

Un module indispensable : NumPy

- http://numpy.org/
- NumPy est un module fondamental pour le calcul scientifique
- Manipulation de matrices
  - unitaire, zéros, identité
- Algèbre linéaire
  - Valeurs propres, vecteurs propres
  - Inverse
  - Déterminant
  - Résolution linéaire
- Appel "classique" import numpy as np

Manipulation de tableaux

- >>> a = np.ones((3,3),int)
- >>> a
  array([[1, 1, 1],
  [1, 1, 1],
  [1, 1, 1]])
  >>> type(a)
  <class 'numpy.ndarray'>
- >>> b = np.identity(3,int)
- >>> b

  array([[1, 0, 0],
  [0, 1, 0],
  [0, 0, 1]])
- >>> c = a + b # surcharge de +
- >>> print (c)
  [[2 1 1]
  [1 2 1]
  [1 1 2]]

Exemple : numpy

```
>>> a = np.identity(2, dtype=np.int16)
>>> a
array([[1, 0],
       [0, 1]], dtype=int16)
>>> b = 2*a # multiplication par un scalaire
>>> b
array([[2, 0],
       [0, 2]], dtype=int16)
>>> print(b)
[[2 0]
 [0 2]]
```

- >>> from LinearAlgebra import *
- >>> a = zeros((3,3),float) + 2.*identity(3)
- >>> print (inverse(a))
- [[0.5, 0., 0.],
- [0., 0.5, 0.],
- [0., 0., 0.5]]
- >>> print (determinant(inverse(a)))
- 0.125
- >>> print (diagonal(a))
- [0.5,0.5,0.5]
- >>> print (diagonal(a,1))
- [0.,0.]

- transpose(a), argsort(), dot()

Algèbres linéaires

```
>>> a = np.array([[1,2],[3,4]])
>>> print(a)
[[1 2]
 [3 4]]
>>> b = np.linalg.inv(a)
>>> print(b)
[[-2.   1. ]
 [ 1.5 -0.5]]
>>> print(np.dot(a,b))
[[1.0000000e+00 0.0000000e+00]
 [8.8817842e-16 1.0000000e+00]]
```

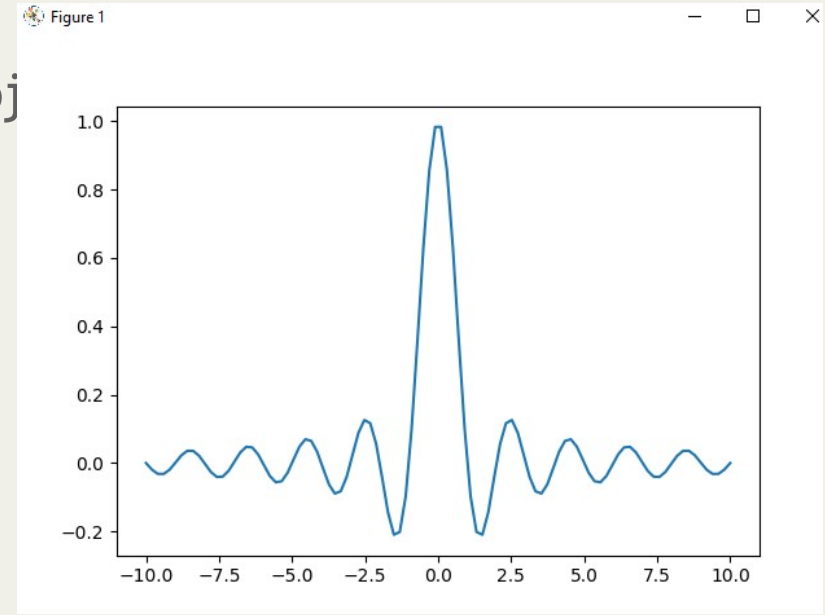
Autres informations

- Beaucoup de ressources sur le net
- http://math.mad.free.fr/depot/numpy/base.html

Matplotlib

- librairie très utilisée pour l'affichage

```
>>> import matplotlib.pyplot as plt
>>> x = np.linspace(-10.0, 10.0, 100)
>>> y = np.sin(np.pi*x)/(np.pi * x)
>>> plt.plot(x,y)
[<matplotlib.lines.Line2D obj
>>> plt.show()
```



Panda

- Ajout des DataFrame et des Series
- Les séries : proche des array de numpy sauf que les données sont indexées
- Assez proche du dictionnaire de Python

Note générale sur Python

- Variables vs objets
- Les pointeurs sont cachés comme en Java
- un changement sur une liste (mutable)impacte toutes les variables qui la référencent

```
Entrée [1]: x = [1, 2, 3]
            a = x
            b = a
            a.append(4)
            print(a)

            [1, 2, 3, 4]

Entrée [2]: print(b)

            [1, 2, 3, 4]

Entrée [ ]:
```

Comment faire ?

- On peut utiliser la méthode copy

```
Entrée [5]: x = [1, 2, 3]
            a = x
            b = x.copy()
            a.append(4)
            print(a, x)

            [1, 2, 3, 4] [1, 2, 3, 4]

Entrée [6]: print(b)

            [1, 2, 3]
```

```
Entrée [7]: a is x

   Out[7]: True

Entrée [8]: b is x

   Out[8]: False
```

Mutables vs immutables

- les « string » sont immutables, une autre est créée à chaque opération

```
Entrée [9]: x = "vive Python"
            id(x)

  Out[9]: 2473474597360

Entrée [10]: y = x
             id(y)

  Out[10]: 2473474597360
```

```
Entrée [12]: x = x.upper()
             id(x)

  Out[12]: 2473469577968

Entrée [13]: id(y)

  Out[13]: 2473474597360
```

Mutables vs immutables

- Les immutables :
  - entier
  - réel
  - complexe
  - booléen
  - string
  - tuple
- Les mutables
  - listes
  - ensembles
  - dictionnaires
  - objets créés par l'utilisateur
  - tableaux numpy
  - objets pandas

Mais !

- Certains opérations transforment les mutables en immutables

```
Entrée [14]: x = [1, 2, 3, 4]
             a = x
             x = x*2
             x

  Out[14]: [1, 2, 3, 4, 1, 2, 3, 4]

Entrée [15]: a

  Out[15]: [1, 2, 3, 4]
```

- Même chose pour les tranches x[2:5]

Un dernier pour la route

- Pas facile

```
Entrée [17]: def ajoute2(l):
                 l.append(2)
                 return l
             x = [1, 2, 3, 4]
             y = x
             ajoute2(x)

  Out[17]: [1, 2, 3, 4, 2]

Entrée [18]: y

  Out[18]: [1, 2, 3, 4, 2]
```

References

- Python Homepage
  - https://www.python.org/
- Python Tutoriel
  - https://www.python.org/about/gettingstarted/
- Python Documentation
  - https://docs.python.org/
- Python Library References
  - http://docs.python.org/release/2.5.2/lib/lib.html
- Python applis :
  - https://www.python.org/about/apps/