# Partie 1 – Bases de Python

**Sujet : Nombres aléatoires, compréhension de liste, f-string**

## 1.2.1 – 1.2.3

On a plusieurs moyens de créer une liste de 1 million d'éléments, pris aléatoirement entre 1 et 10. Une fois cette liste créée, on crée une seconde liste contenant uniquement les éléments pairs. On peut ensuite calculer le pourcentage d'éléments pairs. Le formatage de la chaine de caractère est fait avec une f-string.

```python
import random # -> random.fonction() pour utiliser une fonction
from random import randrange # -> randrange() pour utiliser la fonction
from random import randint   # -> randint() pour utiliser la fonction

# On tire 1 milion de fois un nombre aléatoire entre 1 et 10 qu'on met dans
une liste en utilisant la compréhension de liste.
d = [randrange(1, 11) for _ in range(1_000_000)]

# Idem avec la fonction randint() cette fois
# d = [randint(1,10) for _ in range(1_000_000)]

# La fonction random.choices() retourne directement une liste
# d = random.choices(range(1, 11), k=1_000_000)

# PARTIE 2
# Ne garder que les nombres pair. On utilise la compréhension de liste avec
une condition, en réutilisant notre liste créée précédemment.
d_even = [x for x in d if x % 2 == 0]

# Les f-strings: solution simple de formatage !
print(f'{len(d_even) / 10_000:.2f}%')
```

## 1.2.4 – Description du code

Toutes les solutions appliquent le carré en se servant de la compréhension de liste (applique la fonction "carré" sur tous les éléments d'une liste, et retourne une liste).

- **Solution 1:** On créé d'abord deux listes à l'aide du constructeur list() que l'on concatène, avant de prendre les carrés des éléments. La fonction range() fait ici le travail de générer les bons nombre paires ou impaires.
- **Solution 2:** On créé la liste en une fois directement avec un range de 1 à 20, mais pour chaque élément des conditions vérifient s'il doit être ajouté ou non.
- **Solution 3:** On utilise deux compréhension de liste pour créer la liste finale en deux partie séparée. Comme pour la solution 1 on se sert du fait qu'on peut concaténer des listes, mais cette fois-ci l'opération est faite à la toute fin.

### 1.2.5 – Liste de strings avec condition

Pour la liste de string, il y a de nouveau plusieurs manières de faire. On peut soit vérifier si z ou Z sont présents ou soit vérifier si z est présent dans la chaine de caractère mise en minuscule. La dernière solution présentée ci-dessous est simplement un appel explicit à la méthode utilisée lors que l'on fait … `if ("z" in x).` N'hésitez pas à utiliser par exemple ChatGPT en utilisant comme prompt : « ***In python, what are the differences between x.__contains__("z") and ("z" in x)*** » pour avoir une réponse détaillée.

```
newlist = [x for x in objets if ("z" in x) or ("Z" in x)]

newlist = [x for x in objets if 'z' in x.lower()]

newlist = [x for x in objets if (x.__contains__("z")) or (x.__contains__("Z"))]
```

## Partie 2 - NumPy

Les réponses aux questions se trouvent dans le tutoriel w3school.

1) Pourquoi utiliser NumPy ?

- Pour utiliser des tableaux, Python fournit le type "List". Cependant les calculs sur les listes sont lents.

2) Comment s'appelle (n.b. "de quel type est") l'objet "array" dans NumPy ?

- ndarray

3) Pourquoi utiliser NumPy est-il plus rapide qu'utiliser les listes ?

- Stockage continu en mémoire. Optimisé pour travailler avec les dernières architectures CPU.

4) A quelle question le code "# Question 4" ci-dessous répond-il ?

- La question 2.

5) Affichez la somme des deux derniers éléments du tableau en y accédant de deux manières différentes. Une fois depuis le début, et une fois depuis la fin (indexation).

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr[3] + arr[4])
print(arr[-1] + arr[-2])
print(sum(arr[-2:]))
```

# Partie 3 – Pandas

1) Pourquoi utiliser Pandas ?

- Pandas permet d'analyser des Big Data et d'en extraire des conclusions basées sur les statistiques. Grâce à Pandas, on peut nettoyer des données pour les rendre pertinentes.

2) Que peut faire pandas (d'après le tuto w3schools) ?

- Donner des informations sur les données telles que la moyenne, la valeur max() ou min() de colonnes. Enlever des colonnes inutiles ou non pertinentes. En résumé, nettoyer les données et en extraire des informations.

3) Que fait l'exemple "Question 3" ci-dessous ?

- Crée un DataFrame à partir d'un dictionnaire, puis l'affiche.

4) Compléter la cellule "Question 4" ci-dessous pour afficher les lignes demandées. Utiliser l'attribut loc comme décrit dans le tutoriel. Pour afficher les dernières colonnes, vous pouvez utiliser des indexes fixes qui ne fonctionnent qu'avec un dataframe de cette taille, ou essayer la fonction iloc() avec des indexes négatifs.

Que remarquez vous concernant l'utilisation de simples crochets ([...]) ou doubles crochets ([[x]]) pour extraire une ligne du dataframe ? En utilisant la fonction type(), donnez le type de données retournées avec les simples crochets ([...]) ou doubles crochets ([[x,y]]).

```
# Afficher la ligne qui concerne les Volvo.
print(myvar.loc[1])
print(type(myvar.loc[1]))

print("----------------------")
print(myvar.loc[[1]])
print(type(myvar.loc[[1]]))

print("----------------------")
# Afficher les deux dernières lignes du dataframe
print(myvar.loc[[1,2]])
print("----------------------")
print(myvar.iloc[-1])
```

- On remarque que le format affiché n'est pas le même selon l'utilisation de df.loc[1] ou df.loc[[1]]. Avec les simples crochets, on obtient une série, avec les doubles, un dataframe. Pour utiliser des indexes négatifs, il faut utiliser iloc()

5) Complétez le code comme demandé dans la cellule "Question 5 - exercice". Extrait du tutoriel Pandas de w3school.

```
# Question 5 - Solution

# 1) Enlevez les données manquantes (NaN = Not a Number) et créez un nouveau dataframe appelé df_clean  (n.b ne changez pas le dataframe original, n'utilisez pas *inplace=True*). N'hésitez pas à utiliser "print(df_clean)" pour voir les changements, mais déplacez-le ou commentez-le pour ne pas encombrer l'affichage des résultats la cellule !


df_clean = df.dropna().copy()
# df_clean = df_clean.reset_index(drop = True) # Optionnel

# 2) La ligne 26 contient une date au mauvais format. Pour cela nous allons convertir la colonne "Date".
#    Attention a bien faire cette opération sur df_clean et non sur df (modifié par rapport au tutoriel).
#    En cas de Warnings, vous pouvez l'ignorer.

#df_clean['Date'] = pd.to_datetime(df_clean['Date']) # peut provoquer une erreur
date = pd.to_datetime(df_clean.loc[:, "Date"].copy(),format='mixed')

#df_clean["Date"] = date -> Retourne un warning
df_clean = df_clean.assign(Date = date)

# 3) La valeur à la ligne 7 vous semble suspecte. Vous pouvez choisir de la remplacer par une valeur qui a plus de sens (45) ou vous pouvez simplement supprimer la ligne. Basez-vous toujours sur le tutoriel pour réaliser cette tâche.

df.loc[7, 'Duration'] = 45

# Autre solution:
#for x in df.index:
#    if df.loc[x, "Duration"] > 120:
#        df.drop(x, inplace = True)
#        df.loc[x, "Duration"] = 120 # Solution mauvaise, car on change drastiquement la donnée, mais c'est dans le tuto !!

# Enfin on affiche notre dataframe propre et ses infos. Décommentez les "print" pour afficher le dataframe df_clean et ses infos

print(df_clean)
print("\n\n==================== Infos ====================\n\n")
print(df_clean.info())
```

# Partie 4 – Matplotlib

Pour cette partie le code vous est entièrement donné. Cependant il est important que vous compreniez rapidement ce que représentent les différents graphiques.

Comment chaque enregistrement (Row – ligne) est représenté ? Que représente l'axe X et l'axe Y d'un histogramme ? Que représente chaque groupe d'un boxplot ? Comment un scatterplot représente-t-il une donnée par rapport à une autre ?