

```
from math import sin, pi, ...
sin(pi/4) → 0.707...
cos(2*pi/3) → -0.4999...
acos(0.5) → 1.0471...
sqrt(81) → 9.0          √
log(e**2) → 2.0          etc. (cf doc)
```

bloc d'instructions exécuté tant que la condition est vraie

### Instruction boucle conditionnelle

**while** expression logique :

→ bloc d'instructions

**s = 0**  
**i = 1** } initialisations avant la boucle

condition avec au moins une valeur variable (ici **i**)

**while i <= 100:**

# bloc exécuté tant que  $i \leq 100$

**s = s + i\*\*2**  
**i = i + 1** } faire varier la variable de condition !

**print("somme:", s)** } résultat de calcul après la boucle

attention aux boucles sans fin !

### Contrôle de boucle

**break** sortie immédiate

**continue** itération suivante

$$s = \sum_{i=1}^{i=100} i^2$$

bloc d'instructions exécuté pour chaque élément d'un conteneur ou d'un itérateur

### Instruction boucle itérative

**for** variable **in** séquence :

→ bloc d'instructions

Parcours des valeurs de la séquence

**s = "Du texte"** } initialisations avant la boucle  
**cpt = 0** } variable de boucle, valeur gérée par l'instruction **for**

**for c in s:**  
    **if c == "e":**  
        **cpt = cpt + 1**  
    **print("trouvé", cpt, "e")** } Comptage du nombre de e dans la chaîne.

boucle sur dict/set = boucle sur séquence des clés

utilisation des tranches pour parcourir un sous-ensemble de la séquence

Parcours des index de la séquence

□ changement de l'élément à la position

□ accès aux éléments autour de la position (avant/après)

**lst = [11, 18, 9, 12, 23, 4, 17]**

**perdu = []**

**for idx in range(len(lst)):**  
    **val = lst[idx]**  
    **if val > 15:**  
        **perdu.append(val)**  
        **lst[idx] = 15** } Bornage des valeurs supérieures à 15, mémorisation des valeurs perdues.

**print("modif:", lst, "-modif:", perdu)**

Parcours simultané index et valeur de la séquence:

**for idx, val in enumerate(lst):**

**print("v=", 3, "cm :", x, " ", y+4)**

### Affichage / Saisie

éléments à afficher : valeurs littérales, variables, expressions

Options de **print**:

□ **sep=" "** (séparateur d'éléments, défaut espace)

□ **end="\n"** (fin d'affichage, défaut fin de ligne)

□ **file=f** (print vers fichier, défaut sortie standard)

**s = input("Directives: ")**

**input** retourne toujours une chaîne, la convertir vers le type désiré (cf encadré Conversions au recto).

**len(c)** → nb d'éléments

**min(c)** **max(c)** **sum(c)**

**sorted(c)** → copie triée

**val in c** → booléen, opérateur **in** de test de présence (**not in** d'absence)

**enumerate(c)** → itérateur sur (index, valeur)

Spécifique aux **conteneurs de séquences** (listes, tuples, chaînes) :

**reversed(c)** → itérateur inversé **c\*5** → duplication **c+c2** → concaténation

**c.index(val)** → position **c.count(val)** → nb d'occurrences

### Opérations sur conteneurs

Note: Pour dictionnaires et ensembles, ces opérations travaillent sur les clés.

modification de la liste originale

**lst.append(item)**

**lst.extend(seq)**

**lst.insert(idx, val)**

**lst.remove(val)**

**lst.pop(idx)** suppression de l'élément à une position et retour de la valeur

**lst.sort()** **lst.reverse()** tri / inversion de la liste sur place

ajout d'un élément à la fin

ajout d'une séquence d'éléments à la fin

insertion d'un élément à une position

suppression d'un élément à partir de sa valeur

### Opérations sur listes

### Opérations sur dictionnaires

**d[clé]=valeur** **d.clear()**

**d[clé]→valeur** **del d[clé]**

**d.update(d2)** } mise à jour/ajout des couples

**d.keys()** } vues sur les clés,

**d.items()** } valeurs, couples

**d.pop(clé)**

### Opérations sur ensembles

Opérateurs:

| → union (caractère barre verticale)

& → intersection

- ^ → différence/diff symétrique

< <= > >= → relations d'inclusion

**s.update(s2)**

**s.add(clé)** **s.remove(clé)**

**s.discard(clé)**

stockage de données sur disque, et lecture

### Fichiers

**f = open("fic.txt", "w", encoding="utf8")**

variable nom du fichier mode d'ouverture encodage des

fichier pour sur le disque □ 'r' lecture (read) caractères pour les

les opérations (+chemin...) □ 'w' écriture (write) fichiers textes:

cf fonctions des modules **os** et **os.path** □ 'a' ajout (append)... utf8 ascii latin1 ...

en écriture chaîne vide si fin de fichier en lecture

**f.write("coucou")** **s = f.read(4)** si nb de caractères

lecture ligne suivante pas précisé, lit tout le fichier

**f.close()** ne pas oublier de refermer le fichier après son utilisation !

Fermeture automatique Pythonnesque : **with open(...)** as **f:**

très courant : boucle itérative de lecture des lignes d'un fichier texte :

**for ligne in f:**

→ bloc de traitement de la ligne

### Génération de séquences d'entiers

très utilisé pour les boucles itératives **for** par défaut 0 non compris

**range([début,] fin [,pas])**

**range(5)** → 0 1 2 3 4

**range(3, 8)** → 3 4 5 6 7

**range(2, 12, 3)** → 2 5 8 11

**range** retourne un « générateur », faire une conversion

en liste pour voir les valeurs, par exemple:

**print(list(range(4)))**

# Mémento numérique Python 3

**import matplotlib.pyplot as plt** → charge le module pyplot sous le nom **plt**

**plt.figure('titre')** → crée une fenêtre de tracé vide

**plt.plot(LX, LY, 'o-b')** → trace le graphique défini par les listes LX et LY (abscisses et ordonnées)

couleur : 'b' (blue), 'g' (green), 'r' (red), 'c' (cyan), 'm' (magenta), 'y' (yellow), 'k' (black)  
type de ligne : '-' (trait plein), '--' (pointillé), '-.' (alterné)...  
marque : 'o' (rond), 'h' (hexagone), '+' (plus), 'x' (croix), '\*' (étoile)...

**plt.xlim(xmin, xmax)** → fixe les bornes de l'axe x

**plt.ylim(ymin, ymax)** → fixe les bornes de l'axe y

**plt.axis('equal')** → change les limites des axes x et y pour un affichage avec des axes orthonormés (le tracé d'un cercle

**plt.show()** → affichage de la fenêtre donne un cercle)

**plt.savefig(fichier)** → sauve le tracé dans un fichier  
(le suffixe du nom fichier peut donner le format ; par exemple, 'image.png')

**import numpy as np** → charge le module numpy sous le nom **np**

## Construction de tableaux (de type ndarray)

**np.zeros(n)** → crée un vecteur dont les  $n$  composantes sont nulles

**np.zeros((n,m))** → crée une matrice  $n \times m$ , dont les éléments sont nuls

**np.eye(n)** → crée la matrice identité d'ordre  $n$

**np.linspace(a,b,n)** → crée un vecteur de  $n$  valeurs régulièrement espacées de  $a$  à  $b$

**np.arange(a,b,dx)** → crée un vecteur de valeurs de  $a$  incluse à  $b$  exclue avec un pas  $dx$

**M.shape** → tuple donnant les dimensions de  $M$

**M.size** → le nombre d'éléments de  $M$

**M.ndim** → le nombre de dimensions de  $M$

**M.sum()** → somme de tous les éléments de  $M$

**M.min()** → plus petit élément de  $M$

**M.max()** → plus grand élément de  $M$

argument **axis** optionnel : 0 → lignes, 1 → colonnes :

**M.sum(0)** → somme des lignes

**M.min(0)** → plus petits éléments, sur chaque colonne

**M.max(1)** → plus grands éléments, sur chaque ligne

**import numpy.linalg as la**

**la.det(M)** → déterminant de la matrice carrée  $M$

**la.inv(M)** → inverse de  $M$

**la.eig(M)** → valeurs propres de  $M$

**la.matrix\_rank(M)** → rang de  $M$

**la.matrix\_power(M,n)** →  $M^n$  ( $n$  entier)

**la.solve(A,B)** → renvoie  $X$  tel que  $A X = B$

**import scipy.integrate as spi**

**spi.odeint(F,Y0,LT)**

→ renvoie une solution numérique du problème de Cauchy  $Y'(t) = F(Y(t),t)$ , où  $Y$  est un vecteur d'ordre  $n$ , avec la condition initiale  $Y(t_0) = Y_0$ , pour les valeurs de  $t$  dans la liste **LT** de longueur  $k$  commençant par  $t_0$ , sous forme d'une matrice  $n \times k$

**spi.quad(f,a,b)** → renvoie une évaluation numérique de l'intégrale :  $\int_a^b f(t) dt$

## Conversion ndarray <-> liste

**V = np.array([1,2,3])** →  $V$  : vecteur ( 1 2 3 )

**L = V.tolist()** →  $L$  : liste [1, 2, 3]

**M = np.array([[1,2],[3,4]])** →  $M$  : matrice  $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$

**L = M.tolist()** →  $L$  : liste [[1, 2], [3, 4]]

## Extraction d'une partie de matrice

**M[i], M[i,:]** → ligne de  $M$  d'index  $i$

**M[:,j]** → colonne de  $M$  d'index  $j$

**M[i:i+h, j:j+1]** → sous-matrice  $h \times l$

Copier un tableau avec la méthode **copy** :

**M2 = M1.copy()**

**M1+M2, M1\*M2, M\*\*2** → opérations « terme-à-terme »

**c\*M** → multiplication de la matrice  $M$  par le scalaire  $c$

**M+c** → matrice obtenue en ajoutant le scalaire  $c$  à chaque terme de  $M$

**V1.dot(V2)**  
**np.dot(V1,V2)** → renvoie le produit scalaire de deux vecteurs

**M.dot(V)**  
**np.dot(M,V)** → renvoie le produit d'une matrice par un vecteur

**M1.dot(M2)**  
**np.dot(M1,M2)** → renvoie le produit de deux matrices

**M.transpose()**  
**np.transpose(M)** → renvoie une copie de  $M$  transposée (ne modifie pas  $M$ )

**M.trace()**  
**np.trace(M)** → renvoie la trace de  $M$

## Fonctions mathématiques usuelles

**np.exp, np.sin, np.cos, np.sqrt etc.**

→ fonctions qui s'appliquent sur des réels ou des complexes, mais aussi sur des vecteurs et des matrices (s'appliquent à chaque terme), qui sont optimisées en durée de calcul.

Rappel : ce mémento est fourni à titre indicatif. Il ne faut le considérer ni comme exhaustif, ni comme exclusif, ni comme un minimum à connaître absolument (l'examineur n'attend pas du candidat qu'il connaisse parfaitement toutes ces fonctions et ces commandes).