SAE15 – Référence de codes essentiels

Ce document regroupe les codes Python essentiels utilisés pour le projet SAE15 et répond explicitement aux questions du TP1, sans rajouter les cellules déjà présentes dans le notebook Jupyter. Il sert de référence technique pour la collecte, le stockage, le nettoyage, l'analyse et la visualisation des données.

---

1. Requêtes HTTP et collecte de données (TP1 – Partie 1)

Objectif

Récupérer les données d'occupation des parkings depuis l'API Open Data de Montpellier.

Code essentiel

```
import requests

url = "https://portail-api-data.montpellier3m.fr/offstreetparking?limit=1000"
response = requests.get(url)

# Vérification du statut HTTP
print(response.status_code)
```

Explication (réponse TP1)

- `requests.get()` envoie une requête HTTP GET au serveur.
- `response.status_code == 200` indique que la requête a réussi.
- `response.text` contient la réponse brute du serveur.

---

2. Analyse et parsing JSON (TP1 – Partie 2)

Conversion JSON

```
import json

data = response.json()
```

Réponse aux questions TP1

- La réponse est convertie en **liste de dictionnaires Python**.
- Un fichier JSON est structuré en **paires clé / valeur**, avec des objets imbriqués.

Sauvegarde locale pour éviter les requêtes inutiles

```
with open("data.json", "w", encoding="utf-8") as f:
    json.dump(data, f, indent=4)
```

Rechargement depuis le fichier

```
with open("data.json", encoding="utf-8") as f:
    data = json.load(f)
```

---

3. Extraction des informations utiles (TP1 – Questions 2, 3, 4)

a) Nombre de places libres (parkings ouverts uniquement)

```
for parking in data:
    if parking.get("status", {}).get("value") == "open":
        libres = parking["availableSpotNumber"]["value"]
        print(libres)
```

b) Nom du parking + places libres

```
for parking in data:
    if parking.get("status", {}).get("value") == "open":
        nom = parking["name"]["value"]
        libres = parking["availableSpotNumber"]["value"]
        print(nom, libres)
```

c) Pourcentage de places libres

```
for parking in data:
    if parking.get("status", {}).get("value") == "open":
        nom = parking["name"]["value"]
        libres = parking["availableSpotNumber"]["value"]
        capacite = parking["totalSpotNumber"]["value"]
        taux = libres / capacite * 100
        print(nom, round(taux, 2), "%")
```

---

4. Gestion du temps et automatisation (TP1 – Partie 3)

Timestamp UNIX

```
import time

temps = int(time.time())
print(temps)
```

Réponse TP1

- Ce nombre représente le **temps écoulé depuis le 1er janvier 1970 (epoch UNIX)**.

Suivi d’un parking toutes les 10 secondes pendant 5 minutes

```
import time

duree = 300
```

```
intervalle = 10

debut = time.time()

while time.time() - debut < duree:
    response = requests.get(url)
    data = response.json()
    for p in data:
        if p.get("name", {}).get("value") == "Corum":
            print(time.time(), p["availableSpotNumber"]["value"])
    time.sleep(intervalle)
```

---

5. Passage au DataFrame (fondamental pour le projet)

```
import pandas as pd

df = pd.DataFrame([
    {
        "Nom": p["name"]["value"],
        "Disponibilite": p["availableSpotNumber"]["value"],
        "Capacite": p["totalSpotNumber"]["value"],
    }
    for p in data if p.get("status", {}).get("value") == "open"
])

# Calculs dérivés
df["Places_occupees"] = df["Capacite"] - df["Disponibilite"]
df["Taux_occupation_%"] = df["Places_occupees"] / df["Capacite"] * 100
```

---

6. Codes essentiels d’analyse (au-delà du TP1)

Moyenne par parking

```
df.groupby("Nom")["Taux_occupation_%"].mean()
```

Diagnostic de dimensionnement

```
def diagnostic(taux):
    if taux < 40:
        return "Sur-dimensionné"
    elif taux <= 80:
        return "Bien dimensionné"
    else:
        return "Sous-dimensionné"


df["Diagnostic"] = df["Taux_occupation_%"].apply(diagnostic)
```

---

7. Visualisation essentielle (hors notebook détaillé)

Graphique simple

```
import matplotlib.pyplot as plt

plt.figure(figsize=(8,4))
plt.bar(df["Nom"], df["Taux_occupation_%"])
plt.xticks(rotation=30)
plt.ylabel("Taux d'occupation (%)")
plt.title("Occupation des parkings")
plt.tight_layout()
plt.show()
```

---

8. Carte interactive (relais voiture / vélo)

```
import folium

m = folium.Map(location=[43.61, 3.88], zoom_start=13)

for _, row in df.iterrows():
    folium.Marker(
        [row.get("Latitude", 43.61), row.get("Longitude", 3.88)],
        popup=row["Nom"],
        icon=folium.Icon(icon="car", color="red")
    ).add_to(m)

m
```

---

9. Démarche de création d'une librairie personnelle (TP1 – Question 10)

Principe : - Regrouper les fonctions réutilisables (`fetch_data()`, `clean_data()`, `compute_rates()`) - Les documenter avec des docstrings - Les stocker dans un fichier `utils_parking.py`

```
def fetch_data(url):
    """Récupère les données JSON depuis l'API Montpellier."""
    r = requests.get(url)
    return r.json()
```

Conclusion du mémo

Ce fichier constitue une **boîte à outils complète** répondant aux exigences du TP1 et aux besoins du projet SAE15. Il justifie les choix techniques, centralise les codes essentiels et facilite la maintenance et la réutilisation du travail.