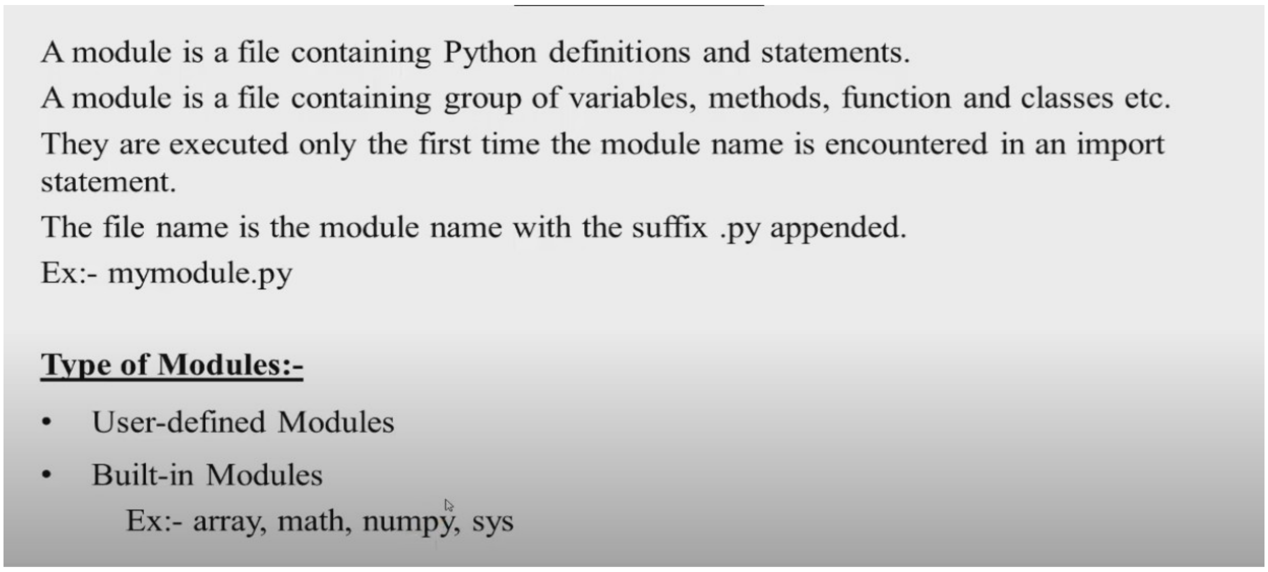
PYTHON NOTES-6

MODULES IN PYTHON

- In programming, a module is a piece of software that has a specific functionality. For example, when building a ping pong game, one module would be responsible for the game logic, and another module would be responsible for drawing the game on the screen. Each module is a different file, which can be edited separately.
- Modules are imported from other modules using the `import` command.
- We can import any number of files to our code.



A module is a file containing Python definitions and statements.

A module is a file containing group of variables, methods, function and classes etc.

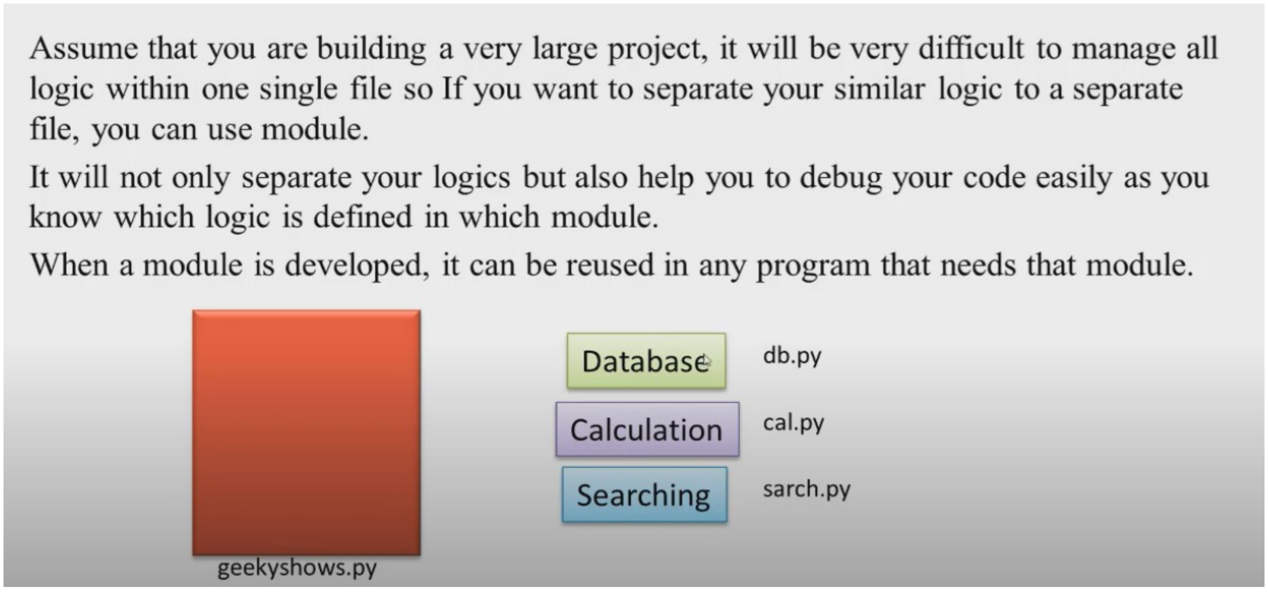
They are executed only the first time the module name is encountered in an import statement.

The file name is the module name with the suffix .py appended.

Ex:- mymodule.py

Type of Modules:-

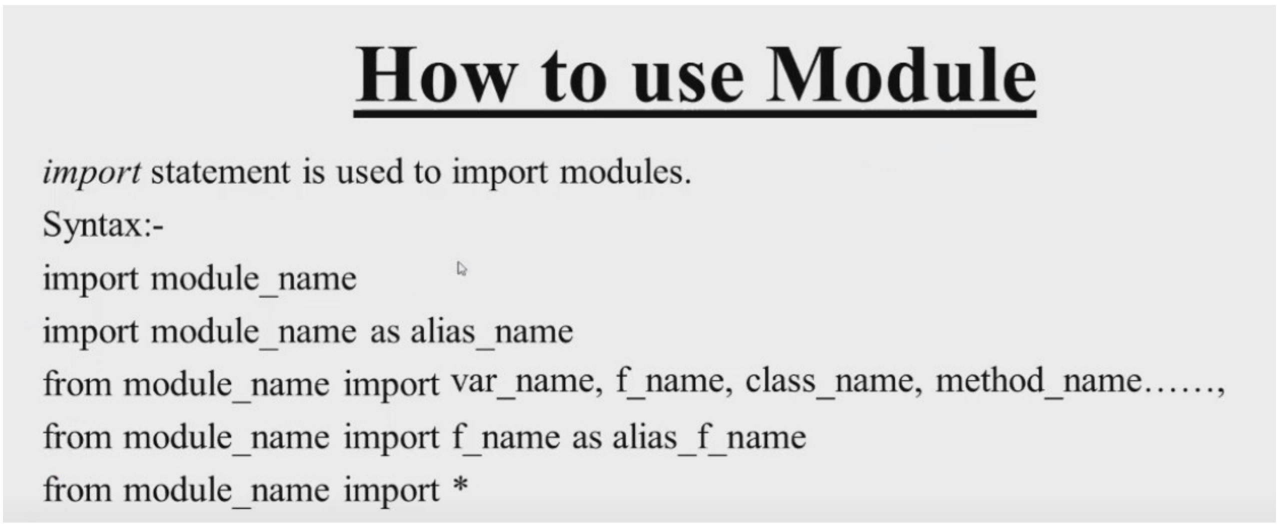
- User-defined Modules
- Built-in Modules
  Ex:- array, math, numpy, sys



Assume that you are building a very large project, it will be very difficult to manage all logic within one single file so If you want to separate your similar logic to a separate file, you can use module.

It will not only separate your logics but also help you to debug your code easily as you know which logic is defined in which module.

When a module is developed, it can be reused in any program that needs that module.



How to use Module

import statement is used to import modules.

Syntax:-

import module_name

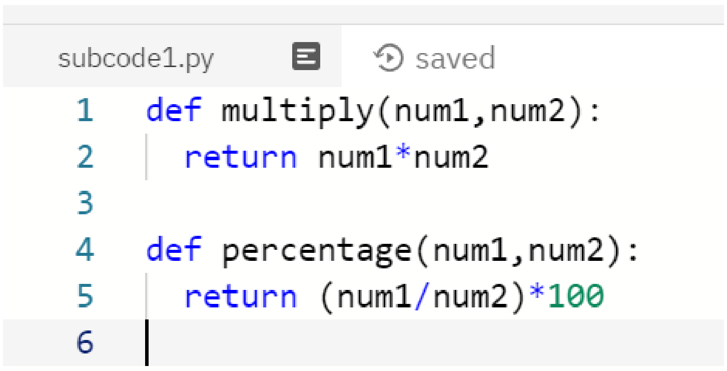
import module_name as alias_name

from module_name import var_name, f_name, class_name, method_name……,

from module_name import f_name as alias_f_name

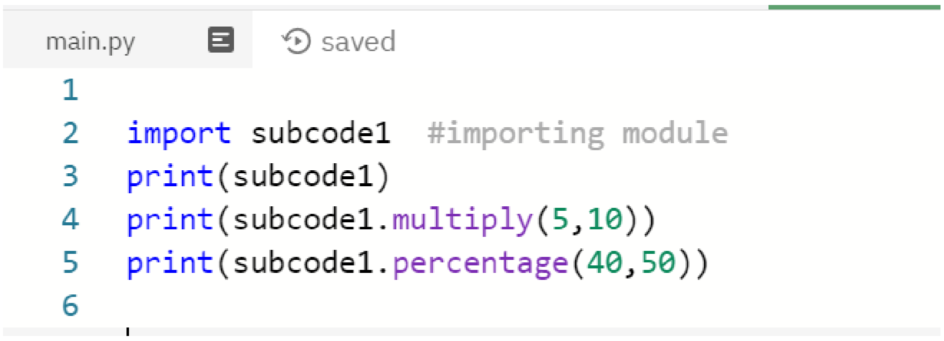
from module_name import *

File1: subcode1.py



```python
def multiply(num1,num2):
    return num1*num2

def percentage(num1,num2):
    return (num1/num2)*100

```

File2: main.py

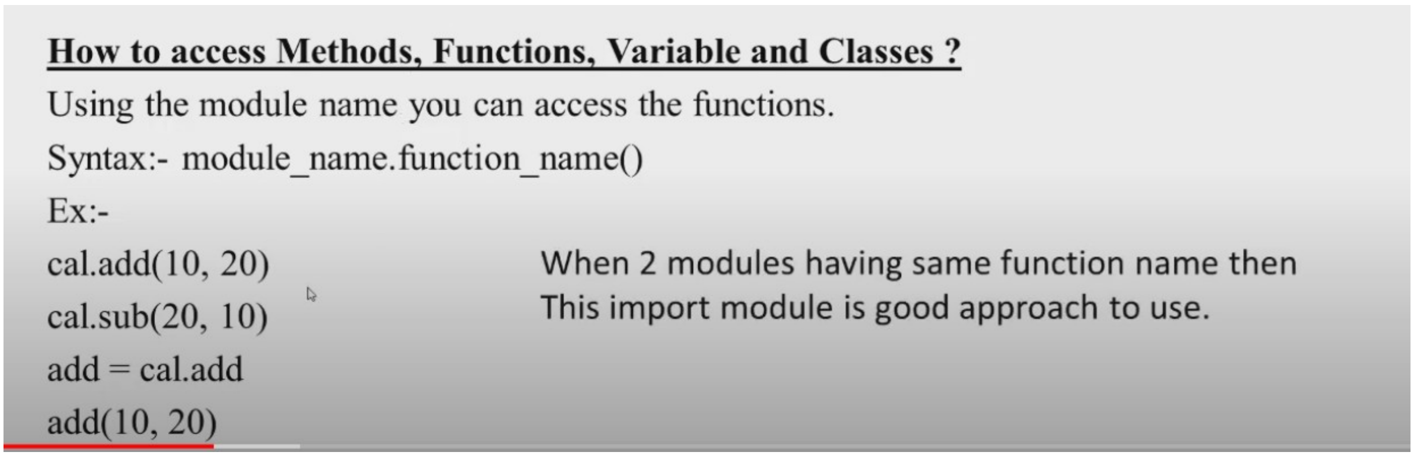


```python

import subcode1  #importing module
print(subcode1)
print(subcode1.multiply(5,10))
print(subcode1.percentage(40,50))

```

So, here we are importing subcode.py file in main.py and using its content.

- In above code we can also import item from subcode1 to main.py like this

```python
from subcode1 import multiply,percentage
```



How to access Methods, Functions, Variable and Classes ?

Using the module name you can access the functions.

Syntax:- module_name.function_name()

Ex:-

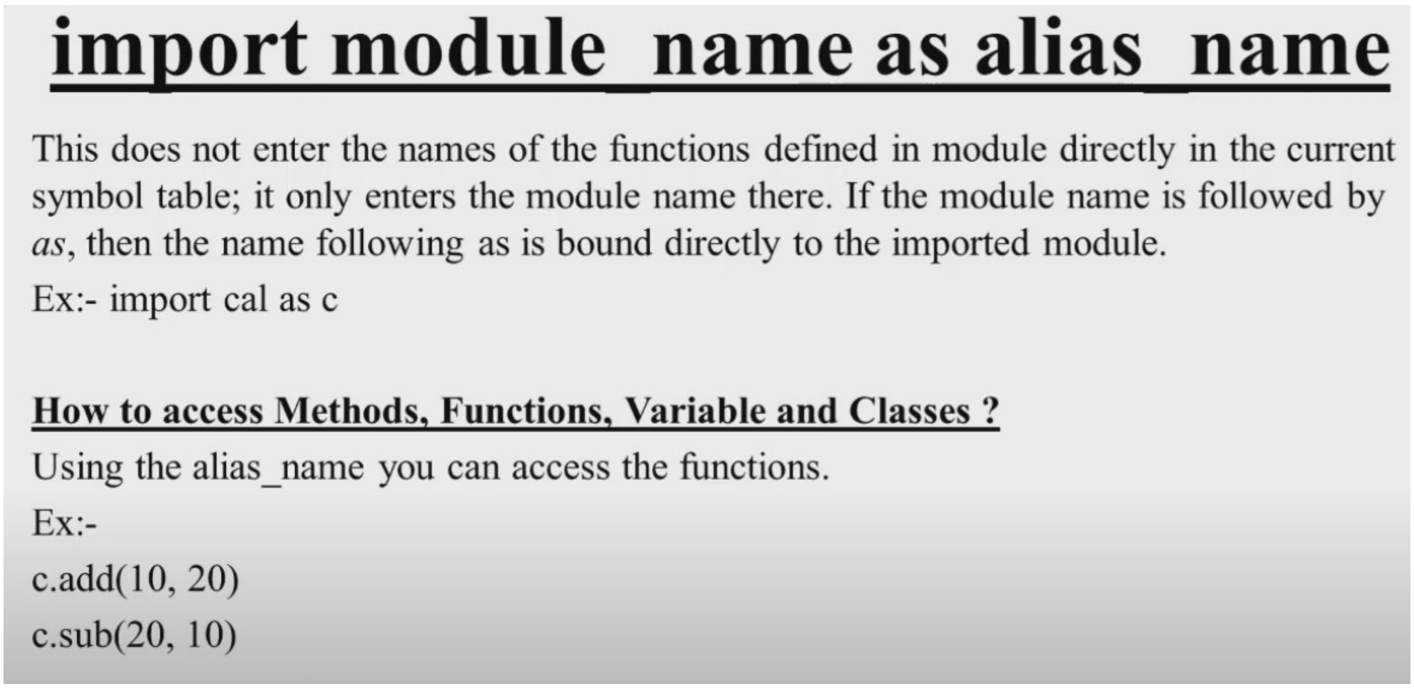
cal.add(10, 20)

cal.sub(20, 10)

add = cal.add

add(10, 20)

When 2 modules having same function name then
This import module is good approach to use.



import module_name as alias_name

This does not enter the names of the functions defined in module directly in the current symbol table; it only enters the module name there. If the module name is followed by *as*, then the name following as is bound directly to the imported module.

Ex:- import cal as c

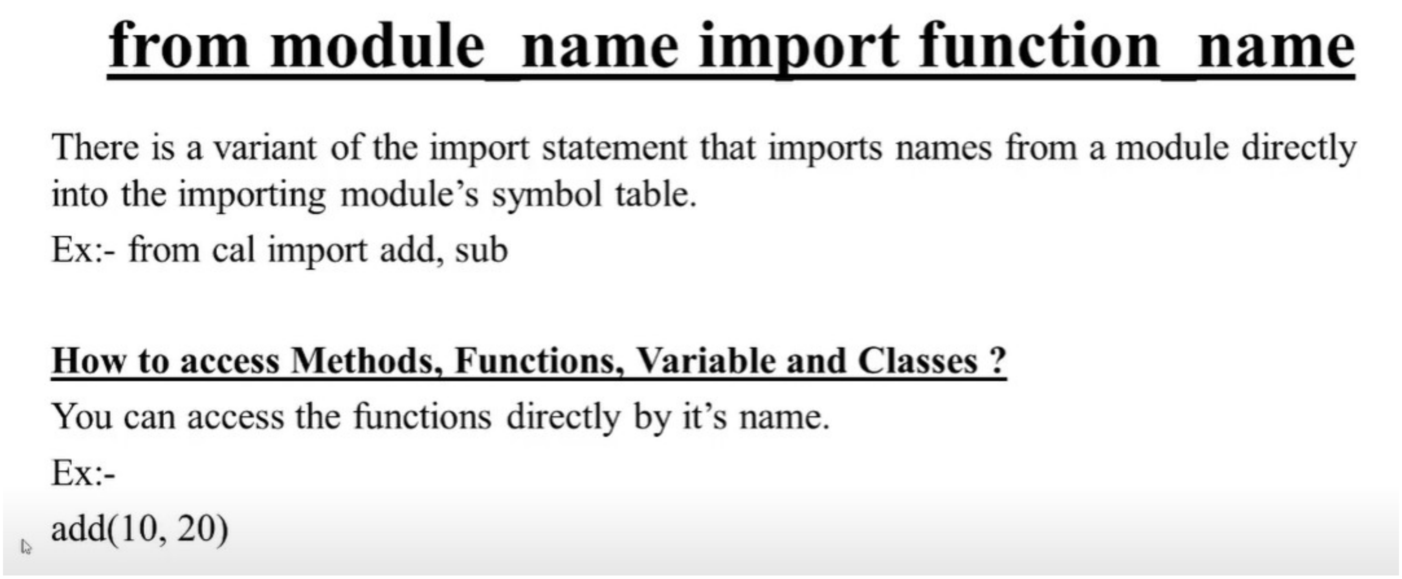
How to access Methods, Functions, Variable and Classes ?

Using the alias_name you can access the functions.

Ex:-

c.add(10, 20)

c.sub(20, 10)



from module_name import function_name

There is a variant of the import statement that imports names from a module directly into the importing module's symbol table.

Ex:- from cal import add, sub

How to access Methods, Functions, Variable and Classes ?

You can access the functions directly by it's name.

Ex:-

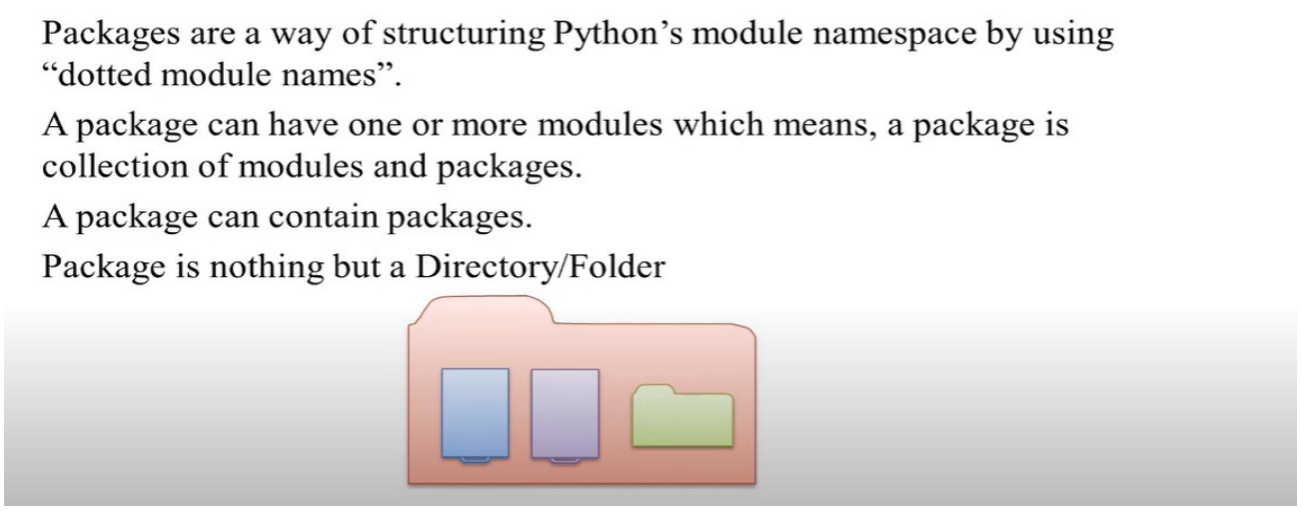
add(10, 20)

Example:

```
import fib from fibonacci
#here above we are importing fib item from fibonacci module
```

PACKAGES

A package is a collection of Python modules, i.e., a package is a directory of Python modules containing an additional __init__.py file. The __init__.py distinguishes a package from a directory that just happens to contain a bunch of Python scripts. Packages can be nested to any depth, provided that the corresponding directories contain their own __init__.py file.

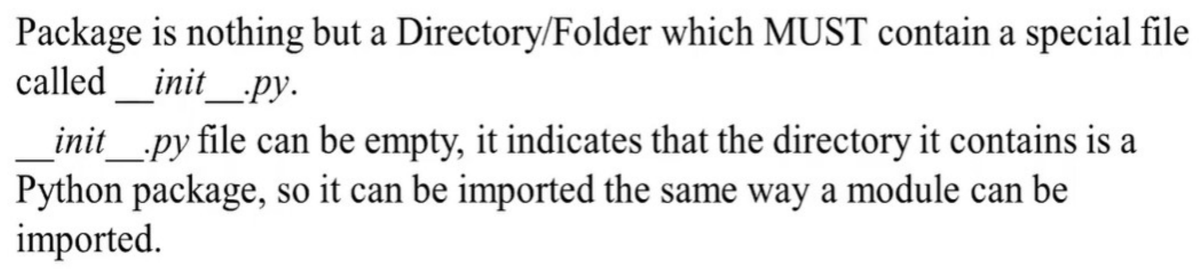


Packages are a way of structuring Python's module namespace by using "dotted module names".

A package can have one or more modules which means, a package is collection of modules and packages.

A package can contain packages.

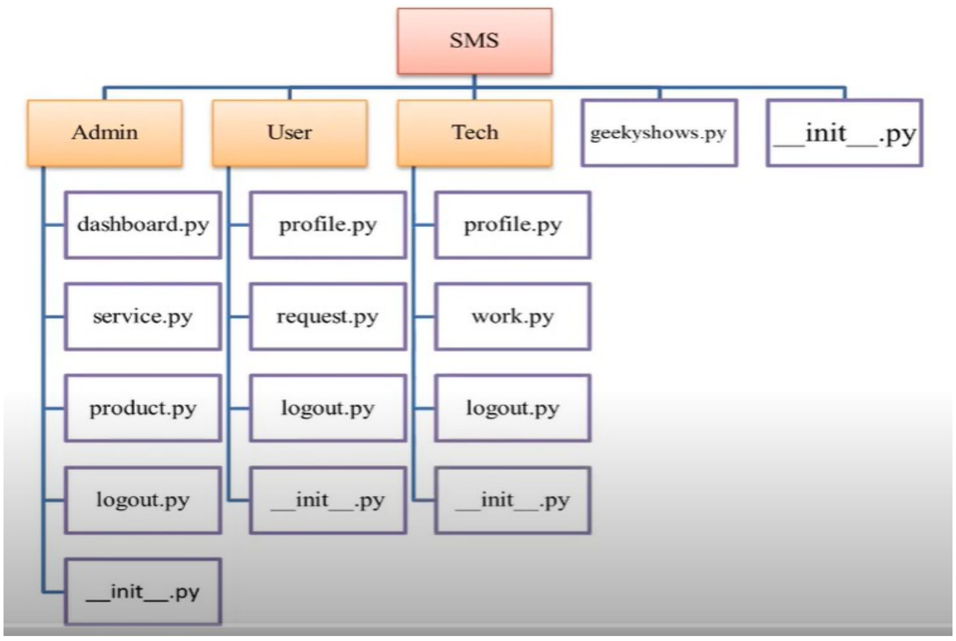
Package is nothing but a Directory/Folder



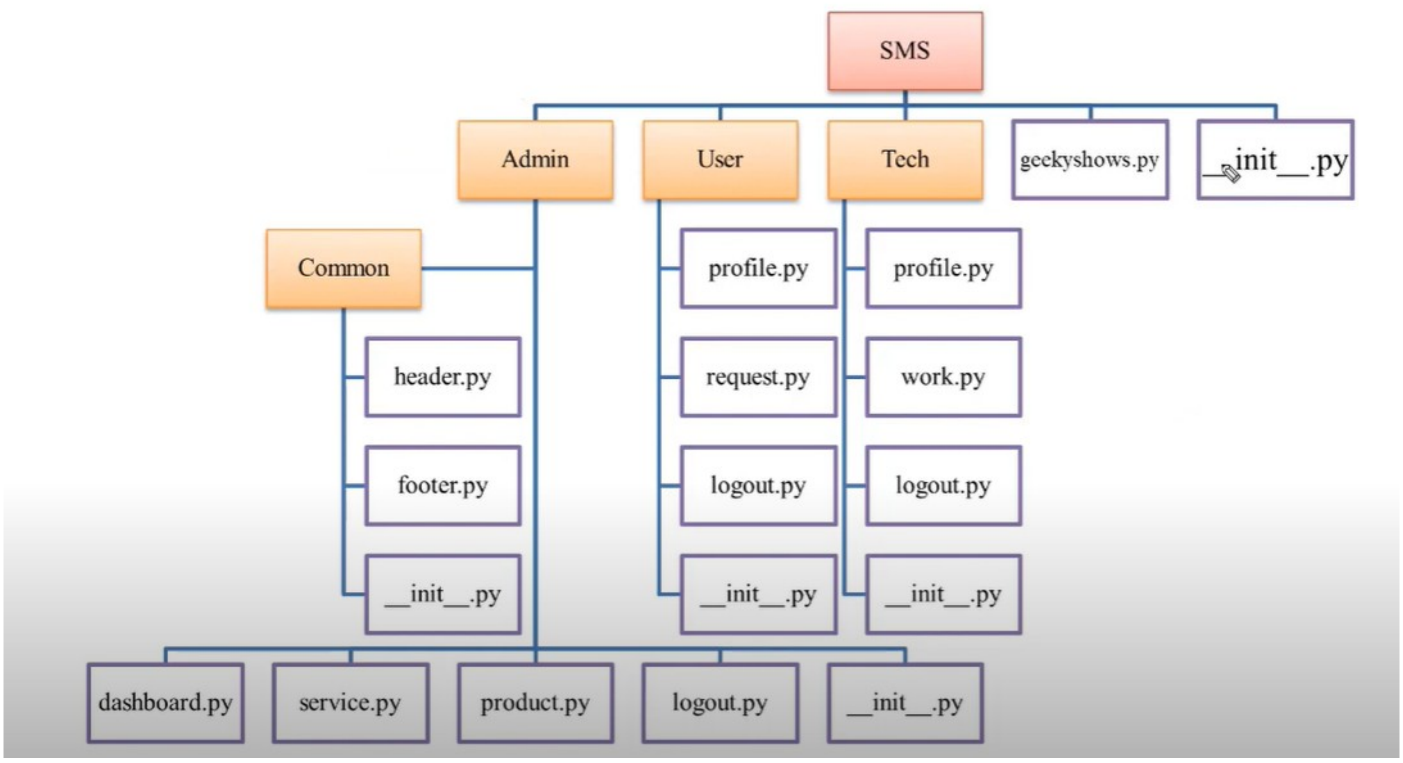
Package is nothing but a Directory/Folder which MUST contain a special file called *__init__.py*.

__init__.py file can be empty, it indicates that the directory it contains is a Python package, so it can be imported the same way a module can be imported.

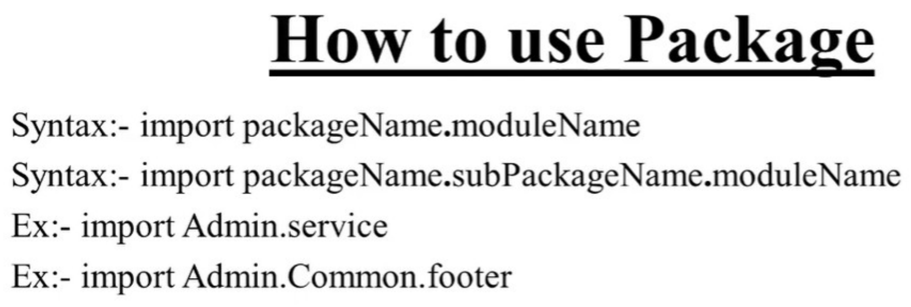
Example 1: Packages and Modules in a Sample Project SMS



Example 2: A package can have sub pacakge and that sub package can have another pacakge inside of it. Here in example, Admin is sub package of SMS package and inside Admin package have another package name as Common.



- A package should have a file name as ***__init__.py*** , so that we can able to recognize it as package.



How to use Package

Syntax:- import packageName.moduleName

Syntax:- import packageName.subPackageName.moduleName

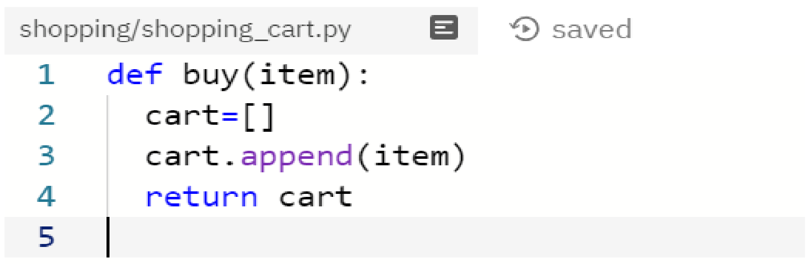
Ex:- import Admin.service

Ex:- import Admin.Common.footer

EXERCISE:

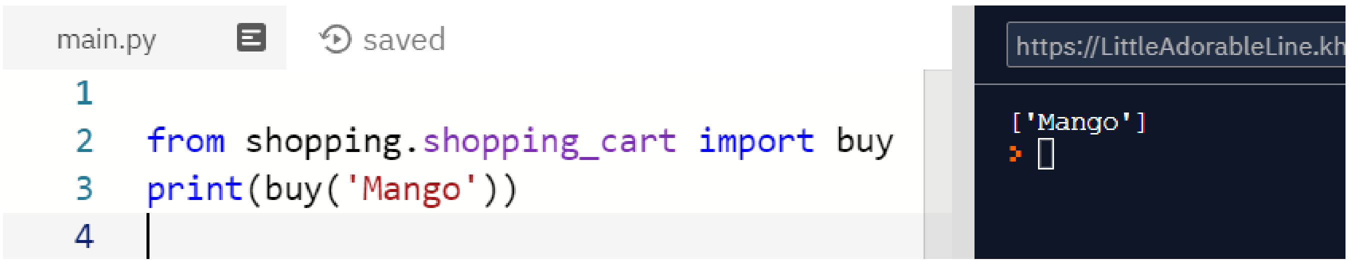
```
Create a file main.py
Create a package named as shopping, inside shopping create a file
shopping_cart having function buy () and then import it to main.py
```

File- shopping_cart.py


shopping/shopping_cart.py saved

```
def buy(item):
  cart=[]
  cart.append(item)
  return cart

```

File-main.py



main.py saved

```
from shopping.shopping_cart import buy
print(buy('Mango'))
```

https://LittleAdorableLine.kh

```
['Mango']
```

- We can import everything from particular module by giving below command-

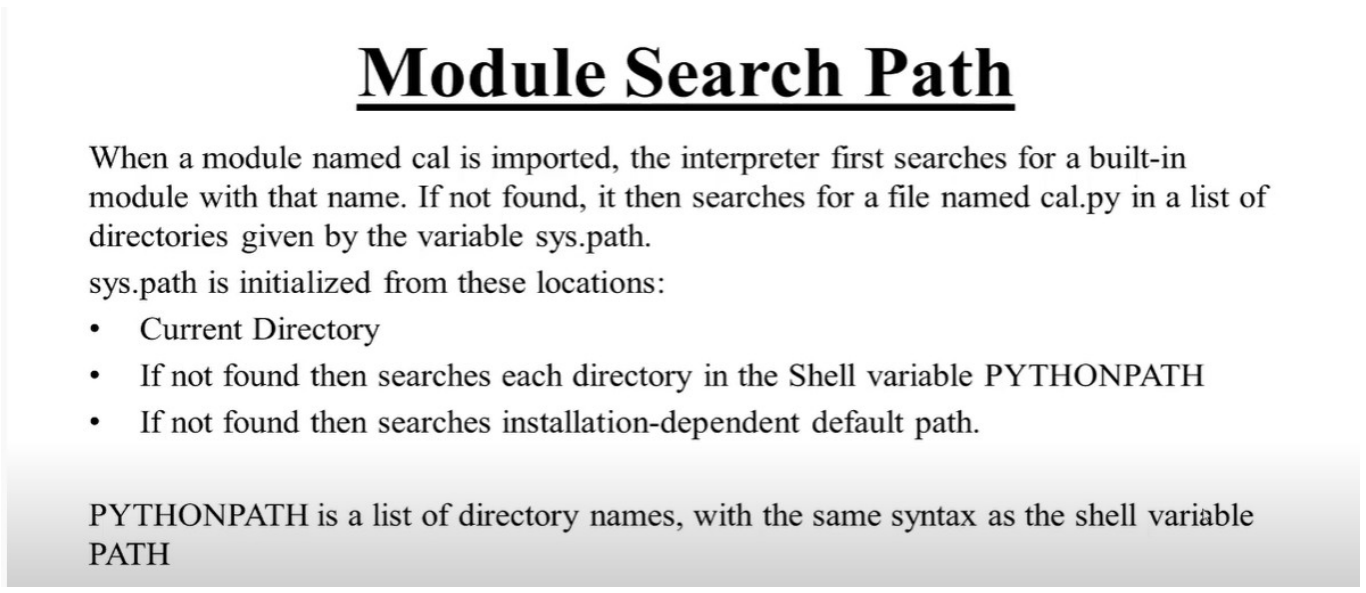
```
from module_name import *
'''here * signifies importing everything'''
```

This imports all names except those beginning with an underscore (_).
Ex:- from cal import *

Command for printing all variables and functions of a particular module

What is the correct syntax of printing all variables and function names of the "mymodule" module?

```
import mymodule

print(dir(mymodule))
```



Module Search Path

When a module named cal is imported, the interpreter first searches for a built-in module with that name. If not found, it then searches for a file named cal.py in a list of directories given by the variable sys.path.

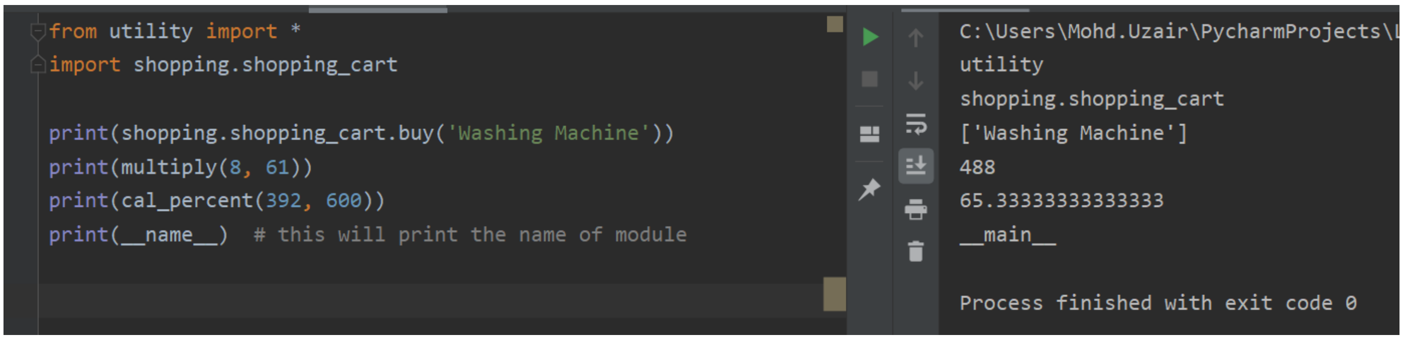
sys.path is initialized from these locations:

- Current Directory
- If not found then searches each directory in the Shell variable PYTHONPATH
- If not found then searches installation-dependent default path.

PYTHONPATH is a list of directory names, with the same syntax as the shell variable PATH

__name__ (A Special variable) in Python

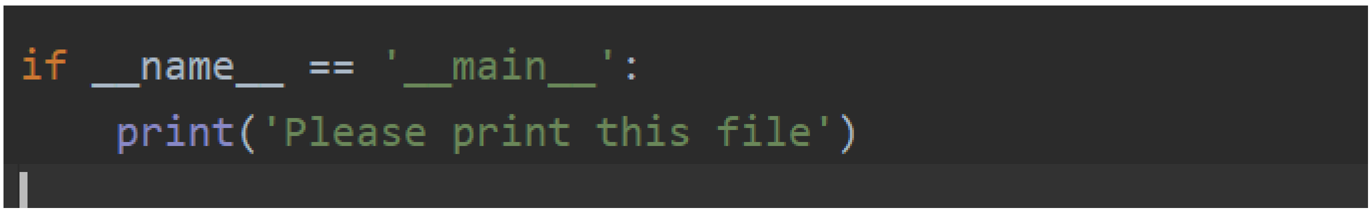
- Since there is no main () function in Python, when the command to run a python program is given to the interpreter, the code that is at level 0 indentation is to be executed. However, before doing that, it will define a few special variables. __name__ is one such special variable. If the source file is executed as the main program, the interpreter sets the __name__ variable to have a value "__main__". If this file is being imported from another module, __name__ will be set to the module's name.
- *__name__ is a built-in variable which evaluates to the name of the current module.*



```
from utility import *
import shopping.shopping_cart

print(shopping.shopping_cart.buy('Washing Machine'))
print(multiply(8, 61))
print(cal_percent(392, 600))
print(__name__)  # this will print the name of module
```

```
C:\Users\Mohd.Uzair\PycharmProjects\
utility
shopping.shopping_cart
['Washing Machine']
488
65.33333333333333
__main__

Process finished with exit code 0
```

- If we want to run only main.py and not any other module file. We put the condition:

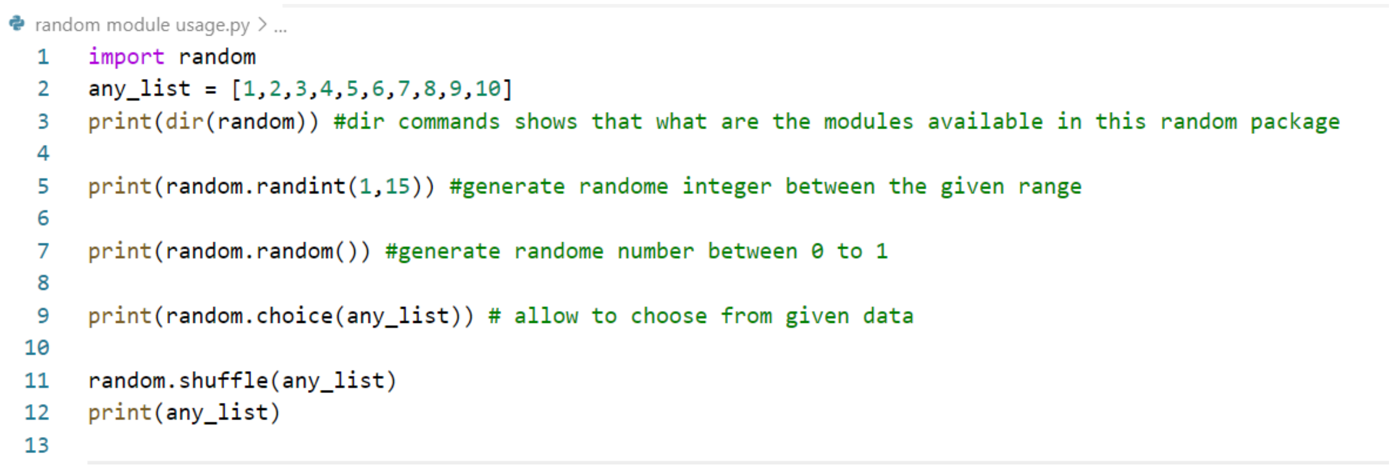
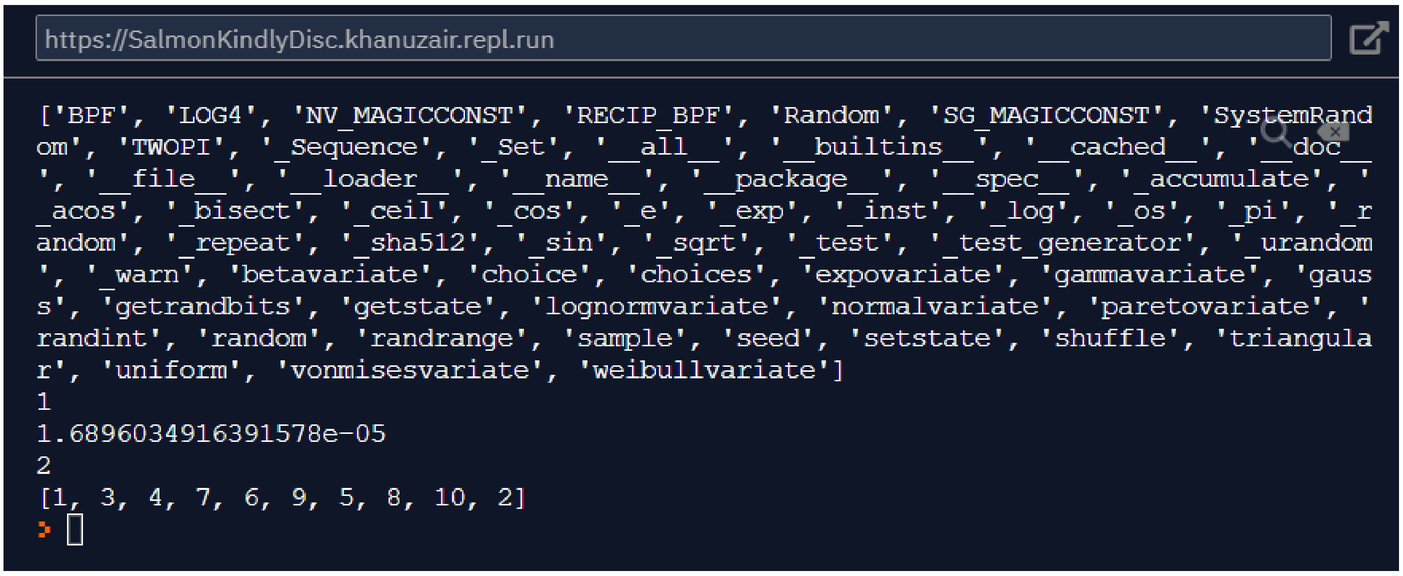


```
if __name__ == '__main__':
    print('Please print this file')
```

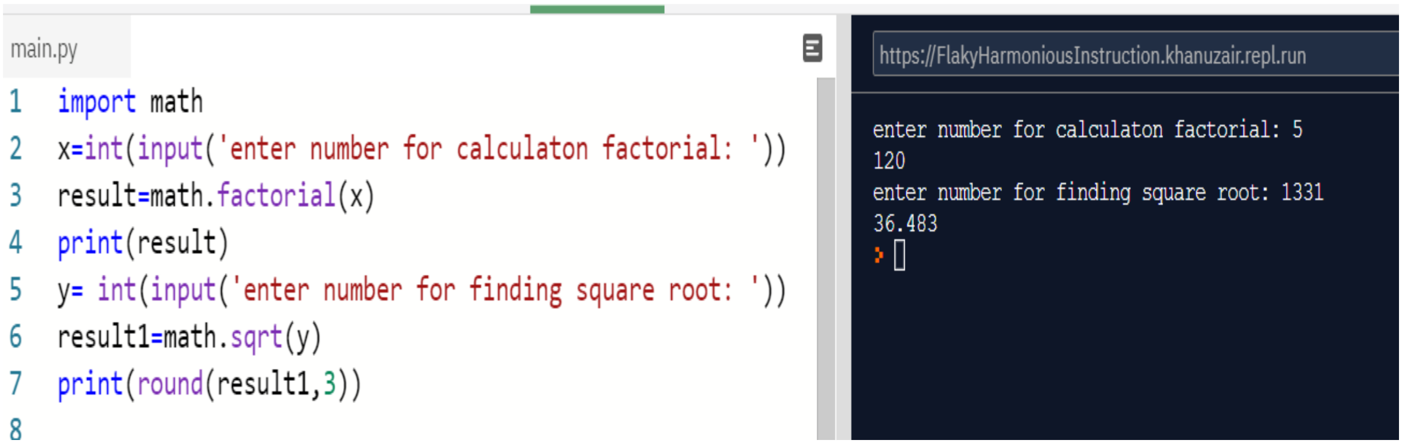
By doing this, only main.py file will run.

Built-in Packages and Built-in Modules

Example: random module


random module usage.py > ...
```python
import random
any_list = [1,2,3,4,5,6,7,8,9,10]
print(dir(random)) #dir commands shows that what are the modules available in this random package

print(random.randint(1,15)) #generate randome integer between the given range

print(random.random()) #generate randome number between 0 to 1

print(random.choice(any_list)) # allow to choose from given data

random.shuffle(any_list)
print(any_list)

```

https://SalmonKindlyDisc.khanuzair.repl.run
```
['BPF', 'LOG4', 'NV_MAGICCONST', 'RECIP_BPF', 'Random', 'SG_MAGICCONST', 'SystemRand
om', 'TWOPI', '_Sequence', '_Set', '__all__', '__builtins__', '__cached__', '__doc__
', '__file__', '__loader__', '__name__', '__package__', '__spec__', '_accumulate', '
_acos', '_bisect', '_ceil', '_cos', '_e', '_exp', '_inst', '_log', '_os', '_pi', '_r
andom', '_repeat', '_sha512', '_sin', '_sqrt', '_test', '_test_generator', '_urandom
', '_warn', 'betavariate', 'choice', 'choices', 'expovariate', 'gammavariate', 'gaus
s', 'getrandbits', 'getstate', 'lognormvariate', 'normalvariate', 'paretovariate', '
randint', 'random', 'randrange', 'sample', 'seed', 'setstate', 'shuffle', 'triangula
r', 'uniform', 'vonmisesvariate', 'weibullvariate']
1
1.6896034916391578e-05
2
[1, 3, 4, 7, 6, 9, 5, 8, 10, 2]
```

Example: math package


main.py
```python
import math
x=int(input('enter number for calculaton factorial: '))
result=math.factorial(x)
print(result)
y= int(input('enter number for finding square root: '))
result1=math.sqrt(y)
print(round(result1,3))

```
https://FlakyHarmoniousInstruction.khanuzair.repl.run
```
enter number for calculaton factorial: 5
120
enter number for finding square root: 1331
36.483
```

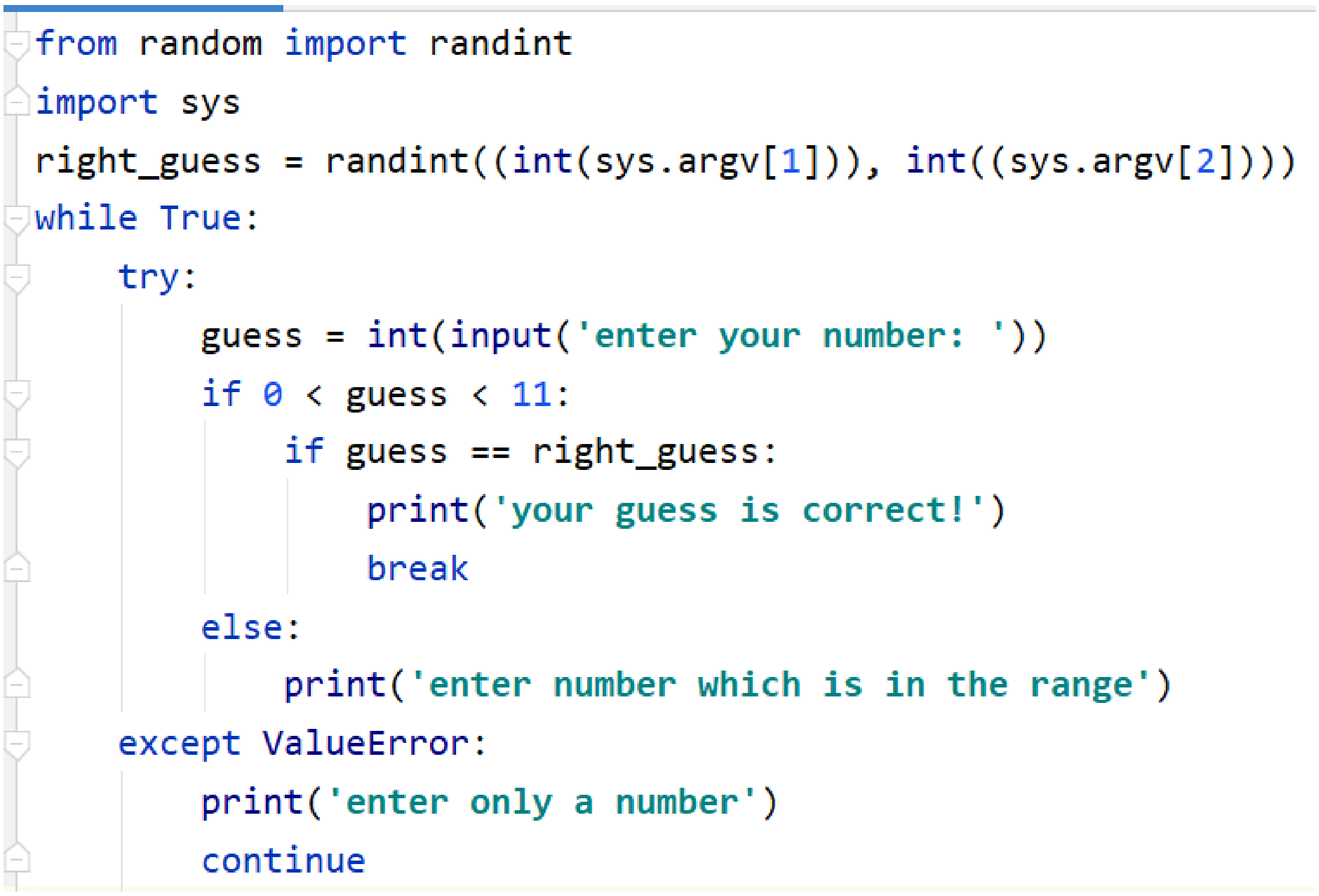
sys — System-specific parameters and functions

- This sys module provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter. It provides information about constants, functions and methods of python interpreter. It can be used for manipulating Python runtime environment.

- Command line arguments are those values that are passed during calling of program along with the calling statement. Thus, the first element of the array ***sys.argv ()*** is the name of the program itself. ***sys.argv ()*** is an array for command line arguments in Python. To employ this module named “sys” is used. ***sys.argv*** is similar to an array and the values are also retrieved like Python array.

Code Exercise:

```
Generate a random number in a range.
Ask user to guess the number
Check whether the guess number is correct or not.
```



```python
from random import randint
import sys
right_guess = randint((int(sys.argv[1])), int((sys.argv[2])))
while True:
    try:
        guess = int(input('enter your number: '))
        if 0 < guess < 11:
            if guess == right_guess:
                print('your guess is correct!')
                break
        else:
            print('enter number which is in the range')
    except ValueError:
        print('enter only a number')
        continue
```

PYTHON PACKAGE INDEX

PyPI is the official third-party software repository for Python. At the time of writing this article, PyPI was already hosting 95971 packages!

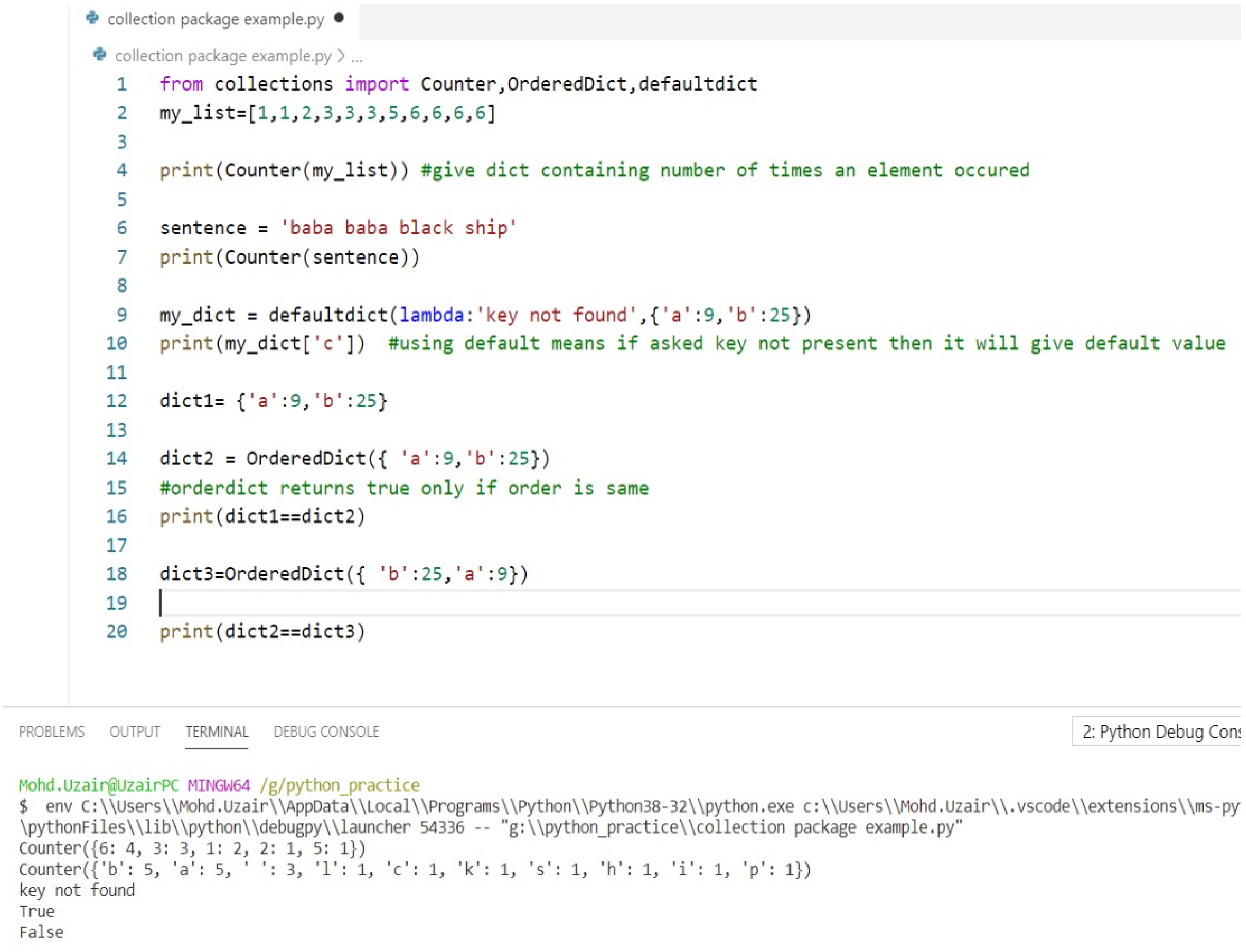
pip uses PyPI as the default source for packages and their dependencies. So whenever you type:

```
pip install package_name
```

pip will look for that package on PyPI and if found, it will download and install the package on your local system.

SOME PACKAGES IN PYTHON:

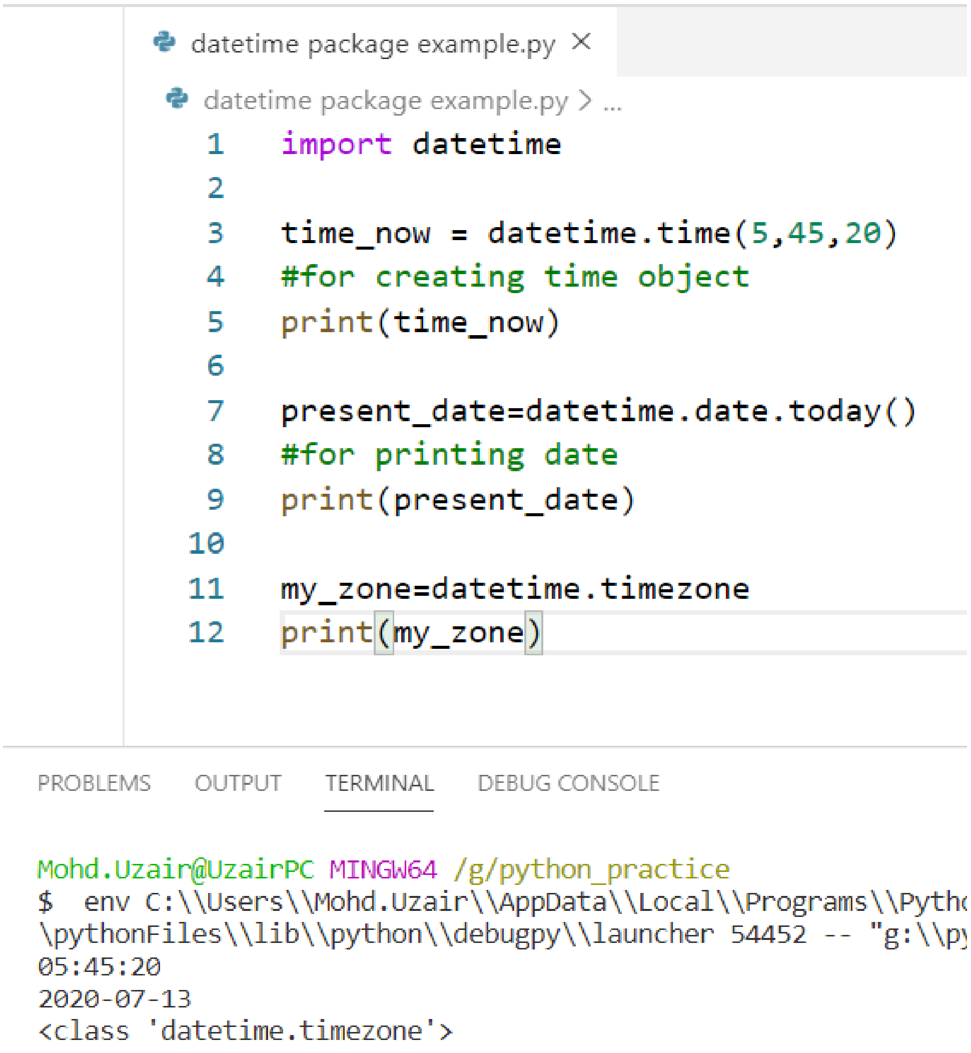
PACKAGE NAME: COLLECTIONS


collection package example.py
collection package example.py > ...

```python
from collections import Counter,OrderedDict,defaultdict
my_list=[1,1,2,3,3,3,5,6,6,6,6]

print(Counter(my_list)) #give dict containing number of times an element occured

sentence = 'baba baba black ship'
print(Counter(sentence))

my_dict = defaultdict(lambda:'key not found',{'a':9,'b':25})
print(my_dict['c'])  #using default means if asked key not present then it will give default value

dict1= {'a':9,'b':25}

dict2 = OrderedDict({ 'a':9,'b':25})
#orderdict returns true only if order is same
print(dict1==dict2)

dict3=OrderedDict({ 'b':25,'a':9})

print(dict2==dict3)
```

```
Mohd.Uzair@UzairPC MINGW64 /g/python_practice
$  env C:\\Users\\Mohd.Uzair\\AppData\\Local\\Programs\\Python\\Python38-32\\python.exe c:\\Users\\Mohd.Uzair\\.vscode\\extensions\\ms-py
\pythonFiles\\lib\\python\\debugpy\\launcher 54336 -- "g:\\python_practice\\collection package example.py"
Counter({6: 4, 3: 3, 1: 2, 2: 1, 5: 1})
Counter({'b': 5, 'a': 5, ' ': 3, 'l': 1, 'c': 1, 'k': 1, 's': 1, 'h': 1, 'i': 1, 'p': 1})
key not found
True
False
```

PACKAGE NAME: DATETIME



datetime package example.py

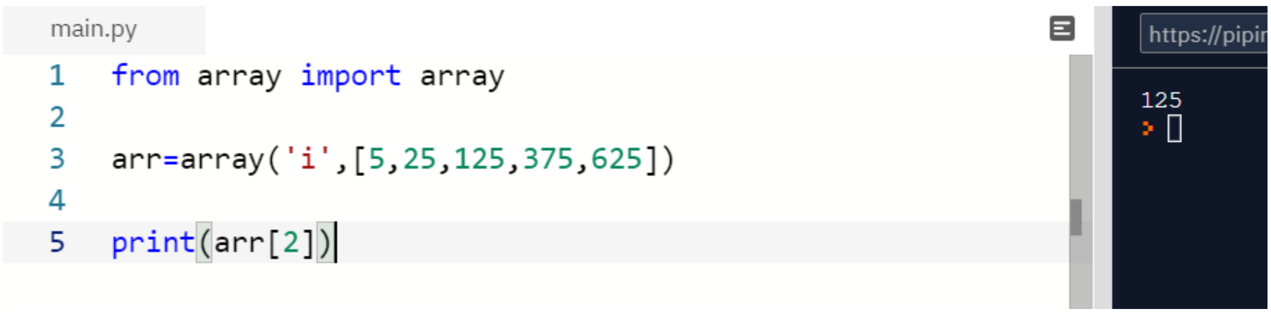
datetime package example.py > ...

```python
import datetime

time_now = datetime.time(5,45,20)
#for creating time object
print(time_now)

present_date=datetime.date.today()
#for printing date
print(present_date)

my_zone=datetime.timezone
print(my_zone)
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
Mohd.Uzair@UzairPC MINGW64 /g/python_practice
$  env C:\\Users\\Mohd.Uzair\\AppData\\Local\\Programs\\Pytho
\pythonFiles\\lib\\python\\debugpy\\launcher 54452 -- "g:\\py
05:45:20
2020-07-13
<class 'datetime.timezone'>
```

PACKAGE NAME: ARRAY



main.py

```python
from array import array

arr=array('i',[5,25,125,375,625])

print(arr[2])
```

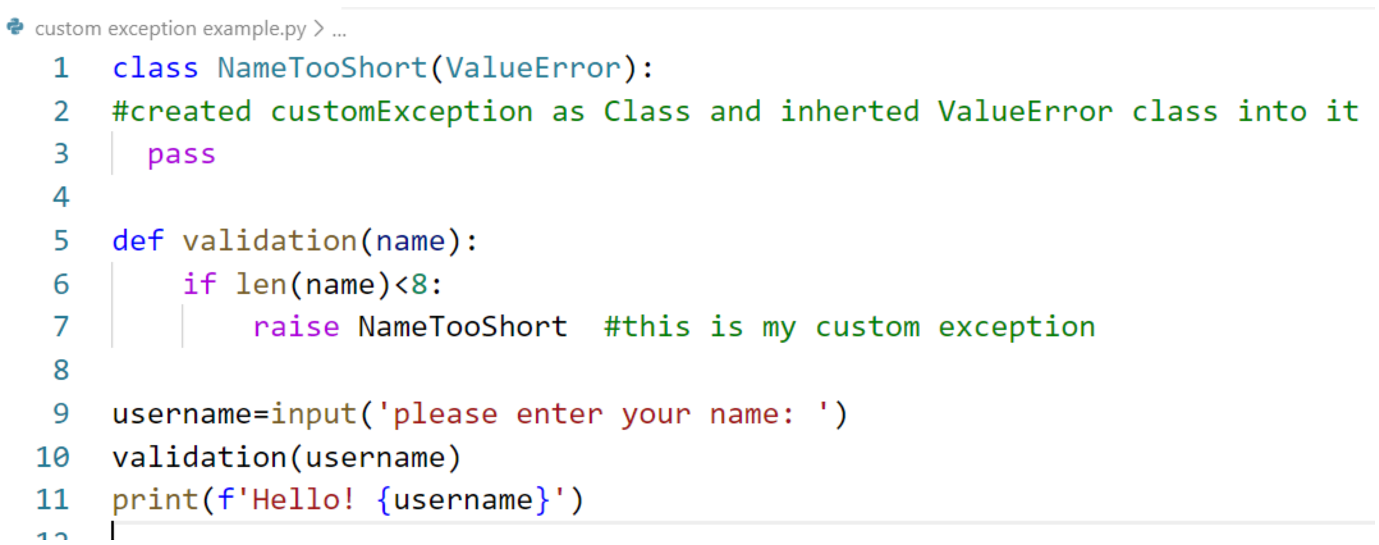
https://pipi

```
125
```

CUSTOM EXCEPTIONS

- Programmers may name their own exceptions by creating a new exception class. Exceptions need to be derived from the Exception class, either directly or indirectly. Although not mandatory, most of the exceptions are named as names that end in **"Error"** similar to naming of the standard exceptions in python.

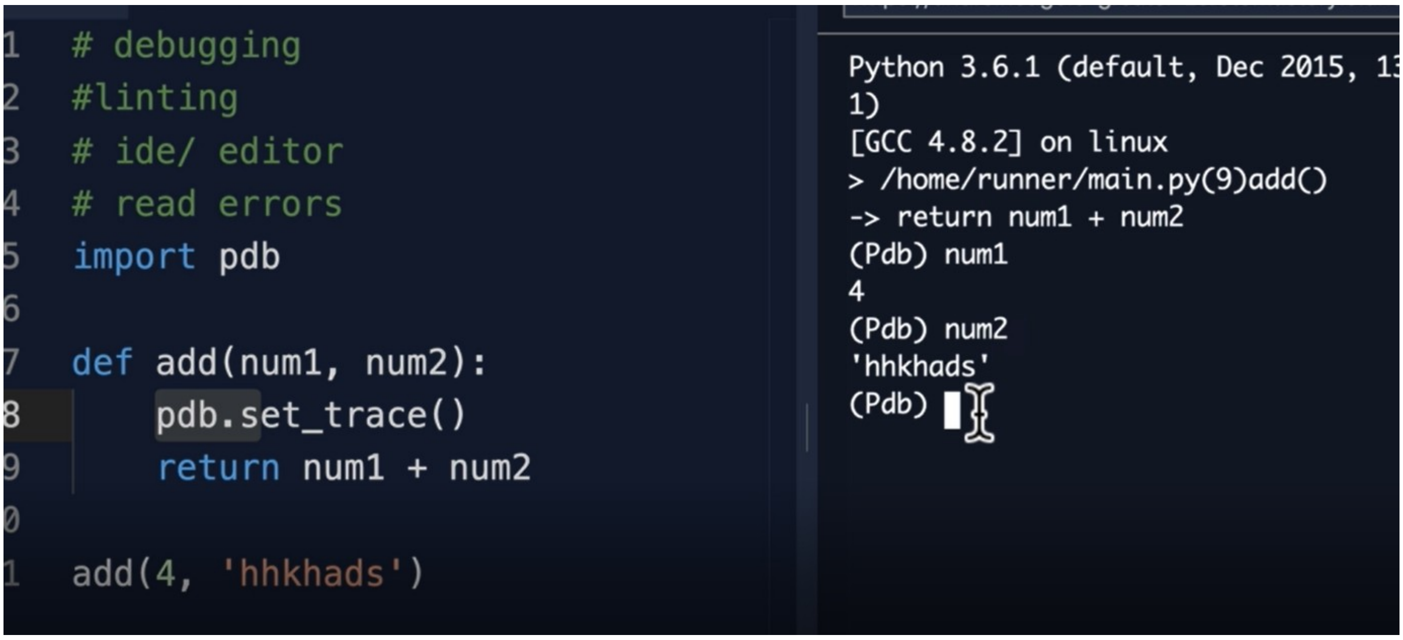
Example:



custom exception example.py > ...

```python
class NameTooShort(ValueError):
#created customException as Class and inherted ValueError class into it
  pass

def validation(name):
    if len(name)<8:
        raise NameTooShort  #this is my custom exception

username=input('please enter your name: ')
validation(username)
print(f'Hello! {username}')
```

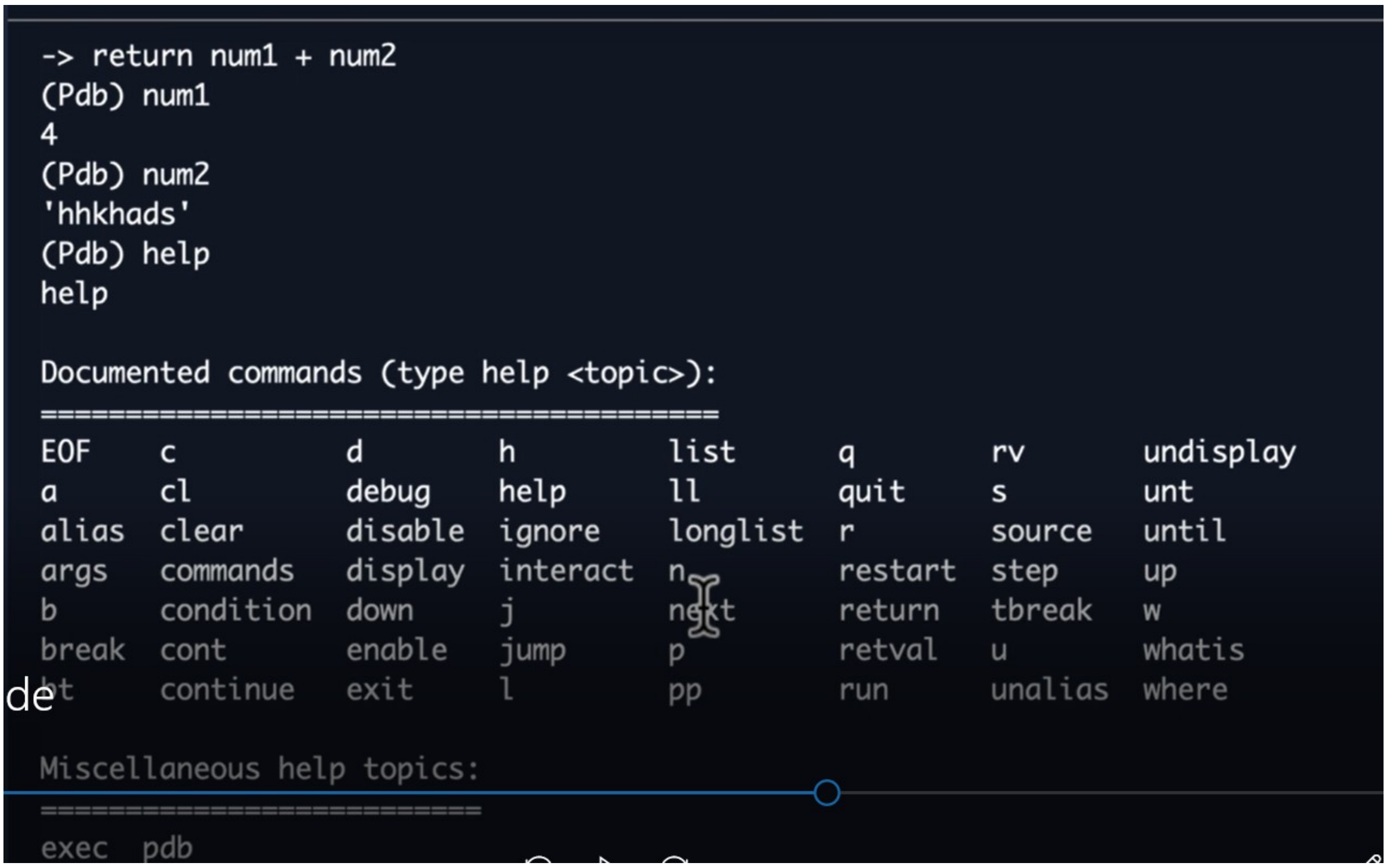
Python Debugger (pdb module for debugging code)



```python
# debugging
#linting
# ide/ editor
# read errors
import pdb

def add(num1, num2):
    pdb.set_trace()
    return num1 + num2

add(4, 'hhkhads')
```

```
Python 3.6.1 (default, Dec 2015, 13
1)
[GCC 4.8.2] on linux
> /home/runner/main.py(9)add()
-> return num1 + num2
(Pdb) num1
4
(Pdb) num2
'hhkhads'
(Pdb) 
```

When u type 'help' in pdb, you can see the various commands which you can use in pdb



```
-> return num1 + num2
(Pdb) num1
4
(Pdb) num2
'hhkhads'
(Pdb) help
help

Documented commands (type help <topic>):
========================================
EOF    c          d        h         list      q        rv       undisplay
a      cl         debug    help      ll        quit     s        unt
alias  clear      disable  ignore    longlist  r        source   until
args   commands   display  interact  n         restart  step     up
b      condition  down     j         next      return   tbreak   w
break  cont       enable   jump      p         retval   u        whatis
bt     continue   exit     l         pp        run      unalias  where

Miscellaneous help topics:
==========================
exec  pdb
```