

1 Fondamentaux

- «Déclaration de variable: » `var=1` sera un entier, `var=1.0` (sera un réel), `var="1"` (sera une chaine de caractère), `var=[var1,var2,var3]` (sera une liste), `var=(var1,var2,var3)` sera un tuple.
- «Changer le type d'une variable: » `var=int("1")` (sera un entier), `float` (pour réel), `str`(pour chaine de caractères).
- «Variable saisie au clavier: »

```
var=input("texte s'affichant à l'écran et demandant de saisir une variable")
```

le contenu de la variable saisie au clavier est par défaut une chaine de caractère, il faudra la convertir si on veut des nombres.
- «Opérateurs: » `+` `-` `*` `%` (modulo) `//` (Quotient de la division euclidienne) `**` (puissance) `=` (affectation)
- «1e5» Écriture scientifique de 10^5 .
- «1j» Nombre complexe i tel que $i^2 = -1$.
- «Opérateurs logiques: » `not` (ou!) `and` (ou &) `or` (ou |) `True` (ou 1) `False` (ou 0)
- «Affichage: »

```
print("var1={0} et var2={1}".format(var1, var2))
```

2 Conditions et boucles

- «Condition si (if)»

```
if (condition1):
    instruction1
elif (condition2):
    instruction2
else:
    instruction final
```

- «Boucle pour (for)»

```
for i in range(1,10,1):
    instruction à répéter pour i allant de 1 à 9 par pas de 1.
```

- «Boucle tant que (while)»

```
while (condition):
    instruction tant que condition est vraie
```

- «Try»

```
try:
    instruction1
except (type d'erreur):
    instruction2 s'il y a une erreur à instruction1
```

Type d'erreur : ValueError RuntimeError, TypeError, NameError , ZeroDivisionError, IOError ...

```
try:
    a/b
except ZeroDivisionError :
    print("impossible de diviser par 0")
```

3 Les listes

Une liste est modifiable.

- «`T = [e11 ,e12 , e13]`» créer une liste L.
- «`list(T)`» Transforme un tuple en une liste.
- «`range(10)`» : liste de 0 à 9 par pas de 1
- «`range (2, 20, 3)`» liste de 2 à 20 (exclue) par pas de 3
- «Concaténation» `L1+L2`
- Sélection de liste
 - «`L[1:6:3]`» sélection du 2eme au 7eme (exclu) éléments par pas de 3.
 - «`L[-2]`» sélection des deux derniers éléments de L.
 - «`L[2:]`» sélection de l'élément 2 jusqu'à la fin de L.
 - «`L[::-1]`» sélection de L mais inversée.
 - «`L[0][-1]`» sélection du premier élément de L et en prendre le dernier élément de celui ci
- «`L.reverse()`» inverser l'ordre des items d'une liste.
- «`var=L.count(élément à compter)` » compter combien de fois un élément apparaît dans une liste
- «`L.append(élément à ajouter)`» ajouter un élément à la fin d'une liste.

- «**L.insert(i,x)**» l'élément x dans la liste avant l'élément à la position i (le reste est décalé)
- «**L.pop(i)**» l'élément x dans la liste avant l'élément à la position i (le reste est décalé)
- «**var = len(L)**» longueur d'une liste

4 chaînes de caractères

Une chaîne de caractère est non modifiable.

- «**s.upper()**» Mettre tout en majuscule
- «**s.lower()** » Mettre tout en minuscule
- «**s.title()**» Mettre la première lettre de chaque mots en majuscule, le reste en minuscule
- «**s.swapcase()**» Mettre les minuscules en majuscule et les majuscules en minuscule
- «**s.islower()**» Indique si s est en minuscule (True/False)
- «**s.isupper()**» Indique si s est en majuscule (True/False)
- «**s.isalpha()**» Indique si la chaîne contient uniquement des caractères alphabétique (True/False)
- «**s.isdigit()**» Indique si la chaîne contient uniquement des caractères numériques (True/False)
- «**s.isalnum()**» Indique si la chaîne contient uniquement des caractères alphanumérique (True/False)
- «**s.replace(old , new)**» Remplacer les bouts de chaîne old par un new
- «**"el".join(L)**» Concaténer une liste avec l'élément 'el' entre chaque items
- «**s.split("el")**» Découper une chaîne en liste en fonction du séparateur 'el'

5 Tuples :

Un tuple est non modifiable.

- «**T = (e11 ,e12 , e13)**» définit un tuple.
- «**tuple(L)**» transforme la liste L en un tuple.
- «**Permuter des données**» var1, var2 = var2, var1
- «**Affichage**»

```
print ("x=%d y=%d"%(1))
```



```
print("x={0[0]} y={0[1]}".format(1))
```

6 Les dictionnaires

- «**D = {"clé1": valeur1, "clé2": valeur2, "clé3": valeur3}**» Définit un dictionnaire D.
- «**print(D["cle1"])**» Affiche valeur1
- «**D["cle"] = valeur**» Ajoute un élément ou modifie la valeur de cle
- «**D.keys()**» renvoie une liste contenant les clés.
- «**D.values()**» renvoi une liste contenant les valeurs du dictionnaire.
- «**D.has_key(clé)**» revoit le booléen True si le dictionnaire contient la clé ou la valeur, False sinon.
- «**D.get(clé, valsinon)**» renvoie la valeur d'une clé ou valsinon si la clé n'existe pas.
- «**len(D)**» renvoi la taille du dico (nombre de clés)
- «**D.items()**» renvoi une liste de tuple, chaque tuple contenant clé puis valeur
- «**del D[clé]**» supprime un élément du dico (clé et valeur)

7 Les fonctions

- «**importer un module**»
 - `import math`, on sera obligé d'utiliser math. avant d'utiliser un fonction du module `math.sqrt(2)`.
 - `import numpy as np`, np sera alors un raccourci du module numpy `np.exp(1)`.
 - `from math import sqrt, sin` (on importe seulement la fonctions *sqrt* et *sin* du module *math*)
 - `from math import *` idem que précédemment mais on importe toutes les fonctions du module.

Avec **from** il ne faut pas mettre le nom du module, seulement le nom de la fonction, par exemple `sqrt(2)` et non `math.sqrt(2)`.

- «**def**»

```
def nomDeLaFonction (param1 , param2=valParDefault2 etc... ):
    """ description de la fonction, Docstring """
    Actions
    return var1, var2
```

Les variables sont locales SAUF pour les listes qui sont définitivement modifiées !

- «modules courants» maths, os (système), random, time, tkinter (fenêtres), numpy, matplotlib (graphique), sympy (calcul formel), http lib (connections http) ...

- «Exemple»

```
def factorielle ( n):
    """ Fonction factorielle """
    facto=1
    for i in range(1,n+1):
        facto*=i
    return facto
```

8 Manipulation de fichier

- «fichier = open("monfichier.txt", "r")» pour ouvrir un fichier en lecture
Les options sont : **r** lecture seule, **w** écriture, **a** pour ajouter du texte **b** binaire
- «contents = fichier.read()» pour la lecture du fichier.
- «lines = fichier.readlines()» pour lire une seule ligne.
- «fichier.close()» pour fermer le fichier.
- «fichier.write("Ceci est un texte")» pour écrire le texte. Si le fichier n'existe pas, il sera créé.
- « fichier.writeline(line)» Pour écrire une ligne supplémentaire.

9 Expressions régulières

Les **Expressions régulières** (ou **rationnelles**, ou encore "pattern" en anglais), communément abrégé en *regex*, consiste en une chaîne de caractères, souvent appelée « motif ».

Un **r** placé devant la chaîne permet de considérer l'antislash `\` comme un caractère normal. Par exemple, on pourra écrire : `regex = r'a0\'`

- «import re» pour importer le module
- «.» désigne n'importe quel caractère ;
- «^» indique que le début de la chaîne doit correspondre
- «\$» indique que la fin de la chaîne doit correspondre
- «{n}» indique que le caractère précédent doit être répété n fois.

- «{n,m} » indique que le caractère précédent doit être répété entre n et m fois.
- «*» le caractère précédent peut être répété 0 ou plusieurs fois.
- «+» le caractère précédent peut être répété 1 ou plusieurs fois.
- «?» le caractère précédent peut être répété zéro ou une fois.
- Les quatre derniers symboles sont dits "gourmands", cela signifie qu'ils chercheront un maximum de caractères, ce qui peut parfois poser problème. Pour éviter ce comportement, on peut rajouter un `?` après ces derniers, c'est à dire utiliser `??`, `+` ? et `*?`.
- L'antislash `\` permet d'échapper tous ces caractères spéciaux.
Par exemple pour le point on utilisera `\.`
- Les crochets `[]` permettent d'indiquer une plage de caractère, par exemple `[e-h]` correspondra à e, f, g ou h.
- Les parenthèses `()` permettent de grouper certaines expressions ce qui peut permettre de s'y référer par la suite avec `\1 \2` etc ...
- «\w» correspond à tout caractère alphabétique, ce qui est équivalent à `[a-zA-Z]` ;
- «\W» correspond à tout ce qui n'est pas un caractère alphabétique ;
- «\d» correspond à tout caractère numérique, ce qui est équivalent à `[0-9]` ;
- «\D» correspond à tout ce qui n'est pas un caractère numérique.
- Il existe deux manières d'utiliser les regex :
 - La première consiste à appeler la fonction avec en premier paramètre le motif, et en deuxième paramètre la chaîne à analyser.
 - La seconde consiste à compiler la regex, et à ensuite utiliser les méthodes de l'objet créé pour analyser une chaîne passée en argument. Cette méthode permet d'accélérer le traitement lorsqu'une regex est utilisée plusieurs fois.

```
prog = re.compile(pattern)
result = prog.match(string)
```

est équivalent à

```
result = re.match(pattern, string)
```

Exemples

```
In [1]: import re
In [2]: text = "Il s'était prudemment déguisé mais fut
          rapidement capturé par la police."
In [3]: # trouver tous les adverbes en -ment
In [4]: re.findall(r"\w+ment", text)
Out[4]: ['prudemment', 'rapidement']
```

```
In [5]: # Du debut jusqu'au dernier mot ayant la syllabe ment.
In [6]: re.findall(r"^.ment", text)
Out[6]: ["Il s'était prudemment déguisé mais fut rapidement"]
In [7]: # idem mais au premier mot ayant la syllabe ment.
In [8]: re.findall(r"^.?ment", text)
Out[8]: ["Il s'était prudemment"]
```

Utilisation de substitute

```
In [9]: text = "Confessions Publiques (Live)/20-Madame rêve.mp3"
In [10]: re.sub(r'[/[0-9]+-', '/', text)
Out[10]: 'Confessions Publiques (Live)/Madame rêve.mp3'
In [11]: regex = re.compile('/[0-9]+-') # ou alors
In [12]: regex.sub('/', text)
Out[12]: 'Confessions Publiques (Live)/Madame rêve.mp3';
```

10 Les classes

• «Exemple»

```
class Edwige:
    """Classe qui définit un individu. Un individu sera définie par
    son nom, son prénom et son âge"""
    def __init__( self, nom, prénom, âge):
        """Constructeur d'un individu """
        âge : l'âge de l'individu"""
        self.n = nom
        self.p = prénom
        self.a = âge
    def __str__( self):
        """Renvoie une chaine de caractères décrivant l'
        individu"""
        return "individu
        : Nom={}, Prénom={}, âge={} ".format(self.n, self.p,
        self.a)
```

11 Sympy

On utilise sympy pour le calcul formel

Dans sympy il faut déclarer les variables symboliques explicitement avec **Symbol** ou **S**.

- «**Rational(1,2)**» ou «**S('1/2')**» détermine le rationnel $\frac{1}{2}$.
- «**sqrt(12)**» détermine le radical $\sqrt{12}$.
- «**pprint(sqrt(12))**» pour afficher $\sqrt{12}$.
- «**x=Symbol('x')**» ou «**x=S('x')**» définit x comme variable
- «**((x+3)**2).expand()**» pour développer $(x+3)^2$
- «**(x**2+2*x-3).factor()**» pour factoriser.
- «**(x**2+2*x-3)**2.subs(x, 1)**» pour substituer x par 1.
- «**limit(sin(x)/x, x, 0)**» pour calculer $\lim_{x \rightarrow 0} \sin x$
- «**limit(1/x,x,0,'+')**» pour calculer $\lim_{x \rightarrow 0^+} \frac{1}{x}$
- «**solve('x^2-2*x-3',x)**» pour résoudre $x^2 - 2 * x - 3 = 0$
- «**integrate(x+1/x, x)**» pour la primitive de $f(x) = x^2 + \frac{1}{x}$
- «**integrate(sin(x), (x, 0, pi/2))**» pour calculer $\int_0^{\frac{\pi}{2}} x + \frac{1}{x} dx$
- «**integrate(exp(-x), (x, 0, oo))**» pour calculer $\int_0^{+\infty} e^{-x} dx$.
- «**A = Matrix([[1,2],[3,4]])**» définit la matrice A.

12 Numpy

- «**int(x)**» Nombre entier le plus proche de x.
- «**rint(x)**» Nombre entier le plus proche de x.
- «**floor(x)**, **ceil(x)**» Partie entière supérieure et inférieure de x.
- «**rand(n,p)**» Générateur aléatoire de matrice de dimension $n \times p$ sur $[0;1]$. (**numpy.random**)
- «**pi**» Valeur approchée de $\pi \approx 3.1415$.
- «**exp(1)**» Valeur approchée de $e \approx 2.7183$.
- Vecteur
 - «**array([1, 2, 3])**» Représente le vecteur horizontal $\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$.

- `«array([[1], [2], [3]])»` Représente le vecteur vertical $\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$.
- `«x.size»` Renvoie la longueur du vecteur x.
- `«arange(a,b,i)»` Construit le tableau $(a \quad a+i \quad a+2i \quad \dots \quad b)$.

• Matrice

- `«array([[1,2], [3,4]])»` Représente la matrice $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$.
- `«m[i,j]»` Représente l'élément a_{ij} de la matrice m
- `«m.shape»` Renvoie la taille de la matrice m sous forme d'une liste.
- `«m[i,:]»` Extrait la i^e ligne de la matrice m.
- `«m[:,i]»` Extrait la i^e colonne de la matrice m.
- `«zeros([n,p])»` Représente la matrice nulle de taille $n \times p$.
- `«eye(n,p)»` Représente la matrice identité de taille $n \times p$.
- `«ones([n,p])»` Représente la matrice de coefficients 1 de taille $n \times p$.
- `«asarray(m)»` Transforme une matrice en tableau.
- `«asmatrix(m)»` Transforme un tableau en matrice.
- `«norm(m)»` Norme de la matrice m. (`numpy.linalg`)
- `«transpose(m)»` Transposée de la matrice m.
- `«det(m)»` Déterminant de la matrice m. (`numpy.linalg`)

• Polynômes

- `«poly1d([1, 2, 3, 5, 8])»` Représente le polynôme $x^4+2x^3+3x^2+5x+8$.
- `«poly1d([1,2, 3, 5, 8],True)»` Représente le polynôme $(x-1)(x-2)(x-3)(x-5)(x-8)$.
- `«p.order»` Degrés du polynôme p.
- `«p.roots»` ou `«roots(p)»` Racines du polynôme p.
- `«p.coeffs»` Coefficients du polynôme p.
- `«p(3)»` ou `«polyval(p,3)»` Évalue le polynôme p en 3 (selon le schéma de Hörner)

- `«imshow(m,c?)»` Affiche la matrice en deux dimensions
- `«show()»` Affiche la figure courante
- `«savefig(name)»` Sauvegarde la figure courante dans le fichier name.
- `«clf()»` Efface la figure courante
- `«legend(array,loc?)»` Dessine une légende contenant les lignes apparaissant dans array (Option : pour définir l'emplacement)
- `«xlabel(str) ylabel(str)»` Imprime une légende pour décrire les axes horizontaux et verticaux
- `«axis([xl,xr,yb,yt])»` Cadre la figure sur le rectangle décrit par les 4 coordonnées.

14 scipy.stats

Pour faire des statistiques comme avec R. Pour toutes les lois la correspondance suivante.

Python	R	
<code>loi.pdf</code>	<code>dloi</code>	densité $f(x)$
<code>loi.pmf</code>	<code>dloi</code>	calcule $P(X = a)$ pour une loi discrète
<code>loi.cdf</code>	<code>ploi</code>	calcule $P(X \leq a)$
<code>loi.ppf</code>	<code>qloi</code>	renvoie a tel que $P(X \leq a) = b$
<code>loi.rvs</code>	<code>rloi</code>	générer des nombres alatoires (r pour random)

- `«choose(10, 6)»` Pour calculer $\binom{10}{6}$
- `«stats.binom»` C'est la loi binomiale. Par exemple si $X \sim B(10, 0.4)$:
 - `«stats.binom.pmf(5,10,0.4)»` Pour calculer $P(X = 5)$.
 - `«scipy.stats.binom(10,0.4).pmf(5)»` Idem pour calculer $P(X = 5)$.
 - `«stats.binom.cdf(5,10,0.5)»` Pour calculer $P(X \leq 5)$.
 - `«stats.binom.rvs(10,0.5)»` Pour générer des nombres alatoires selon la loi $B(10, 0.4)$.
- `«stats.uniforme»` C'est la loi uniforme. Par exemple si $X \sim U(8, 12)$.
 - `«stats.uniform.pdf(9,8,4)»` $f(9)$ et on trouve 0.25. 4 est la longueur de l'intervalle.
 - `«<+>» <+>`
 - `«<+>» <+>`
 - `«<+>» <+>`
- `«stats.norm»` C'est la loi normale

13 matplotlib.pyplot

- `«plot(x,y,c?)»` Affiche les points définis par les vecteurs x et y, (Option : permet de définir le format et la couleur du tracé)

→ «<++>» <++>

→ «<++>» <++>

→ «<++>» <++>

→ «<++>» <++>

- «stats.expon» C'est la loi exponentielle

→ «<++>» <++>

→ «<++>» <++>

→ «<++>» <++>

→ «<++>» <++>