

Devoir Surveillé

Les Dictionnaires

M.Meyroneinc-Condyl

Exercice n°1 : notes d'examen

On modélise des résultats à un examen grâce à des dictionnaires de la forme suivante :

```
candidat_a = {'nom' : 'John' , 'ecrit' : 15, 'oral' : 13, 'projet' :  
17}  
candidat_b = {'nom' : 'Elsa' , 'ecrit' : 18, 'oral' : 17, 'projet' :  
16}  
candidat_c = {'nom' : 'Icham' , 'ecrit' : 11, 'oral' : 17, 'projet' :  
20}  
candidat_d = {'nom' : 'Eva', 'ecrit' : 3, 'oral' : 19, 'projet' :  
14}
```

!!! exemple “Exercice” test ## Question 1 : Ecrire la commande permettant d’accéder à la note de projet du candidat_a

Question 2

Ecrire une fonction `affiche_note(candidat, matiere)` ayant comme paramètres `candidat` et `matiere` et renvoyant le nom du candidat ainsi que la note de la matière choisie.

```
def affiche_note(candidat, matiere):  
    nom=candidat['nom']  
    note=candidat[matiere]  
    return nom, note
```

Tester votre fonction grâce au jeu de tests ci-dessous.

```
assert affiche_note(candidat_a, 'ecrit')==('John', 15)  
assert affiche_note(candidat_b, 'oral')==('Elsa', 17)  
assert affiche_note(candidat_c, 'projet')==('Icham', 20)
```

Compléter la fonction moyenne qui prend en paramètre un dictionnaire modélisant des résultats à un examen et renvoie la moyenne correspondante (chaque épreuve est coefficient 1).

```
def moyenne(candidat):  
    ...
```

Tester votre fonction grâce au jeu de tests ci-dessous.

```
assert moyenne(candidat_a) == 15  
assert moyenne(candidat_b) == 17  
assert moyenne(candidat_c) == 16  
assert moyenne(candidat_d) == 12
```

Question 3 :

Suite à une commission d'harmonisation, il est décidé que :

- les notes de projet sont baissées de 2 points (sans pouvoir devenir négatives),
- les notes d'oral sont baissées de 4 points (sans pouvoir devenir négatives non plus).

Compléter la fonction harmonisation ci-dessous qui prend en paramètre un dictionnaire candidat modélisant des résultats à un examen et **renvoie une copie du dictionnaire candidat dont les notes de projet et d'oral ont été modifiées comme indiqué plus haut.**

```
import copy
```

```
def harmonisation(candidat):  
    copie_candidat = copy.deepcopy(...)  
    ...  
    ...  
    return ...
```

Tester votre fonction grâce au jeu de tests ci-dessous :

```
assert harmonisation({'nom' : 'John' , 'ecrit' : 15, 'oral' : 3,  
    'projet' : 17}) ==  
{ 'nom' : 'John' , 'ecrit' : 15, 'oral' : 0, 'projet' : 15}  
assert harmonisation({'nom' : 'Elsa' , 'ecrit' : 18, 'oral' : 17,  
    'projet' : 1}) ==  
{ 'nom' : 'Elsa' , 'ecrit' : 18, 'oral' : 13, 'projet' : 0}  
assert harmonisation({'nom' : 'Icham' , 'ecrit' : 11, 'oral' : 17,  
    'projet' : 20}) ==  
{ 'nom' : 'Icham' , 'ecrit' : 11, 'oral' : 13, 'projet' : 18}
```

Exercice n°2 : parcours - montant des achats

On considère des dictionnaires mémorisant les sommes dépensées par des clients sur un site de e-commerce, **depuis leur première commande**. Ces dictionnaires sont donc mis à jour :

- dès qu'un des clients déjà inscrit sur le site, dans la catégorie concernée, effectue une nouvelle commande,
- dès qu'un client nouvellement inscrit, dans la catégorie concernée, effectue sa première commande.

Les clients sont identifiés par une référence client de sept caractères.

Voici un exemple possible de dictionnaire pour un ensemble de cinq clients :

```
achats = {'AB45112' : 235.67,  
          'JC56911' : 3496.54,  
          'P098386' : 977.41,  
          'HK53457' : 9.99,  
          'DP51673' : 7531.02}
```

Question 1 :

Compléter la fonction `nouvel_achat` :

- qui prend en paramètres :
 - un dictionnaire `dict_achats` comme évoqué ci-dessus,
 - une référence client `ref`,
 - le montant `m` d'une commande effectuée par le client.
- et **muté** le dictionnaire afin de l'actualiser avec la commande effectuée par le client.

```
def nouvel_achat(dict_achats, ref, m):  
    if ... in dict_achats. ... :  
        dict_achats[...] = ... + ...  
    else:  
        dict_achats[...] = ...
```

Tester votre fonction grâce au jeu de tests ci-dessous.

```
achats = {'AB45112' : 235,  
          'JC56911' : 3496,  
          'P098386' : 977,  
          'HK53457' : 9,  
          'DP51673' : 7531}
```

```
nouvel_achat(achats, 'P098386', 58)
```

```
nouvel_achat(achats, 'HK53457', 12)
nouvel_achat(achats, 'JC56911', 149)
nouvel_achat(achats, 'IQ72342', 1219)
```

```
assert achats == {'AB45112': 235,
                  'JC56911': 3645,
                  'P098386': 1035,
                  'HK53457': 21,
                  'DP51673': 7531,
                  'IQ72342': 1219}
```

Question 2 :

Compléter la fonction `montant_total` :

- qui prend en paramètre un dictionnaire `dict_achats` comme évoqué ci-dessus),
- et renvoie le montant total des achats des clients de cette catégorie.

```
def montant_total(dict_achats):
    total = 0
    for ... in dict_achats. ... :
        total = total + ...
    return total
```

Tester votre fonction grâce au jeu de tests ci-dessous.

```
achats = {'AB45112' : 235,
          'JC56911' : 3496,
          'P098386' : 977,
          'HK53457' : 9,
          'DP51673' : 7531}
assert montant_total(achats) == 12248
```

```
achats = {'AB45112' : 32,
          'JC56911' : 511,
          'P098386' : 118,
          'HK53457' : 9,
          'DP51673' : 2,
          'BC22222' : 12345}
assert montant_total(achats) == 13017
```

```
achats = {}  
assert montant_total(achats) == 0
```

Exercice n°3 : candidats en commun

Deux associées diffusent régulièrement des appels d'offre.

À chaque fois elles attendent de recevoir 15 dossiers, numérotés de 1 à 15.

Chacune des deux associées note alors dans une liste les numéros des dossiers qu'elle présélectionne.

Dans un second temps elles sélectionnent les dossiers qui sont présents **à la fois** dans les deux listes pour organiser des entretiens avec les responsables.

Par exemple :

- l'associée A présélectionne [2, 5, 7, 9, 14, 15],
- l'associée B présélectionne [1, 3, 5, 9, 12, 13, 15],
- les deux associées sélectionnent finalement [5, 9, 15] pour les entretiens.

Question 1:

Compléter la fonction `selectionner_etape_0` qui :

- prend en paramètre une liste contenant des entiers compris entre 1 et 15,
- renvoie un dictionnaire ayant pour clefs les entiers de 1 à 15 et pour valeurs 0 ou 1 selon que la clef correspondante est présente ou pas dans la liste.

On pourra consulter le jeu de tests si besoin.

```
def selectionner_etape_0(preselection):  
    dico = {i:0 for i in range(1, 16)}  
    for ... in preselection :  
        dico[...] = ...  
    return ...
```

Tester votre fonction grâce au jeu de tests ci-dessous.

```
preselection = [3, 5, 12, 15]  
assert selectionner_etape_0(preselection) ==  
{1: 0, 2: 0, 3: 1, 4: 0, 5: 1, 6: 0, 7: 0, 8: 0, 9: 0, 10: 0, 11: 0,  
 12: 1, 13: 0, 14: 0, 15: 1}
```

```
preselection = [1, 2, 3, 4]  
assert selectionner_etape_0(preselection) ==  
{1: 1, 2: 1, 3: 1, 4: 1, 5: 0, 6: 0, 7: 0, 8: 0, 9: 0, 10: 0, 11: 0,  
 12: 0, 13: 0, 14: 0, 15: 0}
```

```

preselection = []
assert selectionner_etape_0(preselection) ==
{1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 0, 10: 0, 11: 0,
 12: 0, 13: 0, 14: 0, 15: 0}

preselection = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
assert selectionner_etape_0(preselection) ==
{1: 1, 2: 1, 3: 1, 4: 1, 5: 1, 6: 1, 7: 1, 8: 1, 9: 1, 10: 1, 11: 1,
 12: 1, 13: 1, 14: 1, 15: 1}

```

Question 2 :

Compléter la fonction `selection_finale` qui :

- prend en paramètres deux listes contenant des entiers entre 1 et 15,
- renvoie une liste contenant les entiers présents **à la fois** dans les deux listes.

L'idée est de créer un dictionnaire comme ci-dessus pour chacune des deux préselections puis de parcourir les entiers de 1 à 15 et de vérifier pour chacun si les deux valeurs associées dans les deux dictionnaires sont égales à 1.

On pourra consulter le jeu de tests si besoin.

```

def selection_finale(preselection_a, preselection_b):
    dico_a = selectionner_etape_0(preselection_a)
    dico_b = selectionner_etape_0(preselection_b)
    selection_finale = []
    for ... in range(1, 16):
        if ...[...] == ... and ...[...] == ... :
            selection_finale.append(...)
    return selection_finale

```

Tester votre fonction grâce au jeu de tests ci-dessous.

```

p_a = [1, 7, 9, 14, 15]
p_b = [3, 7, 11, 12, 14]
assert selection_finale(p_a, p_b) == [7, 14]

p_a = [1, 2, 3, 6, 7, 9, 11, 12, 14, 15]
p_b = [2, 3, 4, 6, 7, 8, 9, 11, 12, 13]
assert selection_finale(p_a, p_b) == [2, 3, 6, 7, 9, 11, 12]

p_a = [1, 3, 5, 7, 9, 11, 13, 15]
p_b = [2, 4, 6, 8, 10, 12, 14]
assert selection_finale(p_a, p_b) == []

```

```
p_a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]  
p_b = [2, 4, 6, 8, 10, 12, 14]  
assert selection_finale(p_a, p_b) == p_b
```