I) PYTHON

Objectif : Réaliser un code clair, facilement lançable (via Docker) et facilement partageable (via Git)

1. L'algorithme

Commencer par résoudre ce problème très simple : https://www.hackerrank.com/challenges/sparse-arrays/problem

2. Première partie du test

Quand vous aurez la solution, je vous propose de nettoyer un peu le code et de le mettre dans une classe qui sera dans un module python séparé (un autre fichier) et de faire un "main" qui se sert de cette classe.

Dans votre dossier, vous pourriez, par exemple, avoir une arborescence du type :
main.py
SparseArray.py
README.md (qui décrit, en anglais, ce que fait le code)

On pourra appeler le main avec la commande suivante (qui reprend l'exemple 1):
python -m main ab,abc,bc
et qui renvoie, en print par exemple, un dictionnaire du type :
{ab: 2, abc: 1, bc: 0}

J'ai volontairement mis un seul argument en entrée du code (la "query" de l'énoncé) :

- l'array "strings" (que vous parcourrez pour la comparer avec la "query") pourrait être une variable d'environnement qu'on pourra lire dans le main et qui sera passée au constructeur de la classe SparseArray.py (vous verrez en dernière étape pourquoi)
- la taille des tableaux "query" et "strings" pourrait être calculée par le code et non passée en input

Je vous laisse un peu vous organiser : ce que je vais regarder c'est votre façon de coder. Au delà de l'algorithmie, je voudrais voir comment vous nommez les variables, les fichiers, les classes (mes noms ne sont que des propositions qui me vont bien mais que vous pouvez changer si vous le souhaitez), comment vous rédigez votre README, les paramètres que vous passez à vos fonctions, ...

Il y-a toujours plusieurs manières d'arriver à un résultat en revanche j'aime bien la rigueur, la propreté et la clarté 🙂

3. Seconde partie du test

Nous utilisons énormément Docker chez Maisons du Monde. Je vous propose donc de continuer le test en containerisant votre code à l'aide de Docker. Vous pourriez réaliser un Dockerfile qui :

- utilisera python:3.7 comme image de base
- assignera la variable d'environnement que vous utilisez dans votre main.py (c'est un des principes de la 12 factor app https://12factor.net/config)
- lancera le code avec la commande ENTRYPOINT

On pourrait :

- builder l'image Docker avec la commande : **docker build . -t test_mdm**
- runner le container avec la commande : **docker run -t test_mdm ab,abc,bc**

4. Pour aller plus loin

Je vous propose de réaliser, toujours avec Docker, une API Flask très simple avec swagger pour le rendu. Le but est d'avoir une url que vous pourriez interroger avec, comme argument, la "query". Ce qui change c'est qu'au lieu d'envoyer le code en ligne de commande, on passe maintenant par une API avec, grâce à swagger, un beau rendu.
Il n'y a que le main et le Dockerfile qui devrait changer.

5. Le rendu

Je propose de me partager votre test sur un repo git hébergé chez le fournisseur de votre choix (chez nous, on utilise GitLab mais si vous préférez GitHub ou BitBucket, c'est vous qui voyez).

II) SQL

Objectif : Réaliser des requêtes SQL claires et facilement compréhensibles

1. Les données

Nous avons les 2 tables suivantes :

TRANSACTIONS
Cette table contient des données transactionnelles. Avec les infos suivantes :

- date : date à laquelle la commande a été passée
- order_id : identifiant unique de la commande
- client_id : identifiant unique du client
- prod_id : identifiant unique du produit acheté
- prod_price : prix unitaire du produit
- prod_qty : quantité de produit achetée

Echantillon de la table TRANSACTIONS :

date	order_id	client_id	prod_id	prod_price	prod_qty
01/01/20	1234	999	490756	50	1
01/01/20	1234	999	389728	3,56	4
01/01/20	3456	845	490756	50	2
01/01/20	3456	845	549380	300	1
01/01/20	3456	845	293718	10	6

PRODUCT_NOMENCLATURE
Cette table contient le référentiel produit c'est à dire les méta-données du produit. On y trouve les infos suivantes :

- product_id : identifiant unique du produit
- product_type : type de produit (DECO ou MEUBLE)
- product_name : le nom du produit

Echantillon de la table PRODUCT_NOMENCLATURE :

product_id	product_type	product_name
490756	MEUBLE	Chaise
389728	DECO	Boule de Noël
549380	MEUBLE	Canapé
293718	DECO	Mug

2. Première partie du test

Je vous propose de commencer par réaliser une requête SQL simple permettant de trouver le chiffre d'affaires (le montant total des ventes), jour par jour, du 1er janvier 2019 au 31 décembre 2019. Le résultat sera trié sur la date à laquelle la commande a été passée.
Je rappelle que la requête doit être claire : n'hésitez pas à utiliser les mot clefs AS permettant de nommer les champs dans SQL.

Echantillon des résultats de la requête :

date	ventes
01/01/2020	524240
02/01/2020	520918
03/01/2020	526983
04/01/2020	520987
05/01/2020	524879
06/01/2020	524436

2. Seconde partie du test

Réaliser une requête un peu plus complexe qui permet de déterminer, par client et sur la période allant du 1er janvier 2019 au 31 décembre 2019, les ventes meuble et déco réalisées.

Echantillon des résultats de la requête :

client_id	ventes_meuble	ventes_deco
999	50	14,24
845	400	60