Using With Python Libraries

Library: - Library is a collection of modules (and packages) that together cater to a specific type of applications or requirements.

Modules: - The act of partitioning a program into individual components (known as modules) is called modularity.
A module is a separate unit in itself.

Advantage of module: -
• It reduces its complexity to some degree.
• It creates a number of well defined, documented boundaries within the program.
• Its contents can be reused in other programs, without having to rewrite or recreate them.

Python module: - A python module is a file (**.py** file) containing variables, class definitions, statement and functions related to a particular task.
• The python module that come preloaded with python are called standard library modules.

Importing Modules in a Python Program

Python provides import statement to import modules in a program. The import statement can be used in two forms.
(1) To import entire module: the import <module> command
(2) To import selected objects from a module: the from <module> import <object> command

Importing Entire Module: -
The imports statement can be used to import entire module and even

for importing selected items.
To import entire module syntax is -
import <module 1 >, <module 2>....
For example:

```
import time
import decimals, fractions
```

Dot notation :- After importing a module, to access one of the functions, you have to specify the name of the module and the name of the function, separated by a dot (also known as a period) this format is called dot notation.
Syntax: -
<module-name>.<function-name> ()
This way of referring to a module's object is called dot notation.
For example:

```
import math
math.sqrt(16)
```

• You can give alias name to imported module as
Syntax :
import <module> as <alias name>

For Example :-

```
import math as a
a.sqrt (16)
```

Importing Select Objects from a Module: -
If you want to import some selected items, not all from a module, then you can use following syntax:-

from <module name >import<object name>

For example:
from math import sqrt

To Import Multiple Objects:-
If you want to import multiple objects from the module then you can use following syntax :-
from <module name >import<object name>,<object name>,<object name>....

For example: -
from math import sqrt, pi, pow

USING PYTHON STANDARD LIBRARY'S FUNCTIONS AND MODULES: -
Python's standard library offers many built in functions and modules for specialized type of functionality.

For example:-
len(),str(),type()
math module, random module , etc.

Using python's built in functions: -
The python interpreter has a number of functions built into it that are always available, you need not import any module for them.

Function name	Description
len( )	Returns the length of a sequence or iterable *e.g.*, len("abc") gives 3.
pow( )	Returns a^b when a and b are given as arguments, *e.g.*, pow(3, 4) gives 81.
str( )	Converts a number to a string, *e.g.*, str(12) will give '12' and str(12.4) will give '12.4'.
int( )	Converts an integer-convertible string to integer, *e.g.*, int('12') will give 12.
float( )	Converts a float-convertible string to integer, *e.g.*, float('12.2') will give 12.2.
range( )	Returns an immutable sequence type, *e.g.*, range(3) will give sequence 0, 1, 2.
type( )	Returns the data type of passed argument, *e.g.*, type(12) will give <class 'int'>.

Let us now talk about some more built-in mathematical functions.

Table 4.1 Some useful built-in mathematical functions

Function name	Description	Examples

abs(x)¶	Takes an integer or a floating point number as argument and returns the absolute value of a number.	>>> abs(-12)
		12
		>>> abs(-12.4)
		12.4
		>>> abs(12.4)
		12.4

divmod(a, b)	Takes two (non-complex) numbers as arguments and returns a pair of numbers consisting of their quotient and remainder.
	For integers, the result is the same as (*a* // *b*, *a* % *b*). For floating point numbers the result is (*q*, *a* % *b*)
		>>> divmod(7, 2)
		(3, 1)
		Pair of quotient and remainder returned
		>>> divmod(7.25, 2.5)
		(2.0, 2.25)
		>>>

sum(iterable)
sum(iterable,arg)
	Returns sum of the items of an ***iterable*** from left to right and returns the total.
	With two arguments iterable and arg, it returns the sum of the items of an ***iterable*** and the ***arg's*** value.
	The ***iterable's*** items are normally numbers, and the ***arg*** value should be a number.
	(Recall that all sequence types are iterable.)
		>>> sum([2, 3, 4])
		9
		Sum of the elements of the iterable returned
		>>> sum((2.5, 6, 4.2))
		12.7
		>>> sum([2, 3, 4], 5)
		14
		Sum of the elements of the iterable along with the argument value returned
		>>> sum([2, 3, 4], 8.3)
		17.3

max(iterable)
max(arg1, arg2,….)
	Returns the largest item in an iterable / sequence or the largest of two or more arguments.
	If one positional argument is provided, it should be an iterable. The largest item in the iterable is returned.
		>>> max(3, 5)
		5
		Maximum of two integer values returned
		>>> max([3, 5, 9,], [7])
		[7]
		Maximum of two list arguments returned – list that begins with a higher value is returned
		>>> max((3, 5, 9, 10), (7,))
		(7,)
		Maximum of two tuple arguments returned – tuple that begins with a higher value is returned
		>>> max((3, 5, 9, 10))
		10
		Maximum of one iterable argument returned – highest element from the tuple iterable
		>>> max([13, 15, 27, 10])
		27
		Maximum of one iterable argument returned – highest element from the list iterable

Function name	Description	Examples

min(iterable)
min(arg1, arg2,….)
	Returns the smallest item in an iterable / sequence or the smallest of two or more arguments.
	If one positional argument is provided, it should be an iterable. The smallest item in the iterable is returned.
		>>> min(3, 5)
		3 *Minimum of two integer values returned*
		>>> min([3, 5, 9,], [7])
		[3, 5, 9] *Minimum of two list arguments returned – list that begins with a higher value is returned*
		>>> min((3, 5, 9, 10), (7,))
		(3, 5, 9, 10) *Minimum of two tuple arguments returned – tuple that begins with a higher value is returned*
		>>> min([13, 15, 27, 10])
		10 *Minimum of one iterable argument returned – smallest element from the tuple iterable*
		>>> min([13, 15, 27, 10])
		10 *Minimum of one iterable argument returned – smallest element from the list iterable*

oct(<integer>)
hex(<integer>)
	returns octal string for given numbers *i.e.*, 00 + octal equivalent of number.
	returns hex string for given numbers *i.e.*, 0x + hexadecimal equivalent of number.
	Please note that *oct( ), hex( )* and *bin( )* do not return a number ; they return a string representation of converted number.
		>>> n = 24
		>>> oct(n)
		'0o30'
		>>> hex(n)
		'0x18'

Consider following program that uses two more built-in functions :

- **int (<number>)**[5] truncates the fractional part of given number and returns only the integer or whole part.
- **round(<number>, [<ndigits>])** returns number rounded to *ndigits* after the decimal points. If *ndigits* is not given, it returns nearest integer to its input.

Python's built in string functions: -

That are ---

• **<str>.join (<string iterable>)** - Joins a string or character after each member of the string iterator.

(I) If the string based iterator is a string then the <str> is inserted after

every character of the string.
For example:

```
>>>"***". join ("Hello")
'H***e***l***l***o'
```

(ii) If the string based iterator is a list or tuple of string then, the given string / character is joined with each member of the list or tuple, But the list or tuple must have all member as strings otherwise Python will raise an error.

```
>>>"***". join (("Hello", "Python"))
'Hello***Python'

>>>"***". join (["Hello", "Python", "Language"])
'Hello***Python***Language'

>>>"***". join ((123,"Hello","Python"))
Error
```

• **<str>. split (<string/char>)** - Split a string based on given string or character and return a list containing split strings as members.

(i) If you do not provide any argument to split then by default it will split the give string considering whitespace as a separator.

For example:

```
>>>"I Love Python". split()
['I', 'Love', 'Python']
```

(ii) If you provide a string or a character as an argument to split (),then

the given string is divided into parts considering the given string/character as separate and separator character is not included in the split string.

For example:
>>>"I Love Python". split ("o")
['I L','ve Pyth','n']

• **<str>. replace (<word to be replaced>,<replace word>)** - Replaces a word or part of the string with another in the given string <str>.

For example:
>>>"I Love Python". replace ("Python", "Programming")

>>>"I Love Programming"

USING RANDOM MODULE: -

Python has a module namely random that provides random number generators.

To use random number generators in your Python program, you first need to import module random using any import command

import random

Some most common random number generator functions in random module are:

random () : - It returns a random floating point number N in range [0.0,1.0] , i.e., 0.0 ≤ N ≥ 1.0.

randint (a, b) : - It returns a random integer N in the range (a, b) , i.e. , a ≤ N ≤ b (both range-limit are inclusive).

random.uniform(a, b) : - It returns a random floating point number N such that
a ≤ N ≤ b for a ≤ b and
b ≤ N ≤ a for b < a and

random.randrange(stop) or random.randrange(start, stop, [steps]) : - It returns a randomly selected element from rang (start, stop, step) .

USING STRING MODULE: -

Python has a module by the name string that comes with many constant and classes.
If you want to use string module, then you must import it by using import command: -
Like that: -

```
import string
```

Some useful **constants** defined in the string module are being listed below :

string.ascii_letters	it returns a string containing all the collection of ASCII letters.
string.ascii_lowercase	it returns a string containing all the lowercase ASCII letters, *i.e.*, 'abcdefghijklmnopqrstuvwxyz'.
string.ascii_uppercase	it returns all the uppercase ASCII letters, *i.e.*, 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'.
string.digits	it returns a string containing all the digits Python allows, *i.e.*, the string '0123456789'.
string.hexdigits	it returns a string containing all the hexadecimal digits Python allows, *i.e.*, the string '0123456789abcdefABCDEF'.
string.octdigits	it returns a string containing all the octal digits Python allows, *i.e.*, the string '01234567'.
string.punctuation	it returns a string of ASCII characters which are considered punctuation characters, *i.e.*, the string '!"#$%&'()*+,-./:;?@[\]^_'{|}~'

The string module also offers a utility function capwords( ) :

capwords(<str>, [sep=None])	it splits the specified string *<Str>* into words using *<Str>.split( )*. Then it capitalizes each word using *<Str>.capitalize( )* function. Finally, it joins the capitalized words using *<Str>.join( )*. If the optional second argument *sep* is absent or is *None*, it will remove leading and trailing whitespaces and all inside whitespace characters are replaced by a single space.

For example: -

```
>>>import string
>>>string.digits
'0123456789'
```

CREATING A PYTHON LIBRARY: -

Package: - A package is a collection of Python modules under a common namespace, created by placing different modules on a single directory along with some special file such (**__init__.py**).

• A library can have one or more packages and sub-packages.

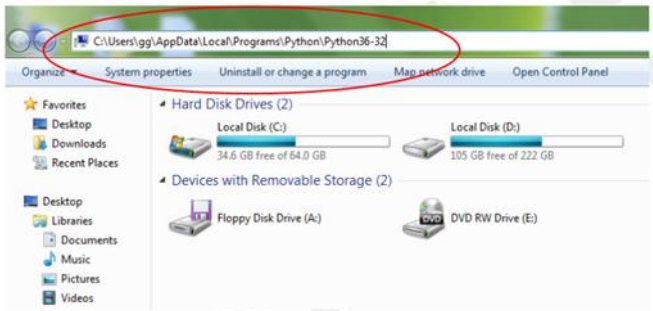
Steps of making package: -

- At first you have known a path where all files of python saved.

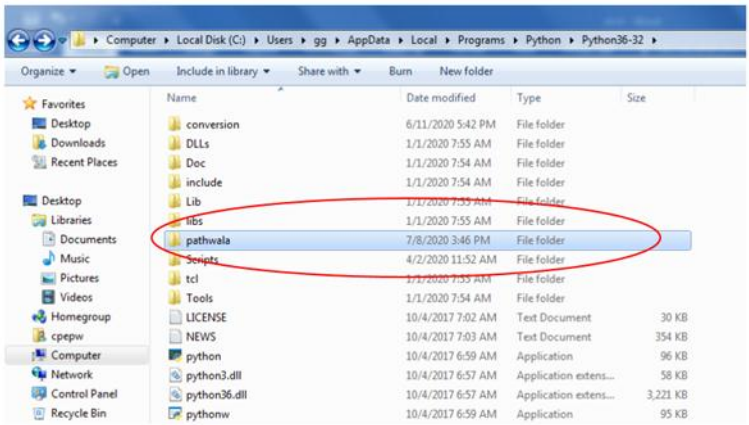
You can find the path of python by following command as shown in figure:-

```
Python 3.6.3 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.6.3 (v3.6.3:2c5fed8, Oct  3 2017, 17:26:49) [MSC v.19
00 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information
.
>>> import os
>>> a = os.getcwd()
>>> print (a)
C:\Users\gg\AppData\Local\Programs\Python\Python36-32
>>>
```

- Copy that path and paste in computer paths like this: -

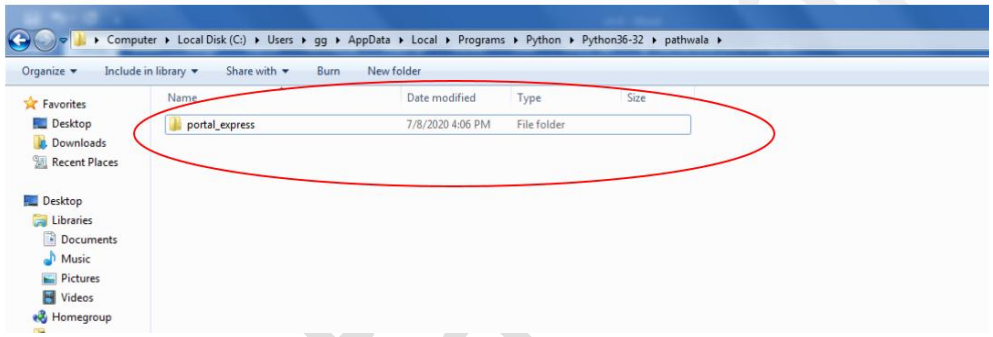


- Make a file (package name). For example we make pathwala (package) folder.



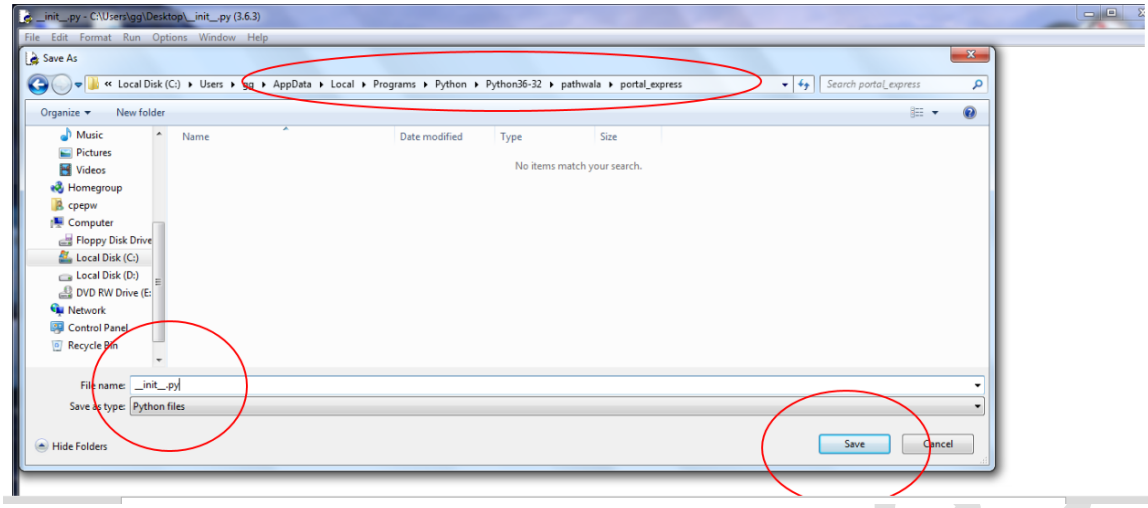
- Now make another folder (Sub Package) in package folder; if you want

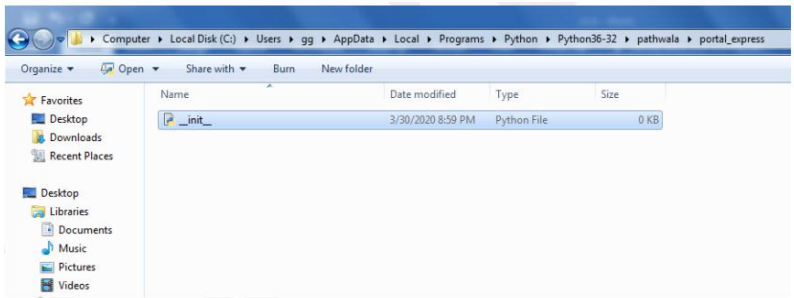
For example we make portal_express (sub package)



- Now make an empty module with name __init__.py in portal_express (sub package)

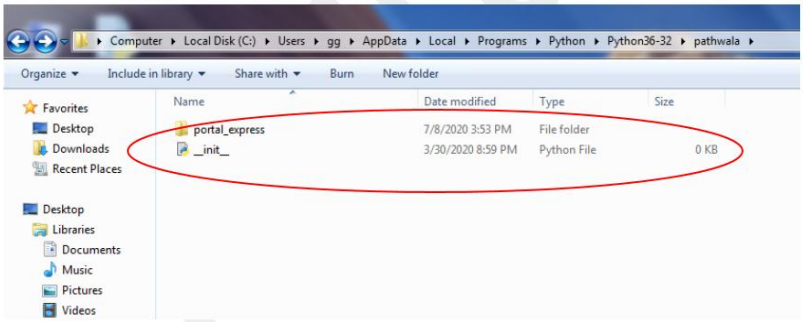
And save that module like that:-





Like that:-

- Similarly also, make an empty module with name __init__.py in pathwala(package)

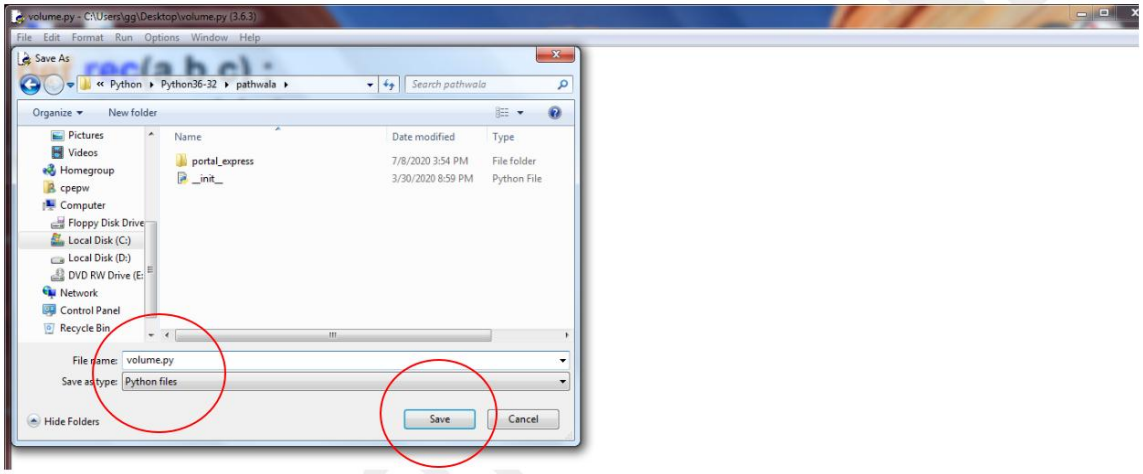


- Now make modules as you want in pathwala(Package)

For example : We make volume module as shown in figure

```
def rec(a,b,c) :
    return a * b * c
def cir(r) :
    return 22/7*r**2
```

And save that module in pathwala (Package)

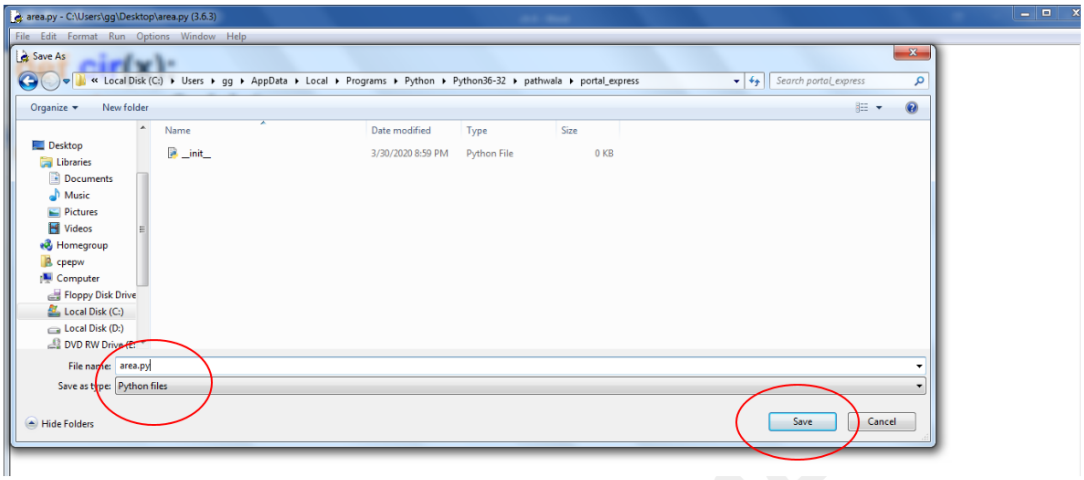


- Now make modules as you want in portal_express(Sub Package)

For example : We make area module as shown in figure

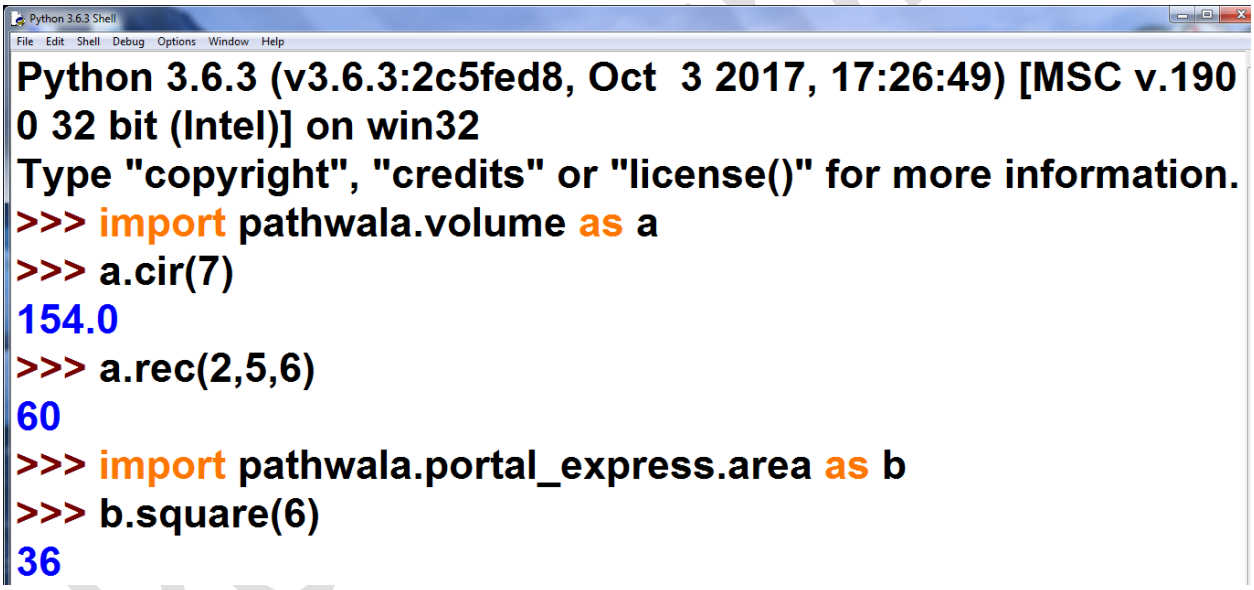
```
area.py - C:\Users\gg\Desktop\area.py (3.6.3)
File Edit Format Run Options Window Help
def cir(x):
    return 3.14 * x
def square(x):
    return x * x
```

- And save that module in portal_express (Sub Package)



- Now, Your package become ready to use in Python

For example as shown in figure: -



Thankyou!!!!!

For Sumita Arora Type C solution (Programing Question) Visit our YouTube Channel [Portal Express]

For Solution of Sumita Arora visit on [Path Wala]