

TP python - Structures de bases

Gaël Guibon

2017-2018

Sommaire

1	Python en un coup d’œil	ii
1.0.1	Philosophie python : clarté, simplicité	ii
1.0.2	Python, un langage interprété	iii
1.0.3	Ecosystème Python	iii
2	Lancer votre installation python en local	iv
3	Syntaxe de base	v
3.0.1	Variables	v
3.0.2	Indentation	v
3.0.3	Lire un fichier	v
4	Structures de données	vi
4.0.1	Afficher une variable	vi
4.0.2	Les listes	vi
4.0.3	La compréhension de liste	vi
4.0.4	Les tuples	vii
4.0.5	Le JSON	vii
4.0.6	Les dictionnaires	vii
4.0.7	Les matrices	viii
4.0.8	Les CSV	viii
4.0.9	Le XML	viii

Chapitre 1

Python en un coup d’œil

1.0.1 Philosophie python : clarté, simplicité

The Zen of Python (TimPeters) :

<https://www.python.org/dev/peps/pep-0020/>

«

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Flat is better than nested.
- Sparse is better than dense.
- Readability counts.
- Special cases aren’t special enough to break the rules.
 - Although practicality beats purity.
- Errors should never pass silently.
 - Unless explicitly silenced.
- In the face of ambiguity, refuse the temptation to guess.
- There should be one— and preferably only one —obvious way to do it.
 - Although that way may not be obvious at first unless you’re Dutch.
- Now is better than never.
 - Although never is often better than right now.
- If the implementation is hard to explain, it’s a bad idea.
- If the implementation is easy to explain, it may be a good idea.
- NameSpaces are one honking great idea – let’s do more of those!

»

1.0.2 Python, un langage interprété

Contrairement à C++ ou Java, Python est un langage interprété. En voici les principales différences :

Compilé	Interprété
Tout le programme est compilé	Chaque instruction est interprétée
Code intermédiaire généré	Pas de code intermédiaire
Instructions de contrôle conditionnelles plus rapide	Instructions de contrôle conditionnelles plus lentes
Plus de mémoire requise	Moins de mémoire requise
Pas besoin de recompiler à chaque fois	Recompilé chaque fois qu'il est interprété
Erreurs au niveau du programme	Erreurs au niveau de l'instruction)
Java	Python

1.0.3 Ecosystème Python

Beaucoup de bibliothèques python existent et permettent de traiter de nombreux cas. Voici un exemple de bibliothèques python très utiles et reconnues :

Nom	Lien
Interfaces graphiques	
PyGTK	http://www.pygtk.org/
Traitement Automatique du Langage	
NLTK	http://www.nltk.org/
Formats de fichiers	
LXML	http://lxml.de/
Beautiful soup	https://www.crummy.com/software/BeautifulSoup/
Json	https://docs.python.org/2/library/json.html
Apprentissage / calculs	
NumPy	http://www.numpy.org/
SciPy	http://www.scipy.org/
Scikit-Learn	http://scikit-learn.org/
TensorFlow	https://www.tensorflow.org/
Base de données	
PyMongo	https://github.com/mongodb/mongo-python-driver
Application Web	
Django	https://www.djangoproject.com/
Parallelisme	
Thread	https://docs.python.org/3/library/threading.html

TABLE 1.1 – Sélection non exhaustive de bibliothèques python

Chapitre 2

Lancer votre installation python en local

- Lancer ubuntu et se connecter avec vos identifiants
- Récupérer les documents de la clef USB ou du git :

```
git clone https://github.com/gguibon/CoursesPython.git
```

- Dans le terminal (CTRL + ALT + T) taper :

```
bash install.sh
```

- Lancer ipython avec la commande suivante :

```
python notebook  
  
# en cas de probleme de droits / adressage :  
~/local/bin/ipython  
OU  
~/local/bin/ipython notebook
```

Pour ce cours nous utilisons Jupyter Notebook comme IDE afin de profiter de l'interpréteur de python. D'autres IDE sont tout aussi viables : Eclipse, PyCharm, Visual Studio Code, SublimeText, Atom, etc.

TIP

PyCharm et Visual Studio Code possèdent une bonne autocomplétion pour python.

Chapitre 3

Syntaxe de base

3.0.1 Variables

Python est un langage à typage dynamique.

Déclarer une variable en java :

```
int myInt = 0;
System.out.println(myInt);
```

Déclarer une variable en python :

```
myInt = 0
print( myInt )
```

3.0.2 Indentation

Les structures sont délimitées par l'indentation. *i.e.* les **tabulations** remplacent les accolades.

Boucle for en java :

```
for(String line : lines){
    System.out.println(line);
}
```

Boucle for en python :

```
for line in lines :
    print( line )
```

3.0.3 Lire un fichier

La méthode open() permet de lire ou écrire dans un fichier. Elle prend en argument le chemin et le type d'ouverture : 'r' pour read, 'w' pour write, 'a' pour append et 'b' pour binaire.

Il est impératif de toujours fermer un fichier ouvert (comme les bufferreader en java).

```
objet_fichier = open("monfichier.txt", "r")
contenu_string_du_fichier = open("monfichier.txt", "r").read()
liste_des_lignes_du_fichier = open("monfichier.txt", "r").readlines()

objet_fichier.close()
```

Chapitre 4

Structures de données

4.0.1 Afficher une variable

Afficher le contenu de l'entrée standard :

```
str_entree = input()
# taper un mot
print( str_entree )
```

4.0.2 Les listes

1. Définir la liste : `liste = [17, 38, 10, 25, 72]`, puis effectuez les actions suivantes ¹ :

- trie et affichez la liste ;
- ajoutez l'élément 12 à la liste et affichez la liste ;
- renversez et affichez la liste ;
- affichez l'indice de l'élément 17 ;
- enlevez l'élément 38 et affichez la liste ;
- affichez la sous-liste du 2^{ème} au 3^{ème} élément ;
- affichez la sous-liste du début au 2^{ème} élément ;
- affichez la sous-liste du 3^{ème} élément à la fin de la liste ;
- affichez la sous-liste complète de la liste ;

2. Ecrire une fonction récursive “palindrome(s)” qui renvoie *True* si la chaîne de caractères *s* est un palindrome, *False* sinon. NB : un palindrome est une chaîne de caractères qui peut se lire indifféremment dans les deux sens. ²

```
#initialisation de liste
ma_liste = []
# autre methode
ma_liste = list()
```

```
# exemple de fonction en python
def bonjour(nom):
    return 'Hola_' + nom
print( bonjour('gael') )
```

4.0.3 La compréhension de liste

Les créations de listes peuvent être abrégées en python. Il est possible d'écrire de créer des listes à partir d'éléments sans avoir à faire de trop nombreuses indentations et retours à la ligne.

1. Exercice issu du cours de l'ut d'Orsay.
2. Exercice de Alain Samuel et Sébastien Mavroumatis.

```
# filtrer une liste normalement
liste = [0, 1, 5, 10, 50, 2]
liste_filtree = list()
for num in liste:
    if num > 3:
        liste_filtree.append(num)

# avec liste comprehension
liste = [0, 1, 5, 10, 50, 2]
liste_filtree = [num for num in liste if num > 3]
```

TIP

Il est conseillé de d'abord dérouler les boucles normalement afin de comparer les résultats avec la compréhension de liste.

1. Lire le fichier "data/critique.txt"
2. Pour chaque phrase (ligne) récupérer uniquement les phrases qui possède le mot "Nolan"
3. Pour chaque phrase (ligne) récupérer les mots commençant par une majuscule (en itérant sur les caractères)
4. Créer une nouvelle liste ne contenant uniquement les phrases qui possèdent au moins un mot commençant par une majuscule et qui ne soit pas le premier mot. (Conseil : Utiliser la méthode `any()` : <https://docs.python.org/3/library/functions.html#any> et la méthode `str.strip()`)

4.0.4 Les tuples

Les tuples sont différents des listes :

- Ils sont plus rapides
- On ne peut y ajouter ou enlever une valeur. Ils sont non-mutables.
- Ils n'ont pas de méthode (`append`, `remove`, etc)

Reprendre la fonction du palindrome en utilisant uniquement des tuples. Cette fonction retournera désormais un tuple contenant la valeur booléenne, et le mot associé.

```
#initialisation de tuple
mon_tuple = ()
```

Named tuples :

Modifier la fonction en retournant un `namedtuple`, puis parcourir les champs du `namedtuple`.

Exemple d'initialisation d'un `namedtuple` :

```
import collections
results = collections.namedtuple( 'Results', ['xtrain', 'ytrain', 'xtest', 'ytest'] )
```

4.0.5 Le JSON

En utilisant la librairie `json` (<https://docs.python.org/3/library/json.html?highlight=json#module-json>), lire le fichier JSON fourni. Vérifier si pour chaque mot du champ "content" il s'agit ou non d'un palindrome. Attribuer un nouveau champ "palindrome" à l'objet "mot" afin d'indiquer si oui ou non il s'agit d'un palindrome. Enfin, réécrire la liste d'objet json en un fichier "mots_checked.json".

```
import json
```

4.0.6 Les dictionnaires

Lire le json et intégrer son contenu dans un dictionnaire (clé = champ content ; valeur = champ palindrome). Trier ce dictionnaire en fonction des valeurs et afficher en premier tous les mots qui sont bel et bien des palindromes.

TIP

Les dictionnaires ne peuvent être triés directement ! Les tuples et les listes seront utiles ici...


```
dico = dict()  
# OU  
dico = {}
```

4.0.7 Les matrices

- Générer une matrice à l'aide la librairie numpy (<http://www.numpy.org/>) :

```
import numpy as np  
a = np.arange(15).reshape(3, 5)  
# ou encore  
a = np.random.rand(3, 5)
```

- Afficher la taille, la forme, le nombre de dimensions et le nombre d'items de la matrice
- Multiplier les matrices et afficher le produit élément par élément

4.0.8 Les CSV

Transformer le JSON complet en un fichier CSV. Utilisez la librairie csv (<https://docs.python.org/3/library/csv.html>) pour cela.

```
import csv
```

4.0.9 Le XML

Transformer le CSV complet en un fichier XML à l'aide de la librairie LXML (<http://lxml.de/>).

```
from lxml import etree  
from copy import deepcopy
```