* Python object and data Structures.

1) Numbers. - 2 types — integers
                        — floating point nos.

- Python 2 performs classical division.
  (i.e) decimal points are truncated off (rounded off).

Python 3 performs true division.
(i.e) decimal points are not rounded.

3/2 → Python 2 = 1 ⇒ To avoid this.
Python 3 = 1.5.

↓

① 3/2.0 = 1.5

or

② Caste 1 nos.
(i.e) float(3)/2 = 1.5
↓
it castes int 3 as floating no.

③ or
- from __future__ import division.
- 3/2 = 1.5
→ this imports division from the future library.

- (2)^3 ⇒ 2**3
  (Power)

- Sqrt
  √4 = 4**0.5.

- Ordered Operation :- $2 + 10 * 10 + 3$

↓ multiplication is given priority

$(2+10) * (10+3)$ → here ( ) given first priority than mul.

* To assign label by using '='

eg:- $a = 5$ → 5 is assigned a label a.

∴ when we call 'a' in the code anywhere it is equal to 5.

∴ $a + a = 10$.

variable name = object you want to label.

eg:-
```
my_income = 100
tax_rate = 0.1

my_taxes = my_income * tax_rate
my_taxes
10.0
```

2) Strings

- It can be created using single quotes or double quotes
  eg:- 'Hello'
  "This is a string".

- printing a string :-
  Python 2 → Print 'Hello world'.
  Python 3 → Print ('Hello world').

- length of string :-
  eg:- len('Hello world')
  ↓↓

- assigning label :-
  s = "Hello world".
  print s. → o/p - Hello world.
  or s → o/p - 'Hello world'.

- Indexing :-
  eg:- s[0] = 'H'
  s[1] = 'E'

- Slicing :-
  eg:- s[1:]
  ↳ from s object (i.e) Hello world grab everything from 1 index to (:) → onwards
  o/p - 'ello world'.

  s[:3] → grab everything upto 3 index.
  o/p - 'Hel'.

s[:] → grab whole string
o/p - 'Hello world'.

s[-1] (reverse)
o/p - 'd'

s[:-1] grab everything upto last letter.
o/p - 'Hello worl'.

s[::1] grab everything with step size 1
o/p - 'Hello World'.

s[::2] step size 2.
o/p 'HLOWRd'

- To reverse the string :- s[::-1]
o/p - 'dlroW olleH'.

* String Properties

- Immutability - Once the elements within string are set it can't be changed.
But we can concatenate the strings (add)

eg:-
(i) s + ' Concatenate me!'
o/p - 'Hello world concatenate me!'

(ii) letter = z
letter * 10
o/p = 'zzzzzzzzzz'.

2) Built in functions :-

s = 'Hello'

s.upper() → converts whole string to upper case

o/p - HELLO.

s. (press tab & we can get pop up of all the built in functions possible)

3) Print formatting.

1) print 'This is a string'.

2) x = string

print 'Place my variable here : %s' % (x)

statement; %s → string; (x) → label given to the string

o/p : Place my variable here : string.

(x can be anything even nos. or decimals, etc)

3) Print 'Floating point number : %1.2f' % (13.145)

precision.

o/p = Floating point number : 13.14.

4) print 'Floating point number : %6f' % (13.145)

(total minimum no. of digits)

o/p = Floating point number : 13.145000

print 'Floating point number : %25.3f' % (13.145)

o/p = 'Floating point number : 13.145

fill the characters which are not there with white space.

* Conversion methods :-

(ii) print 'Convert to string %r' %(123)
o/p = Convert to string 123
uses repr() function.

(iii) print 'Convert to string %s' %(123)
o/p = Convert to string 123
uses str() function.

- print 'First: %s, Second: %s, Third: %s' %('hi
o/p = First: hi!, Second: Two, Third: 3 Two', 3)

- print 'First: {x} Second: {y} Third: {x}'
.Format (x='inserted', y='Two!').
o/p = First: 'inserted' Second: 'Two!' Third: 'inserted'.

4) Lists : (can hold different object type)
(no type contrain & no size constrain).

eg:- (i) my_list = ['string', 23, 1.2, 'o']
my_list
o/p = ['string', 23, 1.2, 'o']

(ii) len(my_list) # length of list.
o/p = 4

(iii) my_list[0]
o/p = 'string'
my_list[1:]
o/p = 23, 1.2, 'o'.
my_list[:3]
o/p = 'string', 23, 1.2.

(iv) Concatenate.

eg:
my_list + ['new item']
o/p = 'string', 23, 1.2, 'o', 'new item'

- after concatenating even if we call
my_list
o/p = 'string', 23, 1.2, 'o'.
(i.e) it does not change permanently (retains the original one), change is temporary.

If we want to make changes permanently we need to reassign it.
eg:- my_list = my_list + ['permanent add']
o/p = 'string', 23, 1.2, 'o', permanent add'.

(v) my_list * 2 ;

o/p = 'string',
23,
1.2,
'0',
'permanent add',
'string',
23,
1.2,
'0',
'permanent add'.

~~doubels~~

(vi) nesting of list (list inside a list)

eg:- L-1 = [1, 2, 3]
L-2 = [4, 5, 6]
L-3 = [7, 8, 9]

matrix = [L-1, L-2, L-3]
matrix
o/p = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

matrix [0]
o/p = [1, 2, 3]

matrix [0][0]
o/p = 1
matrix [2][0]
o/p = 7

5) Dictionaries (collection of objects sorted by key).

eg:- i) my_dict = {'key1': 'value', 'key2': 'value2'}.

my_dict['key1']

o/p = 'value'. → unlike list we cannot use [0] → as index.

dictionaries like list are flexible for datatype.

it can be accessed using 'key' as index.

because here it is mapping & not a sequence.

ii) my_dict = {'k1': 123, 'k2': 3.4, 'k3': 'string'}

my_dict['k3']

o/p = 'string'.

my_dict['k3'][0]

o/p = 's'.

my_dict['k3'][::-1]

o/p = 'gnirts'. #reverse.

my_dict['k3'].upper()

o/p = STRING.

my_dict['k1'] - 120.

o/p = 3. #temp. change.

my_dict['k1'] = my_dict['k1'] - 120.

o/p = 3 #permanent change.

my-dict['k1'] += 100 ⇔ my-dict['k1'] = my-dict['k1'] + 100.
similarly '-', '*', '\' can be done.

(iii) d = {} # empty dictionary
d['animal'] = 'Dog' } # assignment.
d['answer'] = 42. }
d
o/p = {'animal': 'Dog', 'answer': 42}.
key ↙ object associated with that key ↘

(iv) nesting of dictionary.
eg:-
d = {'k1': {'nestkey': {'subnestkey': 'value'}}}
d['k1']
o/p = {'nestkey': {'subnestkey': value'}}

d['k1']['nestkey']
o/p = {'subnestkey': 'value'}

d['k1']['nestkey']['subnestkey']
o/p = 'value'.

d['k1']['nestkey']['subnestkey'].upper()
o/p = 'VALUE'.

- There is no specific order in dictionaries unlike list, dictionary keep track of things wrt the key and its respective value.

+100. eg:- d = { }

```
d['k1'] = 1
d['k2'] = 2
d['k3'] = 3
d
o/p = {'k1': 1, 'k2': 2, 'k3': 3}
```

```
d.keys()
o/p = ['k3', 'k2', 'k1']      # retrives list of all the keys.
```

```
d.values()                    # retrives all the values.
o/p = [3, 2, 1]
```

```
d.items()
o/p = [('k3', 3), ('k2', 2), ('k1', 1)]
```

this is known as "tuples".

6) Tuples. (similar to list but they are immutables).

- Creating tuples. eg:-

```
t = (1, 2, 3)
t
o/p = (1, 2, 3)
```

```
len(t)
o/p = 3.
```

```
t[0]
o/p = 1
```

- slicing eg:- t[-1]
  o/p = 3

- tuple methods :- variable name . method name (arguments)

eg:- t.index ('1')
  o/p = 0
  t.index ('2')
  o/p = 1

t.count ('1')
  o/p = 1
t = (1, 1, 2, 3)
t.count (1)
o/p = 2 .

- immutable → cannot change the items inside it

eg:- t = (1, 2, 3)
  t[0] = 's'
  o/p = error .

L = [1, 2, 3]
L[0] = 's'
L
o/p = ['s', 2, 3]

- When you are passing an object and you don't want the objects to be changed throughout, in such a situation tuples are used.

eg:- tuple representing the dates of a calender which is not going to change throughout the code.

7) Files. (how to access with external files)

pwd - print working directory

its)

i) f = open('test.txt') # file is open
variable → function → name of file.

(ii) f.read()
o/p = 'This is a test line'. # reads the content of the opened file.

f.read()
o/p = # nothing comes as o/p. once the file is read the cursor is at the end of the file ∴ reads nothing

(iii) f.seek(0) → places the cursor at the beginning of the file ← we need to reset the cursor
f.read()
o/p = 'This is a test line'.

(iv) f.readlines()
o/p = ['This is a test line'] # returns every line of the file and treats it as an element & returns in the form of list to you

(v) ~~%%writefile~~
%%writefile new.txt.
First line → name of file
Second line
.
o/p = writing new.txt. # allow you to write in a text file without using any external text editor in ipython itself.

8) Sets and Bodeans

* sets (unordered collection of unique elements)

eg:- x = set()

x

o/p = set()

x.add(1) # for adding elements to the set.

x

o/p = {1}

if we add same elements again which are already present in it, that element won't be repeated

(i.e) the sets add unique elements (no repetation)

eg:- l = [1,1,1,2,2,2,2,3,3,4] # list.

l

o/p = [1,1,1,2,2,2,2,3,3,4]

set(l) # set.

o/p = {1,2,3,4}

* Booleans (predefined true and false displays)

eg:- 1 > 2

o/p = false → boolean.