Python Fundamentals

1. Reading & Writing CSV Files

Change the directory to data file folder.

In [209]:

```
%cd D:\Data Science\GitHub\Python Learning\Python-for-Data-Science\Data Files\Python Le
arning
```

```
D:\Data Science\GitHub\Python Learning\Python-for-Data-Science\Data Files
\Python Learning
```

Import the file as a **list of dictionaries**.

In [210]:

```
import csv

%precision 2

with open('expense_report.csv') as dat0:
    pay_dat = list(csv.DictReader(dat0))


pay_dat[:3]
```

Out[210]:

```
[OrderedDict([('TRXN_MONTH', 'Mar-19'),
              ('TRXN_DT', '01/03/2019'),
              ('TRXN_AMT', '61.2'),
              ('MERCHT_NAME', 'VIRGIN ACTIVE'),
              ('CATG', 'Entertainment')]),
 OrderedDict([('TRXN_MONTH', 'Mar-19'),
              ('TRXN_DT', '04/03/2019'),
              ('TRXN_AMT', '1.31'),
              ('MERCHT_NAME', 'LOON FUNG LIMITED'),
              ('CATG', 'Glossary')]),
 OrderedDict([('TRXN_MONTH', 'Mar-19'),
              ('TRXN_DT', '04/03/2019'),
              ('TRXN_AMT', '16.89'),
              ('MERCHT_NAME', 'LOON FUNG LIMITED'),
              ('CATG', 'Glossary')])]
```

In [211]:

```
# Check length of the dictionary.
len(pay_dat)
```

Out[211]:

```
1207
```

Return the keys of first element in *pay_dat*.

In [212]:

```
pay_dat[0].keys()
```

Out[212]:

```
odict_keys(['TRXN_MONTH', 'TRXN_DT', 'TRXN_AMT', 'MERCHT_NAME', 'CATG'])
```

Calculate the **average transaction** amount in the spreadsheet.

In [213]:

```
sum(float(i['TRXN_AMT']) for i in pay_dat)/len(pay_dat)
```

Out[213]:

```
34.08
```

Create a **set** of categories.

In [214]:

```
exp_cat = set(i['CATG'] for i in pay_dat)
exp_cat
```

Out[214]:

```
{'Business Expense',
 'Cash Out',
 'Eating Out',
 'Electronic Product',
 'Entertainment',
 'Filght Ticket',
 'Finance',
 'Furniture',
 'Glossary',
 'Gym & Sports',
 "Jinlei's Credit Card",
 'Online Course Learning',
 'Other',
 'Property',
 'RMB (Milk Powder)',
 'Rent',
 'Shopping',
 'Transport',
 'Utility Bill'}
```

Calculate **average transaction amount** for each expense cateogy.

In [215]:

```
Ave_TrxnAmt_Cat = []

for cat in exp_cat:
    exp_sum = 0
    exp_count = 0

    for exp_amt in pay_dat:
        if exp_amt['CATG'] == cat:
            exp_sum += float(exp_amt['TRXN_AMT'])
            exp_count += 1
    Ave_TrxnAmt_Cat.append((cat,exp_sum/exp_count))
```

Sort the output list by ***lengh of Category Name***.
Here we need to use *.sort(key = ..)*.

In [216]:

```
Ave_TrxnAmt_Cat.sort(key = lambda x: len(x[0]))
Ave_TrxnAmt_Cat
```

Out[216]:

```
[('Rent', 447.00),
 ('Other', 14.09),
 ('Finance', 100.00),
 ('Glossary', 9.70),
 ('Property', 519.74),
 ('Shopping', 98.73),
 ('Cash Out', 45.00),
 ('Furniture', 268.67),
 ('Transport', 8.76),
 ('Eating Out', 12.07),
 ('Utility Bill', 96.47),
 ('Gym & Sports', 34.29),
 ('Entertainment', 38.08),
 ('Filght Ticket', 745.00),
 ('Business Expense', 57.76),
 ('RMB (Milk Powder)', 172.00),
 ('Electronic Product', 249.00),
 ("Jinlei's Credit Card", 1075.00),
 ('Online Course Learning', 30.00)]
```

2. Python Dates and Times

Frequently used packages in Python for dates & time manipulations.

In [217]:

```
import datetime as dt
import time as tm
```

Check the time stamp of current moment.

In [218]:

```
tm.time()
```

Out[218]:

```
1581261482.79
```

Extract year, month, date etc from the timestamp.

In [219]:

```
dtnow = dt.datetime.fromtimestamp(tm.time())
dtnow
```

Out[219]:

```
datetime.datetime(2020, 2, 9, 15, 18, 2, 798541)
```

In [220]:

```
type(dtnow)
```

Out[220]:

```
datetime.datetime
```

In [221]:

```
delta = dt.timedelta(days = 100)
type(delta)
```

Out[221]:

```
datetime.timedelta
```

In [222]:

```
today = dt.date.today()
```

In [223]:

```
today > today - delta
```

Out[223]:

```
True
```

3. Advanced Python Objects, map()

An example of class.

In [224]:

```
class Person:
    department = 'School of Information'

    def set_name(self, new_names):
        self.name = new_names
    def set_location(self, new_location):
        self.location = new_location
```

- **map()** function

In [225]:

```
store1 = [10.00, 11.00, 90.43, 2.83]
store2 = [16.00, 12.00, 63.33, 1.38]
cheapest = map(min, store1, store2)
cheapest
```

Out[225]:

```
<map at 0x29bed1f9d68>
```

It does not return the **acutal output** of the calculation.

In [226]:

```
for i in cheapest:
    print(i)
```

```
10.0
11.0
63.33
1.38
```

An example to apply **map()** function:

In [227]:

```
people = ['Dr. Christopher Brooks',
          'Dr. Kevyn Collins-Thompson',
          'Dr. VG Vinod Vydiswaran',
          'Dr. Daniel Romero']

def split_title_surname (person):
    return person.split(' ')[0] + person.split(' ')[-1]

list(map(split_title_surname, people))
```

Out[227]:

```
['Dr.Brooks', 'Dr.Collins-Thompson', 'Dr.Vydiswaran', 'Dr.Romero']
```

4. Advanced Python Lambda and List Comprehensions

- **Lambda** example:

In [228]:

```
my_function = lambda a, b, c: a + b + c
my_function(1,2,3)
```

Out[228]:

```
6
```

- **List Comprehension** example:

Commen method to print list of even numbers

In [229]:

```
my_list = []
for number in range(0,10):
    if number % 2 == 0:
        my_list.append(number)
my_list
```

Out[229]:

```
[0, 2, 4, 6, 8]
```

Return the same result by list comprehension

In [230]:

```
my_list = [number for number in range(0,10) if number % 2 ==0]
my_list
```

Out[230]:

```
[0, 2, 4, 6, 8]
```

5. Numpy

Import the package.

In [231]:

```
import numpy as np
```

5.1 Creating Arrays

In [232]:

```
x = np.array([1, 2, 3])
y = np.array([4, 5, 6])
m = np.array([[7, 8, 9],[10, 11, 12]])
m
```

Out[232]:

```
array([[ 7,  8,  9],
       [10, 11, 12]])
```

Check the dimension of an array.

In [233]:

```
m.shape
```

Out[233]:

```
(2, 3)
```

Create a equally spaced 1-dimensional array.

In [234]:

```
n = np.arange(0, 30, 2)
n
```

Out[234]:

```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28])
```

Reshape the 1-dimenional array to be 3-by-5 array.

In [235]:

```
n = n.reshape(3, 5)
n
```

Out[235]:

```
array([[ 0,  2,  4,  6,  8],
       [10, 12, 14, 16, 18],
       [20, 22, 24, 26, 28]])
```

Create a 1-dimensional array by defining number of elements.

In [236]:

```
m = np.linspace(0, 4, 9)
m = m.resize(3, 3)
m
```

5.2 Some useful functions

In [237]:

```
np.ones((3, 3))
```

Out[237]:

```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

In [238]:

```
np.zeros((3, 3))
```

Out[238]:

```
array([[0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.]])
```

In [239]:

```
np.eye(3)
```

Out[239]:

```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

In [240]:

```
np.diag([1,2,3,4])
```

Out[240]:

```
array([[1, 0, 0, 0],
       [0, 2, 0, 0],
       [0, 0, 3, 0],
       [0, 0, 0, 4]])
```

5.3 Array Manipulation

- **Repeat Array Elements**

In [241]:

```
np.array([1, 2, 3] * 3)
```

Out[241]:

```
array([1, 2, 3, 1, 2, 3, 1, 2, 3])
```

In [242]:

```
np.repeat([1,2,3], 3)
```

Out[242]:

```
array([1, 1, 1, 2, 2, 2, 3, 3, 3])
```

- **Stack Arrays**

In [243]:

```
np.vstack([[1, 1, 1], [3, 3, 3]])
```

Out[243]:

```
array([[1, 1, 1],
       [3, 3, 3]])
```

In [244]:

```
np.hstack([[1, 1, 1], [3, 3, 3]])
```

Out[244]:

```
array([1, 1, 1, 3, 3, 3])
```

5.4 Array Operation

- Element-wise Operation

In [245]:

```
print(x)
print(y)
x + y
```

```
[1 2 3]
[4 5 6]
```

Out[245]:

```
array([5, 7, 9])
```

In [246]:

```
x * y
```

Out[246]:

```
array([ 4, 10, 18])
```

In [247]:

```
x**2
```

Out[247]:

```
array([1, 4, 9], dtype=int32)
```

- Dot Product

In [248]:

```
x.dot(y)
```

Out[248]:

```
32
```

- Transpose of Array

In [249]:

```
z = np.array([x, x**2])
z.T
```

Out[249]:

```
array([[1, 1],
       [2, 4],
       [3, 9]])
```

- Change Data Type of Elements

In [250]:

```
print(z.dtype)
print(z.astype('f').dtype)
```

```
int32
float32
```

5.5 Some Useful Methods in Array

In [251]:

```
a = np.array([-4, -2, 1, 4, 8])
```

In [252]:

```
a.sum()
```

Out[252]:

```
7
```

In [253]:

```
a.max()
```

Out[253]:

```
8
```

In [254]:

```
a.min()
```

Out[254]:

```
-4
```

In [255]:

```
a.mean()
```

Out[255]:

```
1.4
```

In [256]:

```
a.std()
```

Out[256]:

```
4.2708313008125245
```

In [257]:

```
a.argmax()
```

Out[257]:

```
4
```

In [258]:

```
a.argmin()
```

Out[258]:

```
0
```

5.6 Copy of Numpy Array

In [259]:

```
r = np.arange(36)
r.resize(6,6)
r
```

Out[259]:

```
array([[ 0,  1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10, 11],
       [12, 13, 14, 15, 16, 17],
       [18, 19, 20, 21, 22, 23],
       [24, 25, 26, 27, 28, 29],
       [30, 31, 32, 33, 34, 35]])
```

In [260]:

```
r_copy = r.copy()
r_copy
```

Out[260]:

```
array([[ 0,  1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10, 11],
       [12, 13, 14, 15, 16, 17],
       [18, 19, 20, 21, 22, 23],
       [24, 25, 26, 27, 28, 29],
       [30, 31, 32, 33, 34, 35]])
```

In [261]:

```
r_copy[:] = 0
print(r_copy)
print(r)
```

```
[[0 0 0 0 0 0]
 [0 0 0 0 0 0]
 [0 0 0 0 0 0]
 [0 0 0 0 0 0]
 [0 0 0 0 0 0]
 [0 0 0 0 0 0]]
[[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]
 [12 13 14 15 16 17]
 [18 19 20 21 22 23]
 [24 25 26 27 28 29]
 [30 31 32 33 34 35]]
```

5.7 Iterating Over Arrays

In [262]:

```
test = np.random.randint(0, 10, (4,3))
test
```

Out[262]:

```
array([[7, 3, 0],
       [5, 0, 2],
       [7, 9, 1],
       [1, 9, 5]])
```

- Iterate by Row

In [263]:

```
for row in test:
    print(row)
```

```
[7 3 0]
[5 0 2]
[7 9 1]
[1 9 5]
```

- Iterate by row & row index

In [264]:

```
for i, row in enumerate(test):
    print('row', i, 'is ', row)
```

```
row 0 is  [7 3 0]
row 1 is  [5 0 2]
row 2 is  [7 9 1]
row 3 is  [1 9 5]
```

- Iterate on two arrays

In [265]:

```
test2 = test ** 2
```

In [266]:

```
for i, j in zip(test, test2):
    print(i, '+', j, '=', i+j)
```

```
[7 3 0] + [49  9  0] = [56 12  0]
[5 0 2] + [25  0  4] = [30  0  6]
[7 9 1] + [49 81  1] = [56 90  2]
[1 9 5] + [ 1 81 25] = [ 2 90 30]
```