* Modules. (built-in-libraries)

import name of module # syntax.

eg:
(i) import math.
math.sqrt(4)
o/p = 2.0.

(ii) from math import sqrt.
sqrt(4)
o/p = 2.0.

- We can download modules that are not built-in, in python by 2 ways.
(i). to check if Anaconda already have the module downloaded.
(∵ Anaconda distribution have many modules already included)

This can be done by:

conda install (name of module we want to install) # syntax.

eg:- conda install flask. (Flask is a web framework of creating websites.)

If anaconda does not include the package that we ~~do~~ want we can use,

pip install *module name*.

eg:- pip install python-pptx. # to download modules online.

* we can either google search for online packages or use github.

* Built-in function:

1) Map().

- takes in 2 arguments : a function and a sequence iterable.

eg:- map(function ; sequence)

↓

name of function

- map() applies the function to all the elements of the sequence. It returns a new list with the elements changed by function.

eg:- (i) def fahrenheit(T):
    return (9.0/5)*T + 32.
fahrenheit(0)
o/p = 32.0

but if we want to have list of items i.e.
(ii)
temp = [0, 22.5, 40, 100].
map(fahrenheit, temp)
o/p = [32.0, 72.5, 104.0, 212.0].
(every element in the list is applied with Fahrenheit function).

- when a function is not defined (i.e) any unknown function we can use "lambda".

eg:- (iii) map(lambda T: (9.0/5)*T + 32, temp)
o/p = [32.0, 72.5, 104.0, 212.0].

(iv) a = [1, 2, 3]
b = [4, 5, 6]
c = [7, 8, 9].
map(lambda x, y, : x + b, a, b)
o/p = [5, 7, 9]

↳ lambda functions can take multiple arguments and expressions.

map(lambda num: num*-1, a)
o/p = [-1, -2, -3].

2) Reduce ()

reduce (function, sequence)
continually applies the function to the sequence.

eg:- (i). lst = [47, 11, 42, 13]
reduce (lambda x, y : x+y, lst)
o/p = 113.

47 + 11 = 58.
58 + 42 = 100
100 + 13 = 113.

(ii) lst = [34, 23, 24, 24, 100, 2333, 2, 11]
max(lst) → built-in function.
o/p = 2333

using reduce() function.
max_find = lambda a, b : a if (a>b) else b.
reduce (max_find, lst)
o/p = 2333

3) Filter()

- filter (function, list)
  offers a convenient way to filter out all the elements of an iterable, for which the function returns true.
- The function filter (.) needs a function as its first argument.
- The function needs to return a Boolean value (either True or False).

eg:-
```
def even_check(num):
    if num%2==0:
        return True
    else:
        return False
```

(i)
```
lst=range(10)
lst
o/p = [0,1,2,3,4,5,6,7,8,9]
filter(even_check, lst)
o/p= [0,2,4,6,8]
```

(ii)
```
filter(lambda num: num%2==0, lst)
o/p= [0,2,4,6,8]
```

(iii)
```
filter(lambda num: num>3, lst)
o/p= [4,5,6,7,8,9].
```

4) Zip()

eg:- (i) x = [1, 2, 3]
y = [4, 5, 6]
zip (x, y)
o/p = [(1, 4), (2, 5), (3, 6)]

(ii) a = [1, 2, 3, 4, 5]
b = [2, 2, 10, 1, 1]
for pair in zip(a, b):
print max(pair)
o/p = 2
2
10
4
5

- zip outputs a tuple whereas map outputs a list.

eg:- (iii) map(lambda pair: max(pair), zip(a, b))
o/p = [2, 2, 10, 4, 5].

(iv) x = [1, 2, 3]
y = [4, 5, 6, 7, 8]
zip (x, y)
o/p = [(1, 4), (2, 5), (3, 6)]

#If 1 iterable is shorter than the other than we are defined (o/p) by the shorter iterable.

- zip for dictionaries

(v)
```
d1 = {'a':1, 'b':2}
d2 = {'c':4, 'd':5}
zip(d1, d2)
o/p = [('a','c'), ('b','d')]
```

- using zip to switch the values and keys of 2 dictionaries.

Eg:- (vi)
```
def switchdicts(d1, d2):
    dout = {}  # empty dictionary
    for d1key, d2val in zip(d1, d2.intervalue()):
        dout[d1key] = d2val
    return dout
d1
o/p = {'a':1, 'b':2}
d2
o/p = {'c':4, 'd':5}
switchdicts(d1, d2)
o/p = {'a':4, 'b':5}
```

- ~~Dictionaries~~

5) Enumerate ()

- allows you to keep a count as you iterate through an object.
- It returns a tuple in the form (count, element)

eg:- (i) L = ['a', 'b', 'c']

```
for count, item in enumerate (L):
    print count
    print item
```

o/p = 0
a
1
b
2
c.

(ii)

```
for count, item in enumerate (L):
    if count >= 2:
        break
    else:
        print item
```

o/p = a
b.

6) all() and any()

- allows us to conviniently check for boolean matching in an iterable.
- all() will return True if all elements in an iterable are True.
- any() will return True if any of the elements in an iterable are True.

Eg:- l = [True, True, False, False]
any(l)
o/p = True
all(l)
o/p = False.

7) Complex()

- returns a complex number with the value real + img*1j or convert a string or number to a complex number.
- If the first parameter is a string, it will be automatically interpreted as a complex number and the function must be called without a second parameter.
- But the second parameter can never be a string.