OBJECTIFS : L'objectif de ce TP est de réaliser des codes pythons utilisant la structure de listes

Pour l'exercice 1, un tableau est à compléter par écrit sur la feuille distribuée. Pour les autres exercices, vous rédigerez un compte-rendu numérique en utilisant un logiciel de traitement de texte (*Word ou Libre Office*). Le fichier constitué sera appelé *tp9B.doc* ou *tp9B.odt* et devra être transféré en fin d'activité **par l'intermédiaire** du site *nsibranly.fr* : se loguer et transférer en utilisant le code **tp9** . Ce compte-rendu contiendra :

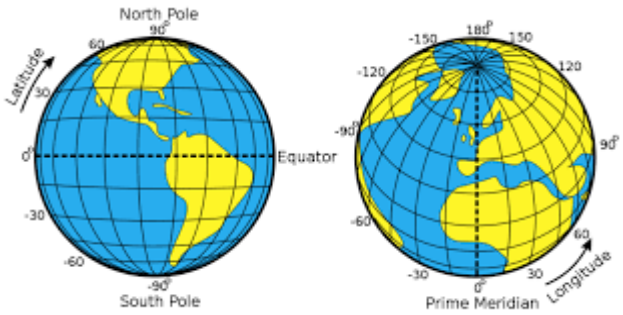
- les réponses aux différentes questions posées,
- les captures d'écran **des morceaux de codes** écrits **et** celles **des résultats des exécutions** données dans le shell. Pour faire ces captures, utiliser *l'Outil Capture d'écran* de Windows.

1. Manipulations de base sur les listes :

On reprend ici les commandes de base qui ont été vues en cours en les appliquant sur la liste suivante :

```
l = ["69005" , "LYON 05" , 45.8 , 4.8 ]
```

qui donne respectivement le code postal d'une commune, le nom de cette commune, sa latitude et sa longitude. Les 2 premiers éléments de cette liste sont des chaines de caractères (type *string* en python), les 2 derniers sont des réels (type *float* en python).



1.1. Accéder au contenu des éléments de la liste :

Ecrire uniquement la ligne `l = ["69005" , "LYON 05" , 45.8 , 4.8 ]` dans un fichier enregistrer sous le nom exercice_1.py. Exécuter ce fichier.

⇨ Dans le *shell*, exécuter les instructions suivantes et en donner le résultat (à écrire sur tableau papier) :

Instruction	`>>> l[0]`	`>>> l[2]`	`>>> l[-1]`	`>>> l[:2]`
Résultat				

Instruction	`>>> l`	`>>> l[2:]`
Résultat		

Instruction	`>>> l[1:3]`	`>>> l[-2]`	`>>> len(l)`
Résultat			

1.2. Ajouter un élément en fin de liste et en supprimer un :

⇨ Dans le *shell*, exécuter les instructions suivantes et en donner le résultat :

Instructions	`>>> l.append(2021)` `>>> l`	`>>> del l[-1]` `>>> l`
Résultat		

Instructions	`>>> del l[2:]`
	`>>> l`
		`>>> l.append(45.8)`
		`>>> l.append(4.8)`
		`>>> l`

Résultat		

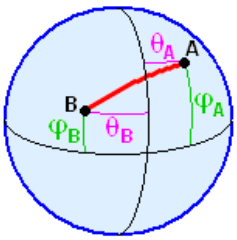
1.3. Ajouter la distance entre la ville concernée et Paris en fin de liste :

Les coordonnées GPS du 5ième arrondissement de Lyon sont $\varphi_A = 45.8°$ de latitude Nord et $\theta_A = 4.8°$ de longitude Est. A partir de ces coordonnées, une formule mathématique peut facilement donner la distance d à vol d'oiseau sur le globe terrestre, entre ce point et un autre point de coordonnées GPS : φ_B et θ_B . On a :

$$a = \sin^2\left(\frac{\varphi_B - \varphi_A}{2}\right) + \cos(\varphi_A) \cdot \cos(\varphi_B) \cdot \sin^2\left(\frac{\lambda_B - \lambda_A}{2}\right)$$

$$c = 2 \arctan\left(\frac{\sqrt{a}}{\sqrt{1-a}}\right)$$

$$d = 6\,371 \times c$$



⇨ Compléter le fichier *exercice_1.py* , en important la valeur du nombre *pi* et les fonctions *sqrt*, *sin*, *cos* et *atan* du module ***math*** et en écrivant le code ci-dessous d'une fonction *distance()* qui retourne la distance en km entre le point dont les coordonnées Gps sont en arguments et la ville de Paris.

```
from math import pi,sqrt,cos,sin,atan
# Fonctions
def distance(phiA,tetA) :
    """
        Retourne la distance en km entre le point de latitude phiA -
        longitude tetA et la ville de Paris.
    """
    phiB = 48.8626304852 * pi / 180  # latitude de Paris
    tetB = 2.33629344655 * pi / 180  # longitude de Paris
    phiA = phiA * pi / 180
    tetA = tetA * pi / 180
    a = (sin((phiA-phiB)/2))**2 + cos(phiA)*cos(phiB)*(sin((tetA-tetB)/2))**2
    c = 2 * atan(sqrt(a) / sqrt(1-a))
    d = 6371 * c
    return d

# Programme principal
l = ["69005" , "LYON 05" , 45.8 , 4.8 ]
```

⇨ Exécuter ce fichier et exécutant dans le shell l'instruction `>>> distance(45.8,4.8)` qui donne la distance entre Paris et Lyon. Quelle est cette distance ?

Latitude et longitude de New York, Manhattan

Latitude de New York, Manhattan	40.779897
Longitude de New York, Manhattan	-73.968565

⇨ En utilisant les coordonnées GPS ci-contre, déterminer la distance entre Paris et New York.

⇨ Compléter la partie *Programme principal* du code du fichier *exercice_1.py*, afin d'ajouter la distance entre Paris et Lyon comme 5ième élément de la liste ℓ :

```
# Programme principal
l = ["69005" , "LYON 05" , 45.8 , 4.8 ]
d = distance( l[2] ,
l.append(
print(l)
```

Donne à l'exécution :

```
>>> (executing file "exercice_1.py")
['69005', 'LYON 05', 45.8, 4.8, 387.8246527635398]
```

1.4. Créer une fonction qui affiche proprement le contenu de la liste ℓ :

⇨ Compléter le fichier *exercice_1.py* en y définissant la fonction *affichage()* ci-dessous, qui prend en argument la liste ℓ pour en afficher le contenu proprement (pas de *return* ici) :

```
def affichage(liste) :
    print("Ville : ",liste[1])
    print(
    print(
    print(
    print("Située à ",round(liste[4],1)," km de Paris")

# Programme principal
l = ["69005" , "LYON 05" , 45.8 , 4.8 ]
d = distance( l[2] , l[3] )
l.append(d)
affichage(l)
```

Ce code donne à l'exécution :

```
>>> (executing file "exercice_1.py")
Ville :  LYON 05
Code postal :  69005
Latitude :  45.8 °
Longitude :  4.8 °
Située à  387.8  km de Paris
```

⇨ Donner le résultat de l'exécution dans le shell de l'instruction :

```
>>> affichage(["10001","New York" , 40.78 , -73.97 , distance(40.78 , -73.97)])
```

⇨ Enregistrer le fichier *exercice_1.py*

2. Manipulations d'une liste composée de 2 listes :

⇨ Enregistrer le fichier *exercice_1.py* sous le nom *exercice_2.py* . Sur ce fichier, garder les fonctions créées précédemment et modifier uniquement le programme principal pour créer une nouvelle liste nommée *villes* définie par :

```
# Programme principal
a = ["69005" , "LYON 05" , 45.8 , 4.8]
b = ["10001" , "New York" , 40.78 , -73.97]
villes = [a , b]
```

Cette liste est elle-même composée de 2 listes.

2.1. Instructions de base pour manipuler la liste *villes* :

⇨ Dans le *shell*, exécuter les instructions suivantes et en donner le résultat (à écrire sur tableau papier) :

Instruction	`>>> villes[0][1]`	`>>> villes[1]`
Résultat		

Instruction	`>>> villes[1][1][0]`	`>>> len(villes)`	`>>> len(villes[0])`
Résultat			

2.2. Ajout d'un contenu dans les différentes sous-listes :

⇨ Compléter le fichier *exercice_2.py* en y définissant la fonction *affichage_distance()* ci-dessous, qui prend en argument la liste *villes*. Cette fonction utilise la fonction *distance()* codée dans le paragraphe précédent et retourne la liste villes dans laquelle chacune des sous-listes aura un 5ième contenu égal à la distance entre la ville et celle de Paris.

```
def ajout_distance(l) :
    
    return l
```

```
# Programme principal
a = ["69005" , "LYON 05" , 45.8 , 4.8]
b = ["10001" , "New York" , 40.78 , -73.97]
villes = [a , b]

villes = ajout_distance(villes)
print(villes)
```

Donnera à l'exécution :

```
>>> (executing file "exercice_2.py")
[['69005', 'LYON 05', 45.8, 4.8, 387.8246527635398], ['10001', 'New York', 40.78, -73.97, 5829.0457534841635]]
```

2.3. Affichage des données d'une la ville correspondant à un code postal :

⇨ Compléter encore le fichier *exercice_2.py* en y définissant la fonction *affichage_ville()* donnée ci-après, qui prend en argument la liste *villes* et une chaine de caractère correspondant à un code postal *cp* . Cette fonction utilise la fonction *affichage()* codée dans le paragraphe précédent pour écrire dans le shell les caractéristiques de la ville dont le code postal correspond à *cp* :

```
def affichage_ville(l , cp) :
```



```
# Programme principal
a = ["69005" , "LYON 05" , 45.8 , 4.8]
b = ["10001" , "New York" , 40.78 , -73.97]
villes = [a , b]

villes = ajout_distance(villes)
affichage_ville(villes , "10001")
```

Donnera :

```
>>> (executing file "exercice_2.py")
Ville :  New York
Code postal :  10001
Latitude :  40.78 °
Longitude :  -73.97 °
Située à  5829.0  km de Paris
```

Avec les arguments : `affichage_ville(villes , "75000")` on aura à l'exécution :

```
>>> (executing file "exercice_2.py")
Aucune ville trouvée
```

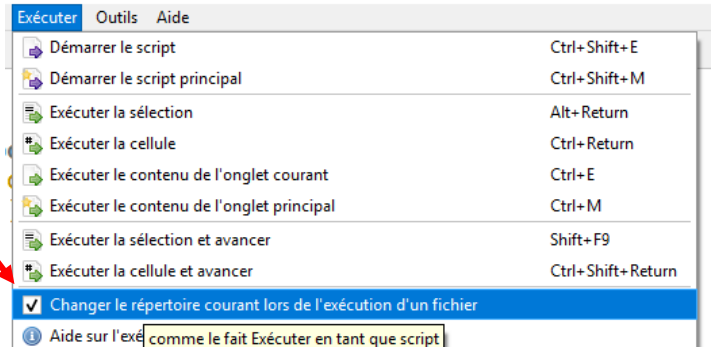
⇨ Enregistrer le fichier *exercice_2.py*

3. Manipulations d'une liste de liste de grande taille :

⇨ Enregistrer le fichier *exercice_2.py* sous le nom *exercice_3.py*

⇨ Télécharger le dossier *tp9B.zip*. Après l'avoir dézippé, copier les fichiers *bibli_tp9.py* et *communes.csv* contenus de ce dossier, dans **dans votre répertoire** de travail.

⇨ Dans Pyzo, s'assurer que la case « *Changer le répertoire courant …* » est cochée :



⇨ Sur le fichier *exercice_3.py* , garder les fonctions créées précédemment et modifier uniquement le programme principal pour créer une nouvelle liste nommée *villes* définie par :

```
# Programme principal

from bibli_tp9 import lecture_fichier
villes =  lecture_fichier("communes.csv")
```

3.1. Prise en main de cette nouvelle liste *villes* :

⇨ Dans le *shell*, exécuter les instructions suivantes et en donner le résultat (à écrire sur tableau papier) :

Instruction	`>>> villes`	`>>> len(villes)`	`>>> villes[30000]`
Résultat	**Trop de contenu**		

3.2. Utilisation des fonctions précédentes sur cette nouvelle liste *villes* :

⇨ On ajoute dans le programme principal : `villes = ajout_distance(villes)`

Que donne alors l'exécution dans le shell de `>>> villes[30000]` ?

⇨ On ajoute dans le programme principal : `affichage_ville(villes , "42000")`

Que donne alors l'exécution du programme ? :

3.3. Recherche de la ville française la plus éloignée de Paris :

⇨ Compléter le fichier *exercice_3.py* en y définissant la fonction *recherche ()* donnée ci-après, qui prend en argument la liste *villes*. Cette fonction retourne l'indice de la liste villes qui correspond à la ville française la plus éloignée de Paris :

```
def recherche(l) :
    
    return indice
```

```
# Programme principal

from bibli_tp9 import lecture_fichier
villes =  lecture_fichier("communes.csv")
villes = ajout_distance(villes)
affichage_ville(villes , "42000")
i_max = recherche(villes)
print("\nLa ville la plus éloignée de Paris est :\n")
affichage(villes[i_max])
```

Permet un retour à la ligne

Donne à l'exécution :

```
>>> (executing file "exercice_3.py")
Ville :  ST ETIENNE
Code postal :  42000
Latitude :  45.4301235512 °
Longitude :  4.37913997076 °
Située à  411.7  km de Paris

La ville la plus éloignée de Paris est :

Ville :  ST PHILIPPE
Code postal :  97442
Latitude :  -21.3040005374 °
Longitude :  55.7454668204 °
Située à  9422.3  km de Paris
```

3.4. Recherche du nombre de villes d'un département :

⇨ Compléter le fichier *exercice_3.py* en y définissant la fonction *villes_departement()* donnée ci-après, qui prend en argument la liste *villes* et une chaine de caractère nommée *num* qui correspond au numéro d'un département. La fonction retourne le nombre de villes de la liste qui appartiennent à ce département.

```
def villes_departement(l , num) :
    
    
    
    return nb
```

```
# Programme principal

from bibli_tp9 import lecture_fichier
villes =  lecture_fichier("communes.csv")
villes = ajout_distance(villes)
affichage_ville(villes , "42000")
i_max = recherche(villes)
print("\nLa ville la plus éloignée de Paris est :\n")
affichage(villes[i_max])
n_69 = villes_departement(villes , "69")
print(f"\nil y a {n_69} communes dans le 69")
```

Nouvelle façon d'insérer des variables dans un champ texte

Donne à l'exécution :

```
il y a 341 villes dans le 69
```

3.5. Recherche du département qui comporte le maximum de communes :

⇨ Compléter le fichier *exercice_3.py* en y définissant la fonction *max_departement()* donnée ci-après, qui prend en argument la liste villes et retourne le n° de département qui comprend le plus de communes. On utilisera dans cette nouvelle fonction la fonction *villes_departement()* écrite précédemment.

```
def max_departement(l) :
    dep = []
    for liste in l :
        n_dep = liste[0][0]+liste[0][1]
        if n_dep not in dep : dep.append(n_dep)
```

```
# Programme principal

from bibli_tp9 import lecture_fichier
villes =  lecture_fichier("communes.csv")
villes = ajout_distance(villes)
affichage_ville(villes , "42000")
i_max = recherche(villes)
print("\nLa ville la plus éloignée de Paris est :\n")
affichage(villes[i_max])
n_69 = villes_departement(villes , "69")
print(f"\nil y a {n_69} communes dans le 69")
n_dep = max_departement(villes)
print(f"\nle département qui a le plus de communes est le {n_dep}")
```

Donne :

```
le département qui a le plus de communes est le 62
```