TP 00

Objectifs :
— initiation au python
— utilisation d'un Jupyter notebook
— utilisation de la documentation python : https://docs.python.org/3/library/index.html

Exercice 1 : Introduction à Python

Python est un langage interprété supportant aussi bien une programmation procédurale qu'objet ou fonctionnelle. Le langage dispose d'un interpréteur interactif que vous pouvez lancer avec la commande `python3`, mais nous utiliserons principalement des *Jupyter notebook*.

Un *notebook* est une page web contenant des cellules de calcul et des cellules textuelles (écrites en `Markdown`). Pour exécuter une cellule de calcul par le bouton ▷ ou au clavier par `Ctrl+Entrée`.

À vous d'utiliser cette feuille Jupyter comme compte-rendu de TP. N'oubliez d'y insérer en `Markdown` la décomposition exercice/question entre les cellules, afin d'y retrouver la structure du TP.

N'oubliez pas que la documentation python est votre amie ! Il est facile d'accéder, directement à partir d'une cellule d'un *notebook* à la documentation d'une fonction donnée. Par exemple, si on cherche des informations sur la fonction `arange(...)` de la bibliothèque `numpy`, il suffit de saisir `numpy.arange?` dans une cellule et de la valider pour obtenir l'aide et les détails sur cette fonction.

Q1. Intéréssons-nous aux manipulations de base ainsi qu'au typage. Utilisons d'abord une cellule comme une calculatrice.

Q1.1. Exécutez les instructions suivantes pour comprendre les opérations réalisées :

```
print(123+456)
print(123*456)
print(123**456)

print(123/45)
print(123//45)
print(123%45)

print(42/0)
```

Q1.2. Vous manipulerez beaucoup de librairie. Calculez `exp(1.0)`.

Q1.3. Vous obtenez une erreur ! Il faut importer une librairie pour cela. Testez les instructions suivantes :

```
# import le plus prudent
import math
print(math.exp(1.0))
# ou
# mais il existe un risque de confusion entre librairies
from math import exp
print(exp(0))
```

Q1.4. Python est fortement typé (ce qui impose des conversions explicites) et le typage est dynamique. Contrairement à Java tout est objet, y compris les types de base : il faut donc bien faire la différence entre "*méthode*" et "*fonction*". Saisissez les instructions suivantes dans une cellule de votre *notebook* pour vérifier ce que ça donne :

```
x=1
print(type(x))
x="toto"
print(type(x))
```

Q1.5. De même, que donne l'instruction `print(3*x)` ?

Q1.6. Que donne les instructions suivantes, et pourquoi ?

```
print(40 + "2")
print(40 + int("2"))
print(str(40) + "2")
```

Q1.7. Définissez de manière classique avec le mot-clé `def` ou via une *lambda-expression* le polynôme suivant et afficher sa valeur pour x=42 :

$$f : x \mapsto x^4 + 10\,x^3 + 100\,x + 1789$$

Q2. Intéressons-nous aux types *séquence* et *dictionnaire*. Une *séquence* est un ensemble indexé à partir de zéro. La longueur d'une séquence peut être obtenue avec la fonction `len(...)`. On peut accéder à un élément par son index ou obtenir un sous-ensemble des éléments.

Q2.1. Commençons avec le type *séquence* le plus simple, à savoir la chaîne de caractères. Écrivez dans une cellule les instructions nécessaires pour :

- définir une variable contenant une chaîne d'au moins 8 caractères
- obtenir la longueur de la chaîne
- afficher le premier caractère d'une chaîne
- afficher le dernier caractère d'une chaîne (via un indice négatif)
- afficher les caractères entre le second et cinquième (bornes incluses, via l'opérateur de *slicing* ':')
- tester l'appartenance du caractère 'z' à votre chaîne (via le prédicat `in`)
- remplacer le second caractère par un 'z'

Q2.2. Un autre type *séquence* **non mutable** est le `tuple`. Il est construit avec des éléments de types possiblement différent, entre parenthèses. À partir du tuple `t = (1, "abc", 1.5, 'a')`, testez les instructions pour :

- afficher la longueur du tuple (via la fonction `len(...)`)
- afficher les éléments, sauf le premier et le dernier (par *slicing*)
- vérifier les types des deux premiers éléments ainsi que du tuple lui-même (via la fonction `type(...)`)

Q2.3. Une *liste* est un type de séquence mutable qui se manipule comme les tuples. On peut en changer les éléments à un index donné et on peut utiliser la fonction `del(...)` pour supprimer des éléments. Une liste est construite avec des éléments entre crochets. À partir de la liste `l = [1, "abc", 1.5, 'a']`, testez les instructions pour :

- afficher la liste
- supprimer l'élément d'indice 1 (via la fonction `del(...)`)
- doubler la valeur de l'élément d'indice 1
- rajouter l'élément "`abc`" à la fin de la liste (via la méthode `append(...)`)
- inverser la liste (via la méthode `reverse()`)

Q2.4. Un *dictionnaire* est un ensemble mutable de valeurs indexées par des clés. Il est représenté entre accolades. À partir du dictionnaire `d = {"forename": "Alice", "age": 42, "height": 1.60}`, testez les instructions pour :

- afficher le dictionnaire ainsi que sa longueur (via la fonction `len(...)`)
- afficher uniquement la taille (de clé `height`)
- ajouter un poids : `weight=53.5`
- ajouter une liste de passe-temps : `hobbies = ["painting", "breaking", "hiking"]`
- supprimer le dernier passe-temps de la liste (via la fonction `del(...)`)
- tester si "*breaking*" est bien un passe-temps (via le prédicat `in`)

Notez que les différents types de données que l'on vient de voir peuvent également être créés avec une syntaxe objet. Par exemple, `l = list()` au lieu de `l = []` pour une liste ou `d = dict()` au lieu de `d = {}` pour les dictionnaires.

Q2.5. Ces différents types peuvent également être initialisés par une méthode algorithmique (appelée *compréhension*) à la création. Saisissez dans une nouvelle cellule les exemples ci-dessous pour comprendre cette utilisation.

```python
l = [ x for x in range(10)]
print(l)

pair = [ x for x in range(10) if x%2 == 0 ]
print(pair)
```

Sur le même modèle, donnez les instructions nécessaires pour :
— créer une liste contenant tous les multiples de 10 entre 0 et 100 (bornes incluses)
— créer un dictionnaire qui prend les chiffres de 0 à 9 en clés et associe à chacune leur carré.

Q2.6. Calculez la somme des carrés des entiers de 1 à 10 inclus. La méthode `sum(...)` va s'avérer utile.

Q2.7. Écrivez la fonction correspondant à la somme de Newton $sN : (k, n) \mapsto \sum_{i=1}^{n} i^k$, si possible avec une *lambda-expression*. Calculez les valeurs associées à (2,10) et (22,100).

Q3. Intéressons-nous aux entrées/sorties en python.

Les lectures clavier peuvent être réalisées avec la fonction `input(...)` qui lit des chaînes de caractères.

Les sorties standards utilisent la fonction `print(...)`. Il existe différentes méthodes pour spécifier le format des affichages. Nous allons essentiellement utiliser des *f-string* pour le formatage. Ce type de chaîne prend la forme `f"..."`. À l'intérieur de cette chaîne les expressions entre accolades sont interprétées et formatées.

Accessoirement en Python, vous pouvez utiliser indifféremment des guillemets `"..."` ou des apostrophes `'...'` pour délimiter une chaîne de caractères. Vous pouvez définir des chaînes de caractères multi-lignes avec `"""..."""` ou `'''...'''`.

Q3.1. Écrivez les instructions nécessaires à la saisie d'un nom et d'un âge.

Q3.2. Recherchez dans la documentation les formats d'affichage utilisables avec les *f-string*.

Q3.3. Ajoutez les instructions d'affichage de l'âge en binaire et de l'âge dans 30 ans.

Q3.4. Ajoutez une instruction permettant d'afficher π avec 6 décimales, en gérant le format d'affichage avec une *f-string*. π doit être importé du module `math` via l'instruction suivante : `from math import pi`.

Q4. Intéressons-nous enfin aux fichiers et à leur manipulation. Qui dit "fichier", dit aussi gestion des exceptions. En python, la syntaxe pour la gestion des exceptions est la suivante :

```python
try:
    <instructions>
except NomException:
    <instructions>
except NomException2 as ex:
    <instructions>
else:
    <instructions si pas d exception>
```

Comme dans de nombreux langages, vous pouvez avoir plusieurs clauses `except`. La première qui correspond est capturée. Vous pouvez avoir une clause `except` avec plusieurs exceptions sous forme de tuples. Le deuxième exemple montre comment récupérer l'exception dans une variable. L'instruction `else` est optionnelle. Vous pouvez levez une exception avec l'instruction `raise NomException`. Les exceptions sont des classes Python et vous pouvez en créer des nouvelles si nécessaires.

Les primitives classiques de manipulation des fichiers sont `open(...)` et `close()`. Toutefois l'utilisation de l'instruction `with` vous garanti une fermeture automatique et propre des fichiers même en cas d'exception.

```python
with open("<chemin vers la ressource>", encoding="utf-8") as f:
   for line in f:
        print(line, end="")
print()
```

Q4.1. Écrivez les instructions nécessaire à la lecture et l'affichage du contenu du fichier `data_mm00_fruits.csv`, en affichant uniquement une ligne sur deux.

Q4.2. Écrivez les instructions nécessaires à l'écriture dans un fichier `result_mm00_alphabet.csv` des lettres de l'alphabet en minuscule puis en majuscule à raison d'une lettre par ligne.

Exercice 2 : Chiffrement et déchiffrement

Le chiffrement César est une méthode simple de cryptographie où chaque lettre d'un message est décalée d'un certain nombre de positions dans l'alphabet. Par exemple, avec un décalage de 3, "A" devient "D", "B" devient "E", etc. Si on atteint la fin de l'alphabet, on revient au début (par exemple, "Z" devient "C"). La fonction doit maintenir la casse des lettres : les majuscules restent majuscules et les minuscules restent minuscules. Les caractères non alphabétiques (comme les espaces, chiffres, ou ponctuation) doivent rester inchangés. Le décalage peut être un entier positif ou négatif. Un décalage positif signifie un déplacement vers la droite dans l'alphabet, tandis qu'un décalage négatif signifie un déplacement vers la gauche.

Q1. Écrire une fonction `caesar_coder` qui chiffre un message donné en utilisant le chiffrement César. La fonction prendra en entrée une chaîne de caractères et un entier représentant le décalage, et retournera la chaîne de caractères chiffrée.

Q2. Écrire une fonction `caesar_decoder` qui déchiffre un message chiffré en utilisant le chiffrement César. La fonction prendra en entrée une chaîne de caractères chiffrée et un entier représentant le décalage utilisé pour le chiffrement, et retournera la chaîne de caractères déchiffrée.

Exercice 3 : Gestion d'un inventaire

Une épicerie souhaite maintenir un inventaire de ses produits frais. Chaque article a un nom, une catégorie (par exemple, "*fruits*" ou "*vegetables*"), et une quantité en stock. Vous devez écrire une fonction qui prend en entrée une liste d'actions à effectuer sur l'inventaire et qui retourne l'état final de l'inventaire sous la forme d'un dictionnaire.

L'inventaire initial est vide. Les actions à effectuer sont données sous la forme d'une liste de tuples, où chaque tuple contient :

- le type d'action : "*add*" ou "*remove*"
- le nom de l'article
- la catégorie de l'article,
- la quantité à ajouter ou retirer.

Si un article n'existe pas encore dans l'inventaire, il doit être ajouté lors d'une action "*add*". Si un article n'existe pas dans l'inventaire et que l'action est "*remove*", il ne se passe rien. La quantité d'un article ne peut jamais être négative. Si une action de retrait demande une quantité supérieure à ce qui est en stock, la quantité restante doit être fixée à 0. La fonction doit retourner un dictionnaire où les clés sont les catégories, et les valeurs sont des dictionnaires contenant les articles comme clés et leurs quantités comme valeurs.

Q1. Écrivez la fonction `manage_inventory` qui prend une action en paramètre et retourne l'inventaire tel que décrit précédemment. Voici un exemple d'utilisation :

```
actions = [
  ("add", "apple", "fruits", 10),
  ("add", "orange", "fruits", 5),
  ("remove", "apple", "fruits", 2),
  ("add", "potatoes", "vegetables", 3),
  ("remove", "orange", "fruits", 7),
  ("remove", "cucumber", "vegetables", 5)
]

inventory = manage_inventory(actions)
print(inventory)
```

L'affichage attendu est :

```
{
  'fruits': {'apple': 8, 'orange': 0},
  'vegetables': {'potatoes': 3}
}
```

Q2. Après avoir géré les actions sur l'inventaire, vous devez écrire une fonction qui calcule le bilan de l'inventaire par catégorie. Cette fonction doit retourner un dictionnaire où chaque clé est une catégorie et chaque valeur est la somme des quantités d'articles dans cette catégorie. Écrivez la fonction `inventory_report` qui réalise cette opération et affiche une sortie de la forme :

```
{'fruits': 8, 'vegetables': 3}
```

Exercice 4 : Analyse du temps de travail

Une entreprise souhaite analyser les heures de travail de ses employés. Les employés pointent lors de leur arrivée et de leur départ. Ces horodatages sont enregistrés dans un fichier CSV. Il convient de lire ce fichier et de calculer par personne le temps de travail de chaque employé. Les résultats doivent être fourni dans un autre fichier CSV.

Les données sont décrites sous le format suivant :

```
employee,date,arrival_time,departure_time
Alice,2024-08-01,09:00,17:30
Bob,2024-08-01,08:45,16:45
Alice,2024-08-02,09:15,17:00
Bob,2024-08-02,09:00,17:15
```

Les résultats doivent respecter le format suivant :

```
employee,total_work_hours
Alice,16.25
Bob,16.25
```

Q1. À partir du fichier `data_mm00_work-times.csv`, calculez le temps de travail de chaque employé et écrivez le résultat dans un autre fichier CSV nommé `result_mm00_work-summary.csv`.