CONTROLE DE FIN DE THEME

Exercice 1 – Indexations de tables (5pts)

Ci-dessous, le **contenu** d'un **fichier CSV** d'un système de gestion des stocks d'un magasin :

```
etiquette;rayon;poids;prix;stock
Pommes;Fruits;1.2;2.99;150
Lait;Crèmerie;1.0;1.15;80
Pain de mie;Boulangerie;0.5;1.80;60
Spaghetti;Épicerie;0.5;0.95;200
Riz;Épicerie;1.0;2.50;120
```

a. Quel est le **stock** du produit **le plus cher** ?

Le code suivant permet **d'indexer** la table en Python sous la forme d'un **Tableau Doublement Indexé** :

```
1  csv_f = [ A ]("magasin.csv", [ B ]) # ouverture du fichier
2  csv_f.readline() # ignorer les en-têtes
3
4  produits = []
5  ligne = csv_f.readline()
6  while [ C ]: # tant que la ligne n'est pas vide
7      produits.append(ligne.strip().split(";"))
8      ligne = csv_f.readline()
9
10 csv_f.close()
```

b. **Quelle est la fonction** qu'il faut utiliser à l'emplacement [A] ?

c. **Par quoi remplacer** l'emplacement [B], qui est le **paramètre « mode de lecture »** de la fonction ?

d. **Quelle est la condition** à indiquer dans l'emplacement [C] pour permettre **d'itérer sur chaque ligne** du fichier jusqu'à les avoir **toutes lues** ?

e. Quelle est la **valeur d'évaluation** de **l'expression suivante**, exécutée après le code ci-dessus ?

```
type(produits[0])
```

Exercice 2 – Sélections dans une table (10pts)

Un **autre code** permet **d'indexer la table** au format d'un **Tableau de P-uplets Nommés**, ce qui nous permet d'obtenir finalement **la table suivante** :

```
stock = [
  { "etiquette": "Pommes", "rayon": "Fruits",
    "poids": 1.2, "prix": 2.99, "stock": 150 },
  { "etiquette": "Lait", "rayon": "Crèmerie",
    "poids": 1.0, "prix": 1.15, "stock": 80  },
  { "etiquette": "Pain de mie", "rayon": "Boulangerie",
    "poids": 0.5, "prix": 1.80, "stock": 60 },
  { "etiquette": "Spaghetti", "rayon": "Épicerie",
    "poids": 0.5, "prix": 0.95, "stock": 200 },
  { "etiquette": "Riz", "rayon": "Épicerie",
    "poids": 1.0, "prix": 2.50, "stock": 120 }
]
```

Il s'agit donc d'une **liste** de **dictionnaires**. Par souci de **place sur la feuille**, **chaque dictionnaire** est écrit sur **deux lignes**.

a. **Écrire une fonction `valeur_rayon(stock, rayon)`**, où **`stock`** est une **table** au format de la table ci-dessus, **`rayon`** est le **nom d'un rayon** et qui **renvoie la somme totale de la valeur des stocks** des produits **qui sont rangés dans ce rayon** (*valeur du stock = prix à l'unité × stock*).

b. **Écrire une fonction `taille_rayon(stock, rayon)`**, où **`stock`** est une **table** au format de la table ci-dessus, **`rayon`** est le **nom d'un rayon** et qui **renvoie le nombre de produits différents qui sont rangés dans ce rayon**.

c. **Compléter le code suivant** qui utilise une **création de liste par compréhension** pour afficher **le poids de stock** du **produit** qui a **le plus grand poids de stock** (*poids de stock = poids à l'unité × stock*). Vous devez compléter **l'intérieur des crochets** ainsi que **la fonction utilisée à l'avant-dernière ligne**.

```
# utiliser la table contenue dans la variable 'stock'
liste_compr = [




                                                    ]
plus_grand_poids = ............(liste_compr)
print(plus_grand_poids)
```

d. **Compléter le code suivant** qui utilise une **création de liste par compréhension** pour afficher **`True`** si **un produit '`produit`'** est **présent dans un rayon '`rayon`'**, et **`False`** dans le cas contraire. Vous devez compléter **l'intérieur des crochets** ainsi que **l'expression utilisée à l'avant-dernière ligne**.

```
# utiliser la table contenue dans la variable 'stock'
rayon = "Épicerie" # utiliser la variable 'rayon'
produit = "Coquillettes" # utiliser la variable 'produit'
liste_compr = [




                                                    ]
produit_present = ................................
print(produit_present)
```

Exercice 3 – Tri de tables (5pts)

Dans cet exercice, on considère **la table ci-dessous** qui est un **Tableau Doublement Indexé** :

```
magasin = [
    ["Pommes", "Fruits", 1.2, 2.99, 150],
    ["Lait", "Crèmerie", 1.0, 1.15, 80],
    ["Pain de mie", "Boulangerie", 0.5, 1.80, 60],
    ["Spaghetti", "Épicerie", 0.5, 0.95, 200],
    ["Riz", "Épicerie", 1.0, 2.50, 120]
]
```

Les **descripteurs** sont respectivement le **nom du produit**, le **rayon**, le **poids**, le **prix unitaire** et le **stock**.

a. **Compléter le code suivant** pour que **`prix_kg`** soit une table des produits **triés par ordre croissant du prix au poids** (*prix au poids = prix à l'unité / poids*).
Vous pouvez utiliser **soit un bloc de définition** de fonction (`def`…), **soit une lambda-expression**.

```


prix_kg = sorted(magasin, key = ................................)
print(prix_kg)
```

On dispose d'un **dictionnaire `prio_reappro`** qui **associe** à chaque **nom de produit** une **priorité de réapprovisionnement** : une **priorité faible** indique un produit à réapprovisionner **moins souvent** (faible péremption), et inversement pour une **priorité élevée** qui indique un produit à réapprovisionner **plus souvent**.

```
prio_reappro = { "Yaourt": 3, "Fromage": 2, "Riz": 1,
                 "Spaghetti": 1, "Pain de mie": 2, "Poulet": 3,
                 "Pomme": 3, "Biscuit": 1 }
```

b. **Compléter le code suivant** pour que **`ordre_prio`** soit une table des produits **triés par ordre croissant de priorité de réapprovisionnement**. Si un produit **est absent du dictionnaire `prio_reappro`**, alors on lui donne **une priorité arbitraire de `10`**.
Vous pouvez utiliser **soit un bloc de définition** de fonction (`def`…), **soit une lambda-expression**.

```
ordre_prio = sorted(magasin, key = ..........................)
print(ordre_prio)
```

c. **Expliquer** quel **ordre de tri** sera utilisé par la **fonction `sorted`** dans l'expression suivante :

```
noms_produits = ["Banane", "Beurre", "Fromage", "Poulet",
                 "Saumon", "Café", "Sucre", "Lessive"]
print(sorted(noms_produits, key = len))
```