PYTHON

DEVOIR MAISON

Distribué le 10/11/2020

À rendre au plus tard le 16/11/2020

Préambule

1. Créez un dossier `dm_python_2_prenom` (en remplaçant `prenom` par votre prénom). Créez un fichier `.py` par exercice, les noms de fichier sont donnés dans les intitulés de chaque exercice. Enregistrez les fichiers dans le dossier que vous avez créez
2. Architecture du dossier une fois DM terminé:
   | dm_python_2_prenom
   | __ dict_students.py
   | __ eCommerce.py
   | __ mystery.py
   | __ students.py
   | __ warmup.py
3. Quand vous avez terminé, compressez votre dossier et envoyez moi le fichier compressé sur Slack en DM
4. Commentez votre code, décrivez ce que vous faites: **pas de commentaire = -2 point**
5. Les questions sont indépendantes les unes des autres, ne bloquez pas sur un bug ou une question. Si vous n'y arrivez pas ou que votre bug vous résiste, commentez ce qui crée le bug et passez à la question suivante

---

Dict

Réalisez les deux exercices suivants en partant de la list `students`

```
students = [
        {
                "email": "lucas@example.com",
                "marks": [ 18, 19, 15 ],
                "name": "lucas",
                "option": None
        },
        {
                "email": "raphael@example.com",
                "marks": [ 19, 20, 16 ],
                "name": "raphael",
                "option": None
        },
        {
                "email": "lucie@example.com",
                "marks": [ 18, 18, 16 ],
                "name": "lucie",
                "option": { "duration": 3, "name": "web" }
        },
        {
                "email": "arena@example.com",
                "marks": [ 15, 17, 18 ],
                "name": "arena",
                "option": None
        },
        {
                "email": "alexandre@example.com",
                "marks": [ 20, 20, 15 ],
                "name": "alexandre",
                "option": { "duration": 9, "name": "python" }
        },
        {
                "email": "marie@example.com",
                "marks": [ 18, 12, 17 ],
                "name": "marie",
                "option": { "duration": 10, "name": "data" }
        },
        {
                "email": "simon@example.com",
                "marks": [ 17, 18, 20 ],
                "name": "simon",
                "option": { "duration": 7, "name": "SEO" }
        },
        {
                "email": "mirana@example.com",
                "marks": [ 17, 17, 16 ],
```

```
                "name": "mirana",
                "option": None
        },
        {
                "email": "lilou@example.com",
                "marks": [ 19, 20, 20 ],
                "name": "lilou",
                "option": { "duration": 10, "name": "SEO" }
        },
        {
                "email": "yann@example.com",
                "marks": [ 14, 13, 14 ],
                "name": "yann",
                "option": { "duration": 10, "name": "UX" }
        },
        {
                "email": "lyvahne@example.com",
                "marks": [ 18, 20, 15 ],
                "name": "lyvahne",
                "option": { "duration": 9, "name": "UX" }
        }
]
```

Warm-Up (1 points - `warmup.py`)

Donnez les réponses sous forme de commentaires.

1. Quel est le type de `students` et quel est le type des objets contenus dans `students` ?
2. Quelle est la méthode pour connaitre le nombre d'éléments contenus dans `students` ?

Dict Usage (4 points - `dict_students.py`)

1. Affichez la liste des emails de tous les étudiants
2. Pour chaque élève, calculez puis affichez sa moyenne
3. Pour chaque élève, ajoutez la clef `"avg_mean"` qui prend pour valeur la moyenne de l'étudiant
4. Pour les étudiants qui ont une option et uniquement pour ceux-la, affichez le prénom de l'étudiant, le nom de l'option et sa durée => `alexandre python 9`
5. Pour les étudiants qui ont une option **et** qui ont au moins un 20 dans leur notes et

uniquement pour ceux-la, affichez le prénom de l'étudiant avec la string `"is taking the hard way"` => `alexandre is taking the hard way`

Functions

Students Generator (8 points - `students.py`)

L'objectif de cet exercice est de créer un dict représentant un étudiant. Au terme de l'exercice, le dict comprendra les clefs suivantes: `"name"`, `"email"`, `"marks"` et `"option"`.
Pour cela, il vous sera demandé dans un premier temps de créer des fonctions réalisant des tâches simples, par exemple créer une adresse email à partir du prénom de l'étudiant.
Dans un second temps, vous vous servirez de ces fonctions pour créez le dict.

Email Generator (2 points)

Fonction qui génère et retourne une adresse email (type `str`) à partir d'un argument de type string. L'adresse email doit être sous la forme `"name@example.com"`

| --- | --- |
Nom	`create_email_address`
Arguments	name (type `str`)
Exemple d'appel	`create_email_address("esd")`
Traitement	Génère une adresse email commençant par le nom de l'étudiant et se terminant par `"@example.com"`
Exemple de retour	`"esd@example.com"`
Type du retour	`str`

Option Generator (2 points)

Fonction qui génère et retourne une option sous forme de `dict`. Une option est définit par son nom (`"name"`) et sa durée (`"duration"`). Le nom et la durée sont choisis au hasard dans les listes passées en argument.

| --- | --- |
Nom	`create_option`
Arguments	options (type `list`)
	durations (type `list`)
Exemple d'appel	`create_option(["python", "web"], [3, 6, 9])`
Traitement	Génère une option en prenant au hasard un élément de la liste `["python", "web"]` pour l'option et au hasard un élément de la liste `[3, 6, 9]` pour la durée
Exemple de retour	`{"name": "web", "duration": 9}`
Type du retour	`dict`
Hint	Utilisez la méthode `choice` du package `random` pour choisir *au hasard* un élément d'une liste: doc random.choice()

Marks Generator (2 points)

Fonction qui génère et retourne une listes de `n` notes

| --- | --- |
Nom	`create_marks`
Arguments	n (type `int`)
Exemple	`create_marks(4)`

| --- | --- |
d'appel	
Traitement	Génère une liste de n (ici 4) notes entre 0 et 20
Exemple de retour	`[12, 18, 15, 11]`
Type du retour	`list`
Attention	Les notes doivent être dans l'interval [0, 20]
Hint	Utilisez la méthode `randint` du package `random` pour générer *au hasard* des int: doc random.randint()

Student Generator (2 points)

Maintenant que vous pouvez créer une adresse email à partir d'une string, et que vous pouvez générer des notes et des options au hasard, créez une fonction qui génère et retourne un étudiant. Une étudiant est définit par son nom (`"name"`), son email (`"email"`), ses notes (`"marks"`) et son option (`"option"`).

Attention: vous devez utilisez vos fonctions créées précédement.

| --- | --- |
Nom	`generate_student`
Arguments	name (type `str`)
	options (type `list`)
	durations (type `list`)
	n (type `int`)
Appel	`generate_student("hubert", ["python", "web"], [3, 6, 9], 3)`
Traitement	Génère un étudiant ayant pour prénom `"hubert"` , génère l'adresse email, les options et les notes via les méthodes créées précédemment

| --- | --- |
Exemple de retour	`{"name": "hubert", "email": "hubert@example.com",`
	`"option": {"name": "python", "duration": 6}, "marks: [15, 11, 18]}`
Type du retour	`dict`

eCommerce Order Report (10 points - `eCommerce.py`)

L'objectif de cet exercice est de générer un rapport basé sur une commande passée en ligne. Ce rapport devra contenir:

1. Le nombre d'articles commandés
2. Le prix total des articles la commande
3. Calculer si des frais de port doivent être ajoutés
4. Un récapitulatif de la commande

Les fonctions que vous définirez doivent être **générique**, c'est à dire qu'elles doivent être en mesure de prendre le dict ci-dessous comme base, mais si une autre commande comprend 10 `products`, votre fonction doit toujours fonctionner.

Le dict `order` ci-dessous représente la commande passée par un utilisateur, il contient les trois clefs suivantes:

1. `user`: les info de l'utilisateur ayant passé la commande
2. `order_details`: des metadata sur la commande
3. `products`: la liste des produits commandés

```
order = {
        "user": {
                "name": "Hubert Bonisseur de La Bath",
                "email": "hubert@example.com",
                "address": {
                        "street_name": "141, Blvd Mortier",
                        "postal_code": "75020",
                        "city": "Paris",
                        "country": "France"
                }
        },
        "order_details": {
                "order_id": "kdu6g479kdf4e9",
                "created_at": "2020-01-01",
                "paid": True,
                "delivered": False,
                "source": "somewebsite.com"
        },
        "products": [
                {
                        "product_name": "iPhone",
                        "product_id": "38y5fz4456789e",
                        "quantity": 1,
                        "price": 999
                },
                {
                        "product_name": "Smart TV",
                        "product_id": "83y5fz44yel899",
                        "quantity": 1,
                        "price": 1999
                },
                {
                        "product_name": "HDMI Cable",
                        "product_id": "j67ffzk7sel8f3",
                        "quantity": 5,
                        "price": 12
                },
                {
                        "product_name": "Smart Watch",
                        "product_id": "k0l33y5fz44y99",
                        "quantity": 1,
                        "price": 699
                }
        ]
}
```

Nombre d'articles commandés (2 points)

Fonction qui calcule et retourne le nombre d'articles (type `int`) commandés à partir d'un argument de type `dict`. L'argument passé aura toujours la même structure que `order` ci-dessus, avec des valeurs différentes, mais toujours avec les clefs `user`, `order_details` et `products`.
Attention: pour le dict ci-dessus, le total d'articles est `1 + 1 + 5 + 1 = 8`

| --- | --- |
Nom	`get_articles_nb`
Arguments	order (type `dict`)
Exemple d'appel	`get_articles_nb(order)`
Traitement	Calcule le nombre d'article d'une commande à partir d'un dict
Exemple de retour	`8`
Type du retour	`int`

Prix total de la commande (2 points)

Fonction qui calcule et retourne le prix total (type `int`) de tous les articles d'une commande à partir d'un argument de type `dict`. L'argument passé aura toujours la même structure que `order`, avec des valeurs différentes, mais toujours avec les clefs `user`, `order_details` et `products`.
Attention: pour le dict ci-dessus, le prix total est
`1 * 999 + 1 * 1999 + 5 * 12 + 1 * 699 = 3 757`

| --- | --- |
Nom	`get_articles_price`
Arguments	order (type `dict`)
Exemple d'appel	`get_articles_price(order)`
Traitement	Calcule le prix total de tous les article d'une commande à partir d'un dict

| --- | --- |
| Exemple de retour | `3757` |
| Type du retour | `int` |

Frais de port (2 points)

Fonction qui calcule si des frais de ports sont applicables en fonction du prix total de la commande. La fonction retourne `0`, ou le montant des frais de port le cas échéant (type `int`). La fonction prend pour argument un `int` représentant le prix total de la commande. Des frais de port (`30`) sont applicables si le prix de la commande est strictement inférieur à 1000.

| --- | --- |
Nom	`get_shipping_costs`
Arguments	total_price (type `int`)
Exemple d'appel	`get_shipping_costs(total_price)`
Traitement	Calcule et retourne le montant des frais de port (`30` pour toute commande dont le prix total est strictement inférieur à 1000)
Exemple de retour	`0`
Type du retour	`int`

Récapitulatif (4 points)

Fonction qui affiche à l'écran:

```
User: Hubert Bonisseur de La Bath (hubert@example.com)
Order Date: 2020-01-01
Order Status:
        >> Paid
        >> Not Delivered

Products:
        >> iPhone 1 units x $999 = $999
        >> Smart TV 1 units x $1999 = $1999
        >> HDMI Cable 5 units x $12 = $60
        >> Smart Watch 1 units x $699 = $699

Total:
        >> Products: 8 units
        >> Product Price: $3757
        >> Shipping Costs: $0
        >> ORDER TOTAL: $3757
```

L'argument passé aura toujours la même structure que `order`, avec des valeurs différentes, mais toujours avec les clefs `user`, `order_details` et `products`.

| --- | --- |
Nom	`order_summary`
Arguments	order (type `dict`)
Exemple d'appel	`order_summary(order)`
Traitement	Affiche un récapitulatif de la commande à l'écran
Exemple de retour	Pas de retour (affichage uniquement, cf. ci-dessus)
Type du retour	Pas de retour (affichage uniquement, cf. ci-dessus)

Mystery Function (4 points - `mystery.py`)

Étudiez et comprennez le traitement de la fonction ci-dessous:

1. Pour chaque `#` dans le code, ajoutez le commentaire décrivant la ou les lignes qui suivent le `#`
2. Écrivez la docstring de la fonction

Remarque:

`pprint` vous sera utile si vous souhaitez afficher le résultat de la fonction

```
from pprint import pprint
from string import ascii_uppercase

def some_function(some_string):
    """[summary]

    Args:
        some_string ([type]): [description]

    Returns:
        [type]: [description]
    """

    #
    letters = []
    res = [[" "] * 10, [" "] * 10, [" "] * 10]

    #
    for letter in some_string.upper():
        #
        if letter in ascii_uppercase and letter not in letters:
            #
            letters.append(letter)

            row = -1
            col = -1

            #
            if letter in "AZERTYUIOP":
                row = 0
                col = "AZERTYUIOP".index(letter)
            elif letter in "QSDFGHJKLM":
                row = 1
                col = "QSDFGHJKLM".index(letter)
            elif letter in "WXCVBN":
                row = 2
                col = "WXCVBN".index(letter)

            #
            if row != -1 and col != -1:
                #
                res[row][col] = letter

    #
    return res
```