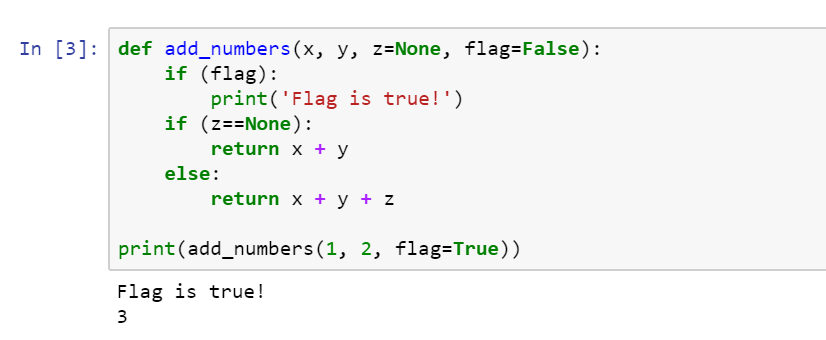
Data Science

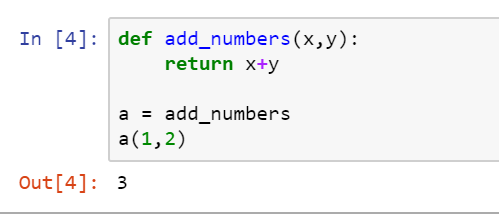
* Hacking Skills
* Math and Statistics Knowledge
* Substantive Expertise

Python is a high-level language:

Readable for people instead of machines

No semicolon and ‘var’, etc.





5 types:

str, NoneType, int, float, function

3 collections:

tuple, list, dictionary

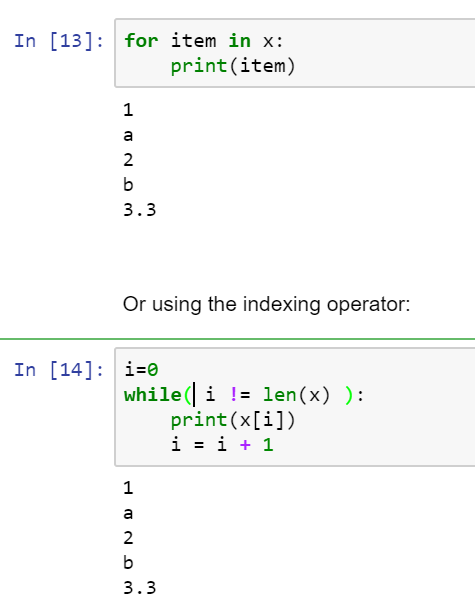
(), [], {}

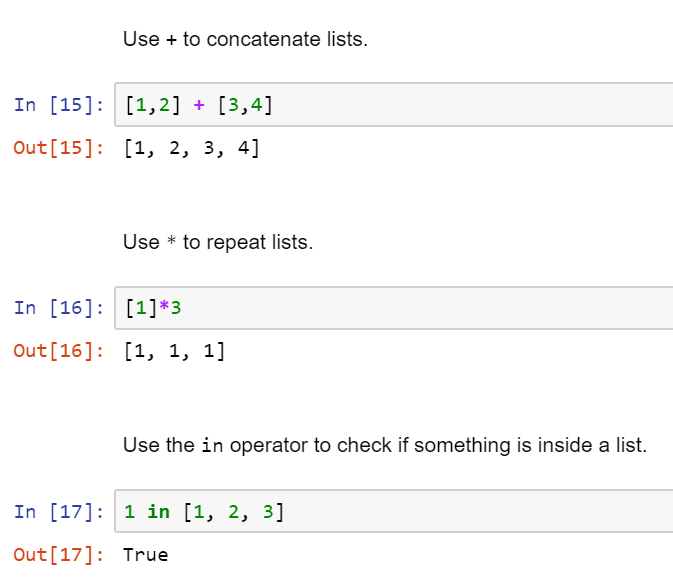
order of tuples cannot be changed (immutable)

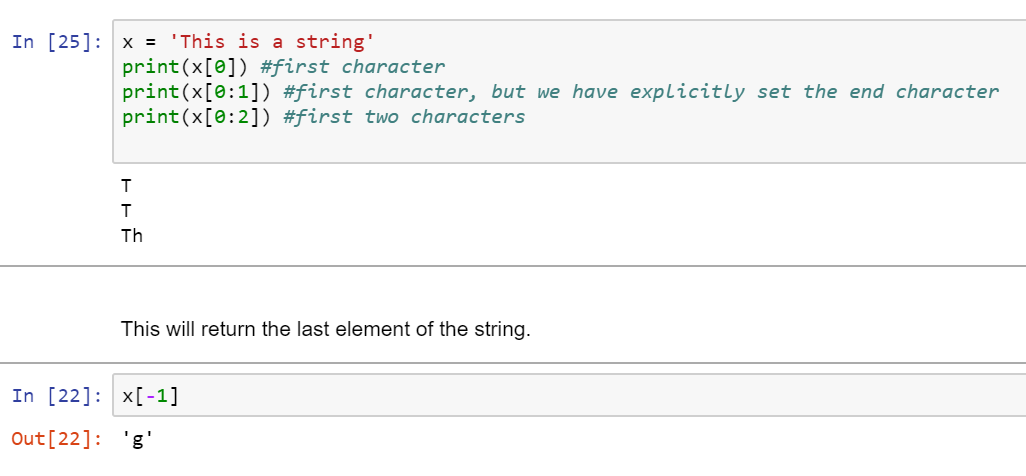
order of lists can be changed (mutable)

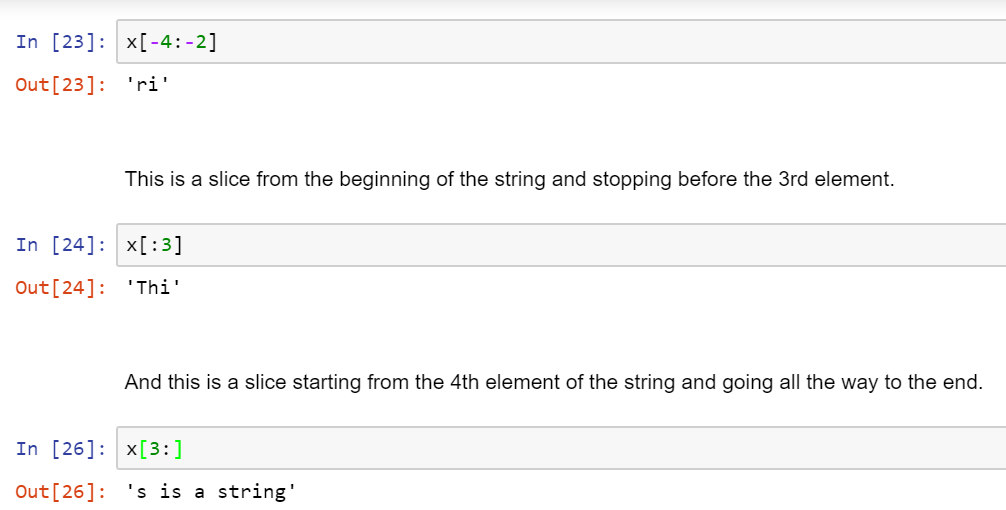
dictionary contains keys and values

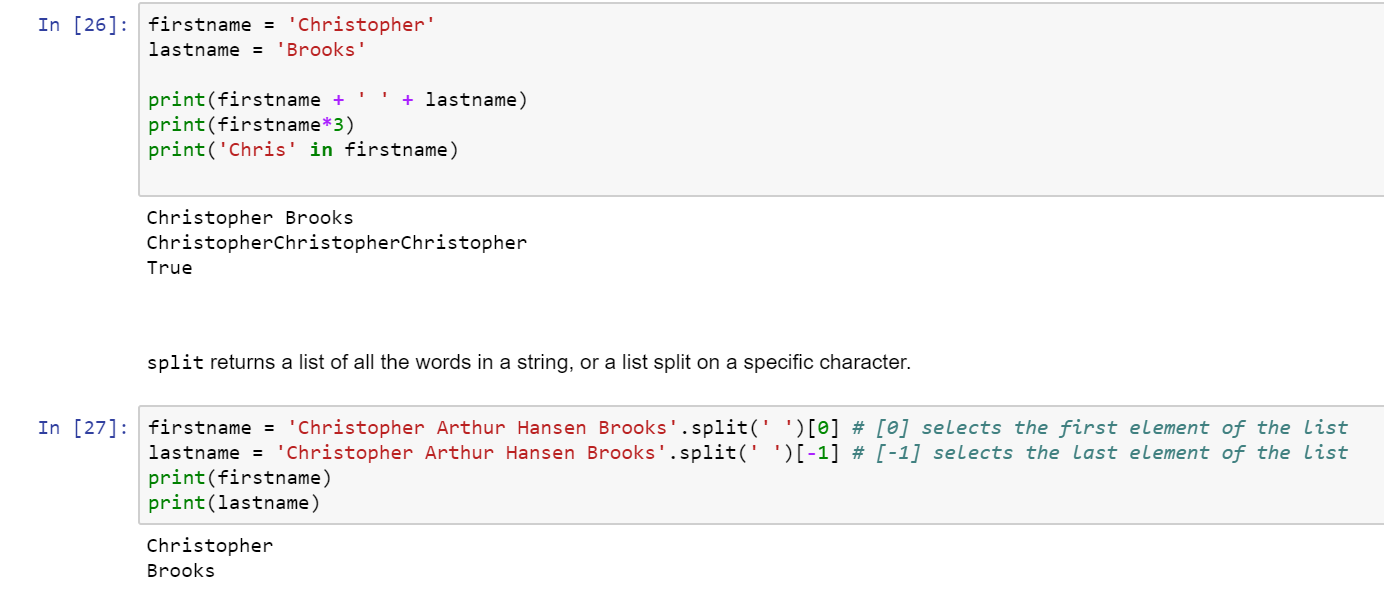
example of loop:



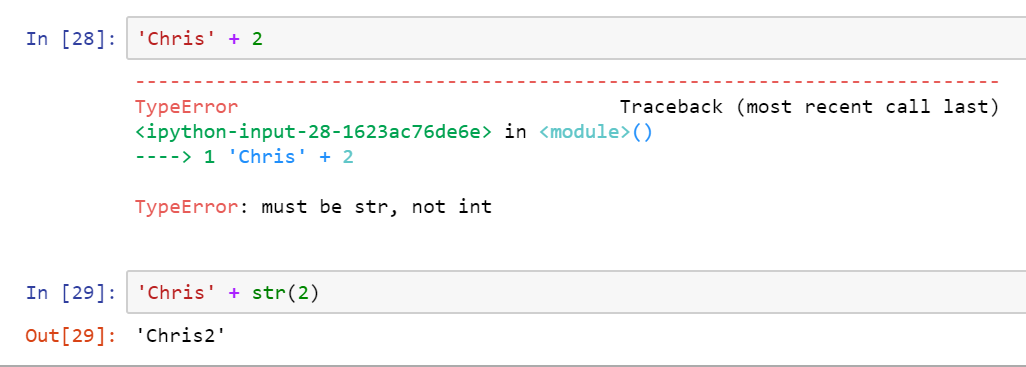




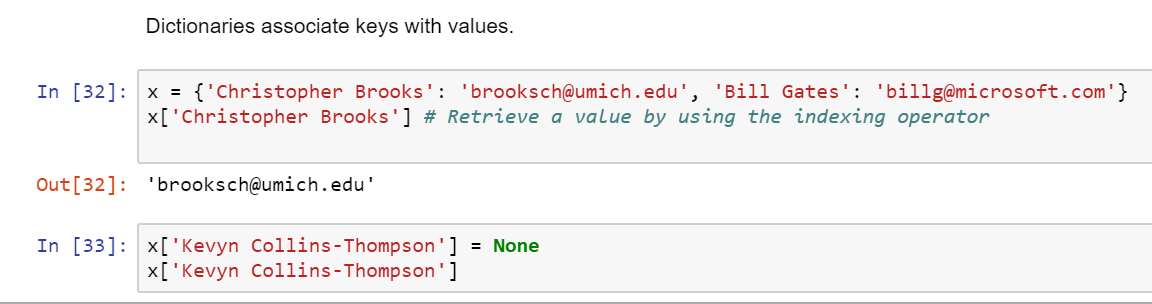


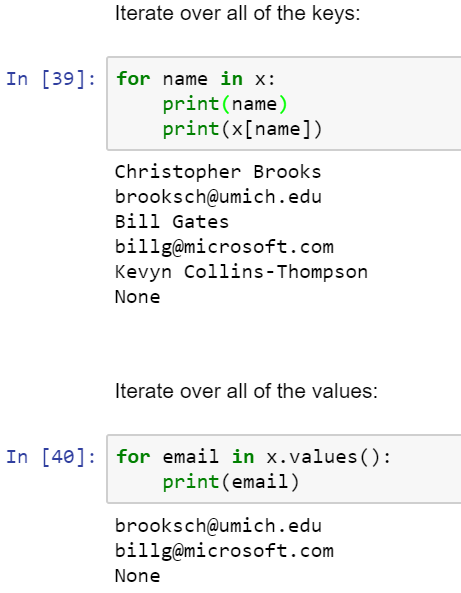


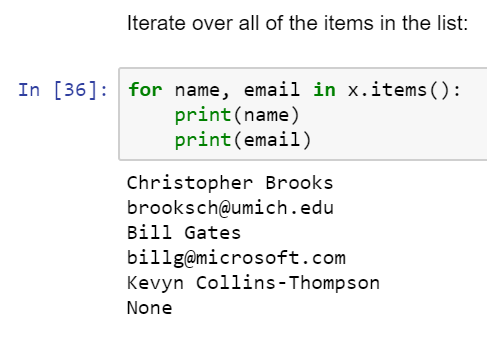
Correct Category:

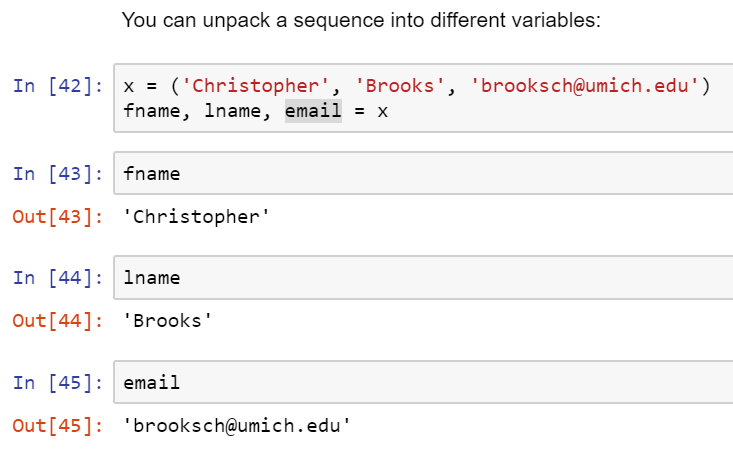


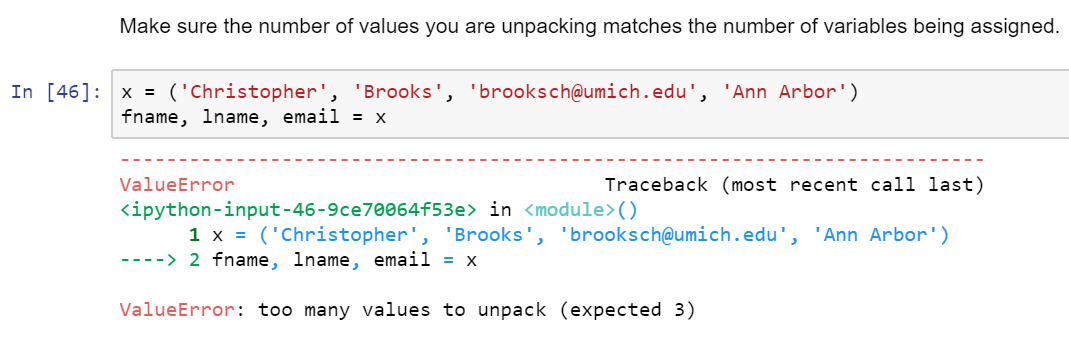
Dictionary:











More on strings:

str(2)

Python use Unicode to handle different languages and even rarely used symbols

a built-in method for string formatting:

**sales\_record = {**

**'price': 3.24,**

**'num\_items': 4,**

**'person': 'Chris'}**

**sales\_statement = '{} bought {} item(s) at a price of {} each for a total of {}'**

**print(sales\_statement.format(sales\_record['person'],**

**sales\_record['num\_items'],**

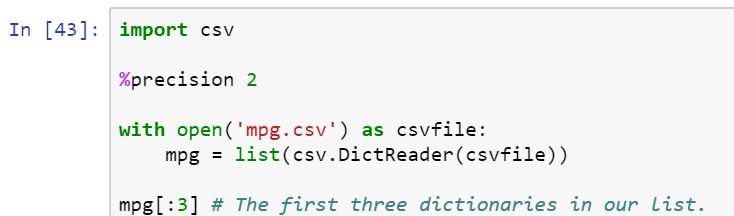
**sales\_record['price'],**

**sales\_record['num\_items']\*sales\_record['price']))**

Chris bought 4 item(s) at a price of 3.24 each for a total of 12.96

XX.format()

Reading and Writing CSV files：



Out [43]: [OrderedDict([('', '1'),

('manufacturer', 'audi'),

('model', 'a4'),

('displ', '1.8'),

('year', '1999'),

('cyl', '4'),

('trans', 'auto(l5)'),

('drv', 'f'),

('cty', '18'),

('hwy', '29'),

('fl', 'p'),

('class', 'compact')]),

OrderedDict([('', '2'),

('manufacturer', 'audi'),

('model', 'a4'),

('displ', '1.8'),

('year', '1999'),

('cyl', '4'),

('trans', 'manual(m5)'),

('drv', 'f'),

('cty', '21'),

('hwy', '29'),

('fl', 'p'),

('class', 'compact')]),

OrderedDict([('', '3'),

('manufacturer', 'audi'),

('model', 'a4'),

('displ', '2'),

('year', '2008'),

('cyl', '4'),

('trans', 'manual(m6)'),

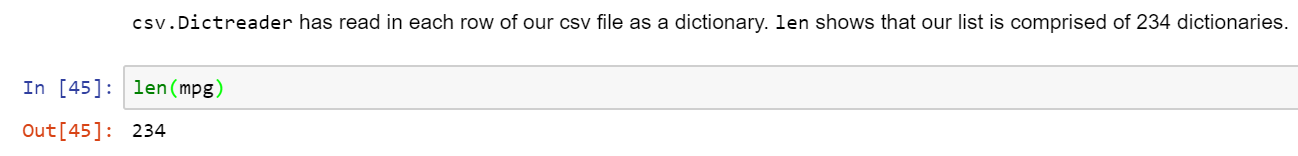
('drv', 'f'),

('cty', '20'),

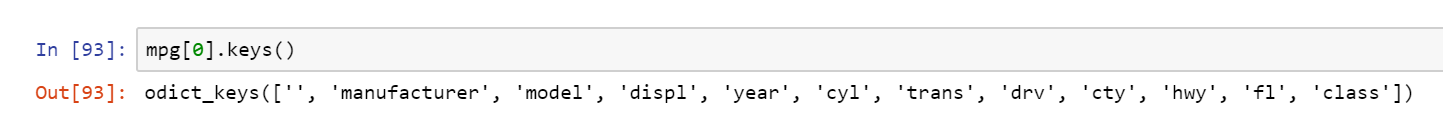
('hwy', '31'),

('fl', 'p'),

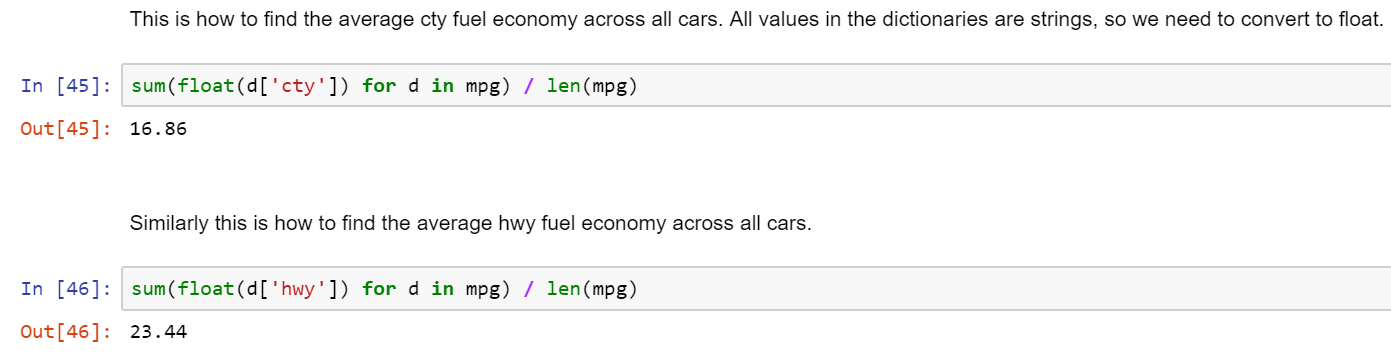
('class', 'compact')])]

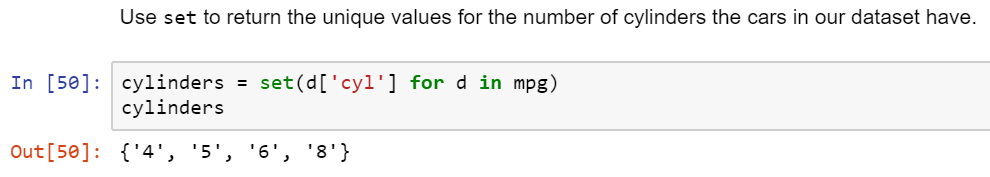


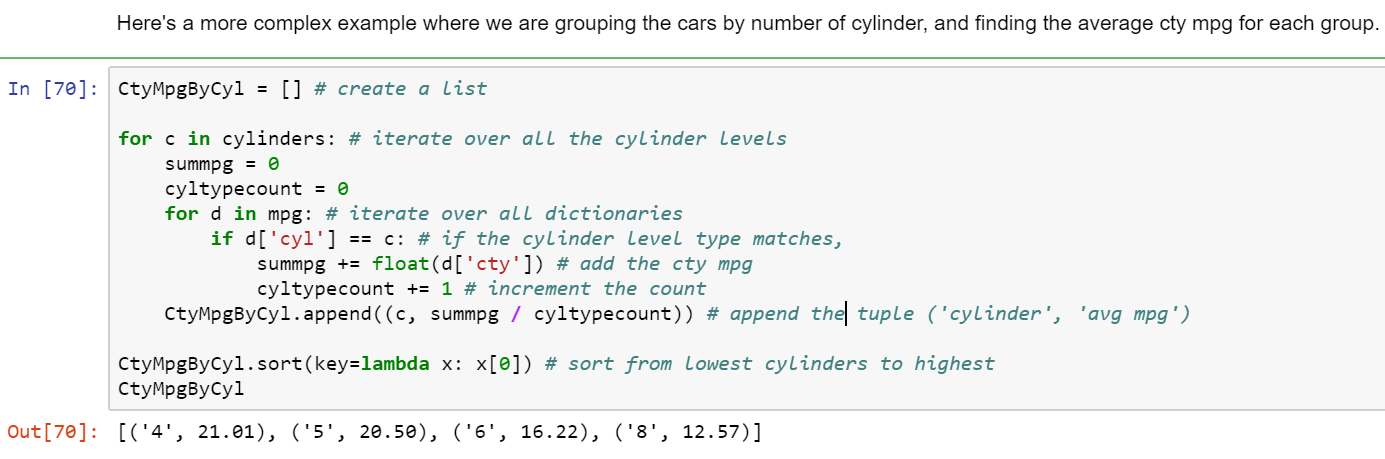
Keys gives us column names of csv:



Average:

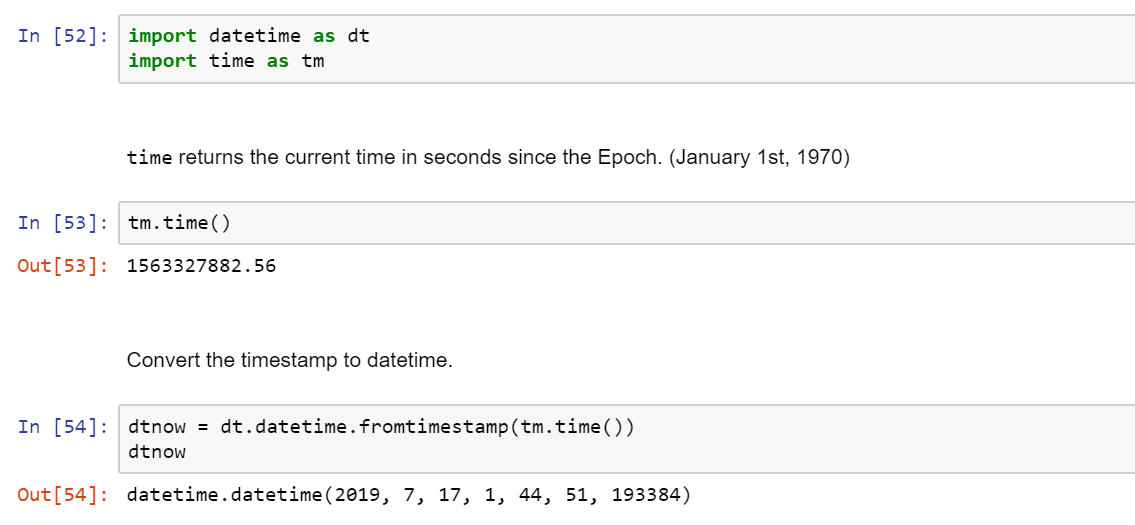


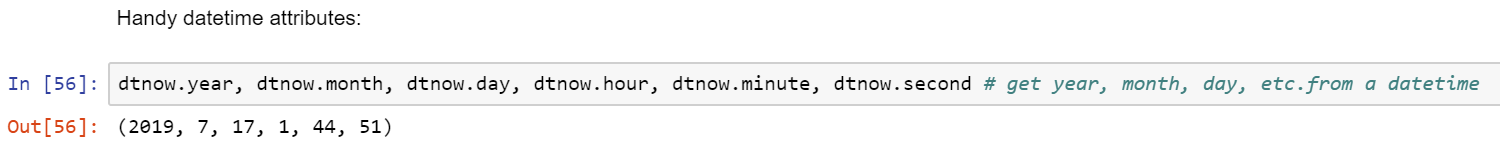


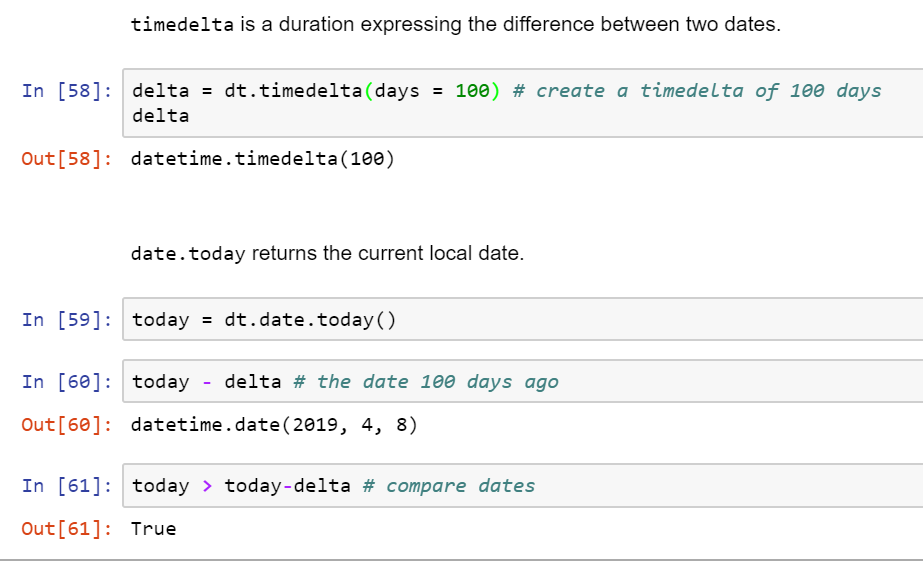


Dates and Times:

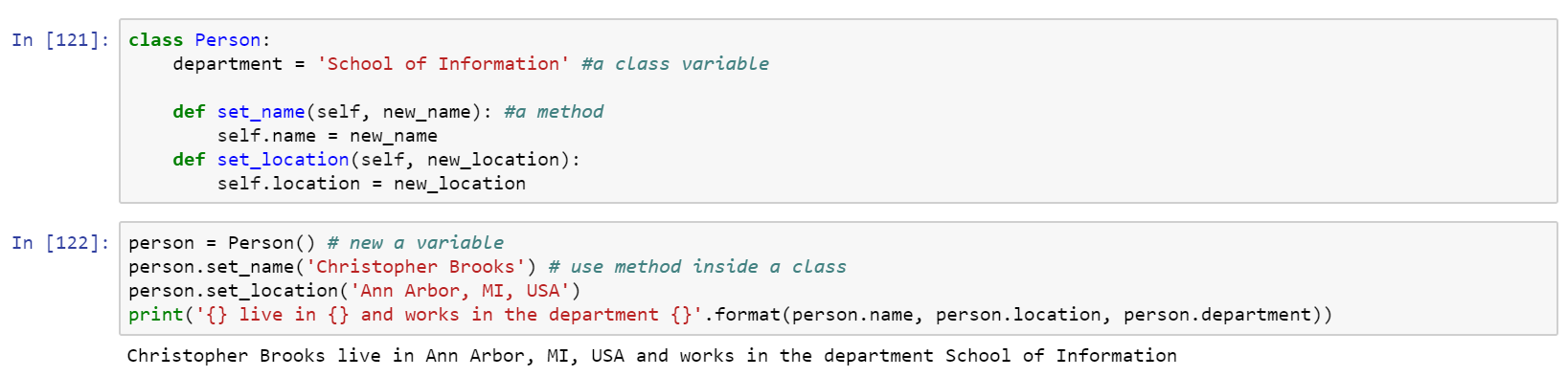
Epoch = 1/1/1970 00:00



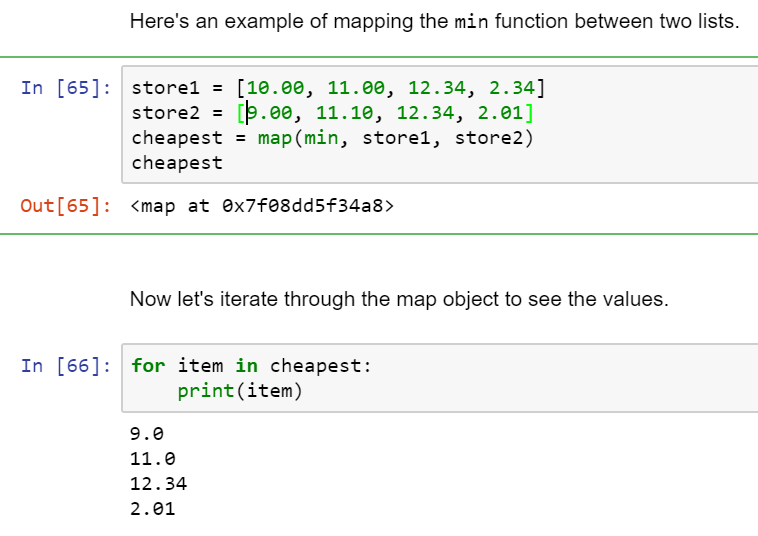


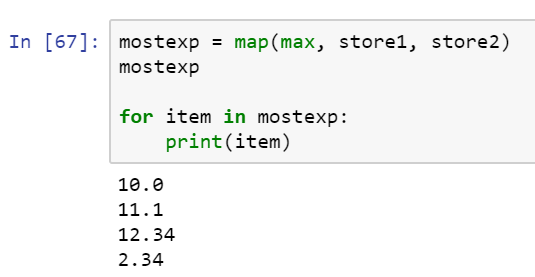


Objects and map():



Map():





**Lambda** and **List Comprehensions:**

**Lambda:** no name functions

Example 1:

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

def split\_title\_and\_name(person):

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

#option 1

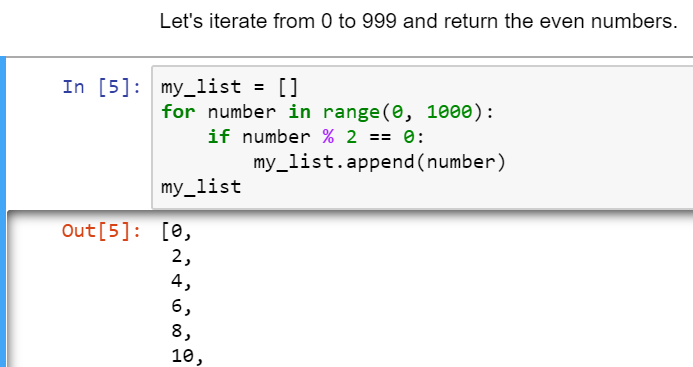
for person in people:

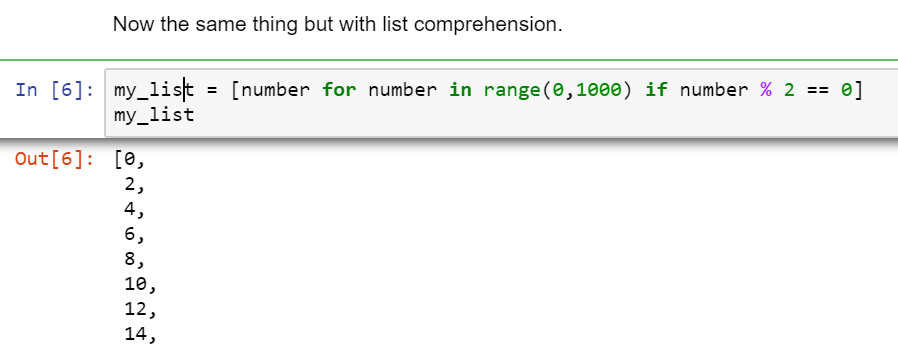
print(split\_title\_and\_name(person) == (lambda x: x.split()[0] + ' ' + x.split()[-1])(person))

#option 2

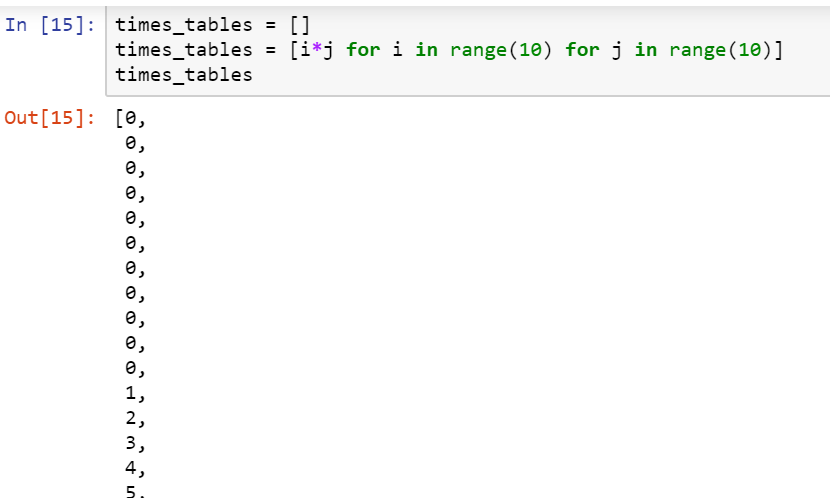
list(map(split\_title\_and\_name, people)) == list(map(lambda person: person.split()[0] + ' ' + person.split()[-1], people))

# actually 0-999 (does not include the last int)





Example 2:



Numerical Python (NumPy):

import **numpy** as np

Creating Arrays:

mylist = [1, 2, 3]

x = np.**array**(mylist)

x

array([1, 2, 3])

y = np.array([4, 5, 6])

y

array([4, 5, 6])

m = np.array([[7, 8, 9], [10, 11, 12]])

m

array([[ 7, 8, 9],

[10, 11, 12]])

m.**shape**

(2, 3)

n = np.**arange**(0, 30, 2) # start at 0 count up by 2, stop before 30

n

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

n = n.**reshape**(3, 5) # reshape array to be 3x5

n

array([[ 0, 2, 4, 6, 8],

[10, 12, 14, 16, 18],

[20, 22, 24, 26, 28]])

o = np.**linspace**(0, 4, 9) # return 9 evenly spaced values from 0 to 4

o

array([ 0. , 0.5, 1. , 1.5, 2. , 2.5, 3. , 3.5, 4. ])

o.**resize**(3, 3)

o

array([[ 0. , 0.5, 1. ],

[ 1.5, 2. , 2.5],

[ 3. , 3.5, 4. ]])

np.**ones**((3, 2))

array([[ 1., 1.],

[ 1., 1.],

[ 1., 1.]])

np.**zeros**((2, 3))

array([[ 0., 0., 0.],

[ 0., 0., 0.]])

np**.eye**(3)

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

[ 0., 1., 0.],

[ 0., 0., 1.]])

np.**diag**(y)

array([[4, 0, 0],

[0, 5, 0],

[0, 0, 6]])

Repeating list:

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

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

Repeating element:

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

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

p = np.ones([2, 3], int)

p

array([[1, 1, 1],

[1, 1, 1]])

stack arrays in sequence **vertically (row wise)**：

np.vstack([p, 2\*p])

array([[1, 1, 1],

[1, 1, 1],

[2, 2, 2],

[2, 2, 2]])

stack arrays in sequence **horizontally (column wise)**：

np.hstack([p, 2\*p])

array([[1, 1, 1, 2, 2, 2],

[1, 1, 1, 2, 2, 2]])

Operations:

print(x + y) # elementwise addition [1 2 3] + [4 5 6] = [5 7 9]

print(x - y) # elementwise subtraction [1 2 3] - [4 5 6] = [-3 -3 -3]

[5 7 9]

[-3 -3 -3]

print(x \* y) # elementwise multiplication [1 2 3] \* [4 5 6] = [4 10 18]

print(x / y) # elementwise divison [1 2 3] / [4 5 6] = [0.25 0.4 0.5]

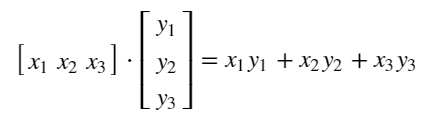
[ 4 10 18]

[ 0.25 0.4 0.5 ]

print(x\*\*2) # elementwise power [1 2 3] ^2 = [1 4 9]

[1 4 9]

Dot product:



x.**dot**(y) # dot product 1\*4 + 2\*5 + 3\*6

32

z = np.array([y, y\*\*2])

print(**len**(z)) # number of rows of array

2

z = np.array([y, y\*\*2])

z

array([[ 4, 5, 6],

[16, 25, 36]])

z.shape

(2, 3)

z.**T**

array([[ 4, 16],

[ 5, 25],

[ 6, 36]])

z.T.shape

(3, 2)

z.**dtype**

dtype('int64')

z = z.**astype**('f')

z.dtype

dtype('float32')

Math functions:

a = np.array([-4, -2, 1, 3, 5])

a.sum()

3

a.max()

5

a.min()

-4

a.mean()

0.59999999999999998

a.std()

3.2619012860600183

index of the maximum and minimum values in the array:

a.argmax()

4

a.argmin()

0

Indexing / Slicing:

s = np.arange(13)\*\*2

s

array([ 0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144])

s[0], s[4], s[-1]

(0, 16, 144)

s[1:5]

array([ 1, 4, 9, 16])

s[-4:]

array([ 81, 100, 121, 144])

s[-5::-2]

array([64, 36, 16, 4, 0])

r = np.arange(36)

r.resize((6, 6))

r

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]])

r[2, 2]

14

r[3, 3:6]

array([21, 22, 23])

r[:2, :-1]

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

[ 6, 7, 8, 9, 10]])

r[-1, : :2] # jump to the 2nd after

array([30, 32, 34])

r[r > 30]

array([31, 32, 33, 34, 35])

r[r > 30] = 30

r

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, 30, 30, 30, 30, 30]])

Copying Data:

r2 = r[:3, :3]

r2

array([[ 0, 1, 2],

[ 6, 7, 8],

[12, 13, 14]])

r2[:] = 0

r2

array([[0, 0, 0],

[0, 0, 0],

[0, 0, 0]])

**r has also been changed!**

r

array([[ 0, 0, 0, 3, 4, 5],

[ 0, 0, 0, 9, 10, 11],

[ 0, 0, 0, 15, 16, 17],

[18, 19, 20, 21, 22, 23],

[24, 25, 26, 27, 28, 29],

[30, 30, 30, 30, 30, 30]])

**Use r.copy to avoid this:**

r\_copy = r**.copy()**

r\_copy

array([[ 0, 0, 0, 3, 4, 5],

[ 0, 0, 0, 9, 10, 11],

[ 0, 0, 0, 15, 16, 17],

[18, 19, 20, 21, 22, 23],

[24, 25, 26, 27, 28, 29],

[30, 30, 30, 30, 30, 30]])

r\_copy[:] = 10

print(r\_copy, '\n') # add a space

print(r)

[[10 10 10 10 10 10]

[10 10 10 10 10 10]

[10 10 10 10 10 10]

[10 10 10 10 10 10]

[10 10 10 10 10 10]

[10 10 10 10 10 10]]

[[ 0 0 0 3 4 5]

[ 0 0 0 9 10 11]

[ 0 0 0 15 16 17]

[18 19 20 21 22 23]

[24 25 26 27 28 29]

[30 30 30 30 30 30]]

Iterating Over Arrays:

test = np.**random.randint**(0, 10, (4,3)) # 4 by 3 array with int from 0 to 9

test

array([[3, 3, 2],

[1, 5, 4],

[4, 0, 0],

[0, 4, 3]])

for row in test:

print(row)

[3 3 2]

[1 5 4]

[4 0 0]

[0 4 3]

for i in range(len(test)):

print(test[i])

[3 3 2]

[1 5 4]

[4 0 0]

[0 4 3]

for i, row in enumerate(test):

print('row', i, 'is', row)

row 0 is [3 3 2]

row 1 is [1 5 4]

row 2 is [4 0 0]

row 3 is [0 4 3]

test2 = test\*\*2

test2

array([[ 9, 9, 4],

[ 1, 25, 16],

[16, 0, 0],

[ 0, 16, 9]])

for i, j in **zip**(test, test2):

print(i,'+',j,'=',i+j)

[3 3 2] + [9 9 4] = [12 12 6]

[1 5 4] + [ 1 25 16] = [ 2 30 20]

[4 0 0] + [16 0 0] = [20 0 0]

[0 4 3] + [ 0 16 9] = [ 0 20 12]