

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 **prix** du produit **au plus grand stock** ?



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** :

```
1 stock = [  
2   { "etiquette": "Pommes", "rayon": "Fruits",  
3     "poids": 1.2, "prix": 2.99, "stock": 150 },  
4   { "etiquette": "Lait", "rayon": "Crèmerie",  
5     "poids": 1.0, "prix": 1.15, "stock": 80 },  
6   { "etiquette": "Pain de mie", "rayon": "Boulangerie",  
7     "poids": 0.5, "prix": 1.80, "stock": 60 },  
8   { "etiquette": "Spaghetti", "rayon": "Épicerie",  
9     "poids": 0.5, "prix": 0.95, "stock": 200 },  
10  { "etiquette": "Riz", "rayon": "Épicerie",  
11    "poids": 1.0, "prix": 2.50, "stock": 120 }  
12 ]
```

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 `stock_poids_max(stock)`**, où **stock** est une **table** au format de la table ci-dessus, et qui **renvoie le nom** (étiquette) du **produit** qui a le **plus grand poids de stock** ($\text{poids de stock} = \text{poids à l'unité} \times \text{stock}$).



- b. **Écrire une fonction `dans_rayon(stock, produit, rayon)`**, où **stock** est une **table** au format de la table ci-dessus, **produit** est le **nom** (étiquette) d'un produit, et **rayon** est le **nom d'un rayon** et qui **renvoie True** si le produit est **dans ce rayon**, et **False** dans le cas contraire.



- c. Compléter le code suivant qui utilise une **création de liste par compréhension** pour afficher la **somme totale de la valeur des stocks** des produits qui sont rangés dans ce rayon (*valeur du stock = prix 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'
rayon = "Épicerie" # utiliser la variable 'rayon'
liste_compr = [

]

somme_valeur_rayon = .....(liste_compr)
print(somme_valeur_rayon)
```

- d. Compléter le code suivant qui utilise une **création de liste par compréhension** pour afficher le **nombre de produits différents qui sont présents dans un rayon 'rayon'**. 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'
rayon = "Épicerie" # utiliser la variable 'rayon'
liste_compr = [

]

nombre_en_rayon = .....(liste_compr)
print(nombre_en_rayon)
```

Exercice 3 – Tri de tables (5pts)

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

```
1 magasin = [
2     ["Pommes", "Fruits", 1.2, 2.99, 150],
3     ["Lait", "Crèmerie", 1.0, 1.15, 80],
4     ["Pain de mie", "Boulangerie", 0.5, 1.80, 60],
5     ["Spaghetti", "Épicerie", 0.5, 0.95, 200],
6     ["Riz", "Épicerie", 1.0, 2.50, 120]
7 ]
```

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**.

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

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

Empty box for explanation.

