

Bases de Python pour SHS

Éric Guichard

2023–2024

Ce document est un « pense-bête » adapté à divers cours tenus à l’université de Lyon et destinés aux étudiants de culture dite « littéraire ». Il ne constitue pas à proprement parler une documentation pour `python`. Pour cela, vous pourrez consulter les sites <https://www.python.org> et <https://fr.scribd.com/document/356740978/courspython3-pdf>.

Vous pouvez aussi profiter des exemples du manuel d’Emilien Schultz : <https://github.com/pyshs/exemples-manuel/tree/master>. Pour les graphiques, le site <https://courspython.com> de David Cassagne est excellent.

1 Variables, listes, dictionnaires

Le « langage » `python` propose 5 catégories d’objets, dont 3 sont présentées ici : les variables, les listes, les dictionnaires. Mieux vaut leur donner des noms explicites :

```
mot="soleil" est mieux que x="soleil".
```

On peut dire, par métaphore, qu’une liste est une succession de variables et qu’un dictionnaire est un tableau indexé : une sorte de couple de listes, la première composée de *clés*, associées à des *valeurs* qui constituent la seconde liste.

Lien avec `perl`, langage pour lequel une variable commence par un `$`, une liste par un `@` et un *hash* (tableau indexé) par un `%`.

1.1 Variables

Une variable peut être du texte (une chaîne de caractères), un nombre (entier ou à virgule : « flottant »), une chose de type vrai/faux (« booléen », souvent apparenté à 1 / 0).

Un roman entier peut être inséré dans une variable, que l’on peut transformer en liste : par exemple de mots, délimités par la ponctuation. Exemples :

```
phrase="Il fait beau ce matin."
abbreviation-mois="03"
nombre=3
```

Une variable textuelle peut être prédéfinie en étant mise à vide : `phrase=""`. De même pour une variable numérique, mise à zéro : `nombre=0`.

1.2 Listes

Une liste mise à vide (initialisée) s’écrit `liste=[]`.

On peut définir une liste de façon exhaustive :
`liste=[2,3,"le soir",18, "hier"]`.

Les éléments d'une liste sont numérotés à partir de zéro ; et ça « boucle » :
`print (liste[2])` donne `le soir`.
`print (liste[-1])` donne `hier`.

Toute liste a une longueur (ici 5) : `len(liste)`. Il s'ensuit que le dernier élément de la liste est `liste[len(liste)-1]`.

1.2.1 Auto-indexation de listes : `enumerate`

Un exemple valant mieux que de longs discours...

```
for index, element in enumerate(liste):  
    print ("l'index auto de l'élément <"+element+"> vaut",index)
```

Résultat :

```
l'index auto de l'élément < 2 > vaut 0  
l'index auto de l'élément < 3 > vaut 1  
l'index auto de l'élément < le soir > vaut 2  
l'index auto de l'élément < 18 > vaut 3  
l'index auto de l'élément < hier > vaut 4
```

La variable `index` est donc créée automatiquement et donne la valeur courante du rang de l'élément sur lequel on travaille.

Une autre façon de voir l'usage de cette fonction consiste à demander le résultat de `print (list(enumerate(liste)))`.

La machine répond : `[(0, 2), (1, 3), (2, 'le soir'), (3, 18), (4, 'hier')]`.

1.2.2 Listes inattendues

Les variables textuelles sont d'étranges listes (de caractères) à leur façon :
`phrase="Il fait beau ce matin."`
`print (phrase[3])` renvoie `f` (le 4^e caractère).

Et donc `print (liste[-1][2])` renvoie `e`.

Étrangement, `print (phrase[3:5])` renvoie `fa` (2 éléments seulement : le 5^e caractère est *exclu* : cf. point 2.1).

Ceci ne fonctionne pas pour les variables nombres, qui ne sont pas découposables en listes :

```
a="123"  
print (a[0]) renvoie 1. Mais  
b=123  
print (b[0]) renvoie TypeError: 'int' object is not subscriptable.
```

1.2.3 Compléter une liste

Ajouter un élément à une liste : avec la fonction `append`. Ex. :
`liste.append(18)`. Un `print (liste)` donne alors
`[2, 3, 'le soir', 18, 'hier', 18]`.

Attention : cet exemple prouve qu'une liste peut contenir plusieurs éléments identiques.

On pourra aisément produire des itérations à partir des éléments d'une liste et transformer une liste en variable.

1.3 Dictionnaires ou tableaux

Un dictionnaire initialisé s'écrit `dict={}`.

Évidemment, toutes les clés d'un dictionnaire sont uniques.

Diverses manières de remplir un dictionnaire

```
dict={"abc":2, "bcd":3}
dict["abc"]=2 etc.
```

Expliciter le contenu d'un dictionnaire

```
for cle in dict.keys():
    print("à la clé", cle, " est associée la valeur ", dict[cle])
```

Autre façon :

```
for cle, valeur in dict.items():
    print("à la clé", cle, " est associée la valeur ", valeur)
```

Récursivité La valeur d'une clé peut être une variable (cas précédents), une liste ou même un dictionnaire. Un dictionnaire est confortable quand on le produit soi-même. S'il est « donné » sans que sa structure soit explicitée (ce qui peut être le cas de certains fichiers `json` importés en tant que dictionnaires), la reconstitution de son squelette peut s'avérer complexe.

2 Boucles et conditions

Cas des `for`, `if`, `while`, etc.

Syntaxe générale :

```
ligne de la condition:
-- > (TAB) actions si condition réalisée
```

En d'autres termes, la ligne du type `for`, `if`, `else`, `elif`, `while` se termine par deux points « : ». Les lignes des commandes correspondantes commencent par une **tabulation**.

2.1 Cas des listes ordonnées

`range(0,10)` contient dix nombres, de **0 à 9**. Le dernier (10) est donc **exclu**.
Preuve-exemple :

```
liste=range(0,10)
for i in liste:
    print (i)
print ("Attention "+str(len(liste))+" n'est pas affiché")
```

Seront affichés 0, 1, etc. jusqu'à 9 et la ligne `Attention 10` n'est pas affiché.

2.2 Sorties de boucles ou conditions

On usera de 3 types d'instructions : `pass` (peu utile), `continue` (sortie temporaire) et `break` (sortie définitive).

Exemples

```
for i in liste:
    if i==3:
        break
    print (i)
```

Seront affichés : 0,1,2.

*

```
    if i==3:
        continue
    print (i)
```

Seront affichés : 0,1,2, puis 4, 5... 9.

*

```
for i in liste:
    if i %2==0: #si i divisé par 2 donne un reste nul
        print (str(i)+ " est pair")
    else:
        pass
```

Dans ce dernier cas, les lignes contenant `else` et `pass` sont en fait inutiles. Mais elles peuvent être confortables

- quand on n'est pas sûr de soi (pas de `if` sans `else`),
- si on veut remettre à plus tard les instructions du `else`.

2.3 Exemple : compter des objets

En `python`, on ne peut créer de tableaux (dictionnaires) à la volée comme en `perl` (même si la fonction `enumerate` a ses charmes, cf. point 1.2.1). Il faut donc initialiser leurs clés. Un cas fréquent consiste à compter tous les mots d'une liste et leurs fréquences.

Exemple : On suppose qu'on demande à des personnes leurs lieux de naissance, et on obtient la liste suivante.

```
liste=[Paris,Lyon,Paris,Rouen,Bron,Lyon]
```

Combien sont nées à Paris, etc. ? Voici une solution, qui sollicite beaucoup les boucles et conditions.

```

liste=["Paris","Lyon","Paris","Rouen","Bron","Lyon"]
sommeliers={}
for v in liste:
    if v in sommeliers.keys(): # si on a déjà vu v
        sommeliers[v]+=1 #on ajoute 1 à sa valeur associée
    else:
        sommeliers[v]=1 # on crée la clé v et on lui associe 1
print (sommeliers)

```

Ce qui donne : {'Paris': 2, 'Lyon': 2, 'Rouen': 1, 'Bron': 1}

On peut remplacer la dernière ligne par celles-ci :

```

for i in sorted(sommeliers.keys()):
    print (sommeliers[i],"personne(s) sont nées à", i)

```

Résultat :

```

1 personne(s) sont nées à Bron
2 personne(s) sont nées à Lyon
2 personne(s) sont nées à Paris
1 personne(s) sont nées à Rouen

```

3 Lire un fichier, écrire dans un fichier

3.1 Écrire simplement

`monfic=open("carres.html", "w")` crée le fichier `carres.html` sur le disque dur ("w" pour *write*). Pour le programme, son nom sera `monfic`.

Pour y inscrire la variable `a`, il suffira d'écrire `monfic.write(a)`.

Pour écrire dans un fichier existant sans le supprimer, c'est-à-dire pour *compléter* ce fichier : choisir l'option "a" pour *append*.

Ex. : `monfic=open("carres.html", "a")`.

3.2 Lire simplement

Pour les fichiers courts et en mode *texte*, la méthode `read` peut fonctionner. Mais elle n'est pas optimale :

```

monfic=open("unfichierbref", "r")
leslignes = monfic.read()
print(leslignes)

```

Sinon, la solution la plus simple est d'utiliser une *library* (`csv.reader`, via un `import csv`), comme indiqué plus bas.

Note Il est aussi possible de lire des fichiers *formatés* par des logiciels externes (ex. : XL ou LibreOffice) avec d'autres bibliothèques (ex. : `openpyxl` ou `pandas`).

3.3 Exemple de lecture avec `csv.reader`

On gagne la suppression des sauts de ligne et une lecture « automatisée ».

Soit un fichier `cercles` qui a l'allure suivante :

X	Y	R
10	90	5
30	60	10
60	80	15
90	85	8
90	45	20

On veut récupérer les coordonnées X, Y des centres (on oublie les rayons).

```
nomInterneDuFichier = csv.reader(
open("cercles", 'r', encoding='UTF-8'), delimiter='\t')
totaldeslignes = list(nomInterneDuFichier)
for lignecourante in totaldeslignes:
    print (lignecourante)
    coord1,coord2,rien=lignecourante #grâce au délimiteur \t
    #la variable inutile (rien) est indispensable...
    #autre solution: coord1=lignecourante[0], etc.
    print (coord2) #à compléter...
```

3.4 Exemple de lecture sans csv.reader

```
nomInterneDuFichier = open("cercles", 'r', encoding='UTF-8')
sommelignes = list(nomInterneDuFichier)
for lignecourante in sommelignes:
    print (lignecourante)
    coord1,coord2,rien=lignecourante.split('\t')
    print (coord2,rien)
    #un saut de ligne en trop
```

Piège python a besoin du nombre exact de variables créées par le séparateur pour lire correctement une ligne. D'où la sollicitation de la variable `rien`, au nom explicite. Une solution consiste à mettre tout ce qui ne nous intéresse pas dans une liste, qui absorbera en vrac tous les objets restants :

```
listerien=[] #on définit cette liste
for... :
    x,y,listerien=lignecourante.split('\t')
```

3.5 Sagesse : fermer un fichier

Quand le fichier nommé (par exemple) `monfic` dans le programme n'est plus utilisé, un `monfic.close()` est fortement conseillé.

3.6 Lire une série de fichiers

Faire usage de la *library os : operating system*. Exemple

```
import os
os.chdir("1erTourbis") # on va dans ce dossier
```

```

print(os.listdir()) #affiche de son contenu
for fichier in os.listdir(): #pour chaque fichier
    if fichier.endswith("TRUC"):
        ...
        with open(fichier, "r",encoding='utf8') as circons:
            lignes=list(circons)
            for chaqueligne in lignes:
                ...

```

4 Commentaires, variables et caractères bizarres, etc.

4.1 Apostrophes et guillemets

En anglais : *single and double quotes* : ' et ".

Attention Il m'arrive souvent d'écrire ' au lieu de '. En ce cas, ne cherchez pas à reproduire l'apostrophe typographique dans vos scripts, utilisez l'apostrophe simple.

Absence de différences Une variable texte peut indifféremment être détaillée entre guillemets ou entre apostrophes :

```
a="bonjour" ou a='bonjour'
```

Remarque J'aurais dû écrire a='bonjour' mais cela me prend beaucoup plus de temps. D'où l'« erreur » évoquée au paragraphe précédent...

Dans la pratique, on essaie d'utiliser le délimiteur qui est le moins utilisé.

Si j'écris a='aujourd'hui', j'induis une erreur : il y a 3 apostrophes. Je préférerai alors écrire a="aujourd'hui".

Une autre solution consiste à *protéger* l'apostrophe du mot :

```
a='aujourd\'hui'
```

Par exemple, si j'ai besoin d'insérer dans une variable l'expression `<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"` j'écrirai `a='<svg xmlns="http...xlink" 'car il n'y a pas d'apostrophes dans cette expression pleine de guillemets.`

4.2 Cas des variables avec plusieurs lignes

Deux solutions :

— avec des sauts de ligne encodés : `a='première ligne\nseconde ligne'`
ou `a="première ligne\nseconde ligne"`.

— en utilisant 3 guillemets : `a=""un paragraphe séparé
par de banals
retours chariots""`

produit le résultat escompté. Cette solution fonctionne aussi avec 3 apostrophes.

Note Perl n'aurait pas eu le même comportement : ce dernier langage interprète ce qui est entre guillemets, mais pas ce qui est entre apostrophes : dans le premier cas `\n` produit un saut de ligne, dans le second, il produit un `\` suivi d'un `n`.

Longs commentaires Comme python ne tient pas compte des formes textuelles non assignées à une variable, tout paragraphe entre 3 apostrophes ou guillemets sera pris comme un commentaire (détournement fréquent) :

```
''' Ici
un long
commentaire'''
```

4.3 Additions et surprenantes concaténations

Selon les contextes, le signe `+` symbolise l'addition entre deux nombres ou la concaténation de deux chaînes de caractères. Exemple.

```
mot="bonjour"
nombre=23
print (mot+mot, nombre+nombre)
```

Réponse: `bonjourbonjour 46`

Si j'écris `a="bonjour"` et `b=" à tous"` un `print (a+b)` donnera donc `bonjour à tous`.

Mais si j'écris `c=2`, un `print (a+b+" "+c)` me renverra un message d'erreur, alors que si j'avais écrit `c="2"`, j'aurais obtenu comme imaginé `bonjour à tous 2`.

Il faut convertir `c` en chaîne de caractères pour l'intégrer en une autre (`string`) :

```
print (a+b+" "+str(c))
```

4.4 D'une chaîne à un nombre

L'inverse peut se produire quand on lit une série dans un fichier : ce qu'on croit être un nombre `n` pourra être pris pour un caractère et on ne pourra alors pas l'utiliser pour des calculs.

En ce cas, la fonction (pour une conversion en entier) est `int()`. Ex. : `int(n)`. Pour une conversion en nombre banal (décimal) : `float(n)`. Pour s'y retrouver, la fonction `type` précise le type de la chose manipulée :

```
print (type(a)) donne <class 'str'>
print (type(c)) donne <class 'int'>
print (type(2.5)) donne <class 'float'>
liiiste=[2,5] suivi d'un print (type(liiiste)) donne <class 'list'>.
```

4.5 Extrapolation

Soit la série d'instructions suivantes, séparées par des lignes.

```
e="2" f=5 g="." h=e+g+str(f) print (h) i=float(h)
Un print (type(i/2)) donnera <class 'float'>. Ouf!
```


5 Formatage de nombres et commande format

Cf. <https://courspython.com/print-format.html?highlight=format>, qui oriente aussi vers https://python.sdv.univ-paris-diderot.fr/03_affichage/.

5.1 Usage des chaînes formatées : *fstring*

Il s'agit en général d'obtenir des expressions simplifiées ou homogènes de valeurs numériques. Exemple, déclinable de façons très variées :

```
a=2
b=3.1278
print(f"a vaut {a:.2f} et b vaut {b:.2f} ")
```

Résultat : a vaut 2.00 et b vaut 3.13 .

5.2 la fonction format

5.2.1 Exemple simplissime

```
print ("x vaut {}, y vaut {} et z vaut {}".format('2','3','4'));
```

Donne comme résultat

x vaut 2, y vaut 3 et z vaut 4.

Très utile pour le svg...

5.2.2 Second exemple

```
stock = ['papier', 'carton', 'chemise', 'encre']
print("Nous avons de l'{} et du {} en stock\n".format(stock[3],stock[0]))
#autre solution, moins lisible:
print("Nous avons de l'{} et du {} en stock\n".format(stock))
```

6 Nettoyages textuels, expressions régulières

6.1 Premiers nettoyages

6.1.1 replace

La fonction `replace` est fort confortable. Exemple, sachant qu'\n est un saut de ligne :

```
phrase="Il fait beau\nce matin"
phrase=phrase.replace('\n'," ")
print (phrase)
```

Résultat : Il fait beau ce matin.

6.1.2 strip

`strip` est utile quand il s'agit de travailler sur des objets comme des noms de fichiers qu'on veut débarrasser des espaces () qui parfois les enveloppent. Ex. :

```
a=" fichier"
a=a.strip()
print ("T"+a+"T")  renvoie TfichierT.
```

Paramètres complémentaires :

`strip('liste de caractères bordants à enlever')` Ces caractères peuvent être dans le désordre mais tous doivent être présents (y compris l'espace au besoin).

Attention `strip` enlève les *whitespaces*, souvent traduits par « espaces » en français. En fait, un *whitespace* est un caractère blanc, donc : une ou plusieurs espaces , une tabulation ou un saut de ligne.

En bref, pour un `a="Essai \n"` (des espaces, des tabulations et un saut de ligne à droite), `a.strip()` renverra un `Essai` tout court.

Seconde remarque La gestion des espaces (et des sauts de ligne) par python peut s'avérer désarçonnante. Par exemple, `print ("XXX", "YYY")` affiche un `XXX YYY` (introduction d'une espace).

6.1.3 Commence ou finit par...

Les commandes `endswith` et `startswith` sont simples et utiles.

```
a="coucou"
if a.endswith("u"):
    print (a)
```

6.2 Vers les expressions régulières

6.2.1 Citation

Source : <https://docs.python.org/3/howto/regex.html#match-versus-search>, fin de page.

Le livre le plus complet sur les expressions régulières est certainement *Mastering Regular Expressions* de Jeffrey Friedl, publié par O'Reilly. Malheureusement, il se concentre exclusivement sur les expressions régulières de Perl et de Java, et ne contient aucune information sur Python, de sorte qu'il ne sera pas utile en tant que référence pour la programmation en Python. (La première édition couvrait le module `regex` de Python, aujourd'hui supprimé, ce qui ne vous aidera pas beaucoup). Pensez à le consulter dans votre bibliothèque.

6.2.2 Substitution

Besoin de la bibliothèque `re` (*regular expression*, expression rationnelle). Exemple simple :

```
import re
a="bonjour"
b=re.sub("jou","soi",a)
print (b)
```

Exemple plus efficace, avec une première expression rationnelle, qui élimine la ponctuation dans une phrase ou un texte (par exemple pour compter des mots). En fait, les caractères cités (éventuellement protégés) sont remplacés par des espaces.

```
import re
lignecourante="Le terme « poésie » et ses dérivés « poète »,
« poème » viennent du grec ancien (poiesis)"
lignesimple=re.sub('[ \.,«»()\']+', ' ', lignecourante)
print (lignesimple)
```

7 Rudiments graphiques

Les exemples suivants donnent une idée des possibilités de matplotlib.

7.1 Listes de nombres

C'est le cas le plus simple. On veut représenter une série de points de coordonnées (x, y) . On fabrique alors la liste des x , notée ici `abscisses`, et celle des y , notée ici `ordonnees`.

```
import matplotlib.pyplot as plt
plt.plot(abscisses, ordonnees)
#La suite est étrangement indispensable:
plt.show()
```

Le graphique 1 page 16 est ainsi construit.

Exemple simple :

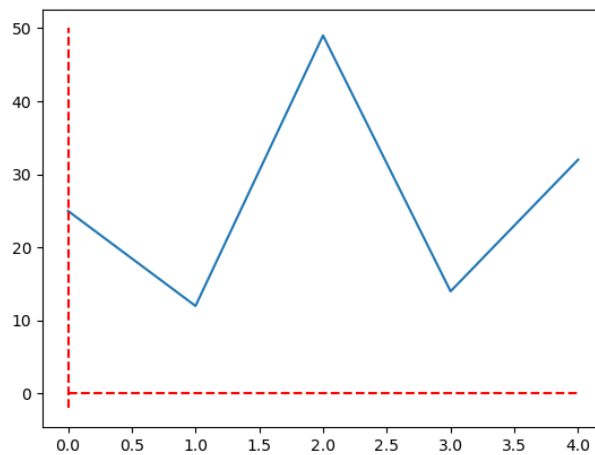
```
import matplotlib.pyplot as plt
abscisses=range(0,5)
freq = [25, 12, 49, 14,32]
plt.plot(abscisses, freq)
plt.show()
```

Pour mieux voir les hauteurs exactes des points, on peut demander (avec la même logique : `plt.plot`) à visualiser les axes :

```
import matplotlib.pyplot as plt
abscisses=range(0,5)
freq = [25, 12, 49, 14,32]
plt.plot(abscisses, freq)
plt.plot((0, 0), (-2, 50), 'r--') #vertical
plt.plot((0, 4), (0, 0), 'r--')

plt.show()
```

On obtient alors le graphique



7.2 Histogrammes

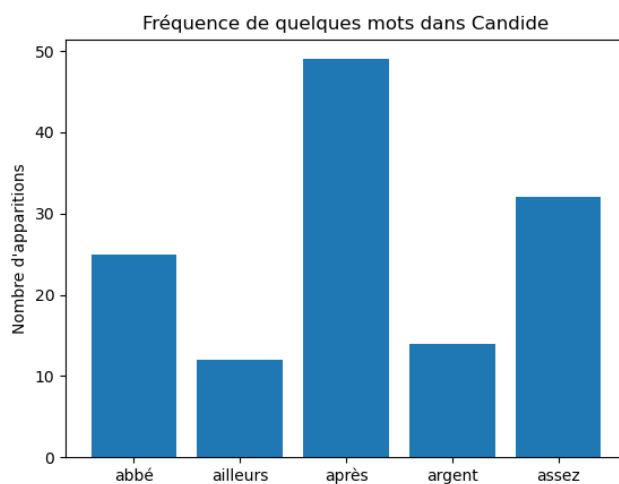
La syntaxe est assez explicite.

```
import matplotlib.pyplot as plt
fig, ax = plt.subplots()

mots= ['abbé', 'ailleurs', 'après', 'argent', 'assez']
freq = [25, 12, 49, 14, 32]

ax.bar(mots, freq)
ax.set_ylabel("Nombre d'apparitions")
ax.set_title('Fréquence de quelques mots dans Candide')
plt.show()
```

Et voici le résultat.

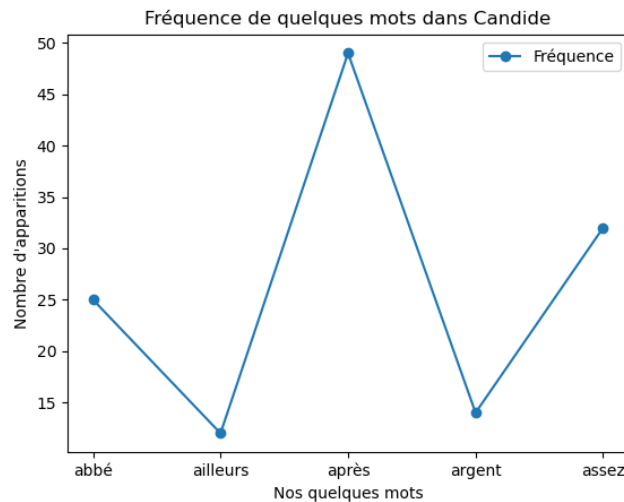


7.2.1 Autre méthode, en combinant les deux méthodes

```
import matplotlib.pyplot as plt
mots= ['abbé', 'ailleurs', 'après', 'argent', 'assez']
freq = [25, 12, 49, 14, 32]
fig, ax = plt.subplots()
ax.plot(mots, freq, "o-", label='Fréquence')

# Options et visualisation
ax.set_title('Fréquence de quelques mots dans Candide')
ax.set_xlabel("Nos quelques mots")
ax.set_ylabel("Nombre d'apparitions")

plt.legend()
plt.show()
```



7.3 Graphes de fonctions

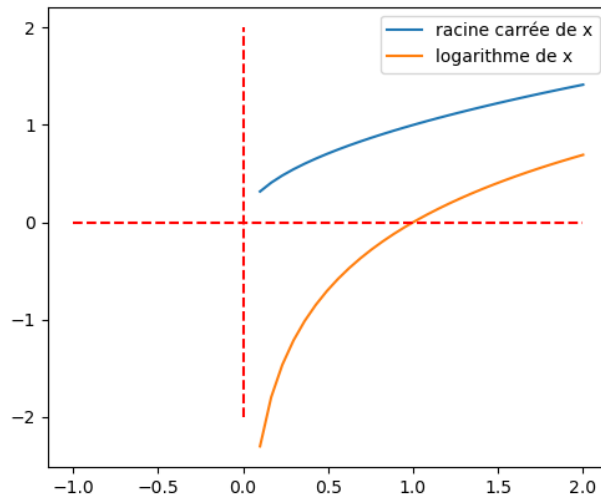
C'est assez simple. Exemple.

```
import matplotlib.pyplot as plt

x = np.linspace(0.1, 2, 30)
#fabrique une liste de 30 éléments entre 0.1 et 2
y1 = np.sqrt(x) #racine de x
y2 = np.log(x) #log de x

fig, ax = plt.subplots()
ax.plot(x, y1, label="racine carrée de x")
ax.plot(x, y2, label="logarithme de x")
ax.legend()
plt.plot((0, 0), (-2, 2), 'r--') #repérage axes
plt.plot((-1, 2), (0, 0), 'r--')
plt.show()
```

Ce qui donne



8 Un programme de comptage de mots : Zipf.py

Un programme à lancer et à commenter. Attention les indentations sont peut-être imparfaites.

```
import re
fichier=open("PoesieWP", "r")
leslignes = list(fichier)
totalmots={}
for lignecourante in leslignes:
    print (lignecourante)
    lignesimple=re.sub('[ \.,«»()\' ]+', ' ', lignecourante)
    mots=lignesimple.split(" ")
    for m in mots:
        if m in totalmots:
            print ("déjà vu")
            totalmots[m]=1+totalmots[m]
        else:
            print ("nouveau")
            totalmots[m]=1
    print (m)

for i in totalmots:
    if totalmots[i]>2:
        print (i, " apparaît ",totalmots[i]," fois")

print (sorted(totalmots.keys(),key=str.lower))
print (sorted(totalmots.values(),reverse=True))
print (totalmots.items())
```

8.1 Final : visualisation de la loi de Zipf

```
import re
import math
import matplotlib.pyplot as plt
import numpy as np

fichier= open('CandideFr.txt', "r",encoding='utf8')
fresu= open('DicoCandideFr', "w",encoding='utf8')

leslignes = list(fichier)
totalmots={}
totalhapax=0
leslogs=[]
abscisses=[]
lesinfos=""

for lignecourante in leslignes:
    print (lignecourante)
    lignesimple=re.sub('[ \.,«»()\`!-\?;\n\t]+', ' ', lignecourante)
    mots=lignesimple.split(" ")
    for m in mots:
        m=m.lower()
        if len(m)==0:
            continue
        if m in totalmots:
            totalmots[m]=1+totalmots[m]
        else:
            print ("nouveau")
            totalmots[m]=1
            print (m)
#Fin calculs

for i in totalmots:
    if totalmots[i]>1:
        leslogs.append(math.log(totalmots[i]))
    else:
        totalhapax+=1

lesinfos+="nb d'hapax: "+str(totalhapax)+"\n"
lesinfos+="nb mots différents: "+str(len(totalmots))+"\n"

print (sorted(totalmots.keys()))
print (sorted(totalmots.values(),reverse=True))

for m in sorted(totalmots.keys()):
    ligne=m+"\t"+str(totalmots[m])+"\n"
    fresu.write(ligne)

fresu.write("\n\nLes infos\n")
fresu.write(lesinfos)
fresu.close

leslogs=sorted(leslogs,reverse=True)

for i in range (1,len(leslogs)+1):
```

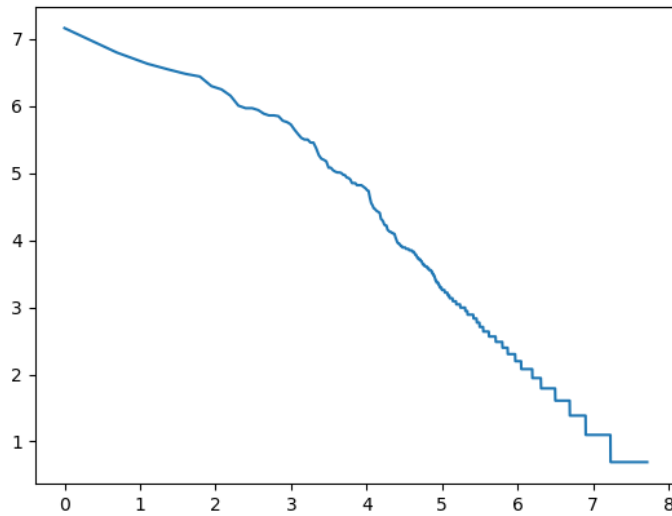


Figure 1 – Loi de Zipf. On remarque l’allure linéaire du graphique *log-log*.

```

j=math.log(i)
abscisses.append(j)

plt.plot(abscisses, leslogs)
#La suite est étrangement indispensable:
plt.show()
#Attention, le graphique peut masquer la fin de l'écriture de fresu
exit()

```

8.2 Graphique de la loi de puissance

Et voici le résultat : fig. 1.

9 Installations, usages

Cette section aurait peut-être dû apparaître au début. Mais comme elle est peu appétissante, elle a été rétrogradée ici. Elle sera prochainement complétée et elle évoquera pip, VSC, jupyter, les comportements variables de python selon les OS, etc. Idées :

Il faut tout d’abord installer pip (le gestionnaire de paquets pour Python) : https://www.odoo.com/fr_FR/forum/aide-1/how-to-install-pip-in-python-3-on-ubuntu-18-04-167715.

Cela peut se faire dans un terminal avec les commandes `sudo apt update`, puis `sudo apt install python3-pip` — `sudo` pour effectuer ces commandes en tant que superutilisateur (*root*).

Ensuite on sollicite pip pour installer la bibliothèque de son choix. Exemple, pour *Beautiful Soup* : `pip install beautifulsoup4`.