

TP : Script de statistiques

Objectif

Pendant la chasse au trésor, toute la promo a passé des commandes au bar. Le patron d'API-ritif veut maintenant des **statistiques**, et il compte sur vous.

Vous allez compléter un script **Python** qui récupère les données par l'API et calcule des statistiques.

Ce TP est individuel et noté.

Le fichier de départ

Copiez le code ci-dessous dans un fichier `stats.py`. La partie réseau est **déjà écrite** : la fonction `get_all` parcourt toutes les pages et gère les `429` (voir la page [Pagination, erreurs, 401 et 429](#)). Votre travail porte uniquement sur les données.

Le script fonctionne dès le départ : chaque question affiche `None` tant que vous ne l'avez pas faite. Commencez par le lancer, et regardez les deux exemples affichés en haut : ils montrent les champs d'une commande et d'un cocktail.

```
# TP2 - Script de statistiques sur API-ritif
#
# Lancement :
#   export API_KEY=ma-cle          (sous Windows PowerShell : $env:API_KEY="ma-cle")
#   python stats.py > resultat.txt
#
# Le script fonctionne dès le départ : chaque question affiche None
# tant que vous ne l'avez pas faite. Complétez les blocs « À FAIRE ».

import os
import time
from urllib.parse import urljoin
```

```

import requests

BASE = "http://api-ritif.portlatch.com/v1"

# La clé est lue dans une variable d'environnement, jamais écrite dans le code.
HEADERS = {"X-API-Key": os.environ["API_KEY"]}

# =====
# FOURNI : ne pas modifier
# =====

def polite_get(url):
    """GET qui attend et recommence tant que le barman répond 429."""
    while True:
        res = requests.get(url, headers=HEADERS)
        if res.status_code != 429:
            res.raise_for_status()
            return res.json()
        wait = int(res.headers.get("Retry-After", 1))
        print("(429 reçu, pause de", wait, "s)")
        time.sleep(wait)

def get_all(path):
    """Renvoie TOUS les éléments d'une liste paginée, en suivant le lien next."""
    items = []
    url = urljoin(BASE, path)
    while url:
        data = polite_get(url)
        items.extend(data["items"])
        next_link = data["_links"].get("next")
        url = urljoin(BASE, next_link["href"]) if next_link else None
    return items

orders = get_all("/v1/orders?scope=all&per_page=50")
cocktails = get_all("/v1/cocktails?per_page=50")

```

```

# Regardez bien ces deux exemples : ils montrent les champs disponibles.
print("Exemple de commande :", orders[0])
print("Exemple de cocktail :", cocktails[0])
print()

# =====
# QUESTIONS OBLIGATOIRES
# =====

# ----- Question 1 : le nombre total de commandes -----

total = None # À FAIRE

print("Q1 - Nombre de commandes :", total)

# ----- Question 2 : le nombre de commandes par statut -----
# EXEMPLE RÉSOLU : c'est le modèle pour compter, à réutiliser plus bas.
# On part d'un dictionnaire vide, et pour chaque commande on ajoute 1
# au compteur de son statut (get renvoie 0 si le statut n'est pas encore vu).

by_status = {}
for order in orders:
    status = order["status"]
    by_status[status] = by_status.get(status, 0) + 1

print("Q2 - Commandes par statut :", by_status)

# ----- Question 3 : le cocktail le plus commandé -----
# Indice 1 : comptez les commandes par cocktail, comme dans la question 2.
# Indice 2 : max(d, key=d.get) renvoie la clé du dictionnaire d
#           qui a la plus grande valeur.

most_ordered = None # À FAIRE

print("Q3 - Cocktail le plus commandé :", most_ordered)

```

```

# ----- Question 4 : l'ardoise de la promo -----
# Le montant total des commandes, SANS compter les commandes annulées
# (statut "cancelled").
# Indice : une variable qui commence à 0, une boucle, un if.

tab = None # À FAIRE

print("Q4 - Ardoise de la promo :", tab, "€")


# =====
# BONUS
# =====

# ----- Bonus 1 : le client qui a passé le plus de commandes -----
# Indice : même principe que la question 3.

best_customer = None # À FAIRE

print("B1 - Meilleur client :", best_customer)


# ----- Bonus 2 : la note moyenne de chaque cocktail -----
# Les avis d'un cocktail : get_all("/v1/cocktails/" + str(cocktail["id"]) + "/reviews")
# Moyenne = somme des notes / nombre de notes, arrondie au dixième avec round(x, 1).
# Attention : un cocktail peut n'avoir aucun avis (division par zéro !).

averages = None # À FAIRE : un dictionnaire nom du cocktail -> note moyenne

print("B2 - Notes moyennes :", averages)


# ----- Bonus 3 : la part de cocktails sans alcool parmi les commandes -----
# Le champ alcohol_free est dans la carte (cocktails), pas dans les commandes.
# Résultat en pourcentage, arrondi au dixième.

alcohol_free_percent = None # À FAIRE

```

```
print("B3 - Commandes sans alcool :", alcohol_free_percent, "%")
```

Consignes

1 - Questions obligatoires

1. Le nombre total de commandes.
2. Le nombre de commandes par statut : **déjà résolue**, c'est le modèle à réutiliser pour compter.
3. Le cocktail le plus commandé.
4. « L'ardoise » de la promo : le montant total des commandes, sans compter les commandes annulées.

2 - Bonus

1. Le client qui a passé le plus de commandes.
2. La note moyenne de chaque cocktail, arrondie au dixième (une requête par cocktail).
3. La part de cocktails sans alcool parmi les commandes, en pourcentage.

3 - Défis (non notés)

- L'heure de pointe : la tranche de 10 minutes avec le plus de commandes.
- Un classement des cocktails par note moyenne, en n'incluant que ceux qui ont au moins 3 avis.

Contraintes

- La clé d'API est lue dans la variable d'environnement `API_KEY`, jamais écrite dans le code.
- Aucune donnée n'est saisie à la main : tout vient de l'API.
- Ne modifiez pas la partie « FOURNI ».

Résultat attendu

- Votre script complété (`stats.py`).
- Un fichier `resultat.txt` contenant sa sortie : `python stats.py > resultat.txt`.

Vos résultats seront comparés à ceux du formateur.

Revision #5

Created 2 October 2026 20:00:47 by Thibaud FRICHET

Updated 13 October 2026 20:41:01 by Thibaud FRICHET