TP_Recherche_Tris_Correction

March 11, 2021

1 Import des modules et outils

[4]:
```
import csv


def lecture_csv(fichier, delimiteur):
    """
    Paramètre : un chemin vers un fichier CSV
    Valeur renvoyée : un tableau de dictionnaires, extraction de la table␣
 ↪contenue dans le fichier
    """
    f = open(fichier, mode = 'r', encoding = 'utf8', newline='')
    reader = csv.DictReader(f, delimiter = delimiteur)  #création de l'objet␣
 ↪reader
    table = [dict(enregistrement) for enregistrement in reader]
    f.close()
    return table

def ecriture_csv(table, fichier, delimiteur):
    """
    Paramètre :
        un chemin vers un fichier CSV
        une table comme tableau de dictionnaires partageant les mêmes clefs, de␣
 ↪valeurs str
    Valeur renvoyée :
        None, écrit table dans fichier avec Dictriter du  module CSV
    """
    g = open(fichier, mode = 'w', encoding = 'utf8', newline='')
    attributs = list(table[0].keys())
    writer = csv.DictWriter(g, delimiter = delimiteur, fieldnames = attributs)␣
 ↪#création de l'objet writer
    writer.writeheader()   #écriture des attibuts
    for enregistrement in table:
        writer.writerow(enregistrement)  #écriture des enregistrements
    g.close()
```

2 Exemple 1 : table clients

2.1 Importation de la table stockée dans clients.csv dans un tableau de dictionnaires

[11]:
```
table_clients = lecture_csv('clients.csv', ',')
```

[3]:
```
# postconditions
def test_import_table_clients(table):
    assert len(table) == 50000
    assert table[0] == {'nom': 'Gomes',
                        'prénom': 'Brigitte',
                        'email': 'bgomes@laposte.net',
                        'département': '79',
                        'naissance': '1960-08-21',
                        'visites': '57',
                        'dépenses': '2371.58'}
    print("tests d'importation réussis pour table_clients")

# décommenter pour tester
test_import_table_clients(table_clients)
```

```
tests d'importation réussis pour table_clients
```

2.2 Recherches sur une seule ligne

2.2.1 Recherche 1 : recherche linéaire en fonction d'un attribut

[16]:
```
def recherche_attribut(table, attribut, valeur):
    """Paramètres :
        table un tableau de dictionnaires table  de clients.csv
        attribut de type str, valeur du type d'attribut dans table
    Valeur renvoyée:
        Un booléen indiquant si table contient un enregistrement e
        tel que e[attribut] == valeur"""
    for enregistrement in table:
        if enregistrement[attribut] == valeur:
            return True
    return False
```

[18]:
```
def test_recherche_attribut_table_clients(table):
    assert recherche_attribut(table, "prénom", "Frédéric") == True
    assert recherche_attribut(table, "prénom", "Nestor") == False
    assert recherche_attribut(table, "naissance", "1975-02-28") == False
    assert recherche_attribut(table, "naissance", "1975-05-13") == True
    print("tests de recherche d'attributs réussis pour table_clients")

# décommenter pour tester
```

```
test_recherche_attribut_table_clients(table_clients)
```

```
tests de recherche d'attributs réussis pour table_clients
```

2.2.2 Recherche 2 : recherche linéaire d'une valeur en fonction d'une condition sur plusieurs attributs

[4]:
```
def recherche_attributs_et(table, attribut1, valeur1, attribut2, valeur2):
    """Paramètres :
        table un tableau de dictionnaires table  de clients.csv
        attribut1 de type str, valeur1 du type d'attribut1
        attribut2 de type str, valeur2 du type d'attribut2
    Valeur renvoyée:
        Un booléen indiquant si table contient un enregistrement e
        tel que e[attribut1] == valeur1 et e[attribut2] == valeur2 """
    for enregistrement in table:
        if enregistrement[attribut1] == valeur1 and enregistrement[attribut2]␣
↪== valeur2:
            return True
    return False

def test_recherche_attributs_et(table):
    assert recherche_attributs_et(table, "prénom", "Frédéric", "département",␣
↪"69") == True
    print("tests de recherche de conjonction d'attributs réussis pour␣
↪table_clients")

# décommenter pour tester
test_recherche_attributs_et(table_clients)
```

```
tests de recherche de conjonction d'attributs réussis pour table_clients
```

[7]:
```
def recherche_attributs_condition(table, tab_attribut, tab_valeur, condition):
    """Paramètres :
        table un tableau de dictionnaires table  de clients.csv
        tab_attribut un tableau de noms d'attributs
        tab_valeur un tableau de valeurs
        condition une condition booléenne
    Valeur renvoyée : un booléen : True si au moins un enregistrement
    de table  vérifie condition(attribut, valeur) et False sinon"""
    for enregistrement in table:
        if condition(enregistrement, tab_attribut, tab_valeur):
            return True
    return False

def condition_and(enregistrement, tab_attribut, tab_valeur):
    """Renvoie True si tous les attributs listés dans tab_attribut
```

```
    ont la valeur correspondante dans tab_valeur et False sinon"""
    assert len(tab_attribut) == len(tab_valeur)
    for  k in range(len(tab_attribut)):
        if enregistrement[tab_attribut[k]] != tab_valeur[k]:
            return False
    return True


def condition_or(enregistrement, tab_attribut, tab_valeur):
    """Renvoie True si au moins un attribut listé dans tab_attribut
    a la valeur correspondante dans tab_valeur et False sinon"""
    assert len(tab_attribut) == len(tab_valeur)
    for  k in range(len(tab_attribut)):
        if enregistrement[tab_attribut[k]] == tab_valeur[k]:
            return True
    return False


#postconditions
def test_recherche_attributs_condition(table):
    assert recherche_attributs_condition(table_clients, ["prénom",␣
 ↪"département"], ["Frédéric", "69"], condition_and) == True
    assert recherche_attributs_condition(table_clients, ["prénom",␣
 ↪"département"], ["Frédéric", "69"], condition_or) == True
    assert recherche_attributs_condition(table_clients, ["prénom",␣
 ↪"département"], ["Nestor", "96"], condition_or) == False
    print("tests de recherche de condition sur attributs réussis pour␣
 ↪table_clients")

test_recherche_attributs_condition(table_clients)
```

```
tests de recherche de condition sur attributs réussis pour table_clients
```

2.3 Agrégations (requêtes sur plusieurs lignes et calcul d'une valeur)

2.3.1 Comptage d'occurences

[12]:
```
def nombre_departement(table, departement):
    """Paramètres :
        table un tableau de dictionnaires, table  de clients.csv
        departement de type str, un numéro de département
    Valeur renvoyée :
        Nombre d'occurences de departement dans table"""
    compteur = 0
    for enregistrement in table:
        if enregistrement["département"] == departement:
            compteur = compteur + 1
```

```
    return compteur

def nombre_occurences(table, attribut, valeur):
    """Paramètres :
        table un tableau de dictionnaires, table  de clients.csv
        attribut de type str, valeur du type d'attribut dans table
    Valeur renvoyée :
        Nombre d'occurences d'attribut avec valeur dans table"""
    compteur = 0
    for enregistrement in table:
        if enregistrement[attribut] == valeur:
            compteur = compteur + 1
    return compteur

def test_nombre_occurences(table):
    assert nombre_occurences(table_clients, "département", "69") == 481
    assert nombre_occurences(table_clients, "prénom", "Frédéric") == 235
    print("tests de recherche de nombre d'occurences réussis pour␣
 ↪table_clients")

# postcondition
assert nombre_departement(table_clients,  "69") == 481
test_nombre_occurences(table_clients)
```

```
tests de recherche de nombre d'occurences réussis pour table_clients
```

Somme et moyenne

[8]:
```
def moyenne_visites(table):
    """Paramètres :
        table un tableau de dictionnaires, table  de clients.csv
    Valeur renvoyée :
        Moyenne des visites par enregistrement de type float"""
    somme = 0
    taille = 0
    for enregistrement in table:
        somme = int(enregistrement["visites"]) + somme
        taille = taille + 1
    return somme / taille

# postcondition
assert moyenne_visites(table_clients) == 76.2807
```

2.3.2 Recherche d'extremum

[13]:
```
def minimum_visites(table):
    """Paramètres :
        table un tableau de dictionnaires, table  de clients.csv
    Valeur renvoyée :
        nombre minimum de visites de type int"""
    min_visites = int(table[0]['visites'])
    for enregistrement in table[1:]:
        nb_visites = int(enregistrement["visites"])
        if nb_visites < min_visites:
            min_visites = nb_visites
    return min_visites

assert minimum_visites(table_clients)  == 2
```

[14]:
```
def departement_max_occurence(table):
    """
    Paramètre : table sous forme de tableau de dictionnaires
    Valeur renvoyée : tuple formé du nombre d'occurences maximal parmi les␣
 ↪départements
    et du tableau des  départements réalisant ce maximum
    """
    histo = {table[0]['département'] : 1}
    max_occurence = 1
    tab_depart_max = []
    for enregistrement in table[1:]:
        depart = enregistrement['département']
        if depart not in histo:
            histo[depart] = 1
        else:
            histo[depart] = histo[depart] + 1
        if histo[depart] > max_occurence:
            max_occurence = histo[depart]
            tab_depart_max = [depart]
        elif histo[depart] == max_occurence:
            tab_depart_max.append(depart)
    return max_occurence, tab_depart_max

assert departement_max_occurence(table_clients) == (547, ['59'])
```

Sélection de lignes

[8]:
```
def selection_departement(table, departement):
    """
    Paramètres :
        table une table sous forme de tableau de dictionnaires
        department une chaine de caractères représentant un département
```

```
    Valeur renvoyée :
        tableau de dictionnaires contenant les enregistrement de table dont␣
 ↪l'attribut "département"
        a la valeur passée en paramètre
    """
    return [enregistrement for enregistrement in table if␣
 ↪enregistrement["département"] == departement]

def selection_depart_visites_min(table, departement, visites_min):
    """
    Paramètres :
        table une table sous forme de tableau de dictionnaires
        department une chaine de caractères représentant un département
        visites_min un entier naturel
    Valeur renvoyée :
        tableau de dictionnaires contenant les enregistrement de table dont␣
 ↪l'attribut "département"
        a la valeur passée en paramètre et l'attribut visites est >= visites_min
    """
    return [enregistrement for enregistrement in table if␣
 ↪enregistrement["département"] == departement
                                                        and␣
 ↪int(enregistrement["visites"]) >= visites_min]

assert selection_departement(table_clients, "69")[0]['email'] == 'nnguyen@noos.
 ↪fr' and len(selection_departement(table_clients, "69")) == 481
assert len(selection_depart_visites_min(table_clients, "69", 100)) == 171
```

Projection sur des colonnes

[9]:
```
def projection_visites(table):
    """
    Paramètres :
        table une table sous forme de tableau de dictionnaires
    Valeur renvoyée :
        tableau des valeurs des attributs "visites" pour les enregistrements de␣
 ↪table
        avec conversion des nombres de visites en entiers
    """
    return [ int(enregistrement["visites"]) for enregistrement in table ]

def selection_departement_projection_visites(table, departement):
    """
    Paramètres :
        table une table sous forme de tableau de dictionnaires
    Valeur renvoyée :
```

```
        tableau des valeurs des attributs "visites" pour les enregistrements de␣
↪table du département
        passé en paramètre avec conversion des nombres de visites en entiers
    """
    return [ int(enregistrement["visites"]) for enregistrement in table if␣
↪enregistrement['département'] == departement]

assert projection_visites(table_clients)[:10] == [57, 145, 67, 131, 76, 52, 65,␣
↪3, 101, 18]
assert selection_departement_projection_visites(table_clients, "69")[:10] ==␣
↪[43, 52, 127, 53, 41, 117, 31, 86, 107, 145]
```

[86]:
```
def projection_departement_age(table, annee):
    """
    Paramètres :
        table une table sous forme de tableau de dictionnaires
    Valeur renvoyée :
        tableau de dictionnaires avec deux attributs le département et l'âge du␣
↪client
        calculé à partir de sa date de naissance.
    """
    return [{'département' : enreg['département'], 'âge' : annee -␣
↪int(enreg['naissance'][:4])}
            for enreg in table]

assert projection_departement_age(table_clients, 2021)[:3] == [{'département':␣
↪'79', 'âge': 61}, {'département': '10', 'âge': 27},
 {'département': '73', 'âge': 20}]
```

2.4 Tris

Tris selon un attribut

[87]:
```
def clef_departement(enreg):
    return enreg['département']

def clef_visites(enreg):
    return int(enreg['visites'])

def clef_departement_visites(enreg):
    return (enreg['département'], int(enreg['visites']))

# tri par département dans l'ordre alphabétique croissant
table_tri_departement = sorted(table_clients, key = clef_departement)

assert table_tri_departement[:2] == [{'nom': 'Marie',
  'prénom': 'Denise',
```

```
  'email': 'dmarie@sfr.fr',
  'département': '01',
  'naissance': '1984-11-04',
  'visites': '80',
  'dépenses': '5881.15'},
 {'nom': 'Lebon',
  'prénom': 'Luc',
  'email': 'llebon@free.fr',
  'département': '01',
  'naissance': '1972-02-07',
  'visites': '58',
  'dépenses': '1610.89'}]

# tri par visites dans l'ordre décroissant

table_tri_visites_decroissant = sorted(table_clients, key = clef_visites,␣
 ↪reverse = True)
assert table_tri_visites_decroissant[:2]  == [{'nom': 'Ribeiro',
  'prénom': 'Chantal',
  'email': 'cribeiro@dbmail.com',
  'département': '43',
  'naissance': '1961-03-10',
  'visites': '151',
  'dépenses': '9556.76'},
 {'nom': 'Lebreton',
  'prénom': 'Zoé',
  'email': 'zlebreton@wanadoo.fr',
  'département': '05',
  'naissance': '1946-05-12',
  'visites': '151',
  'dépenses': '8118.48'}]

# tri lexicographique par  département puis visites dans l'ordre croissant

table_tri_dep_vis_croissant = sorted(table_clients, key =␣
 ↪clef_departement_visites)
assert table_tri_dep_vis_croissant[:2] == [{'nom': 'Jacquet',
  'prénom': 'Benjamin',
  'email': 'bjacquet@orange.fr',
  'département': '01',
  'naissance': '1985-07-05',
  'visites': '2',
  'dépenses': '257.71'},
 {'nom': 'Gros',
  'prénom': 'Danielle',
  'email': 'dgros@tele2.fr',
  'département': '01',
```

```
  'naissance': '1962-07-20',
  'visites': '2',
  'dépenses': '265.76'}]

# comme le précédent mais en chainant les tris

table_tri_dep_vis_croissant2 = sorted(sorted(table_clients, key =␣
 ↪clef_visites), key = clef_departement)
assert table_tri_dep_vis_croissant2 == table_tri_dep_vis_croissant


# si l'ordre du tri diffère selon les attributs, on ne peut pas procéder par␣
 ↪ordre lexicographique
# il faut chainer les tris
# par exemple par département croissant et nombre de visites décroissant
table_tri_dep_crois_vis_decrois = sorted(sorted(table_clients, key =␣
 ↪clef_visites, reverse = True), key = clef_departement)
assert table_tri_dep_crois_vis_decrois[:2] == [{'nom': 'Dumont',
  'prénom': 'Bertrand',
  'email': 'bdumont1@sfr.fr',
  'département': '01',
  'naissance': '1957-07-15',
  'visites': '151',
  'dépenses': '8531.65'},
 {'nom': 'Legros',
  'prénom': 'Emmanuelle',
  'email': 'elegros@voila.fr',
  'département': '01',
  'naissance': '1998-04-18',
  'visites': '151',
  'dépenses': '899.3'}]
```

2.4.1 Application du tri : mise à jour d'une table triée par recherche dichotomique

[9]:
```
from copy import deepcopy #pour réaliser une copie de table

def recherche_dicho_croissant(element, table, attribut):
    """Paramètres :
         table un tableau de dictionnaires
         attribut de type str
         element une valeur possible pour l'attribut
    Valeur renvoyée : index de la valeur element de attribut dans table"""
    debut = 0
    fin = len(table) - 1
    while fin - debut >= 0:
        milieu = (debut + fin) // 2
```

```
        if table[milieu][attribut] < element:
            debut = milieu + 1
        elif table[milieu][attribut] > element:
            fin = milieu - 1
        else:
            return milieu
    return None

def clef_email(enreg):
    return enreg['email']

def maj_depenses_table(table, transactions):
    """Paramètres : table et transactions deux  tables sous forme de tableaux␣
 ↪de dictionnaires
    Valeur renvoyée :
        table_tri un tableau de dictionnaires mise à jour des attributs␣
 ↪'visites' et 'dépenses'
        de table par les valeurs de transactions"""
    table_tri = sorted(deepcopy(table), key = clef_email)
    table_cible = []
    for enreg in transactions:
        index_email = recherche_dicho_croissant(enreg['email'], table_tri,␣
 ↪'email')
        if index_email is not None:
            table_tri[index_email]['visites'] =␣
 ↪str(int(table_tri[index_email]['visites']) + 1)
            table_tri[index_email]['dépenses'] =␣
 ↪str(float(table_tri[index_email]['dépenses']) + float(enreg['dépenses']))
    return table_tri

def test_application_tri():
    table_clients = lecture_csv('clients.csv', ',')
    transactions = lecture_csv('transactions.csv', ',')
    table_tri = maj_depenses_table(table_clients, transactions)
    assert [enreg for enreg in table_clients if enreg['email'] ==␣
 ↪'araynaud@tiscali.fr'] == [{'nom': 'Raynaud',
    'prénom': 'Alfred',
    'email': 'araynaud@tiscali.fr',
    'département': '13',
    'naissance': '1961-06-18',
    'visites': '119',
    'dépenses': '3876.63'}]
    assert [enreg for enreg in table_tri if enreg['email'] == 'araynaud@tiscali.
 ↪fr'] == [{'nom': 'Raynaud',
    'prénom': 'Alfred',
    'email': 'araynaud@tiscali.fr',
```

```
    'département': '13',
    'naissance': '1961-06-18',
    'visites': '120',
    'dépenses': '3979.36'}]
    ecriture_csv(table_tri, 'clients_maj.csv', ',')
    print("Tests réussis")

#Décommenter pour exécuter le test
test_application_tri()
```

```
Tests réussis
```

3 Exemple 2 : table 'countries'

[11]:
```
table_pays = lecture_csv('countries.csv', ';')
```

[12]:
```
assert table_pays[:2] == [{'iso': 'AD',
  'name': 'Andorra',
  'area': '468.0',
  'population': '84000',
  'continent': 'EU',
  'currency_code': 'EUR',
  'currency_name': 'Euro',
  'capital': '6'},
 {'iso': 'AE',
  'name': 'United Arab Emirates',
  'area': '82880.0',
  'population': '4975593',
  'continent': 'AS',
  'currency_code': 'AED',
  'currency_name': 'Dirham',
  'capital': '21'}]
```

[35]:
```
def nombre_europe(table):
    """Paramètre : table de countries.csv
    Valeur renvoyée : compteur de type int représentant le nombre de pays du␣
 ↪continent européen
    """
    compteur = 0
    for enregistrement in table:
        if enregistrement['continent'] == 'EU':
            compteur = compteur + 1
    return compteur

assert nombre_europe(table_pays) == 52
```

[36]:
```
def selection_europe(table):
    """Paramètre : table de countries.csv
    Valeur renvoyée : table des enregistrements des pays du continent européen
    """
    return [enregistrement for enregistrement in table if␣
↪enregistrement['continent'] == 'EU']

europe = selection_europe(table_pays)
assert len(europe) == 52 and europe[0]['name'] == 'Andorra'
```

[122]:
```
def selection_europe_non_euro(table):
    """Paramètre : table de countries.csv
    Valeur renvoyée : table des enregistrements des pays du continent européen␣
↪qui n'ont pas pour monnaie l'euro
    """
    return [enregistrement for enregistrement in table if␣
↪enregistrement['continent'] == 'EU'
                and enregistrement['currency_code'] != 'EUR']

europe_non_euro = selection_europe_non_euro(table_pays)
assert len(europe_non_euro) == 27 and europe_non_euro[0]['name'] == 'Albania'
```

[37]:
```
def projection_aire(table):
    """Paramètre : table de countries.csv
    Valeur renvoyée : tableau des aires (type float) de tous les␣
↪enregistrements"""
    return [float(enregistrement['area']) for  enregistrement in table ]

assert  projection_aire(table_pays)[:5]  == [468.0, 82880.0, 647500.0, 443.0,␣
↪102.0]
```

[42]:
```
def projection_pays_densite(table):
    """Paramètre : table de countries.csv
    Valeur renvoyée : nouvelle table avec deux attributs 'pays' et 'densité' de␣
↪population"""
    return [ {'pays' : enreg['name'], 'densité' : float(enreg['population']) /␣
↪float(enreg['area'])}
            for enreg in table]

assert projection_pays_densite(table_pays)[:3] == [{'pays': 'Andorra',␣
↪'densité': 179.48717948717947},
 {'pays': 'United Arab Emirates', 'densité': 60.033699324324324},
 {'pays': 'Afghanistan', 'densité': 44.974959073359074}]
```

[47]:
```
def maximum_densite(table):
    """Paramètre : table de countries.csv
```

```
    Valeur renvoyée : tuple avec le nom du pays de densité maximale de␣
↪population
    et cette densité maximale"""
    table_densite = projection_pays_densite(table)
    pays_max = table_densite[0]['pays']
    densite_max = table_densite[0]['densité']
    for enreg in table_densite[1:]:
        if enreg['densité'] > densite_max:
            densite_max = enreg['densité']
            pays_max = enreg['pays']
    return (pays_max, densite_max)

assert maximum_densite(table_pays) == ('Monaco', 16905.128205128207)
```

[51]:
```
def population_par_continent(table):
    """Paramètre : table de countries.csv
    Valeur renvoyée : dictionnaire de clefs les identifiants des continents
    et de valeurs les populations cumulées des pays leur appartenant"""
    pop_continent = dict()
    for enregistrement in table:
        continent = enregistrement['continent']
        population = int(enregistrement['population'])
        if continent not in pop_continent:
            pop_continent[continent] = population
        else:
            pop_continent[continent] += population
    return pop_continent

assert population_par_continent(table_pays) == {'EU': 740017414,
 'AS': 4119426856,
 'NA': 539886359,
 'AF': 1018849428,
 'SA': 400143568,
 'OC': 36066083}
```

[130]:
```
## top 20 des pays les plus densément peuplés

def densite_max_top5(table):
    """Paramètre : table de countries.csv
    Valeur renvoyée : table avec les noms et les densités des 5 pays les plus
    densément peuplées dans l'ordre décroissant des densités de population"""

    def clef_densite(enreg):
        return float(enreg['densité'])

    table_densite = projection_pays_densite(table_pays)
```

```
    table_densite_tri_decroissant = sorted(table_densite, key = clef_densite,␣
↪reverse = True)
    return table_densite_tri_decroissant[:5]

assert densite_max_top5(table_pays) == [{'pays': 'Monaco', 'densité': 16905.
↪128205128207},
 {'pays': 'Singapore', 'densité': 6786.5872672152445},
 {'pays': 'Hong Kong', 'densité': 6317.478021978022},
 {'pays': 'Gibraltar', 'densité': 4289.846153846154},
 {'pays': 'Vatican', 'densité': 2093.181818181818}]
```