

* Listes:

Declarer une liste: $l = [1, 2, 3]$ $l = []$ (liste vide)

$l[i]$: retourne le $i^{\text{ème}}$ élément commençant à 0.

$l[-i]$: / / / / / / / / / de la fin de la liste

$l[i:h:j]$: ^{une nouvelle liste qui contient} retourne les éléments ~~de~~ ^{de} i au $-h$ ^{ème} avec un pas $-j$ (de la fin de la liste au début)

$l[i:h:j]$: retourne une nouvelle liste qui contient les éléments $i^{\text{ème}}$ au $h^{\text{ème}}$ élément avec un pas j (du début à la fin)

$l.sort()$: permet le tri croissant

$l.sort(reverse=True)$: permet le tri décroissant

$l.append(\text{element})$: permet d'ajouter un élément à la fin d'une liste.

$l.insert(\text{indice}, \text{element})$: permet d'insérer l'élément à la position de l'indice

$l.remove(\text{element})$: permet de supprimer l'élément (si y a beaucoup de même élément le 1^{er} sera supprimé)

$l.pop(\text{indice})$: permet de supprimer l'élément de l'indice indiqué.

$set(l)$: conversion d'une liste à un set

~~$list(s)$: conversion d'un set à une liste~~ $tuple(l)$: conversion d'une liste à un tuple.

$sorted(l)$: permet de créer une nouvelle liste triée. $mol_liste = sorted(l)$

$sorted(l, reverse=True)$: même chose que $sorted(l)$ mais tri décroissant. $mol_liste = sorted(l, reverse=True)$

$for v \text{ in } l:$
 $\quad print(v)$ } boucle sur une liste avec ses valeurs

$for i \text{ in } range(len(l)):$
 $\quad print(l[i])$ } boucle sur une liste avec ses indices.

* Tuple:

$t = (1, 2, 3)$: déclaration d'un Tuple.

Ex. les mêmes que liste en ce qui concerne $t[i]$, $t[-i]$, $t[i:h:j]$, $t[i:h:j]$

Les Tuples ne sont pas modifiables. (Affiche de boucle set...)

$s = set(t)$: conversion d'un Tuple à un ensemble.

$l = list(t)$: conversion d'un Tuple à une liste.

*Set: collections non ordonnées d'éléments uniques.

$s = \{1, 2, 3\}$: déclaration d'un ensemble

$s.add(element)$: ajouter un élément à un ensemble. (si un même élément existe déjà y'aura pas d'ajout)

$s.remove(element)$: supprimer un élément d'un ensemble (cette méthode génère une erreur si l'élément n'est pas présent)

$s.discard(element)$: même chose que $remove$ mais ne retourne pas d'erreur en cas de non présence de l'élément

$list(s)$: conversion d'un ensemble à une liste

$tuple(s)$: conversion d'un ensemble à un tuple

*Dictionnaire: ils peuvent être modifiable, pas ordonné, pas de clé en double mais les valeurs sont

$d = \{ \text{clé 1: valeur 1,} \}$
 clé 2: valeur 2,
 \vdots
 $\}$ déclaration d'un dictionnaire

$d.keys()$: méthode pour afficher avoir les clés du dictionnaire

$d.values()$: méthode pour avoir les valeurs du dictionnaire

$d[\text{clé}] = \text{valeur}$: ajoute une clé et une valeur au dictionnaire.

$d.items()$: affiche une clé avec sa valeur exemple $[\text{'nom': 'samy'}, (\text{'age', '20'})]$

$\text{for k in d.keys():}$
 print(k) Boucler sur les clés et les afficher une par une

$\text{for v in d.values():}$
 print(v) Boucler sur les valeurs et les afficher une par une

$\text{for m, p in d.items():}$
 print(m, p) Boucler sur les clés et les valeurs de chaque élément et l'afficher (élément par élément).

del d[clé] : supprimer la clé avec sa valeur.

$d.clear()$: permet de supprimer tout les éléments du dictionnaire.

*Structure Conditionnelle: *Boucle:

if C1:
 I1
 elif C2:
 I2
 else:
 I3

$i = 0$
 $\text{while } i < \dots:$
 I1
 I2
 $i += 1$

$\text{for i in range(len(l))}$

I1
 I2

range : utilisé pour générer une seq d'entiers dans un intervalle spécifique

*Fonction:

$\text{def nom_fonction(les_parametres):}$

*Bibliothèque:

$\text{from random import randint}$

#Numpy: Les éléments du Tableau Numpy sont du même type.
import numpy as np: importation de la bibliothèque.

np.array([element, element...]): crée un Tableau array [1, 2, 3] } vecteur horizontal

np.array([[element], [element], ...]): crée un Tableau array $\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$ } vecteur vertical

np.array([[element, element], [element, element]]): crée un Tableau array $\begin{bmatrix} 1, 2, 3 \\ 4, 5, 6 \end{bmatrix}$ } matrice

Type (Tableau): retourne le Type du Tableau "numpy.ndarray"

Tableau.dtype: retourne le Type de données du Tableau (int32, int64, float64)

Tableau.ndim: retourne le nbr de dimensions (axes) qui définit le Tableau

Tableau.shape: retourne la forme du tableau (taille de chaque dimension) (n, m) n étant le nombre de ligne et m le nombre de colonnes.

Tableau.size: retourne la Taille du Tableau n*m, autrement dit le nombre total d'éléments dans le Tableau.

np.arange(start, stop, step): permet de créer un Tableau à 1 dimension allant de la valeur "start" jusqu'à "stop" (excl.) avec un pas "step".

np.linspace(start, stop, nbr): permet de créer un Tableau à 1 dimension allant de la valeur "start" jusqu'à "stop" (inclus) avec "nbr" éléments aux valeurs réparties de façon équitable allant de la valeur "start" jusqu'à "stop" (inclus)

np.zeros(n, m) ou np.ones(n, m): permet de créer un Tableau 2D, avec n lignes et m colonnes, remplis de zéro ou de 1.

np.tile(tableau, y): permet de créer une nouvelle matrice en répétant une Matrice ou un Tableau donné un certain nombre de fois dans différentes directions.
 np.tile(a, (3, 2))
 3 lignes, 2 colonnes

Tableau.T: permet de faire la Transposée d'une Matrice ou (np.transpose(tab))

np.add(a, b): permet d'additionner 2 Matrices.

np.multiply(a, b): permet le produit de 2 Matrices des éléments a et b.

np.matmul(a, b): permet le produit Matriciel.

Remarque:

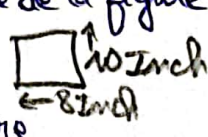
- Pour faire un produit de 2 Matrices faut que le nbr de lignes de la 1^{ère} matrice soit égal au nbr de colonnes de la 2^{ème} matrice

- array occupe moins d'espace et il est plus rapide que la liste mais les Tableaux sont moins flexibles car ils acceptent pas de mix de types

* Matplotlib:

from matplotlib import pyplot as plt: importation de la bibliothèque.

* Methode Simple:

plt.figure(figsize=(8,6)): créer la fenêtre de la figure, avec comme paramètre figure pour attribuer une taille à la fenêtre. 
plt.title('...'): mettre un titre à la figure
plt.subplot(): permet de créer un axe
plt.plot(x,y): dessiner la courbe définie par les données x et y
plt.xlabel('...'): mettre un nom ('libellé') à l'axe des abscisses (x)
plt.ylabel('...'): mettre un nom ('libellé') à l'axe des ordonnées (y)
plt.show(): Afficher la figure à l'écran.

exemple:
 $x = np.arange(0, 10, 2)$
 $y = x ** 2$

Remarque: si on a plus d'un graphe,
- plt.subplot(nrows, ncols, index) d'où "nrows" nombre de lignes dans la grille, "ncols" nombre de colonnes dans la grille et "index" est l'index du sous graphique de l'écran.
exemple: plt.subplot(2,1,1): diviser la figure en une grille de 2 lignes par 1 colonne et index 1 donc c'est le premier sous graphique (il se place en haut)
- on peut rajouter c='couleur', ls='-', lw='3' épaisseur dans plt.plot

* Methode orienté objet:

fig, ax = plt.subplots(): subplots avec S initialise fig avec l'objet figure crée un axe ax.
fig.suptitle('...'): mettre un titre à la figure.
ax.plot(x,y): dessiner la courbe définie par les données x et y.
ax.set_xlabel('...'): mettre un nom ('libellé') à l'axe des abscisses.
ax.set_ylabel('...'): mettre un nom ('libellé') à l'axe des ordonnées.
plt.show(): Afficher la figure à l'écran.

Remarque: si on a plus d'un graphe,
- fig, ax = plt.subplots(nbr ligne, nbr col) dans la grille.
ax[0].plot(x,y): Tracer le premier graphique sur le premier axe.
ax[0].set_ylabel('...'): titre pour l'axe y.
ax[0].set_title('...'): titre premier graphique
ax[1].plot(x,y)
ax[1].set_ylabel('...')
plt.scatter('...'): Tracer un nuage de points.

* Pandas:

import pandas as pd : importation de la bibliothèque (X)

Dataframe: structure tabulaire à 2 dimensions permettant de stocker des données, où les lignes représentent des observations, les colonnes représentent des attributs, avec des indices pour les lignes et des noms pour les colonnes, simplifiant ainsi la Manip des données.

* À partir d'un Tableau Tuple:

data = [(1, 2.0, "Hello"), (2, 3.0, "World")]

df = pd.DataFrame(data) # Affiche le dataframe avec index et nom de colonnes par défaut.

print(df)

	Nom colonne		
	0	1	2
Index: 0	1	2.0	Hello
1	2	3.0	World

df1 = pd.DataFrame(pd.DataFrame(data, index=["premier", "second"], columns=["A", "B", "C"]) # Avec une liste d'index et de colonnes
print(df1)

	A	B	C
premier	1	2.0	Hello
second	2	3.0	World

* À partir d'un dictionnaire ndarray:

d = {"one": [1.0, 2.0, 3.0], "two": [4.0, 3.0, 2.0]}

df = pd.DataFrame(d) # index par défaut.
Les clés du dictionnaire représentent les noms des colonnes

print(df)

	one	two
0	1.0	4.0
1	2.0	3.0
2	3.0	2.0

df1 = pd.DataFrame(d, index=["a", "b", "c"]) # avec une liste d'index.
print(df1)

	one	two
a	1.0	4.0
b	2.0	3.0
c	3.0	2.0

* À partir d'une liste des dictionnaires:

data2 = [{"a": 1, "b": 2}, {"a": 5, "b": 10, "c": 20}]

df = pd.DataFrame(data2) # index par défaut.
Les clés du dictionnaire représentent les noms des colonnes.

print(df)

	a	b	c
0	1	2	NaN
1	5	10	20.0

df1 = pd.DataFrame(data2, index=["first", "second"], columns=["a", "b"])
print(df1)

	a	b
first	1	2
second	5	10

NaN = Not a Number, valeur par défaut pour les valeurs manquantes.

* Importation de données externes:

df = pd.read_csv(chemin vers fichier.csv) lire fichier csv (valeurs séparées par ;)
df = pd.read_excel(chemin vers fichier.xlsx) lire fichier (xls, xlsx, xlsm, xlst, ...)

* Fonctions et attributs d'un DataFrame :

df.head(): retourne les 5 premières lignes du jeu de données.

df.tail(): retourne les 5 dernières lignes du jeu de données.

df.describe(): retourne une description des données, avec ^{des} indicateurs de statistiques (moyenne, min, max, fréquence...)

df.shape: retourne les dimensions sous forme d'un tuple (nb. lignes, nb. colonnes). La ligne d'entête n'est pas comptabilisée dans le Nbr de lignes.

df.size: retourne la Taille des données = nb. lignes x nb. colonnes.

df.columns: retourne une liste des noms des différentes colonnes.

df.index: retourne une liste ~~des~~ des indices.

df.dtypes: retourne le Type de chaque colonne.

* Manipulation des données :

* Sélection de colonne :

print(df["A"]) # fonction aussi avec print(df.A)

print(df["A", "B"]) # sélectionne 2 colonnes à la fois.

print(df["A"][0]) # sélectionner le 1^{er} élément de la colonne A.

print(df["A"][1:2]) # sélectionner les éléments de la colonne A (de 0 à 1)

* Ajout de colonne :

df["D"] = df["A"] + df["B"] # ajouter une colonne D contenant la somme des valeurs de A et B.

df["F"] = "OK" # ajouter une colonne F dont les valeurs = "OK"

* Suppression de colonne :

del df["C"] # Supprimer définitivement la colonne C.

valeurs_F = df.pop("F") # extraire le contenu de la colonne F dans valeurs_F et la supprimer de df.

* Suppression de lignes et/ou de colonnes :

df.drop("A", axis=1, inplace=True) # Supprimer la colonne A et applique le changement sur place.

df.drop("1", axis=0, inplace=True) # Supprimer la ligne 1.

(Autre Exemple page 14 du Cours Python)

6

* Indexing et Slicing:

- loc: sélectionne les lignes et les colonnes par leur label (nom)
- iloc: sélectionne / / / / / par leur numéro de position.

exemple:

df2 {

	A	B	C	D
a	-1.515	-0.850	-0.933	-1.192
b	-0.581	-0.232	0.953	-0.272
c	1.335	-0.102	-0.045	-1.395
d	0.897	1.156	0.410	-0.818
e	-1.329	-0.337	-0.193	0.472

print(df2.loc["b"]) # sélectionne la ligne b.

print(df2.iloc[1]) # sélectionne la ligne 1 qui est la ligne b. x

print(df2.loc[:, "B"]) # sélectionne toutes les lignes de la colonne B

print(df2.iloc[:, 1]) # sélectionne toutes les lignes de la colonne qui est B. x

* isna() ; notna() ; fillna() et dropna():

isna(): retourne True si Valeur = NaN, False sinon

notna(): retourne True si Valeur différente de NaN, False sinon.

df.fillna(): remplacer les valeurs manquantes par de nul valeurs ^{df.fillna(0, inplace=True)}

df.dropna(): ~~supprimer~~ les lignes ou les colonnes qui ont des valeurs manquantes.

* Sklearn:

* Chargement de données: (exemple: Page 25 du Cours python)

from sklearn import datasets: importation de la bibliothèque.

load_boston(): charge et retourne le data set des prix de maison de boston (régression)

load_iris(): charge et retourne le dataset de fleurs d'iris (classification)

load_diabetes(): charge et ret / / / / / pour le diabète (régression)

load_digits(n=10): / / / / / pour la reconnaissance de digits (classification)

* Diviser le jeu de données: (exemple: page 24 du Cours python)

from sklearn.model_selection import train_test_split: Bibliothèque

train_test_split(array, test_size=..., train_size=..., random_state=...):

Cette fonction reçoit notre jeu de données (X et Y) et nous donne en sortie notre jeu de données divisé en un jeu d'entraînement (X_train, Y_train) et un jeu de test (X_test, Y_test)

- arrays: représente les x et y du jeu de données de base.
- test-size: float allant de 0.0 à 1.0, indiquant la proportion allant dans la partie test. optionnel si train-size est déjà spécifié
- train-size: même chose que test-size indiquant la proportion allant dans la partie train.
- random-state: permet d'avoir des données reproductibles à chaque appel à la fonction (pour garder la même division de données à chaque fois)
- la fonction retourne une liste des nouveaux set de données: $x_{\text{train}}, x_{\text{test}}, y_{\text{train}}, y_{\text{test}}$.

* Entraînement des Modèles: (X)

- Importer l'estimateur de l'algo à utiliser:
 - Initialiser les arguments du constructeur de l'estimateur avec les paramètres du Modèle (si nécessaire).
 - Entraîner le Modèle: sur les données d'entraînement en utilisant la méthode `fit`
- exemple TP: (X)

entraînement du Modèle k plus proche voisins

`from sklearn.neighbors import KNeighborsClassifier` } Importer l'estimateur

considérer les 3 plus proches voisins

`classifier_knn = KNeighborsClassifier(n_neighbors=3)` } Initialiser les Arg du Constructeur

entraîner le modèle:

`classifier_knn.fit(x_train, y_train)`

* Phase de Test et Mesure d'évaluation: (X)

`from sklearn import metrics`: importer toutes les métriques.

`y_pred = model.predict(x_test)` tester le modèle en faisant une prédiction sur les données de test.

`accuracy = metrics.accuracy_score(y_test, y_pred)` estimer les performances du Modèle en termes d'accuracy, en comparant les prédictions du Modèle (`pred`) avec les valeurs réelles (`y_test`)