université angers

Licence 3 Informatique
Python

Introduction à NumPy et MatPlot

Pour l'ensemble des TP vous pourrez utiliser au choix l'EDI PyCharm ou le docker Jupyter notebook. Pour lancer les éditeurs :

- Ouvrir un terminal,
- `pycharm-community` ou `_jupyter_notebook`
- Créer un nouveau projet pour chaque TP et un fichier pour chaque exercice.

ATTENTION : Chaque TP fera l'objet d'un dépôt sur Moodle qui sera noté à chaque séance.
Date limite de dépôt pour le TP1 **jeudi 6 Février 2020, 23h59.**

Une documentation Python est disponible à cette adresse : https://docs.python.org/3/tutorial/
Une documentation NumPy est disponible à cette adresse : https://docs.scipy.org/doc/numpy/reference/
Une documentation MatPlotLib : https://matplotlib.org/3.1.1/api/index.html

Exercice 1 - Utilisation des listes en Python

1. Construire une liste `liste1` qui contient les nombres de 1 à 99 (avec la fonction `range`).
2. Ajouter à l'aide d'une boucle les éléments de 100 à 199, à une liste vide `liste2`.
3. Ajouter la liste `liste2` à la liste `liste1`.
4. Inverser `liste1` pour qu'elle aille de 199 à 1.
5. Afficher `liste1` dans l'ordre croissant (2 possibilités en modifiant la liste et sans la modifier).
6. Avec la fonction `range()`, construire une `liste3` composée de tous les nombres pairs de 0 à 100 (inclus).
7. À l'aide d'une boucle supprimez tous les multiples de 10 de `liste3`.
8. Affichez la longueur de `liste3`
9. Affichez tous les multiples de 8 de `liste3`
10. Affichez les 20 premiers éléments de `liste3`
11. Affichez les éléments suivants les 20 premiers éléments de `liste3`
12. Affichez les 10 derniers éléments de `liste3`

    Python propose la compréhension de listes permettant de construire une liste en une seule ligne avec la forme :

    ```
    newList = [valeur for element in liste if condition]
    ```

    La nouvelle liste sera composée des éléments `valeur` si la condition est respectée.
13. Créer une `liste4` composée uniquement des nombres impairs en une ligne

14. Construire une `liste5` comprenant la suite de fibonacci avec une compréhension de liste en initialisant les deux premières valeurs de la liste à 1

    Il est aussi possible d'appliquer une fonction à une liste à l'aide de fonctions de callback. Une fonction de callback a pour forme :

    ```
    def mafonctioncallback(callback, liste):
        nouvelle_liste = []
        for element in liste:
    ```

```
        [action a realiser sur liste a l'aide de la fonction callback]
    return nouvelle_liste

def mafonction(x): return [calcul sur x]
```

15. Écrire une fonction de callback qui permet d'effectuer un map (appliquer le carré sur une liste et appliquer un test de parité (renvoie une liste contenant True ou False pour chaque élément))
16. Écrire une fonction de callback qui permet d'effectuer un filtre (retourne une liste contenant uniquement des nombres pairs)
17. Écrire une fonction de callback qui permet d'effectuer un reduce (retourner la somme des éléments, retourner le produit des éléments)
18. Écrire une fonction minMaxMoy() qui reçoit une liste d'entiers et qui renvoie le minimum, le maximum et la moyenne de cette liste. Le programme principal appellera cette fonction avec la liste : [10, 18, 14, 20, 12, 16] et sauvegardera ces trois résultats dans un fichier "minMaxMoy.txt"
19. Créer une `liste6` avec les valeurs [1,2,3] puis créer une copie de `liste6` dans une `liste6bis` (attention si on change une valeur de `liste6bis`, la `liste6` ne doit pas être modifiée)

Exercice 2 - Utilisation des tableaux NumPy

NumPy est une bibliothèque qui propose des traitements efficaces sur les tableaux de nombres. Il est possible d'utiliser des tableaux d'objets mais cela réduit l'efficacité des fonctions intégrées à NumPy.

Tableaux à une dimension

1. Importer la librairie NumPy.
2. Créer un numpy array vide de dimension 1 et de taille 10.
3. Créer un numpy array de dimension 1 et de taille 10 contenant toujours la valeur 0.
4. Créer un numpy array de dimension 1 et de taille 10 contenant toujours la valeur 1.
5. Créer un numpy array de dimension 1 et de taille 10 contenant toujours la valeur 5 (2 méthodes).
6. Créer un numpy array de dimension 1 contenant les 10 premiers entiers au format chaîne de caractères.
7. Créer un numpy array `tab1` de dimension 1 contenant tous les nombres de 1 à 10 avec un pas de 0,5.
8. Créer un numpy array `tab2` de dimension 1 contenant 20 nombres de 1 à 10 (à intervalle régulier).
9. Créer un numpy array `tab3` de dimension 1 avec les entiers de 10 à 30.
10. Calculer la somme des entiers de `tab3` (une seule ligne).
11. Calculer la somme cumulée des entiers de `tab3` (une seule ligne).
12. Calculer la moyenne des entiers de `tab3` (une seule ligne).
13. Effectuer le produit scalaire des vecteurs `tab2` et `tab3`.
14. Sélectionner les éléments de `tab3` dont la valeur est supérieure à 20.
15. Sélectionner les éléments 1, 5, 10 et 15 de `tab3`.

Les matrices

1. Générer une matrice identité de taille 9.
2. Créer un numpy array de dimension 2 et de taille 10×10 contenant toujours la valeur 1.
3. Afficher la taille de la mtrice précédente.
4. Afficher la dimension de la matrice précédente.
5. Afficher la forme de la matrice précédente.
6. Créer un numpy array de dimension 1 avec les valeurs de 0 à 8 et le transformer en matrice 3×3
7. Créer la matrice A suivante :

$$\begin{Bmatrix} 0.01 & 0.02 & 0.03 & 0.04 & 0.05 & 0.06 & 0.07 & 0.08 & 0.09 & 0.1 \\ 0.11 & 0.12 & 0.13 & 0.14 & 0.15 & 0.16 & 0.17 & 0.18 & 0.19 & 0.2 \\ 0.21 & 0.22 & 0.23 & 0.24 & 0.25 & 0.26 & 0.27 & 0.28 & 0.29 & 0.3 \\ 0.31 & 0.32 & 0.33 & 0.34 & 0.35 & 0.36 & 0.37 & 0.38 & 0.39 & 0.4 \\ 0.41 & 0.42 & 0.43 & 0.44 & 0.45 & 0.46 & 0.47 & 0.48 & 0.49 & 0.5 \\ 0.51 & 0.52 & 0.53 & 0.54 & 0.55 & 0.56 & 0.57 & 0.58 & 0.59 & 0.6 \\ 0.61 & 0.62 & 0.63 & 0.64 & 0.65 & 0.66 & 0.67 & 0.68 & 0.69 & 0.7 \\ 0.71 & 0.72 & 0.73 & 0.74 & 0.75 & 0.76 & 0.77 & 0.78 & 0.79 & 0.8 \\ 0.81 & 0.82 & 0.83 & 0.84 & 0.85 & 0.86 & 0.87 & 0.88 & 0.89 & 0.9 \\ 0.91 & 0.92 & 0.93 & 0.94 & 0.95 & 0.96 & 0.97 & 0.98 & 0.99 & 1.0 \end{Bmatrix}$$

8. Extraire la matrice B suivante de la matrice A précédente :

$$\begin{Bmatrix} 0.61 & 0.62 & 0.63 & 0.64 \\ 0.71 & 0.72 & 0.73 & 0.74 \\ 0.81 & 0.82 & 0.83 & 0.84 \\ 0.91 & 0.92 & 0.93 & 0.94 \end{Bmatrix}$$

9. Générer une matrice de 1 de la taille de la matrice B.
10. Extraire la matrice C suivante de la matrice A :

$$\begin{Bmatrix} 0.21 & 0.23 & 0.25 & 0.27 \\ 0.31 & 0.33 & 0.35 & 0.37 \end{Bmatrix}$$

11. Calculer le produit matriciel entre la matrice C et la matrice B.
12. Récupérer la valeur 0.55 dans la matrice A
13. Récupérer la 4ème colonne de la matrice A
14. Récupérer la 4ème ligne de la matrice A
15. Récupérer la matrice A en ne gardant que les lignes telles que la valeur de la colonne 4 est supérieure à 0.5
16. Calculer la moyenne globale de cette matrice A.
17. Calculer la moyenne de chaque ligne de la matrice A.
18. Calculer la moyenne de chaque colonne de la matrice A.
19. Calculer le produit scalaire de la première colonne de la matrice A avec la première ligne de la matrice A.
20. Remplacer le centre de la matrice par la valeur 0 comme l'exemple suivant :

$$\begin{Bmatrix} 0.01 & 0.02 & 0.03 & 0.04 & 0.05 & 0.06 & 0.07 & 0.08 & 0.09 & 0.1 \\ 0.11 & 0.12 & 0.13 & 0.14 & 0.15 & 0.16 & 0.17 & 0.18 & 0.19 & 0.2 \\ 0.21 & 0.22 & 0.23 & 0.24 & 0.25 & 0.26 & 0.27 & 0.28 & 0.29 & 0.3 \\ 0.31 & 0.32 & 0.33 & 0 & 0 & 0 & 0 & 0.38 & 0.39 & 0.4 \\ 0.41 & 0.42 & 0.43 & 0 & 0 & 0 & 0 & 0.48 & 0.49 & 0.5 \\ 0.51 & 0.52 & 0.53 & 0 & 0 & 0 & 0 & 0.58 & 0.59 & 0.6 \\ 0.61 & 0.62 & 0.63 & 0 & 0 & 0 & 0 & 0.68 & 0.69 & 0.7 \\ 0.71 & 0.72 & 0.73 & 0.74 & 0.75 & 0.76 & 0.77 & 0.78 & 0.79 & 0.8 \\ 0.81 & 0.82 & 0.83 & 0.84 & 0.85 & 0.86 & 0.87 & 0.88 & 0.89 & 0.9 \\ 0.91 & 0.92 & 0.93 & 0.94 & 0.95 & 0.96 & 0.97 & 0.98 & 0.99 & 1.0 \end{Bmatrix}$$

21. Récupérer les indexes auxquels sont situées les valeurs 0

Les nombres aléatoires

1. Générer un numpy array de 10 valeurs aléatoires
2. Générer un numpy array de 100 valeurs entières entre 0 et 100
3. Générer un numpy array de taille 25 avec des nombres tirés aléatoirement suivant une distribution normale centrée réduite.
4. Générer un numpy array de taille 25 avec des nombres tirés aléatoirement suivant une loi uniforme.
5. Générer une matrice aléatoire de dimension 2 et de taille 10×10.
6. Modifier votre programme pour que les nombres aléatoires générés soient toujours les mêmes à chaque exécution.

Les entrées/sorties

1. Sauvegarder une matrice aléatoire de taille 10×10 dans un fichier matrice1.csv avec ' ;' comme séparateur.
2. Charger le fichier matrice1.csv

Exercice 3 - Utilisation de Matplot

1. À partir de deux tableaux [1,4,8,9] et [12,25,34,78] afficher un nuage de points.
2. À partir d'un tableau numpy aléatoire de 1000 entiers afficher un nuage de points.
3. À partir de deux tableaux [1,4,8,9] et [12,25,34,78] tracer une courbe.
4. À partir d'un tableau numpy aléatoire de 100 nombres suivant une distribution normale tracer une courbe.
5. Tracer trois courbes avec x allant de 0 à 2 avec 100 valeurs, représentant une fonction linéaire, carrée et cubique sur la même figure.
6. Ajouter une légende à ces trois courbes signifiant linéaire, quadratique, cubique.
7. Ajouter un titre "Comparaison des fonctions linéaire, quadratique et cubique".
8. Changer les labels par abscisse et ordonnée.
9. Retracer les trois courbes mais dans 3 figures les unes à côté des autres en attribuant les couleurs bleue, orange et verte.
10. Changer la taille de la figure par (20,7).
11. À partir d'un tableau numpy aléatoire uniforme de 100 éléments allant de 0 à 10 tracer un histogramme constitué de 10 bandes verticales.

Exercice 4 - Activité pratique Cette activité pratique sera à déposer sur moodle. L'activité est divisée en deux parties et vous devrez déposer un fichier par partie.

Partie 1 : Génération d'un histogramme de la répartition du QI

1. Choisir une graine pour la génération de nombres aléatoires.
2. Créer un numpy array avec une distribution normale de 500 nombres aléatoires.
3. Affecter à cette distribution un coefficient de σ + μ avec par défaut $\sigma = 15$ et $\mu = 100$ ($fx = x \times \sigma + \mu$)
4. Générer l'histogramme correspondant avec 50 bandes verticales.
5. Changer l'axe y par "Probabilité de densité"
6. Changer l'axe x par "Quotient intellectuel"
7. Ajouter un quadrillage
8. Calculer la fonction représentant une loi normale à partir de σ et μ. Rappel de la fonction :

$$y = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

   Ici x sera remplacé par $bins$ qui correspond aux valeurs des abscisses de chaque bande verticale de l'histogramme
9. Tracer la fonction y en pointillés.
10. Changer le titre par "Histogramme de la répartition du QI : $\mu = 100,\ \sigma = 15$"

Partie 2 : Gestion de données issues d'un fichier au format csv Charger les fichiers redwine.csv et whitewine.csv dans deux numpy array redwine et whitewine. Les données sont stockées de cette manière :

0	"fixed acidity"
1	"volatile acidity"
2	"citric acid"
3	"residual sugar"
4	"chlorides"
5	"free sulfur dioxide"
6	"total sulfur dioxide"
7	"density"
8	"pH"
9	"sulphates"
10	"alcohol"
11	"quality"

1. À partir du fichier redwine.csv, dans un graphique pyplot :
2. Afficher le degré d'alcool en fonction de sa qualité.
3. Changer les labels pour afficher qualité et degré d'alcool.
4. Afficher la moyenne à l'aide de points rouges pour chaque valeur de qualité.
5. Effectuer le même traitement avec le fichier whitewine.csv.
6. Sur un même graphique, comparer les moyennes de dégré d'acools en fonction de la qualité des deux tableaux de données.
7. Pour les deux fichiers, afficher la valeur médiane et les quartiles pour chaque valeur de qualité sous la forme d'un box-plot (boite à moustache).