

# Mémento Python 3 pour l'Oral de l'ESM St-Cyr

©2018 – Éric Ducasse

Version ESM-01

Licence Creative Commons Paternité 4

Forme inspirée initialement du memento de Laurent Pointal, disponible ici : <https://perso.limsi.fr/pointal/python:memento>

Cette version est disponible à : <http://www.st-cyr.terre.defense.gouv.fr/index.php/Les-ecoles-de-Saint-Cyr-Coetquidan/Menu-Haut/mediatheque/Concours-ESM-Filiere-scientifique/ESM-Filiere-scientifique-Rapports-de-jury/Rapport-du-jury-ESM-2018-concours-scientifique>

Ceci est une version abrégée du memento pédagogique utilisé à l'ENSAM disponible ici : <https://savoir.ensam.eu/moodle/course/view.php?id=1428>

**help**(*nom*) aide sur l'objet *nom*  
**help**("nom\_module.nom") aide sur l'objet *nom* du module *nom\_module*  
**dir**(*nom*) liste des noms des méthodes et attributs de *nom*

## Aide

Entier, décimal, complexe, booléen, rien

## Types de base

**int** 783 0 -192 0b010 objets non mutables  
**float** 9.23 0.0 -1.7e-6 zéro binaire (→ -1,7×10<sup>-6</sup>)  
**complex** 1j 2+3j 1.3-3.5e2j  
**bool** True False  
**NoneType** None (une seule valeur : « rien »)

## Conteneurs numérotés (listes, tuples, chaînes de caractères)

**list** [1, 5, 9] ["abc"] [] ["x", -1j, ["a", False]]  
**tuple** (1, 5, 9) ("abc",) () 11, "y", [2-1j, True]  
**str** "abc" "z" ""  
Objets non mutables  
Conteneurs hétérogènes  
expression juste avec des virgules → tuple

## Objets itérables

**Nombre d'éléments**

**len**(*objet*) donne : 3 1 0 3

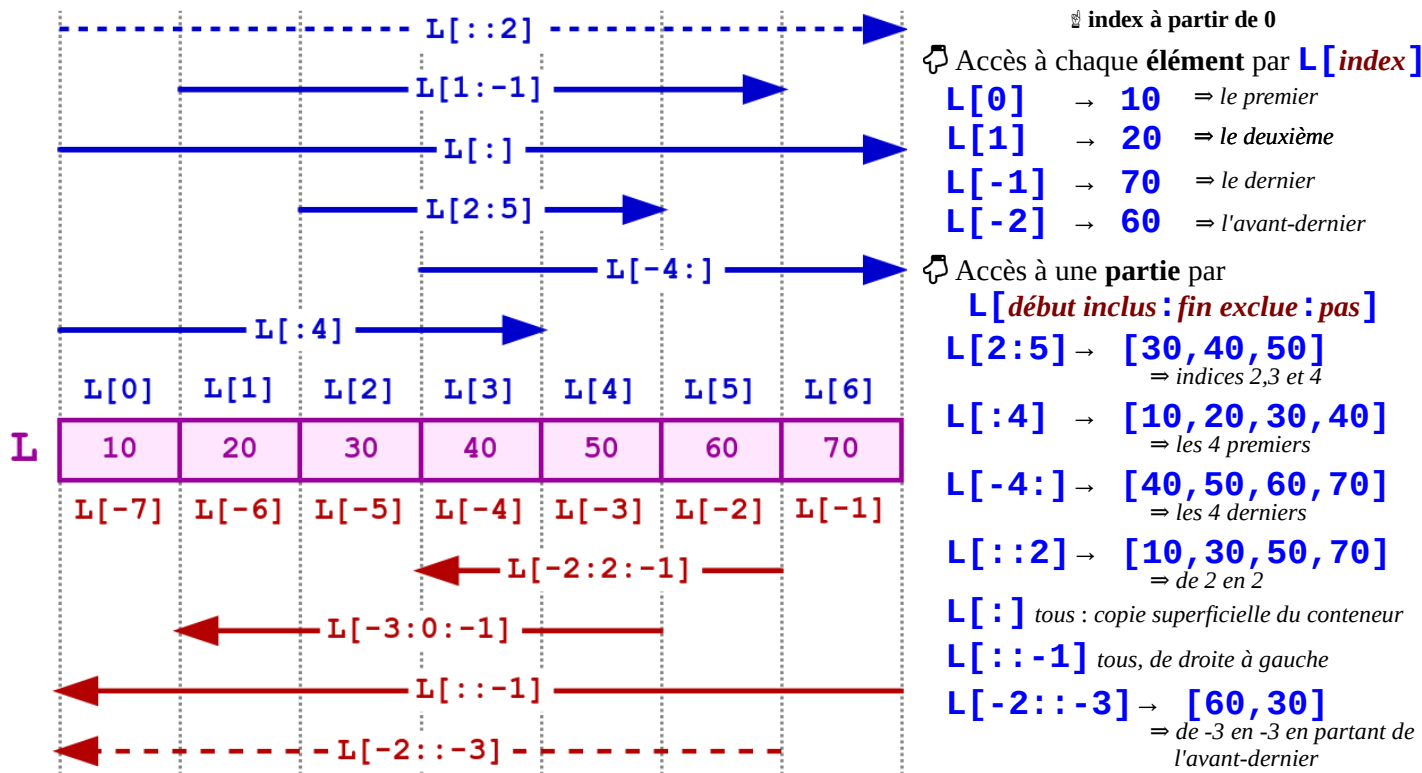
## Itérateurs (objets destinés à être parcourus par in)

**range**(*n*) : pour parcourir les *n* premiers entiers naturels, de 0 à *n*-1 inclus.

**range**(*n*, *m*) : pour parcourir les entiers naturels de *n* inclus à *m* exclus par pas de 1.

**range**(*n*, *m*, *p*) : pour parcourir les entiers naturels de *n* inclus à *m* exclus par pas de *p*.

## Parcours de conteneurs numérotés



## Conteneurs : opérations génériques

**len(c)** **min(c)** **max(c)** **sum(c)**  
**nom in c** → booléen, test de présence dans **c**  
d'un élément identique (comparaison ==) à **nom**  
**nom not in c** → booléen, test d'absence  
**c1 + c2** → concaténation  
**c \* 5** → 5 répétitions (**c+c+c+c+c**)  
**c.index(nom)** → position du premier élément identique à **nom**  
**c.index(nom, idx)** → position du premier élément identique à **nom** à partir de la position **idx**  
**c.count(nom)** → nombre d'occurrences

## Opérations sur listes

**modification « en place » de la liste L originale**  
ces méthodes **ne renvoient rien en général**  
**L.append(nom)** ajout d'un élément à la fin  
**L.extend(itérable)** ajout d'un itérable converti en liste à la fin  
**L.insert(idx, nom)** insertion d'un élément à la position **idx**  
**L.remove(nom)** suppression du premier élément identique (comparaison ==) à **nom**  
**L.pop()** renvoie et supprime le dernier élément  
**L.pop(idx)** renvoie et supprime l'élément à la position **idx**  
**L.sort()** ordonne la liste (ordre croissant)  
**L.sort(reverse=True)** ordonne la liste par ordre décroissant  
**L.reverse()** renversement de la liste  
**L.clear()** vide la liste

**Caractères spéciaux :** **"\n"** retour à la ligne **"\t"** tabulation  
**"\\"** « backslash \ » **"\""** ou **"'"** guillemet " **"'"** ou **"\'"** apostrophe '

## Chaînes de caractères

### Méthodes sur les chaînes

Une chaîne n'est pas modifiable ; ces méthodes **renvoient en général une nouvelle chaîne** ou un autre objet

**"b-a-ba".replace("a", "eu")** → 'b-eu-beu' remplacement de toutes les occurrences  
**"\tUne phrase.\n".strip()** → 'Une phrase.' nettoyage du début et de la fin  
**"des mots\tespacés".split()** → ['des', 'mots', 'espacés']  
**"1.2,4e-2,-8.2,2.3".split(",")** → ['1.2', '4e-2', '-8.2', '2.3']  
**" ; ".join(["1.2", "4e-2", "-8.2", "2.3"])** → '1.2 ; 4e-2 ; -8.2 ; 2.3'

## Conversions

**int("15")** → 15  
**int(-15.56)** → -15 (troncature)  
**round(-15.56)** → -16 (arrondi)  
**float(-15)** → -15.0  
**float("-2e-3")** → -0.002  
**complex("2-3j")** → (2-3j)  
**complex(2, -3)** → (2-3j)  
**list(x)** Conversion d'un itérable en liste  
**list(range(4, -1, -1))** → [4, 3, 2, 1, 0]  
**sorted(x)** Conversion d'un itérable en liste ordonnée (ordre croissant)  
**sorted(x, reverse=True)** Conversion d'un itérable en liste ordonnée (ordre décroissant)  
**tuple(x)** Conversion en tuple  
**str(x)** Conversion en chaîne de caractères

## Mathématiques

**Opérations**  
**+ - \* /**  
**\*\*** puissance **2\*\*10** → 1024  
**//** quotient de la division euclidienne  
**%** reste de la division euclidienne  
**Fonctions intrinsèques**  
**abs(x)** valeur absolue / module  
**round(x, n)** arrondi du **float x** à **n** chiffres après la virgule  
**z.real** → partie réelle de **z**  
**z.imag** → partie imaginaire de **z**  
**z.conjugate()** → conjugué de **z**

## Logique booléenne

**Opérations booléennes**  
**not A** « non A » **True** ou **False**  
**A and B** « A et B »  
**A or B** « A ou B »  
**(not A) and (B or C)** exemple  
**Opérateurs renvoyant un booléen**  
**nom1 is nom2** 2 noms du même objet ?  
**nom1 == nom2** valeurs identiques ?  
**Autres comparateurs :**  
**< > <= >= !=** (≠)  
**nom\_objet in nom\_iterable**  
l'itérable **nom\_iterable** contient-il un objet de valeur identique à celle de **nom\_objet** ?

**import time** **time**  
**time.sleep(s)** pause de **s** secondes  
Évaluation d'une durée d'exécution, en secondes  
**debut = time.time()**  
**:** (instructions)  
**duree = time.time() - debut**

## Liste en compréhension

**Inconditionnelle / conditionnelle**  
**L = [ f(e) for e in itérable ]**  
**L = [ f(e) for e in itérable if b(e) ]**

## Importation de modules

Module **mon\_mod** ↔ Fichier **mon\_mod.py**  
**Importation d'objets par leurs noms**  
**from mon\_mod import nom1, nom2**  
**Importation avec renommage**  
**from mon\_mod import nom1 as n1**  
**Importation du module complet**  
**import mon\_mod**  
:  
... **mon\_mod.nom1** ...  
**Importation du module complet avec renommage**  
**import mon\_mod as mm**  
:  
... **mm.nom1** ...

L'examineur n'attend pas du candidat une connaissance encyclopédique du langage Python, mais une utilisation raisonnée des principes algorithmiques et une mise en pratique des connaissances de base. L'utilisation de l'aide en ligne est encouragée, mais ne doit pas masquer une ignorance sur ces aptitudes. Ce memento est fourni à titre indicatif. Il ne faut le considérer :

- ◇ ni comme exhaustif (en cas de problème sur un exercice particulier, si une fonction ou une commande indispensable était absente de la liste, l'interrogateur pourrait aider le candidat),
  - ◇ ni comme exclusif (une fonction ou une commande absente de ce memento n'est pas interdite, si elle est utilisée à bon escient et que le candidat sait répondre aux questions sur les fonctions de base),
  - ◇ ni comme un minimum à connaître absolument (l'examineur n'attend pas du candidat qu'il connaisse parfaitement toutes ces fonctions et ces commandes).
- Les fonctions et commandes présentées doivent simplement permettre de faire les exercices proposés aux candidats.

# Mémento Python 3 pour l'Oral de l'ESM St-Cyr

## Simulation numérique et calcul scientifique

**import numpy as np**

**Fonctions mathématiques**

Les fonctions de numpy sont vectorisées

**np.pi**, **np.e** → Constantes  $\pi$  et  $e$   
**np.abs**, **np.sqrt**, **np.exp**, **np.log**, **np.log10**, **np.log2** → **abs**, racine carrée, exponentielle, logarithmes népérien, décimal, en base 2  
**np.cos**, **np.sin**, **np.tan** → Fonctions trigonométriques (angles en radians)  
**np.arccos**, **np.arcsin**, **np.arctan** → Fonctions trigonométriques réciproques  
**np.arctan2(y, x)** → Angle dans  $]-\pi, \pi]$   
**np.cosh**, **np.sinh** (trigonométrie hyperbolique)

**Tableaux numpy.ndarray : généralités**

Un tableau **T** de type **numpy.ndarray** (« n-dimensional array ») est un conteneur homogène dont les valeurs sont stockées en mémoire de façon séquentielle.

**T.ndim** → « dimension **d** » = nombre d'indices (1 pour un vecteur, 2 pour une matrice)  
**T.shape** → « forme » = plages de variation des indices, regroupées en tuple  $(n_0, n_1, \dots, n_{d-1})$  : le premier indice varie de 0 à  $n_0-1$ , le deuxième de 0 à  $n_1-1$ , etc.

**T.size** → nombre d'éléments, soit  $n = n_0 \times n_1 \times \dots \times n_{d-1}$

**T.dtype** → type des données contenues dans le tableau

**shp** est la forme du tableau créé, **data\_type** le type de données contenues dans le tableau (**np.float** si l'option **dtype** n'est pas utilisée)

**T = np.zeros(shp, dtype=data\_type)** → tout à 0/False  
**T = np.ones(shp, dtype=data\_type)** → tout à 1/True

**Création d'un tableau**

- Un vecteur **V** est un tableau à un seul indice
- Comme pour les listes, **V[i]** est le (i+1)-ième coefficient, et l'on peut extraire des sous-vecteurs par : **V[:2]**, **V[-3:]**, **V[::-1]**, etc.

Si **c** est un nombre, les opérations **c\*V**, **V/c**, **V+c**, **V-c**, **V//c**, **V%c**, **V\*\*c** se font sur chaque coefficient

Si **U** est un vecteur de même dimension que **V**, les opérations **U+V**, **U-V**, **U\*V**, **U/V**, **U//V**, **U%V**, **U\*\*V** sont des opérations terme à terme

- Produit scalaire** : **U.dot(V)** ou **np.dot(U, V)** ou **U@V**

**Vecteurs**

**Aide numpy/scipy**

**np.info(nom\_de\_la\_fonction)**

**Générateurs**

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

**np.diag(V)** → matrice diagonale dont la diagonale est le vecteur **V**

**Générateurs**

**np.linspace(a, b, n)** → **n** valeurs régulièrement espacées de **a** à **b** (bornes incluses)

**np.arange(x\_min, x\_max, dx)** → de  $x_{min}$  inclus à  $x_{max}$  exclu par pas de **dx**

**Statistiques**

Sans l'option **axis**, un tableau **T** est considéré comme une simple séquence de valeurs

**T.max()**, **T.min()**, **T.sum()**

**T.argmax()**, **T.argmin()** → indices séquentiels du maximum et du minimum

**T.sum(axis=d)** → sommes sur le (d-1)-ème indice

**T.mean()**, **T.std()** → moyenne, écart-type

**Graphiques**

**import matplotlib.pyplot as plt**

**plt.figure(mon\_titre, figsize=(W, H))** crée ou sélectionne une figure dont la barre de titre contient **mon\_titre** et dont la taille est **W** × **H** (en inches, uniquement lors de la création de la figure)

**plt.plot(X, Y, dir\_abrg)** trace le nuage de points d'abscisses dans **X** et d'ordonnées dans **Y** ; **dir\_abrg** est une chaîne de caractères qui contient une couleur ("r"-ed, "g"-reen, "b"-lue, "c"-yan, "y"-ellow, "m"-agenta, "k" black), une marque ("o" rond, "s" carré, "\*" étoile,...) et un type de ligne (" " pas de ligne, "-" plain, "--" dashed, ":" dotted, ...); options courantes : **label=...**, **linewidth=...**, **markersize=...**

**plt.axis("equal")**, **plt.grid()** repère orthonormé, quadrillage

**plt.xlim(a, b)**, **plt.ylim(a, b)** plages d'affichage ; si  $a > b$ , inversion de l'axe

**plt.xlabel(axe\_x, size=s, color=(r, g, b))**, **plt.ylabel(axe\_y, ...)** étiquettes sur les axes, en réglant la taille **s** et la couleur de la police de caractères (**r**, **g** et **b** dans [0,1])

**plt.legend(loc="best", fontsize=s)** affichage des labels des "plot" en légende

**plt.matshow(M, fignum=n)** affichage une représentation de la matrice **M** dans la figure **n**.

**plt.show()** affichage des différentes figures et remise à zéro

Modules **random** et **numpy.random** Tirages pseudo-aléatoires

**import random**  
**random.random()** → Valeur flottante dans l'intervalle [0,1[ (loi uniforme)  
**random.randint(a, b)** → Valeur entière entre **a** inclus et **b** inclus (équiprobabilité)  
**random.choice(L)** → Un élément de la liste **L** (équiprobabilité)  
**random.shuffle(L)** → **None**, mélange la liste **L** « en place »

**import numpy.random as rd**  
**rd.rand(n0, ..., nd-1)** → Tableau de forme  $(n_0, \dots, n_{d-1})$ , de flottants dans l'intervalle [0,1[ (loi uniforme)  
**rd.randint(a, b, shp)** → Tableau de forme **shp**, d'entiers entre **a** inclus et **b** exclu (équiprobabilité)  
**rd.choice(Omega, shp)** → Tableau de forme **shp**, d'éléments tirés avec remise dans **Omega** (équiprobabilité)

**Conversion ndarray ↔ liste**

**T = np.array(L)** → Liste en tableau, type de données automatique  
**L = T.tolist()** → Tableau en liste

**Matrices**

- Une matrice **M** est un tableau à deux indices
- M[i, j]** est le coefficient de la (i+1)-ième ligne et (j+1)-ième colonne
- M[i, :]** est la (i+1)-ième ligne, **M[:, j]** la (j+1)-ième colonne, **M[i:i+h, j:j+l]** une sous-matrice  $h \times l$

- Opérations terme à terme : voir « Vecteurs » ci-contre

- Produit matriciel** : **M.dot(V)** ou **np.dot(M, V)** ou **M@V**

**M.transpose()**, **M.trace()** → transposée, trace

Matrices carrées uniquement (algèbre linéaire) :

**import numpy.linalg as la** ("Linear algebra")

**la.det(M)**, **la.inv(M)** → déterminant, inverse

**la.eigvals(M)** → vecteur des valeurs propres (**vp**)

**vp, P = la.eig(M)** → **P** matrice de passage

**la.matrix\_rank(M)**, → rang

**la.matrix\_power(M, p)** → puissance

**X = la.solve(M, V)** → Vecteur solution de  $MX = V$

**Intégration numérique**

**import scipy.integrate as spi**

**spi.odeint(F, Y0, Vt)** → renvoie une solution numérique du problème de Cauchy  $Y'(t) = F(Y(t), t)$ , avec la condition initiale  $Y(t_0) = Y_0$ , pour les valeurs de **t** dans le vecteur **Vt** commençant par  $t_0$ , sous forme d'une matrice de taille  $\text{len}(Y_0) \times \text{len}(Vt)$

**spi.quad(f, a, b)[0]** → renvoie une évaluation numérique de l'intégrale :  $\int_a^b f(t) dt$

**Lecture de fichier texte**

Le « chemin » d'un fichier est une chaîne de caractères

- Lecture intégrale d'un seul bloc

**with open(chemin, "r") as f:**  
→ **texte = f.read()**

ou

**f = open(chemin, "r")**  
**texte = f.read()**  
**f.close()** (ne pas oublier de fermer le fichier)

- Lecture de la liste de toutes les lignes

**with open(chemin, "r") as f:**  
→ **lignes = f.readlines()**

ou

**f = open(chemin, "r")**  
**lignes = f.readlines()**  
**f.close()**

(Nettoyage éventuel des débuts et fins de lignes)

**lignes = [c.strip() for c in lignes]**

- Lecture et traitement simultanés, ligne par ligne

**with open(chemin, "r") as f:**

→ **for ligne in f:**  
→ (traitement sur ligne)