__/__/__

Intro to Data Science in Python - Course 1

⊛ Python Functions

⊛ Types & Sequences

type (None) → Nonetype

Tuple - immutable sequence ( )

```
x = (1, 'a', b, 'c')
type(x)  # tuple
x[1] = 'y' X  bcoz immutable
```

List - mutable sequence [ ]   ⊛ x.append('e')

```
x = [1, 'a', c, 'd']   ⊛ ~~for~~
type(x)  # mutable
x[1] = 'y' ✓
```

⊛ Both Lists & tuples are iterable.

```
for item in x:
    print(item)
```

⊛ $[1, 2] + [3, 4] \xrightarrow{\text{Concat}} [1, 2, 3, 4]$

$[1] * 3 \longrightarrow [1, 1, 1]$

⊛ (in) 1 in [1, 2, 3] # True

$x[\overset{\uparrow \text{starting index}}{a} : \overset{\nearrow \text{s-i + len of string}}{a + len}]$

Slice (exclusive slice)

```
↳ x = 'This is a string'
print (x[0:2]) → [0,2)
print (x[-4:-2]) → # ri
print (x[3:]) → till end
```

Split

```
firstname = ' This is a Sabuj'
    x = firstname.split()
```

Dict

```
→ x = { 'CB' : 'ab',
        'Bill' : 20,
        'hello' : 19
      }
→ x['goodbye'] = 21     # append
                 key
→ for name in x:
      print(x[name])
                          value
→ for email in x.values():
      print(email)
                                        # iterate thru Dict
→ for name,email in x.items():
  print(name,email)
```

Tuple Unpacking

```
✓ x= ('Christ', 'Jesus', '25')
  fname, lname, age = x

✓ fname #  'Christ'
✓ lname #  'Tens'
✓ age #    25
```

__/__/__

Move on Strings

✓ 'Chris' + 2 X

✓ 'Chris' + str(2) ✓

✓ x = ' {} bought {} from {}'. format ('Salvij', 'Jane', 20)

→ csv files.

```
import csv
fname = open('mpg.csv')
print(fname)   # TextIOWrapper
print(fname.read())  ← if you do this, fname is destroyed.
print        csv.DictReader(fname)

mpg = list(csv.DictReader('mpg.csv'))
```

Scanned by CamScanner

__/__/__

CSV (Corey) comma, separated, values

> import csv

```
with open('names.csv', 'r') as csv_file:
    csv_reader = csv.reader(csv_file)   # object
    for line in csv_reader:
        print(line)        ['John', 'Doe', 'abc@gmail']
```

next(csv_reader) ←
↓
omits the first row.

write to new csv

```
with open('new.csv', 'w') as new_file:
    csv_writer = csv.writer(new_file), delimiter = '-'
                                                  '\t')

    for line in csv_reader:
        csv_writer.writerow(line)
```

read using a delimiter

```
csv_reader = csv.reader(~~csv file~~ new csv file, delimiter='\t')

for line in csv_reader:
    print(line)
```

DictReader

```
csv_reader = csv.DictReader(csv_file
for line in csv_reader:
    print(line['email'])
```

/ Dict Writer (Here you've to pass in the field-names)

```
with open('new.csv', 'w') as new_file:
    ~~new_file~~
    fieldnames = ['FN', 'LN', 'email']
    csv_writer = csv.DictWriter(new_file, fieldnames=fieldnames)
    csv_writer.writeheader()
    for line in csv_reader:
        csv_writer.writerow(line)
del line['email']
```

Python Objects, Map

⊗
```
class Person:
    dept = 'School of Information'

    def set_name(self, new_name):
        self.name = new_name
    def set_location(self, new_loc):
        self.location = new_loc
```

① No labels
② No need for constructor

⊗
```
person = Person()
person.set_name('CB')
person.set_location('USA')
```

⊗ MAP

```
map(function, iterable, ...)
↓ returns a map object
  data → [ d1 , d2 , - - - - , dn ]
        f
map(f, data)   # iterator map obj
list(map(f, data));
```

__/__/__

LAMBDA (anon)

fn name — keyword — inputs — return value

full_name = lambda fn, ln: fn.strip().title() + " " + ln.strip().title()

single expression

full_name("   leo", "EULER") # 'Leo Euler'

✓ lambda x1, x2, ... xn: x1 ± x2 ± ....

no inp.

✓ lambda :

sorted_auth.sort(key=lambda name: name.split(" ")[-1].lower())

Lambda + List Comprehensions

LC

→ lst = [expr for val in collection if <test1> and <test2>]

→ squares = [i**2 for i in range(1,101)]

Numpy

→ gmovies = [title for title in movies if title.startswith("G")]

g[('A', 1990), ('SB', 2000)]

<2000

→ preah = [title for (title, year) in movies if year < 2000]

Adding Lists in Python = Concat

x

w = [4**x for x in v]

Cartesian product

A = [1, 3, 5, 7] c_p = [(a,b) for a in A for b in B]

B = [2, 4, 6, 8]

epoch Jan 1 '1970

type(lambda x: x+1) → FUNCTION

NUMPY

⊛. Saves coding time
- same data-type to avoid type-checking
- uses less-memory

⊛ Python Lists → use pointers to objects to store "data

[Py list]

Mem Addr	Ptr			
0050h	1204h	→	1204h	5.25
0080h	1206h	→	1206h	1.97

[Numpy]

Mem Addr	Data
0050h	5.25
0054h	38.6

```
>import numpy as np
> a= np.array([2,3,4])
#
> a= np.arange(1, 12, 2)   → exclusive
# 1, 3, 5, - - , 11
> a= np. linspace(1,12,6)  # linearly space 6 elements from 1-12
# 1., 3.2, 5.4, 7.6, 9.8, 12.
> a= a. reshape(3,2)  # reshape to 3X2 array.
a= ([[1, 3.2],
     [5.4, 7.6],
     [9.8, 12.]])
```

__/__/__

variable

> a.size #6

> a.shape (3,2)

variable not a function

> a.dtype # 'float64'

> a.itemsize # 8 bytes

> b = np.array([(1.5,2.3), (4.5,6)]) # 2D array (represented with tuples inside [ ])

> a<4

T T

F T

F F

> a * 3 → each element gets multiplied ✓

> a = np.zeros ((3,4)) ✓

> a = np.ones ((2,3)) ✓

> a = np.ones (10) ✓

> a = np. array ([2,3,4], dtype = ~~int~~ np.int16)

np.int

np float

>

a = np.eye(2), → 2x2 identity matrix

Random

> a = np.random.random ((2,3))

↓ values from [0,1)

> a = np.random. randint (0,10,5)

jovanip

np.set_printoptions (precision = 2, suppress = True)

2 dec place

No scientific notation.

> a. sum()

a. min()

a.max()

a. mean()

~~[illegible]~~ a.var() # variance

a.std() # std dev

a. sum (axis=1). # horizontal row wise

a. sum (axis=0) axis=1 # vertical row wise