TP 1 - Introduction à l'IA

I. Les notions abordées

Environnement python, jupyter-notebook, tableaux numpy, import de fichiers csv, affichage de fonctions et d'images.

II. Objectifs

Comprendre/créer/utiliser un environnement python avec conda.
Savoir utiliser un jupyter-notebook.
Installer et utiliser les packages de bases de l'IA, comme Numpy, Pandas, Matplotlib, ?.
Manipuler des tableaux numpy, importer des données d'un fichier csv avec pandas, afficher des fonctions et des images avec matplotlib.

III. Tables des matières

IV. Enoncé du TP

Ouvrir une session sous Linux.
Dans le terminal vérifier si python est installé et sa version.

Si python n'est pas installé, installer Anaconda.

1. Création de l'environnement python et installation des packages

Créer votre environnement python avec conda :

```
$ conda create  -- name myenv
$ conda activate myenv
```

Expliquer l'intérêt d'utiliser un environnement python.
Une fois dans l'environnement installer les packages suivant :
• Numpy
• Jupyter-notebook
• Pandas
• Matplotlib

La documentation des environnements conda est dispo à : https://conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html

2. Création du jupyter-notebook

Dans le terminal lancer l'interface jupyter notebook dans le dossier de votre choix, avec la commande :

```
$ jupyter notebook
```

Ouvrer un notebook vide.

3. Introduction au langage Python

- **Aide en ligne de commande :**

```
t = (8,9)
print(t)
```

Si l'on veut savoir quelles sont les méthodes de l'objet t, on peut utiliser la fonction dir :
dir(t)
Pour plus d'informations sur les méthodes :
help(t)

Injecter le contenu d'une variable dans une chaine de caractères :

```
x=5
message = f'2+3 = {x}'
print(message)
```

Ecrire une fonction qui prend en argument deux entiers et qui print :
« arg1 + arg2 est égal à arg1 +arg2 »
La fonction doit retourner un tuple contenant arg1, arg2, et la somme.

- **Les types de bases :**
    - bool (booléen),
    - int (entier),
    - float (réel),
    - str (chaîne de caractères),
    - tuple (vecteur)

ex:

```
boolean = 1 < 2   #prend 2 valeurs possibles : True ou False
entier = 1
reel = 3.14
chaine = "pi"
tuple  = (entier, reel, chaine)
```

Faire un print de la variable tuple.

Le type de tuple et des éléments de tuple sont ils différents ?

- **Listes**

```
liste = [entier, reel, chaine]
print(liste)
```

Afficher le 3e élément de liste.
Générer une liste de longueur aléatoire comprenant des entiers aléatoires entre 0 et 10.

```
import random
longueur = random.randint(1,5000)
liste = random.sample(range(0, 10000), longueur)
```

Print la longueur de la liste sans regarder la valeur de la variable longueur.
Print le dernier élément de la liste de deux manières différentes.
Print l'entier le plus grand de la liste en triant la liste par ordre croissant.

-> Utiliser la mehtode .sort()

- **Dictionnaires**

Les dictionnaires sont des tableaux associatifs, autrement dit des ensembles de paires {clé : valeur}, par exemple :

```
cours = { 'RCP208' : 2016, 'RCP209' : 2017, 'RCP216' : 2017 }
print(cours)
```

Obtenir les clés du dictionnaire
print(dico.keys())

Obtenir les valeurs du dictionnaire
print(dico.values())

Obtenir les paires (clé, valeur)
print(dico.items())

- **Structures de contrôle**
  if / elif / else

Créer une liste de 20 entiers aléatoires (range de valeurs entre 0 et 50).
Créer une fonction qui prend en argument la liste et qui retourne la valeur et l'indice du plus grand nombre de la liste.

Créer une deuxième fonction qui prend en argument la liste et qui retourne une nouvelle liste contenant uniquement les nombres pairs de la liste.

- **Ecriture et lecture de fichiers .txt**

Ouverture du fichier en mode écriture ('w' pour 'write')

L'instruction with permet de ne pas avoir à fermer manuellement le fichier en utilisant un gestionnaire de contexte.

```
with open('texte.txt', 'w') as fichierTexte:
    fichierTexte.write('Mon texte\n')
```

Le mode 'a' permet d'ajouter des lignes sans écraser le fichier ('append')

```
with open('texte.txt', 'a') as fichierTexte:
    fichierTexte.write('Et une ligne de plus\n')
```

```
with open('texte.txt') as fichierTexte:
    print(fichierTexte.read())
```

4. Manipulation de tableaux numpy.

Importer le package numpy dans le notebook.

Essayer la création de tableau numpy 1D:

```
np.array([a, b, c, d])
np.arange(start, stop, step)
np.linspace(start, stop, nombre de val)
```

Création de matrices (tableaux 2D):

```
np.array([[a, b, c],[d, e, f]])
np.ones((nb de lignes, nb de colonnes))
np.zeros((nb de lignes, nb de colonnes))
```

Créer une fonction qui prend en argument un tableau numpy 2d et qui retourne sa transposée.

Créer une fonction qui prend un argument une matrice et un vecteur (tableau 1D), et qui retourne le produit matrice vecteur. La fonction doit tester au préalable si le produit est possible. La fonction doit utiliser 2 boucles for et vous ne devez pas utilisé np.transpose ou @.

Créer une fonction qui prend en argument deux tableaux 2D numpy et qui retourne le produit matricielle des deux matrices. La fonction doit vérifier que le produit matricielle est possible. La fonction doit utiliser 3 boucles for et vous ne devez pas utilisé np.transpose ou @.

Les méthodes suivantes permmettent de réaliser ces trois fonctions directement:

```
# A, B: matrices
# y = vector

A_transposée = A.T
b = A@y # produit matrice vecteur
C = A@B # produit matricielle
```

Créer une fonction qui prend en arg un vecteur et un entier p. La fonction doit retourner la norme p du vecteur. Vous pouvez tester la fonction en comparant avec la fonction np.linalg.norm