[TP1]

Analyser et manipuler des données avec Python

Sommaire

1. Variables et types

1.1. Variables

1. Déclarer et initialiser deux variables x et y, puis calculer et afficher :
    a. La somme
    b. La multiplication
    c. La division
2. Afficher les types des résultats de vos calculs
    a. Utiliser **print** pour plusieurs outputs
3. Déclarer une variable de type chaine de caractère, puis afficher la valeur et le type de cette variable.
4. Expliquer et commenter ce code :

```
[ ]  print(s[0])
     print(s[0:2])
     print(s[0:4])
```

```
[ ]  print (s[-1])
     print (s[-4:-2])
     print (s[3:])
```

1.2. Opérations avec des chaînes

1. Déclarer deux variables de type chaine de caractères ; firstname et lastname, puis concaténer les deux variables avec une séparation par espace, et afficher le résultat.
2. Expliquer et commenter

```
[2]  fullname= 'SLIMANI Khadija'
     fullname.split('M')  # Split the string on the space character
```

```
[ ]  'khadija'+ 2
```

Assurez-vous de convertir les objets en chaînes avant de concaténer.

- Python a une méthode intégrée pour faciliter formatage de chaîne.

```
[ ] price = 3.24
    num_items = 5
    year = 2018
    print('the price for the year ',year,' is',price)
    print("the price is %i" %num_items)
    print("the price is %f" %price)
    print("the price for the year %i is %f" %(year,price))
    print('Jean bought {} item(s) at a price of {} each for a total of {}'.format(num_items, price, num_items*price))
```

1.3. Les fonctions

1. Ecrire une fonction **add_numbers** qui prend deux nombres et les additionne.
2. Est-ce qu'une fonction peut être affectée à une variable ?

1.4. Expressions conditionnelles

- Tester ce code et commenter

```
x1 = 2
x2 = 7
if x1 > 5:
    print ('x1 is greater than 5')
else:
    print ('x1 is smaller than 5')
```

```
if x1> 5 and x2 > 5:
    print ('x1 and x2 are greater than 5')
elif x1 > 5 or x2 >5:
    print ('x1 or x2 is greater than 5')
else:
    print ('x1 and x2 are smaller than 5')
```

2. Types avancés

2.1. Listes

1. Déclarer et initialiser une liste
2. Afficher le contenu de la liste et son type
3. Les listes sont indexées de la même manière que les chaînes, tester et commenter chaque ligne de ce code :

```
[ ] print (x[0])
    print (x[1:3])
    print (x[:2])
    print (x[-2:])
    print (x[-1])
    print (x[::-1])
```

4. Les listes sont une structure de données mutable ==> modifiée après la création, expliquer ce code :

```
[ ]  x.append(3.3)  # Use append to append an object to a list
     print (x)
     x[0] = -1    # Change the value of the first element from 1 to -1
     print (x)
```

5. Ecrire le code qui permet de parcourir et afficher chaque élément de la liste.
6. Expliquer et ajouter un commentaire pour ce code.

```
[ ]  print ([1,2] + [3,4])   # Use + to concatenate lists
     # Use * to repeat lists
     print ([1]*3)
     # Use the in operator to check if something is inside a list
     print (1 in [1, 2, 3])
     # Use the in operator to check if something is inside a list
     print ('Mimo' in ['Kimo', 'Mimo', 'dodo'])
```

2.2. Tuples

1. Déclarer un tuple, puis afficher son contenu et son type.
2. Les tuples sont indexés de la même manière que les chaînes et les listes, ils sont une structure de données immuable ==> ne peuvent pas être modifiés. Commenter les lignes de code suivantes :

```
[ ]  print (t[0])
     print (t[1:3])
     print (t[-2:])
```

```
[ ]  t[0] = -1
```

2.3. Dictionnaires

Les dictionnaires associent les clés aux valeurs.

```
[ ]  d = {'khadija SLIMANI': 'slimani.khadija@qassil.com', 'Khadija SLIMANI': 'pr.kslimani@gmail.com'}
     print(d)
     print (d, type(d))
```

Les dictionnaires ne sont pas indexés de la même manière que les chaînes, les listes ou les tuples, nous utilisons les clés à la place.

```
[ ]  d['Khadija SLIMANI'] # Retrieve a value by using the indexing operator
```

1. Récupérer les clés du dictionnaire
2. Récupérer les valeurs du dictionnaire
3. Itérer sur tous les éléments (afficher « clés : valeur » de tous les éléments)

3. Dates et heures

Détailler et expliquer chaque ligne de ce code :

```
[ ]  import datetime as dt
     import time as tm
```

```
[ ]  print(tm.time())
     print(dt.datetime.now())
```

```
[ ]  dtnow = dt.datetime.fromtimestamp(tm.time())
     dtnow
```

```
[ ]  # get year, month, day, etc.from a datetime
     dtnow.year, dtnow.month, dtnow.day, dtnow.hour, dtnow.minute, dtnow.second
```

```
[ ]  print(dtnow.month)

     dtnow.strftime('%b')
```

```
[ ]  delta = dt.timedelta(days = 100) # create a timedelta of 100 days
     print(type(delta))
     print(delta)
```

```
[ ]  today = dt.date.today()
     print(today)
     print(type(today))
```

```
[ ]  today - delta # the date 100 days ago
```

```
[ ]  today > today-delta # compare dates
```

4. Compréhensions Lambda et List

Commenter et expliquer le résultat de ce code :

```
[ ]  my_function = lambda a, b, c : a + b
```

```
[ ]  my_function(1, 2, 3)
```

```
[ ]  #my_list= list()
     my_list = []
     for number in range(0, 20):
         if number % 3 == 0:
             my_list.append(number)
     print('my_list contins the following items:',my_list)
```

```
[ ]  my_list = [number for number in range(0,20) if number % 3 == 0]
     my_list
```

5. Python numérique (NumPy)

```
[ ]  import numpy as np
```

5.1. Création de tableaux

1. Créer une liste et la convertir en un tableau numpy.
2. Passer une liste directement avec numpy.
3. Passer une liste de listes pour créer un tableau multidimensionnel.
4. Trouver les dimensions du tableau. (rows, columns).
5. Expliquer le rôle et le résultat obtenu après l'utilisation de ces fonctions **la classe numpy**
    a. **arrange()**
    b. **reshape()**
    c. **linspace()**
    d. **ones()**
    e. **zeros()**
    f. **eye()**
6. Créer un tableau à l'aide d'une liste répétitive.
7. Répète les éléments d'un tableau en utilisant la fonction .**repeat()**.

5.2. Opérations

1. Utiliser +, -, *, / et ** pour effectuer des additions, des soustractions, des multiplications, des divisions et des puissances élémentaires.
2. Utiliser .**dtype()** pour voir le type de données des éléments du tableau.
3. Utiliser .**astype()** pour convertir en un type spécifique.
4. Expliquer le résultat de **print(x**2)** ; (par exemple : x= [1 2 3])
5. Calculer the dot product ; (par exemple : x= [1 2 3] et y= [3 2 4])

NB : Dot Product:

$$\begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = x_1 y_1 + x_2 y_2 + x_3 y_3$$

5.3. Fonctions mathématiques

Numpy a de nombreuses fonctions mathématiques intégrées qui peuvent être exécutées sur des tableaux.

1. Créer un tableau en utilisant numpy, puis calculer la somme, la moyenne, le maximum, le minimum et l'écart-type.
2. Afficher l'index des valeurs maximales et minimales du tableau.

5.4. Indexation / Tranchage

```
[ ]  s = np.arange(13)**2
     s
```

Utilisez la notation entre parenthèses pour obtenir la valeur à un index spécifique. N'oubliez pas que l'indexation commence à 0.

```
[ ]  s[0], s[4], s[-1]
```

Un deuxième **`:`** peut être utilisé pour indiquer la taille du pas. **`tableau[start:stop:stepsize]`**

1. **Expliquer le résultat de: s[-5 ::-2]**
2. **Considérons le tableau multidimensionnel r,**

```
[ ]  r = np.arange(36)
     r.resize((6, 6))
     r
```

Utilisez la notation entre parenthèses pour trancher **: `tableau[ligne, colonne]`**

```
[ ]  r[2, 2]
```

Et utilisez : pour sélectionner une plage de lignes ou de colonnes

```
[ ]  r[3, 3:6]
```

Ici, nous sélectionnons toutes les lignes jusqu'à (et n'incluant pas) la ligne 2, et toutes les colonnes jusqu'à (et n'incluant pas) la dernière colonne.

```
[ ]  r[:2, :-1]
```

3. **Nous pouvons également effectuer une indexation conditionnelle. Expliquer le code suivant :**

```
[ ]  r[r > 30]
```

```
[ ]  r[r > 30] = 30
     r
```

5.5. Copie de données

r2 est une tranche de r

```
[ ]  r2 = r[:3,:3]
     r2
```

1. Définissez les valeurs de cette tranche sur zéro ([:] sélectionne l'ensemble du tableau)
2. Afficher r, vos remarques ?
3. Ecrire un code qui permet de créer une copie qui n'affectera pas le tableau d'origine.
4. Afficher encore une fois le tableau d'origine r, vos remarques ?

5.5.1. Itérer sur des tableaux

Créons un nouveau tableau 4 par 3 de nombres aléatoires 0-9.

```
[ ]  # Generate a random integer in range 0-9 into a 4*3 array
     test = np.random.randint(0, 10, (4,3))
     test
```

```
[ ]   # Generate a random integer in range 0-9 into a 4*3 array
      test = np.random.randint(0, 10, 2)
     print(test)
```

1. Itérer le tableau par ligne.
2. Itérer le tableau par index.
3. Itérer le tableau par ligne et index.

Good luck...!!