USING PYTHON LIBRARIES

Create and Import Python Libraries

- Introduction
- Library?
- Importing Modules in a Python Program
- Using Python Standard Library's Function and Modules
- Creating a Python Library

MODULE ?

- A Python module is a file (.py file) containing variables, class definitions, statements and functions related to a particular task.

NEED FOR MODULES

- Modularity – The act of partitioning a program into individual components (modules).
- A module is a separate unit in itself.
- To organize code into small pieces that are easier to manage, separate code groupings called modules are created.

NEED FOR MODULES

- Python’s standard library is very extensive it offers range of modules and functions.
- Library contains built-in modules(written in C) that provide access to system functionalities such as file I/O and provide standard solutions for many problems that occur in everyday programming.

ADVANTAGES

- It reduces the complexity to some degree
- Categorization - Creates a number of well-defined, documented boundaries within the program(similar types of attributes in a single module)
- Reusable
- Grouping related code into a module makes the code easier to understand and use.
- Putting code into modules helps to import its functionality

MODULES

- A Python module is a file (.py) containing variables, definitions, statements and functions related to a particular task.
- It is independent grouping of code and data (variables, definitions, statements and functions)
- It can be re-used in other programs
- Can depend on other modules

LIBRARY OR PACKAGE

- Refers to a collection of modules that together cater to specific type of needs or applications.
- Commonly used modules that contain source code for generic needs are called libraries.
- Common Python Standard Library –
  - math
  - cmath
  - random
  - statistics
  - Urllib
  - Matplotlib

MODULE NAMING CONVENTION

- Python file with extension .py
- __name__ variable – holds the name of module being referred from __main__ module(current module)
- No keywords to be used

IMPORTING MODULE

- import statement
  - import <module_name>
  - import module1[,module2[,….moduleN]]

To access the method in the module
<module_name>.<function_name>

- from statement
  - from <module_name> import <function_name(s)>
  - from <module_name> import function_name1 [,…function_name2[,….function_nameN]]
  - from <module_name> import *

To access the method in the module
<function_name> [no need of . operator]

PROGRAM #1

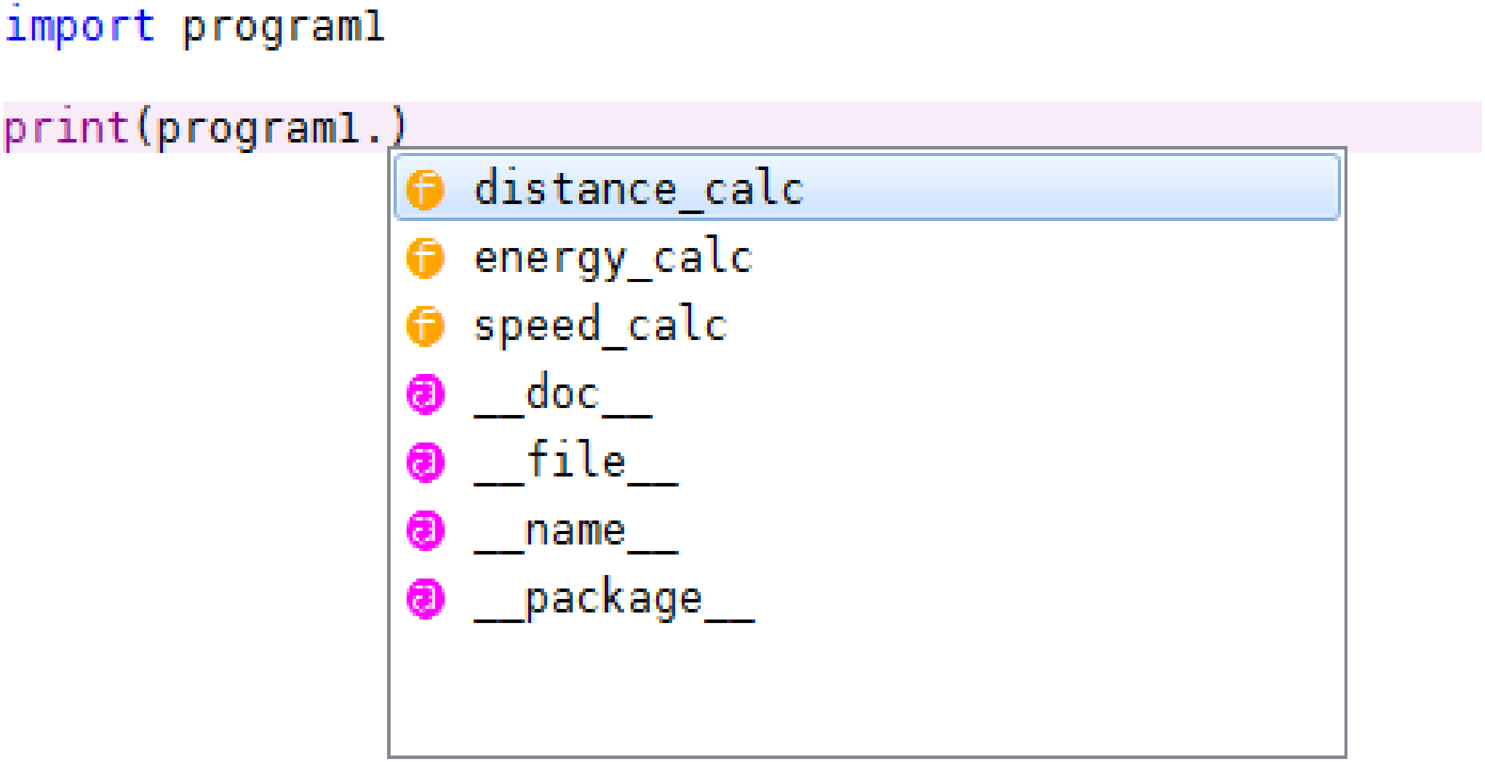
Write a program to calculate the following using modules:

a) Energy = m * g * h
b) Distance = ut + 1/2at^2
c) Speed = Distance/time

EXAMPLE 1: MODULE – SAVE THIS AS PROGRAM1.PY

```
# program to illustrate using modules

# energy = m*g*h

def energy_calc(m,g,h):
    return m*g*h

#distance = ut + 1/2at*t

def distance_calc(u,a,t):
    return u*t + 0.5*a*t**2

#speed = distance /time

def speed_calc(d,t):
    return d/t
```

HOW TO INCLUDE THIS MODULE? LET’S CREATE PROGRAM2.PY WHICH NEEDS TO USE THIS MODULE



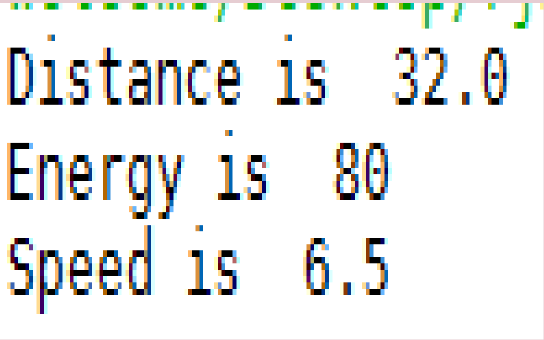
```
import program1

print(program1.)
```

PROGRAM2.PY (USING IMPORT)

EXAMPLE1(Cont..)

```
#program to invoke the module program1.py
#this is saved as program2.py
import program1

u,a,t=2,3,4
print('Distance is ',program1.distance_calc(u,a,t))

m,h,g = 2,5,8
print('Energy is ',program1.energy_calc(m,h,g))

d,t=13,2
print('Speed is ',program1.speed_calc(d,t))
```

OUTPUT



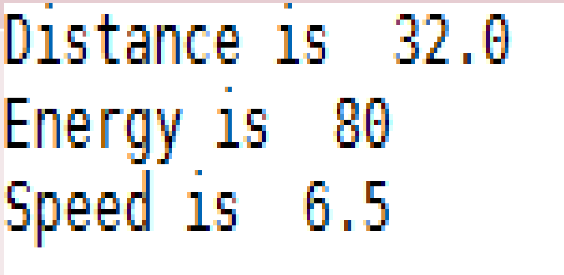
```
Distance is  32.0
Energy is  80
Speed is  6.5
```

PROGRAM2A.PY (USING FROM)

EXAMPLE2

```
#program to invoke the module program1.py
#this is saved as program2a.py
from program1 import distance_calc,speed_calc,energy_calc

u,a,t=2,3,4
print('Distance is ',distance_calc(u,a,t))

m,h,g = 2,5,8
print('Energy is ',energy_calc(m,h,g))

d,t=13,2
print('Speed is ',speed_calc(d,t))
```

OUTPUT



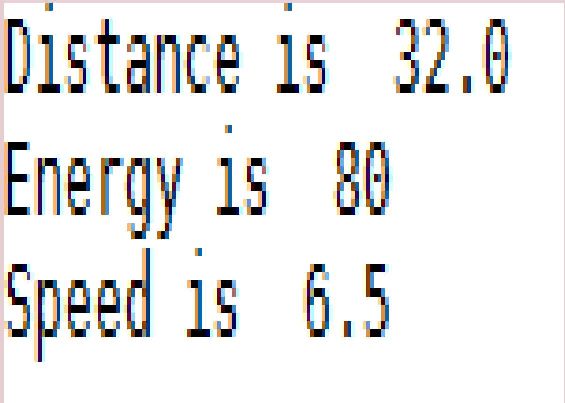
```
Distance is  32.0
Energy is  80
Speed is  6.5
```

PROGRAM2B.PY (USING FROM WITH *)

EXAMPLE2

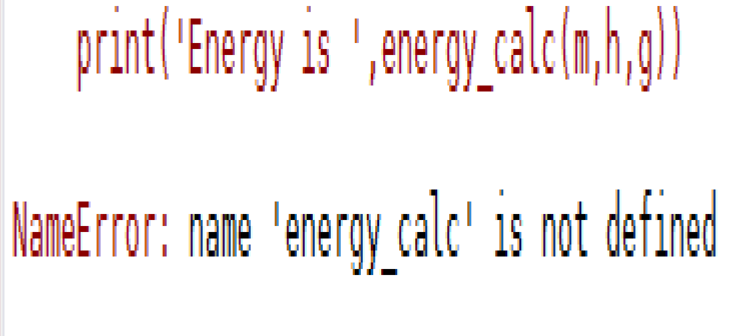
```
#program to invoke the module program1.py
#this is saved as program2b.py
from program1 import *
u,a,t=2,3,4

print('Distance is ',distance_calc(u,a,t))
m,h,g = 2,5,8

print('Energy is ',energy_calc(m,h,g))

d,t=13,2
print('Speed is ',speed_calc(d,t))
```

OUTPUT

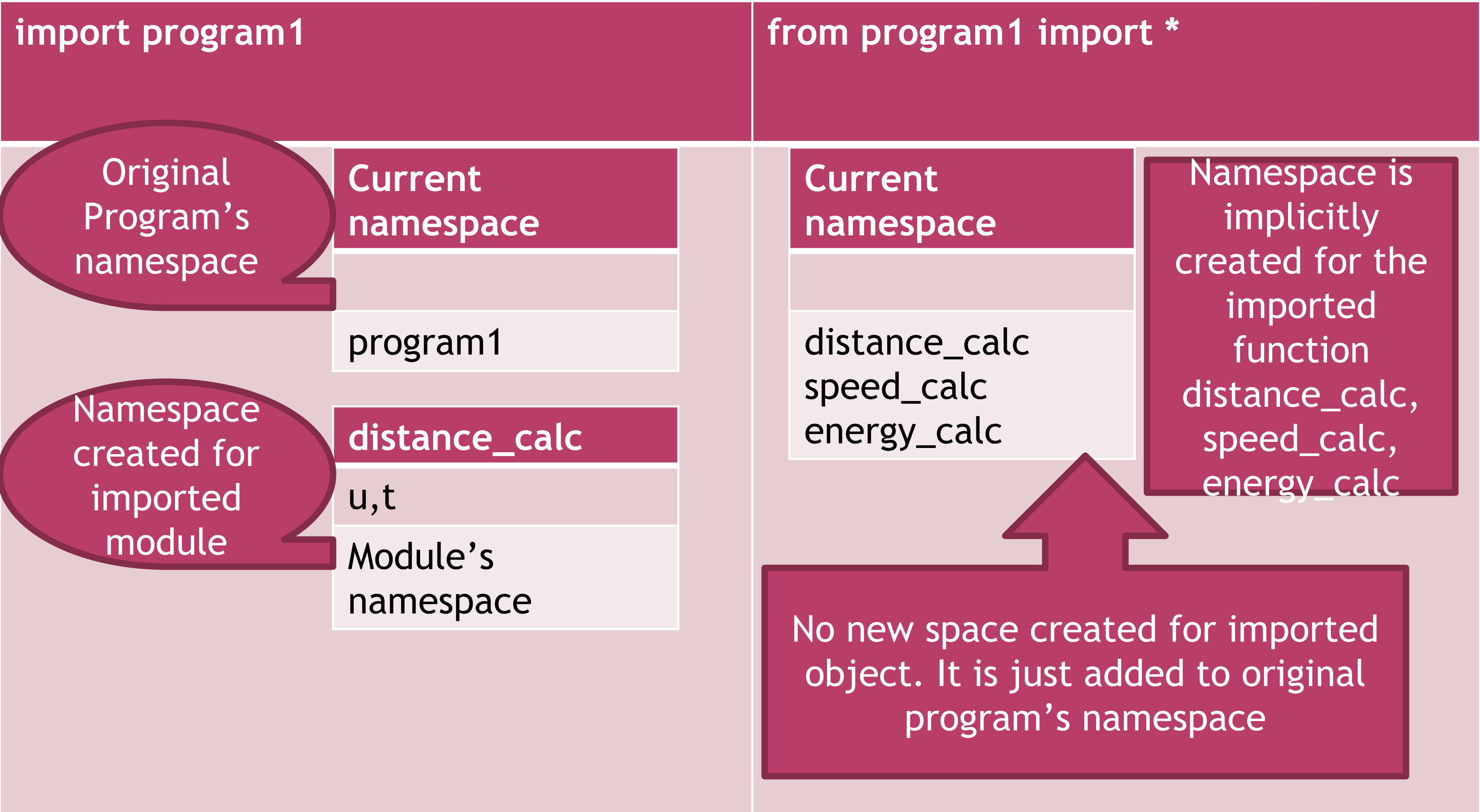


```
Distance is  32.0
Energy is  80
Speed is  6.5
```

PROGRAM3.PY
(USING FROM WITH SELECTED MODULES)

EXAMPLE3	OUTPUT
#program to invoke the module program1.py #this is saved as program3.py from program1 import distance_calc,speed_calc u,a,t=2,3,4 print('Distance is ',distance_calc(u,a,t)) m,h,g = 2,5,8 print('Energy is ',energy_calc(m,h,g)) d,t=13,2 print('Speed is ',speed_calc(d,t))	 print('Energy is ',energy_calc(m,h,g)) NameError: name 'energy_calc' is not defined

DIFFERENCE IMPORT VS FROM STATEMENTS



import program1

Original Program's namespace

Current namespace

program1

Namespace created for imported module

distance_calc

u,t

Module's namespace

from program1 import *

Current namespace

distance_calc
speed_calc
energy_calc

Namespace is implicitly created for the imported function distance_calc, speed_calc, energy_calc

No new space created for imported object. It is just added to original program's namespace

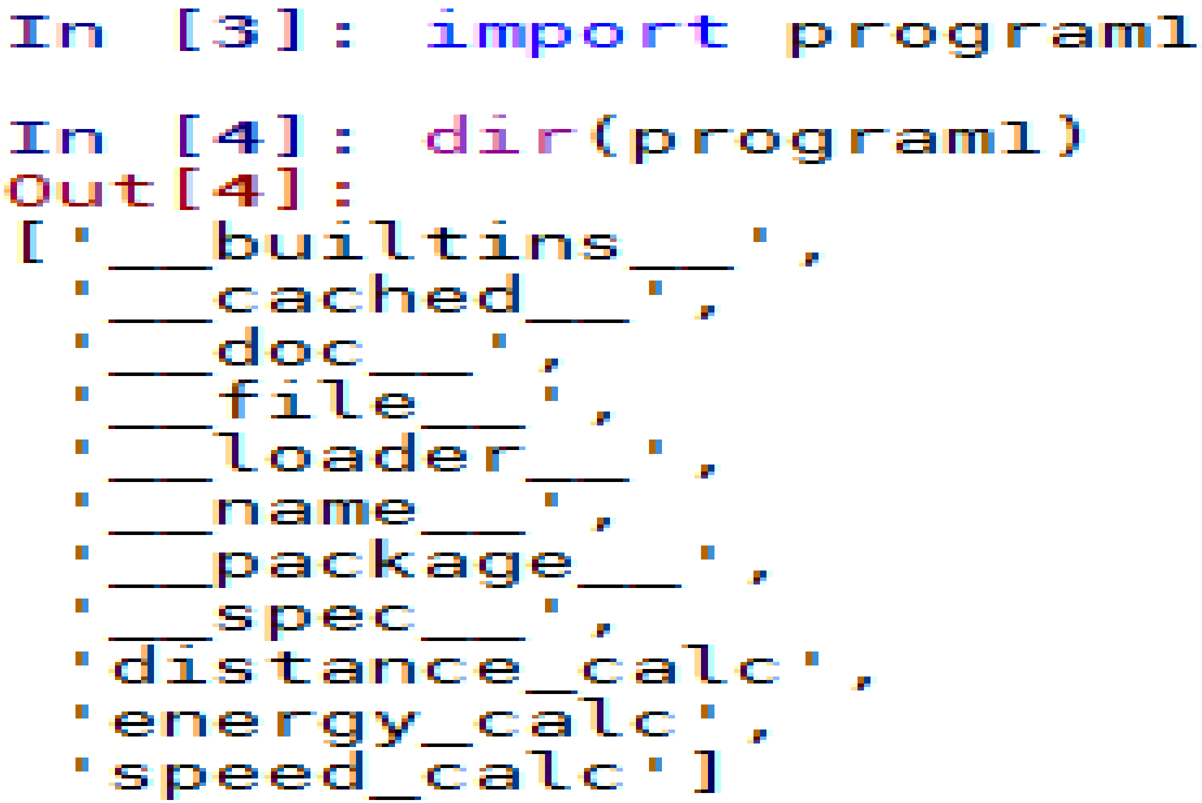
IMPORT <MODULE> COMMAND – PROCESS INVOLVED

- The code of imported module is interpreted and executed.
- Defined functions and variables created in the module are now available to the program that imported module.
- For imported module, a new namespace is setup with the same name as that of the module

FROM <MODULE> IMPORT <OBJECT> COMMAND – PROCESS INVOLVED

- The code of imported module is interpreted and executed.
- Only the asked functions and variables from the module are made available to the program.
- No new namespace is created, the imported definition is just added in the current namespace.

RETRIEVING OBJECTS FROM A MODULE



```
In [3]: import program1

In [4]: dir(program1)
Out[4]:
['__builtins__',
 '__cached__',
 '__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 'distance_calc',
 'energy_calc',
 'speed_calc']
```

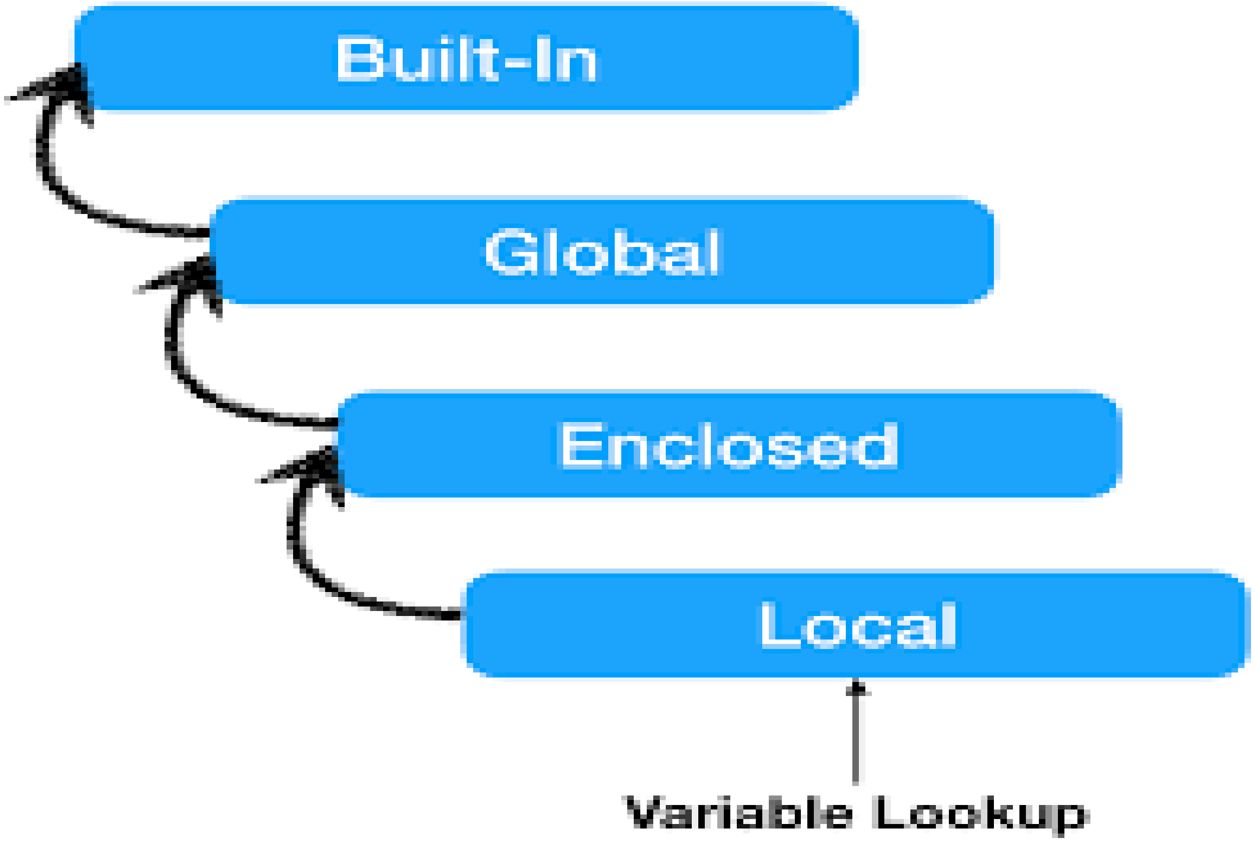
NAMESPACES IN PYTHON

- Namespace in Python is a collection of names.
- Namespace is essentially a mapping of names to corresponding objects
- Ensures that names are unique and won’t lead to any conflict
- Implemented in the form of dictionaries
- Name act as keys and objects as values.
- Types of Python namespaces – global, local, built-in

NAMESPACES IN PYTHON

- Function: Local Namespaces
- Module: Global Namespaces
- Built-in Namespaces

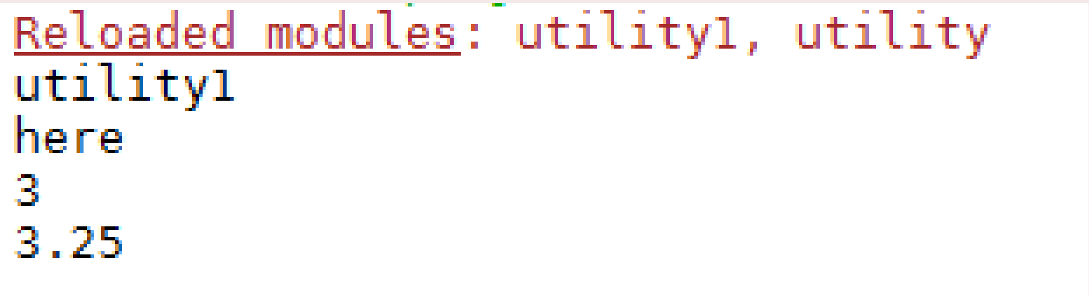
HOW DOES PYTHON DECIDE ON SCOPE OF VARIABLES?



HOW NAMESPACE WORKS ?

utility.py	utility1.py	new1.py

def divide(n1,n2):
    return n1/n2
	print(__name__)
	if __name__=='utility1' :
	    print('here')
	
	def divide(n1,n2):
	
	    return n1//n2
		from utility1 import *
		from utility import *
		print(utility1.divide(13,4))
		print(divide(13,4))



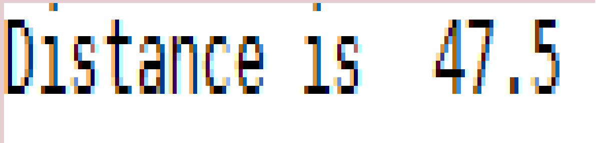
```
Reloaded modules: utility1, utility
utility1
here
3
3.25
```

MODULE ALIASING

Syntax

import <module_name> as <alias_name>

Example	Output

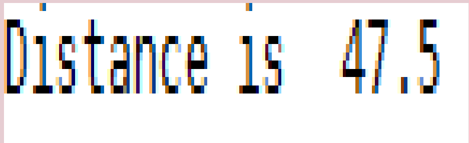
import program1 as pg
print(pg.distance_calc(2,3,5))
	
	Distance is 47.5

MEMBER ALIASING

Syntax

from <module_name> import function as <alias_name>

Example	Output

from program1 import distance_calc as d
print('Distance is ',d(2,3,5))
	
	Distance is 47.5

PACKAGE/LIBRARY

- Python package is a collection of related modules.
- The main difference between module and package is
- Package -
  - collection of several modules
  - __init__.py file(should be present)

__INIT__.PY FILE?

- The __init__.py is the first file to be loaded in a module.
- It makes Python treat directories containing it as modules.
- __init__ method is used to initialize new objects, not create them.
- The sole purpose of __init__ is to initialize the values of instance members for the new object.(class creation – OOP concept)
- The file __init__.py in a folder, indicates it an importable Python package.
- Without __init__.py, a folder is not considered a Python package.
- Standard Python libraries – Datetime Library, Math library, String library etc.

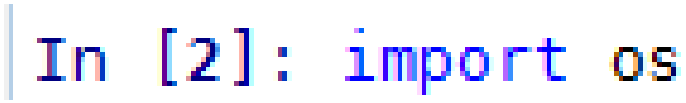
STEPS TO CREATE A PACKAGE/LIBRARY

- Decide about the basic structure of the package.

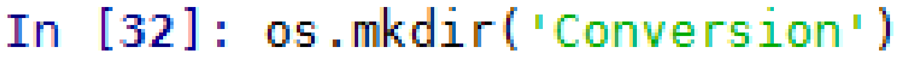
  - Conversion
    - __init__.py
    - lengthconversion.py
      - feettoinches()
      - inchestofeet()
    - massconversion.py
      - kgtotonne()
      - tonnetokg()

HOW TO MAKE DIRECTORY?

In spider environment type the following



```
In [2]: import os
```

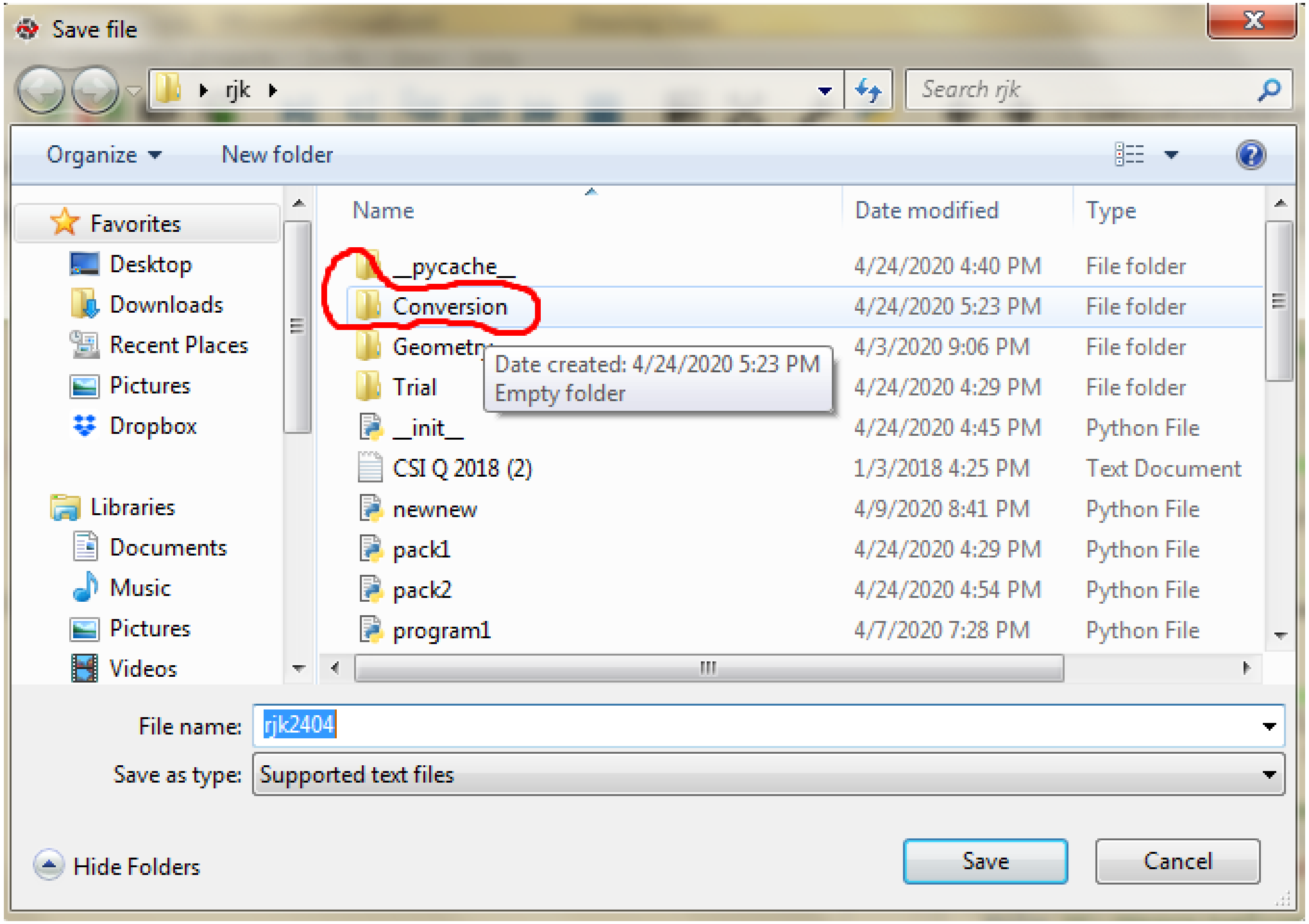


```
In [32]: os.mkdir('Conversion')
```

CREATE LENGTHCONVERSION.PY PROGRAM

```
#lengthconversion.py

def feettoinches(feet):
    return feet // 12

def inchestofeet(inches):
    return inches * 12
```

DOUBLE CLICK ON THE FOLDER CONVERSION

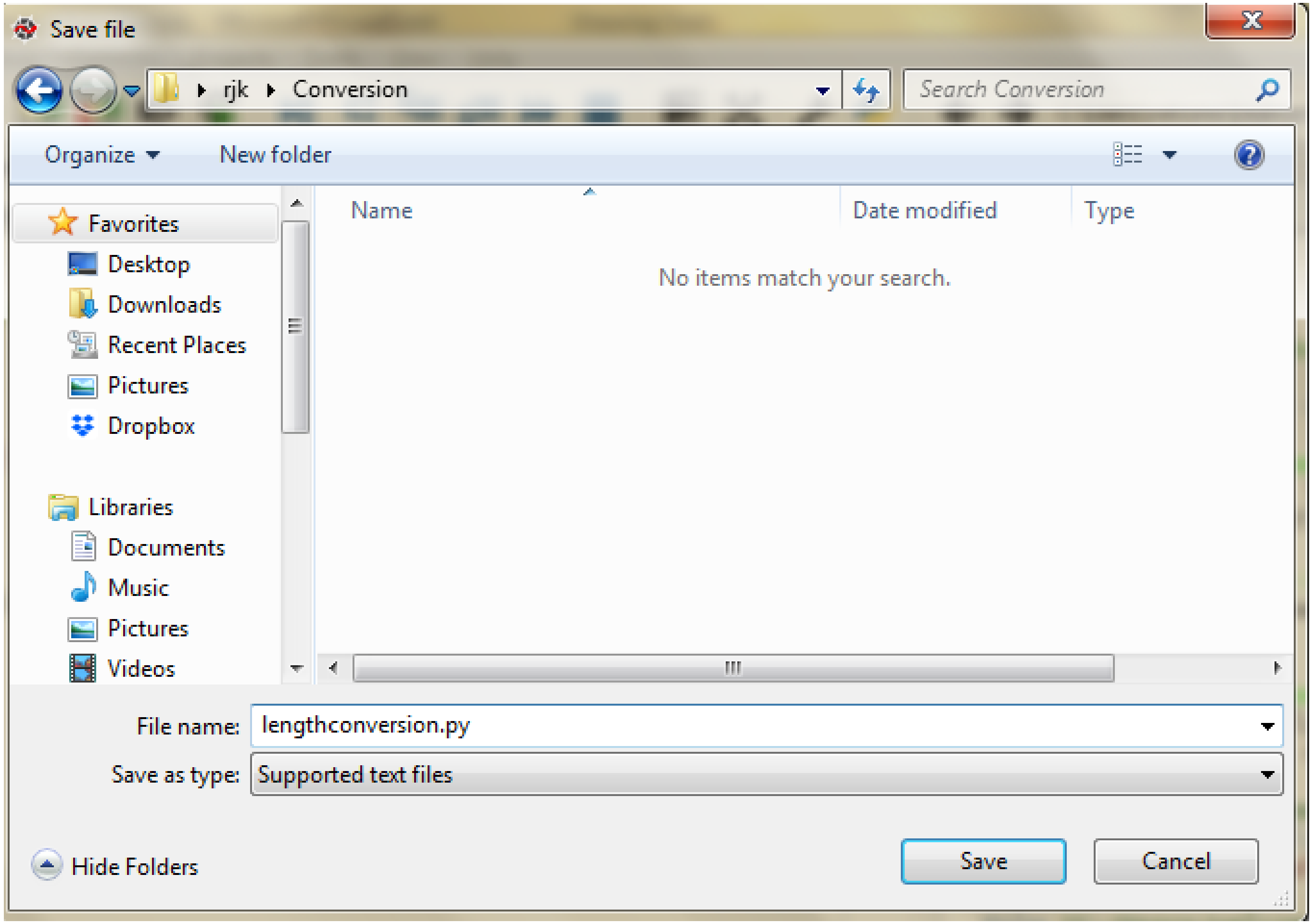


Name	Date modified	Type
__pycache__	4/24/2020 4:40 PM	File folder
Conversion	4/24/2020 5:23 PM	File folder
Geometry	4/3/2020 9:06 PM	File folder
Trial	4/24/2020 4:29 PM	File folder
__init__	4/24/2020 4:45 PM	Python File
CSI Q 2018 (2)	1/3/2018 4:25 PM	Text Document
newnew	4/9/2020 8:41 PM	Python File
pack1	4/24/2020 4:29 PM	Python File
pack2	4/24/2020 4:54 PM	Python File
program1	4/7/2020 7:28 PM	Python File

File name: rjk2404

Save as type: Supported text files

AFTER THAT SAVE THE FILE NAME AS LENGTHCONVERSION.PY IN CONVERSION FOLDER


Save file

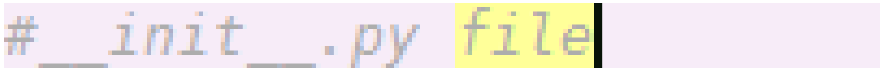
rjk ▸ Conversion

No items match your search.

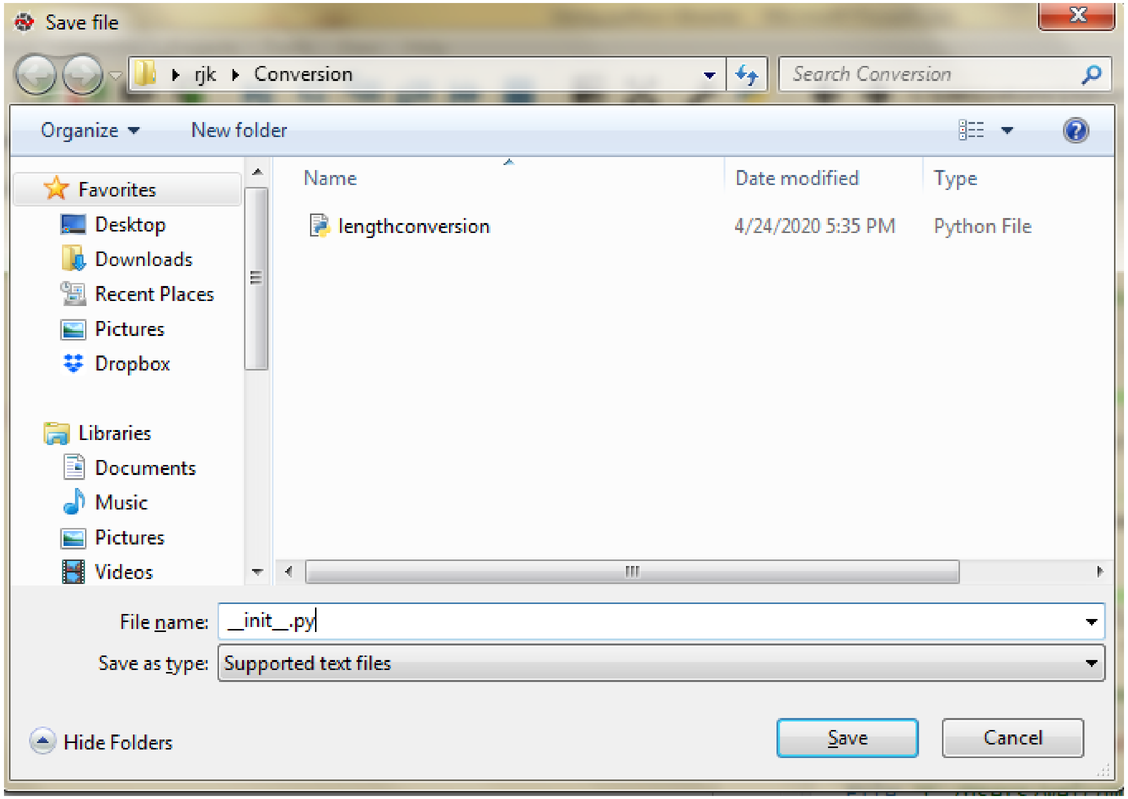
File name: lengthconversion.py

Save as type: Supported text files

CREATE AN NEW PYTHON FILE AND NAME IT AS __INIT__.PY (LET IT BE EMPTY)



```
#__init__.py file
```



Name	Date modified	Type
lengthconversion	4/24/2020 5:35 PM	Python File

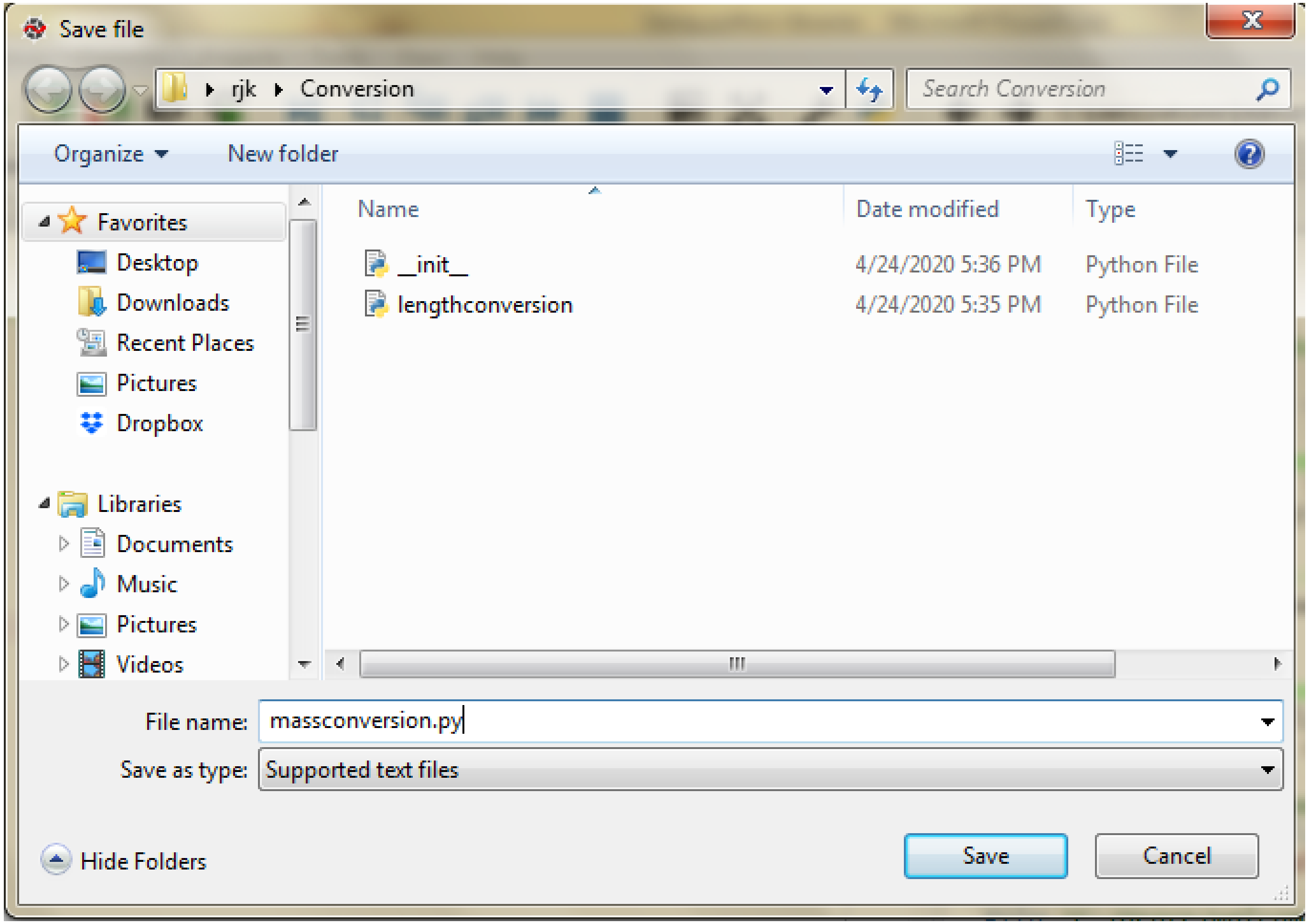
File name: __init__.py

Save as type: Supported text files

SAVE THIS PROGRAM AS MASSCONVERSION.PY IN CONVERSION FOLDER

```
#massconversion.py
def kgtotonne(kg):
    return kg//1000

def tonnetokg(ton):
    return ton * 1000
```

CLICK SAVE BUTTON



Name	Date modified	Type
__init__	4/24/2020 5:36 PM	Python File
lengthconversion	4/24/2020 5:35 PM	Python File

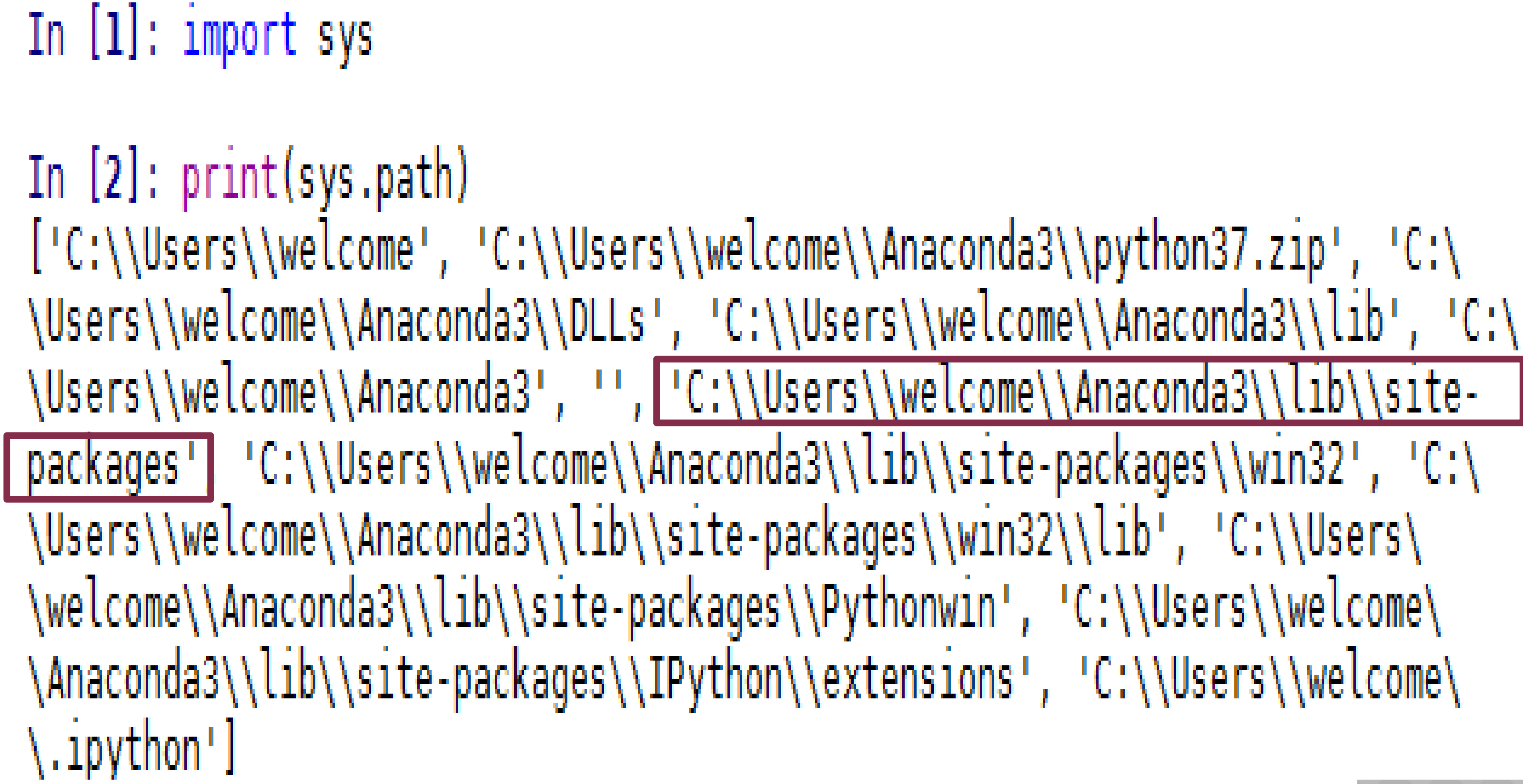
File name: massconversion.py

Save as type: Supported text files

ASSOCIATE IT WITH PYTHON INSTALLATION

- In order to import a package using import command, the package and its contents must be attached to site-packages folder of Python installation as this is the default place from where Python interpreter imports Python library and packages.
- So in order to import our package with import command in our programs, we must attach it to site-packages folder of Python installation.
- In Spyder prompt
  - import sys
  - print(sys.path)
- Once the path of site-packages is figured out, just copy the package folder (Conversion) and paste it
- After copying your package in site-packages folder your current Python installation, now has become a Python library so that you can import its modules and use its functions.

ASSOCIATE IT WITH PYTHON INSTALLATION

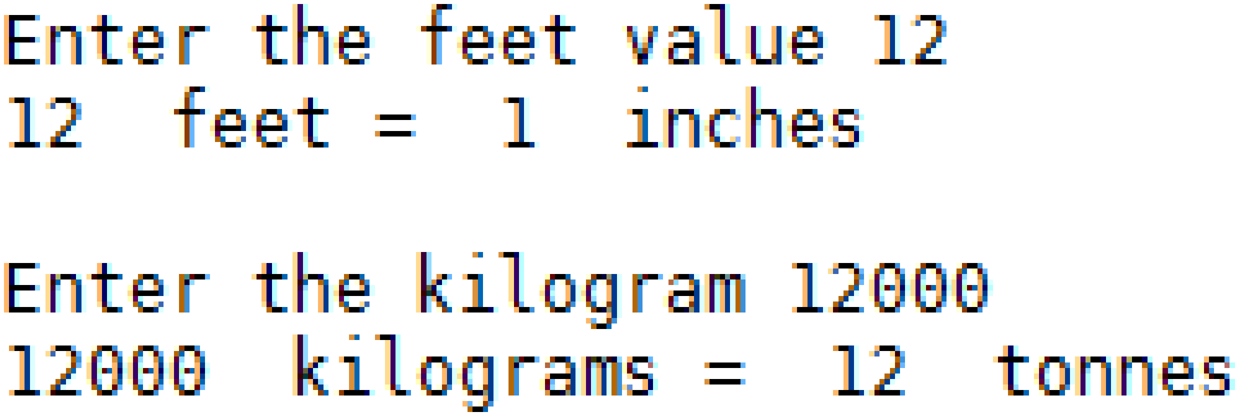


```
In [1]: import sys


In [2]: print(sys.path)
['C:\\Users\\welcome', 'C:\\Users\\welcome\\Anaconda3\\python37.zip', 'C:\
\Users\\welcome\\Anaconda3\\DLLs', 'C:\\Users\\welcome\\Anaconda3\\lib', 'C:\
\Users\\welcome\\Anaconda3', '', 'C:\\Users\\welcome\\Anaconda3\\lib\\site-
packages'  'C:\\Users\\welcome\\Anaconda3\\lib\\site-packages\\win32', 'C:\
\Users\\welcome\\Anaconda3\\lib\\site-packages\\win32\\lib', 'C:\\Users\
\welcome\\Anaconda3\\lib\\site-packages\\Pythonwin', 'C:\\Users\\welcome\
\Anaconda3\\lib\\site-packages\\IPython\\extensions', 'C:\\Users\\welcome\
\.ipython']
```

TRIAL.PY

```
#trial.py

from Conversion import lengthconversion,massconversion


f = eval(input('Enter the feet value '))
print(f,' feet = ',inchestofeet(f),' inches')

kg = eval(input('Enter the kilogram '))
print(kg,' kilograms = ',kgtotonne(kg),' tonnes')
```

OUTPUT



```
Enter the feet value 12
12  feet =  1  inches

Enter the kilogram 12000
12000  kilograms =  12  tonnes
```

PREDICT THE OUTPUT

```
#prg1.PY
def alter(b):
     b = b*2
     return b
#prg2.PY
def alter(b):
     b = b*b
     return b

#main.py
from prg1 import alter
from prg2 import alter
print(alter(3))
```

PREDICT THE OUTPUT

```
d=90
def display():
    global d
    e,d=89,-90
    print(e,d)
display()
print(d)
```

PREDICT THE OUTPUT

```
#mod1
def change(a):
    b=[x*2 for x in a]
    print(b)
#mod2
def change(a):
    b = [x*x for x in a]
    print(b)

#main.py
from mod1 import change
from mod2 import change
s=[1,2,3]
change(s)
```

a) [2,4,6]   b) [1,4,9]
c) [2,4,6][1,4,9]   d) name clash

7)OBSERVE THE FOLLOWING CODE SEGMENT AND ANSWER QUESTIONS BELOW

```
#math_operation
def add(a,b):
    return a+b
def subtract(a,b):
    return a-b
```

1) To import math_operation
2) To print the name of imported module
3) To print the added value of 1,2 using the add function

7)OBSERVE THE FOLLOWING CODE SEGMENT AND ANSWER QUESTIONS BELOW

```
#math_operation
def add(a,b):
    return a+b
def subtract(a,b):
    return a-b
```

1) To import math_operation

```
import math_operation
```

2) To print the name of imported module

```
print(math_operation.__name__)
```

3) To print the added value of 1,2 using the add function

```
print(math_operation.add(1,2))
```

REVISED SYLLABUS 2020-21

Pie (∏) is a well known mathematical constant, which is defined as a the ratio of the circumference to the diameter of a circle and its values is 3.141592653589793. In order to import this single object pie from the math module one can write the following statements

```
import math
print(math.pi)
```

Another well-known mathematical constant defined in the math module is e. It is called Euler’s number and it is a base of the natural logarithm. Its value is 2.718281828459045. To use this in program

```
import math
print(math.e)
```