Apprentissage artificiel

Initiation au langage python

Résumé

Ceci est est un tutorial (très accéléré) de Python afin de voir les bases nécessaires pour le cours d'apprentissage artificiel

1 L'environnement python

1.1 Utilisation des environnements virtuels

1.1.1 Mise en place

Pour utiliser Python nous allons utiliser un environnement virtuel. Avant toute chose il vous faut installer le paquet `python3-virtualenv`.

Ensuite nous allons créer un environnement `apprentissageArtificiel`.

```
python3 -m virtualenv ~/apprentissageArtificiel
source ~/apprentissageArtificiel/bin/activate
```

Pour sortir de l'environnement il faut simplement taper `deactivate`.

1.2 Script/interactif

Il exsite deux façons d'utiliser Python.

— Mode intéractif : pour celà il vous suffit de taper `python` dans votre console ou votre environnement. Il est très utile pour rapidement tester du code ou des fonctionnalités.

— Mode "script" : pour celà vous allez écrire un fichier contenant votre code, et ensuite exécuter ce fichier. Par convention on utilisera l'extention *.py* pour les fichiers python. En haut du fichier il faut spécifier que l'on écrit un script python :

```
#!/usr/bin/python3
```

Une fois le fichier écrit il faut donner les droits d'exécution au fichier puis le lancer :

```
chmod +x monfichier.py
./monfichier.py
```

1.3 Les librairies

Une des forces du langage Python réside dans le nombre important de librairies disponibles. Ces librairies sont à télécharger et à installer. Pour celà il faut utiliser le gestionnaire de paquets `pip`. *Attention à bien activer le* ***`venv`*** *voulu pour éviter les conflits.*

Les principales commandes sont :

```
pip search <paquet> # pour chercher un paquet, ou vérifier s'il est à jour.
pip install <paquet> # installe un paquet
pip install <paquet> --upgrade # pour mettre à jour un paquet.
pip list  # pour lister les paquets installés
```

Nous pouvons maintenant installer les librairies nécessaires au module de M1.

```
pip install numpy matplotlib pandas scikit-learn
```

Nous présenterons plus en détail ces librairies ultérieurement. Lorsque les librairies sont installées il faut les importer pour pouvoir les utiliser :

```
import pandas # On importe la librairie pandas
import matplotlib.pyplot as plt # On importe le module pyplot de
                                # de la librairie matplotlib avec l'alias plt
```

2 Les bases du langage Python

2.1 Commentaires et docstring

— Les commentaires en python sont signalés par un #.
— Les docstrings sont des chaînes de caractères permettant de décrire une fonction ou un module en python.

```
# Définition de la fonction somme
def somme(a, b):
    """
        Retourne la somme des deux nombres passés en paramètre
    """
    return a + b

# Usage
help(somme)
Help on function somme in module __main__:

somme(a, b)
Retourne la somme des deux nombres passés en paramètre
(END)
```

2.2 Les blocs

La mise en page des blocs se fait par des indentations. Ca demande donc une certaine rigueur d'implémentation. *Attention à ne pas mélanger les espaces et les tabulations.*
Par exemple :

```
# parcourt d'une liste
for i in [1, 2, 3, 4, 5]:
    print(i) # affichage de la variable i
for j in [1, 2, 3, 4, 5]:
    print(j)
    print(i+j)
print("Fin de bloc")
```

On peut noter :
— Les commentaires sont définis à l'aide de #.
— La syntaxe des boucles for.
— La définition d'une liste.
— L'indentation fait office de séparateur de bloc.
Une liste de liste peut s'écrire de la façon suivante :

```
liste_de_listes = [ [1, 2, 3],
                    [4, 5, 6],
                    [7, 8, 9] ]
```

3 Les structures des données

3.1 Les chaines de caractères

```
chaine1 = 'une chaine'
chaine2 = "une autre chaine"
taille = len(chaine1)
print("La chaine 1 est {0} et a une taille de {1} la chaine 2 est {2}"
        .format(chaine1,taille,chaine2))
chaine3 = chaine1 + chaine2
print("La chaine 3 est {}".format(chaine3))
bjr = 'bonjour'
print(bjr[1]) # affiche o
print(bjr[0:2]) # affiche bo
print(bjr[:2]) # affiche bo
print(bjr[2:]) # affiche njour
```

3.2 Les listes

```
li = [] # Déclare une liste vide
li.append(1) # li est maintenant [1]
li.append("bonjour") # li est maintenant [1,'bonjour'] (liste hétérogène)
autre_li = [1,2,3,4,5]
liste_de_liste = [ li, autre_li ] # [[1, 'bonjour'], [1, 2, 3, 4, 5]]
autre_li[2:4] # [3,4]
del autre_li[3] # suppression du 3ieme élément
autre_li[2:4] # [3, 5]
autre_li[-1] # 5 (dernier élément)
autre_li[-2] # 3 (avant dernier élément)
len(li) # taille d'une liste
li.extend(autre_li)
```

3.3 Les tuples (ou n-uplet)

Les tuples sont des listes immutables. On y retrouve les mêmes opérations essentielles. Ils sont surtout très pratiques pour récupérer des valeurs de retour de fonctions.

```
def square_and_cube(x):
    return(x*x,x*x*x)

a,b = square_and_cube(3) # a=9, b=27
_,c = square_and_cube(2) # c=8
```

3.4 Les dictionnaires

Les dictionnaires permettent d'associer des clés à des valeurs.

```
d = dict() # Déclare un dictionnaire vide
nombres = { "un" : 0, "deux" : 2, "dix" : 10 }
nombres["un"] = 1 # remplace la valeur
nombres["cinq"] = 5 # ajoute l'entrée
nombres.get("deux") # 2
nombres.keys() # liste de toutes les clés
nombres.values() # liste de toutes les valeurs
nombres.items() # liste de tuples des pairs clés/valeurs

for k,v in nombres.items():
    print("La valeur de la clé {} est {}".format(k,v))
```

3.5 Les ensembles

Les ensembles sont comme des listes sans doublon.

```
ens = set([1,2,2,3,4,5]) # ens : 1,2,3,4,5
ens2 = set([1,3,5,7])
ens & ens2 # intersection (1,3,5)
ens | ens2 # union (1,2,3,4,5,7)
ens - ens2 # différence (2,4)
ens ^ ens2 # différence symétrique (2,4,7)
```

4 Les structures de contrôle

4.1 Les branchements conditionnels

```
temp = 10
if temp > 25:
    print("Il fait chaud")
elif temp < 10:
    # optionnel
    print("Je mets un pull")
else:
    # optionnel
    print("J'hésite")
```

4.2 Les boucles tant que

```
x = 0
while x < 10:
    print(x) # 0,1,2,3,4,5,6,7,8,9
    x+=1
```

4.3 Les boucles for

```
for x in range(10):
    print(x) # 0,1,2,3,4,5,6,7,8,9
for i in range(2,4):
    print(i) # 2,3

for couleur in ["rouge", "vert", "bleu", "jaune"]:
    print("{} est une couleur".format(couleur))
```

4.4 Les fonctions

Il faut utiliser le mot clé `def` pour créer de nouvelles fonctions.

```
def somme(a,b):
    print("{} + {} = {}".format(a,b,a+b))
    return a+b

somme(3,4)
somme(a=3,b=4)
somme(b=4,a=3)

# Portée des variables
x = 42

def set_x(a):
    x = a  # ici x est une nouvelle variable locale
    x+=1
print(x)

def set_global_x(a):
    global x
    x = a  # ici x est une variable globale (à éviter)
    x+=1
print(x)
```

5 Fonctionnalités avancées

Nous allons voir quelques fonctionnalités avancées aidant à la manipulation de données en python.

5.1 Trier

```
a = [2,1,5,7,4]
b = sorted(a) # b = [1,2,4,5,7]
a.sort()      # a = [1,2,4,5,7]

c = sorted(a, reverse=True) # c = [7,5,4,2,1]
a.sort(reverse=True) # a = [7,5,4,2,1]
```

5.2 Les listes en compréhension

Il s'agit de listes dont le contenu est le résultat d'une transformation (un filtrage) sur une autre liste.

```
liste =[-1,-2,-3,-4,-5]
liste_pos = [abs(x) for x in liste] # [1, 2, 3, 4, 5]
liste_carre = [x*x for x in liste_pos] # [1, 4, 9, 16, 25]
cases = [(x,y) for x in range(8) for y in range(8)]
#[(0, 0), (0, 1), (0, 2), (0, 3),....(7, 5), (7, 6), (7, 7)]
```

5.3 Les valeurs aléatoires

```
import random
random.seed(42)
reel_uniforme_0_1 = random.random()
entier_uniforme_0_4 = random.randrange(5)
entier_uniforme_2_4 = random.randrange(2,5)
liste = list(range(10)) # [0,1,2,3,4,5,6,7,8,9]
random.shuffle(liste) # [3, 6, 2, 0, 4, 1, 8, 9, 7, 5]
random.choice(liste) # returne aléatoirement une valeur de la liste
donnees = list(range(100)) # [0,..,100]
entrainement = random.sample(donnees, 50)
# 50 valeurs prises aléatoirement dans la liste
```