Techniques web : Contrôle 1

Nom Louis
Prénom JOURDAIN
N étudiant 49.3

I) Questions de cours (/12, chaque question /2)

1) Dumping sauvage

```
import json
data = {"nom": "Alice", "age": 30,
"langages": ["Python", "C++"]}
json_texte = json.dumps(data)
with open("data.json", "w") as f:
    json.dump(data, f)
with open("data.json", "r") as f:
    contenu = f.read()
```

🔹 **Documentation**

`json.dump(obj, fp, *, skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, cls=None, indent=None, separators=None, default=None, sort_keys=False, **kw)`

Serialize un objet Python `obj` vers un **fichier texte** ou un flux `fp` (par ex. un fichier ouvert) au format JSON.

`json.dumps(obj, *, skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, cls=None, indent=None, separators=None, default=None, sort_keys=False, **kw)`

Serialize un objet Python `obj` en **chaîne de caractères JSON** (retourne une `str`).

À quoi servent les fonctions `dump()` et `dumps()` du module `json` ?
Quelle est la différence entre les deux ?
Dans quel cas utiliser l'une ou l'autre ?
Expliquez à partir du code donné en exemple ci-dessus et de la documentation.

Ces fonctions servent toutes deux à exporter des données Python au format JSON, c'est-à-dire à sérialiser (transformer un objet en une suite de caractères ou de bit) un objet Python (dans l'exemple, un dictionnaire) en une représentation conforme au standard JSON. Cependant, elles ne le font pas de la même manière.
La fonction json.dump() sert à écrire directement le JSON dans un fichier. Elle prend en paramètre l'objet Python à sérialiser ainsi qu'un objet fichier déjà ouvert ; elle se charge ensuite d'écrire dans ce fichier le contenu JSON correspondant. Selon le mode d'ouverture du fichier ("w", "a", etc.), elle créera, écrasera ou ajoutera du contenu. Dans le code fourni, le fichier data.json est ouvert en écriture, et json.dump(data, f) écrit immédiatement le JSON dans ce fichier.
À l'inverse, la fonction json.dumps() ne modifie aucun fichier : elle renvoie simplement une chaîne de caractères contenant la représentation JSON de l'objet Python. On obtient donc une valeur de type str, comme la variable json_texte dans l'exemple. Cette chaîne peut ensuite être affichée, stockée dans une variable, envoyée sur un réseau, ou écrite manuellement dans un fichier selon les besoins.

2) Un tableau pour les définir tous

Complétez le tableau suivant en précisant pour chaque langage :

S'il agit côté client ou côté serveur, son rôle principal dans le développement Web,un ou deux exemples concrets d'usage

Langage	Côté (client / serveur)	Rôle principal	Exemples d'usage	Ordre d'apparition
JavaScript	Client (et serveur avec Node.js)	Rendre les pages web interactives, exécuter du code dynamique	Animations, formulaires interactifs, appels API (AJAX)	1995
HTML	Client	Structurer le contenu des pages web	Titres, paragraphes, liens, formulaires	1991
Node.js	Serveur	Exécuter du JavaScript côté serveur	Applications web, serveurs HTTP, API REST	2009
CSS	Client	Définir la présentation et le style des pages web	Couleurs, polices, mise en page, animations	1996

3)Vous êtes le panda :

🐼 **Exercice : You are the pandas**

On dispose du tableau suivant sur des pandas dans un zoo :

Nom	Age	Sexe	Poids_kg	Zone
Pandy	5	M	80	Nord
Lulu	2	F	40	Sud
Bao	7	M	90	Est
Mei	4	F	50	Ouest
Tao	3	M	60	Sud

Ecrivez le résultat des lignes de code suivantes :

print(pandas_zoo.head(2)) => affiche les deux premières observations
Pandy 5 M 80 Nord
Lulu 2 F 40 Sud

print(pandas_zoo['Age'].mean()) => calcule l'âge moyen des pandas
4.2

print(pandas_zoo['Sexe'].value_counts()) => crée un autre df pour lequel la clé de chaque observation est une valeur possible de la propriété «Sexe » et la valeur est le nombre d'apparition de cette valeur dans le df initial
M 3
F 2

pandas_zoo['Categorie_Poids'] = pandas_zoo['Poids_kg'].apply(lambda x: 'Léger' if x < 60 else 'Lourd')
print(pandas_zoo)
=> Crée une nouvelle colonne « Catégorie poids » dans le df assignant l'étiquette « Léger » si le poids est *strictement* inférieur à 60 kg et « Lourd » sinon

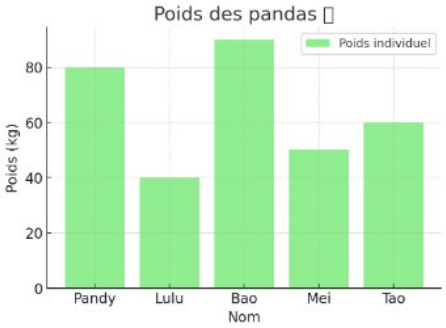
Pandy Lourd
Lulu Léger
Bao Lourd
Mei Léger
Tao Lourd (inégalité stricte)

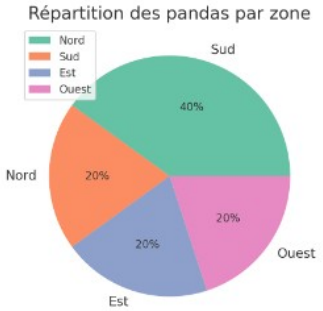
4) Encore plus de pandas

Le code ci-dessous crée deux graphiques à partir d'un DataFrame Pandas. Expliquez ligne par ligne ce qu'il réalise et à quoi servent les légendes.

```
plt.figure(figsize=(6,4))
plt.bar(df["Nom"], df["Poids_kg"], color="lightgreen")
plt.title("Poids des pandas 🐼")
plt.xlabel("Nom")
plt.ylabel("Poids (kg)")
plt.legend(["Poids individuel"])
plt.show()

plt.figure(figsize=(5,5))
df["Zone"].value_counts().plot(kind="pie", autopct="%1.0f%%", c
plt.title("Répartition des pandas par zone")
plt.ylabel("")
plt.legend(df["Zone"].unique(), loc="upper left")
plt.show()
```





a) créé un histogramme représentant le poids de chaque panda.
D'abord on crée une figure (objet de la librairie matplotlib.pyplot, de 6 centimètres sur 4)
On crée ensuite l'histogramme avec sur l'axe des abscisses (x) le nom des pandas et sur l'axe des ordonnées (y) leur poids. On demande explicitement que les barres soient en vert.
On ajoute ensuite un titre au graphique (avec un emoji dans le titre, ne faites jamais ça dans la vraie vie !)
Puis on ajoute des légendes sur chacun des axes et ajoute une légende
Enfin on affiche le graphique avec la commande show() (c'est la dernière figure créée qui est affichée)

b) crée un « pie chart » représentant la répartition des pandas selon leur aire géographique d'origine.
On commence par créer la figure.
Puis on crée un nouveau df qui associe à chaque zone géographique le nombre de pandas qui en est originaire. On utilise la fonction .plot() **de la librairie pandas** pour configurer directement le graphique à partir des données du df. Cette fonction prend en argument le type de graphique qu'on veut (pie chart ici) alors qu'avec matplotlib.pyplot il faut utiliser la fonction associée au type de graphique que l'on veut obtenir.
On ajoute un titre, une légende dans le coin supérieur gauche et on affiche.

5) Rapide et efficace : Réécrivez les 4 extraits de code suivants de façon idiomatique

```
nombres = [1, 2, 3, 4, 5, 6]
pairs = []
for n in nombres:
    if n % 2 == 0:
        pairs.append(n)
print(pairs)
```

pairs = [nombre for nombre in nombres if nombre % 2 == 0]

```
cles = ["nom", "âge", "ville"]
valeurs = ["Alice", 30, "Paris"]
personne = {}
for i in range(len(cles)):
    personne[cles[i]] = valeurs[i]
print(personne)
```

personne = {cle : valeur for cle, valeur in zip(cles,valeurs)}

```
x = 10
y = 20
tmp = x
x = y
y = tmp
print(x, y)
```

y,x = x, y

```
mots = ["Python", "code", "élégant", "lisible"]
tous_longs = True
for mot in mots:
    if len(mot) < 5:
        tous_longs = False
print(tous_longs)
```

tous_longs = all(len(mot) >= 5 for mot in mots)

6) Les précautions :

```
import os, json, logging
from tqdm import tqdm
DOSSIER = "donnees_json"

for nom_fichier in tqdm(os.listdir(DOSSIER), desc="Mise à jour des fichiers JSON"):
    if not nom_fichier.endswith(".json"):
        continue
    chemin = os.path.join(DOSSIER, nom_fichier)
    try:
        with open(chemin, "r", encoding="utf-8") as f:
            data = json.load(f)
        data["universite"] = "INALCO"
        with open(chemin, "w", encoding="utf-8") as f:
            json.dump(data, f, ensure_ascii=False, indent=4)
        logging.info(f"      {nom_fichier} mis à jour.")
    except (json.JSONDecodeError, OSError) as e:
        logging.error(f"      Erreur sur {nom_fichier}: {e}")
```

Que fait ce code ? Qu'est ce qui pourrait mal se passer ?Citez au moins deux sources d'erreur possible.

=> Ce code tente d'ouvrir tous les fichiers du répertoire « données json » qui ont une extension .json, de charger chaque fichier comme un json, de modifier ou ajouter le champs « université » en lui assignant la valeur « INALCO » puis de sauver la version modifiée du fichier avec de bonnes indentations.

=> Ce code pourrait cracher pour plusieurs raisons qui ne dépendent pas de sa qualité d'écriture. Par exemple le dossier « donnees_json » pourrait ne pas exister, il pourrait contenir un fichier qui n'est pas un json (ce qui causerait un crash au moment d'utiliser « load ». Ou encore peut être que l'utilisateur n'a pas les droits d'écriture sur les fichiers (et le dump échouerait). le fichier est très gros ou pas en UTF-8, cela peut causer des erreurs de décodage ou de mémoire.

II) Algorithmique (/10)

Voici un code en python

```python
def encore_des_pandas(num):
    pandas_string = []
    sign_to_unit = {"V": "I", "X": "I", "L": "X", "C": "X", "D": "C", "M": "C"}
    value_to_sign = {1000: 'M', 500: 'D', 100: 'C', 50: 'L', 10: 'X', 5: 'V', 1: 'I'}

    for step in sorted(value_to_sign.keys(), reverse=True):
        roman_letter = value_to_sign[step]
        division = num // step
        if division > 0:
            pandas_string.extend([roman_letter] * division)
            num -= division * step

        reste_dizaine_suivante = int(str(num % step)[0]) if len(str(num)) == len(str(step)) - 1 else 0

        if reste_dizaine_suivante == 4 and int(str(step)[0]) != 5:
            num += step / 10
            pandas_string.append(sign_to_unit[roman_letter])

        if reste_dizaine_suivante == 9 and int(str(step)[0]) != 5:
            num -= 9 * step / 10
            pandas_string.append(sign_to_unit[roman_letter])
            pandas_string.append(roman_letter)
    return pandas_string
```

1) Expliquez à quoi il sert.
Exécutez l'algorithme avec en entrée 419 et profitez-en pour expliquer le code ligne à ligne. /2

=> Ce code prend en entrée un entier (num) et retourne (sous forme de liste) sa représentation en chiffre romains

=> Si on l'exécute avec comme entrée 419 :
* On initialise la liste pandas_string par la liste vide []
* On parcourt le dictionnaire value_to_sign de la valeur la plus grande à la plus petite.
On commence donc avec step = 1000.
On va chercher le signe romain correspondant, roman_letter = M
num représente à chaque instant de l'algorithem **le nombre qu'il reste à écrire**
A ce stade de l'algorithme on a écrit autant de fois qu'on le devait les signes unitaires et multiples de 5 de façon additive.

* On calcule le quotient de la division euclidienne de num par step.
Si le résultat est non nul on écrit autant de fois le symbole roman_letter, et on retire à num
Avec step = 1000, le quotient est nul et il ne se passe rien.

On calcule ensuite le reste de la division de num par step et on ne considère que le chiffre de la dizaine inférieure à step, soit le chiffre des centaines (ou alors le quotient de la division de num par step /10...)

avec step = 1000 reste_dizaine_suivante vaut 4

* On traite ensuite le cas complexe des notations soustractives dans les chiffres romains.

Si reste_dizaine_suivante vaut 4, et que step n'est pas un multiple de 5,
alors on ajoute step/10 à num (pour pouvoir déclencher l'écriture du chiffre suivant dont on soustrait l'unité) et on écrit l'unité inférieure correspondant au step (pour le millier, c'est la centaine…) pour anticiper la notation soustractive.
Comme c'est le cas on a num = 519 et pandas_string = [« C »]

reste_dizaine_suivante vaut 9, et que step n'est pas un multiple de 5,
alors on enlève à num le chiffre de ce reste (anticipation du traitement d'un step plus petit) et on écrit à la fois l'unité à soustraire et la lettre roman_letter (signe correspondant à step)
Ce n'est pas le cas pour step = 1000 car on a reste_dizaine_suivante = 4

pour step = 500, division = 1 donc on écrit D et retire 500 à num
On a pandas_string = [« C », « D »] et num = 19. Aucune des deux conditions n'est satisfaite.

Pour step = 100, division = 0, reste_dizaine_suivante = 1!= 4 ou 9, pas de changement
Pout step = 50,   division = 0, reste_dizaine_suivante = 1!= 4 ou 9, pas de changement

Pour step = 10, division = 1, donc on écrit X et on retire 10 à num
On a pandas_string = [« C », « D », « X »] et num = 9
 reste_dizaine_suivante = 9
La deuxième condition est satisfaite donc on retire à num 9 * step/10 donc 9
num = 0
puis on écrit la valeur du signe de l'unité correspondant à step = 10, soit I
puis on écrit la valeur du signe correspondant à step = 10 soit X
pandas_string = [« C », « D », « X », « I », « X »]

pour step = 5, division = 0, reste_dizaine_suivante = 0, rien ne se passe
pour step = 1, division = 0, reste_dizaine_suivante = 0, rien ne se passe
NB : on aurait pu sortir de l'algorithme plus tôt avec un test if num == 0 : break

2) Quelle est la complexité de ce programme ? Justifiez /1

Toutes les opérations au sein de la boucle sont des opérations élémentaires (conditions, lectures dans un dictionnaire, opérations mathématiques, écritures dans une liste)
Le temps d'exécution du programme est donc linéaire en fonction de la longueur du dictionnaire « value_to_sign »
Or il y a eu un tri de ce dictionnaire en fonction des valeurs. Les valeurs sont une liste.
Or la complexité du tri d'une liste est de O(n log n).
La complexité du programme est donc de O(n log n) avec n longueur du dictionnaire value_to_sign

3) Que pourrait-on écrire pour écrire les lignes 6 et 7 en une seule ligne ? /1

for step, roman_letter in sorted(value_to_sign.items()):

4) Pourquoi doit-on traiter le cas du 4 et du 9 différemment ? /1

Si le reste de l'unité suivante vaut 4, on ajoute à num et ne note que l'unité à soustraire alors que quand il est à 9 on diminue num et écrit à la fois l'unité à soustraire et le chiffre des dizaines.
On doit traiter ces deux cas différemment selon la valeur de step
si le reste de l'unité vaut 4 , le step suivant vaut 5 fois l'unité, or num est inférieur à 5 fois l'unité (puisque le reste est 4, pas ≥ 5!) donc on doit ajouter une unité à num pour s'assurer que le chiffre correspondant au step multiple de 5 sera écrit lors de l'étape suivante de la boucle.
On aura donc noté l'unité à soustraire puis au début de l'étape suivante avec division = 1 on écrit le multiple de 5

Si le reste de l'unité vaut 9, on écrit l'unité à soustraire, puis on doit écrire aussi le symbole suivant (celui qui correspond à step) immédiatement, car au tour suivant de la boucle step n'aura plus cette valeur et elle ne sera plus jamais accessible (l'ordre de la boucle est du plus grand au plus petit).
Puisque on a noté à la fois la soustraction et le multiple de 10, on a noté l'équivalent de 9*X donc on doit le retirer immédiatement de num pour éviter d'écrire plusieurs fois (sinon au tour suivant, le reste par le multiple de 5 serait de 1.

5) Cet algorithme « lit le chiffre de gauche à droite ». /5
Écrivez un algorithme qui fait la même chose mais en lisant la chaîne dans l'autre sens (c'est à dire en commençant par les unités).

on lit les chiffres en sens inverse et on construit progressivement la chaîne final. On va réfléchir ici par dizaine au lieu de réfléchir par signe (ce qui est possible aussi !)

```
def encore_des_pandas_inverse(num):
    pandas_string = []
    value_to_sign = {1: 'I', 5: 'V', 10: 'X', 50: 'L', 100: 'C', 500: 'D', 1000: 'M'}

    # On lit le nombre chiffre par chiffre en partant des unités
    chiffres = list(str(num))[::-1]  # renverse la chaîne
    for i, chiffre in enumerate(chiffres):
        chiffre = int(chiffre)
        if chiffre == 0:
            continue

        # détermine la puissance de 10 correspondante
        base = 10 ** i
        unité = value_to_sign[base]
        cinq = value_to_sign.get(5 * base, "")
        dix = value_to_sign.get(10 * base, "")

        if chiffre <= 3:
            pandas_string.insert(0, unité * chiffre)
        elif chiffre == 4:
            pandas_string.insert(0, unité + cinq)
        elif chiffre <= 8:
            pandas_string.insert(0, cinq + unité * (chiffre - 5))
        else:  # 9
            pandas_string.insert(0, unité + dix)
    return pandas_string
```