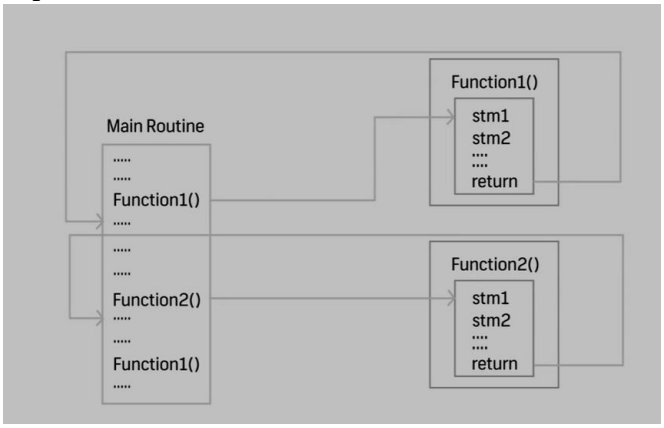
UNIT-III

Python Function, Module and Packages

1. **State Modular Programming.(2M)**
   Functions are independent and reusable blocks of instructions. When the programming solution is long and complex, it is good idea to break it into smaller blocks which are independent and reusable. This approach to software development called modular programming.
   ---

2. **Explain Function in Python.(4M)**
   - Functions are **independent and reusable** blocks of instructions. Each function is a separate, complete and reusable software component.
   - When the programming solution is long and complex, it is good idea to break it into smaller blocks which are independent and reusable. This approach to software development called **modular programming**. It makes the code easy to follow, develop and maintain.



   - When we call a function, it performs a specified task and returns a control to the calling routine. So functions can be thought of organization tools that keep your code neat and tidy. Also functions make it easy to reuse the instructions that you have created.
   - Python offers several built in functions, infact we have already used such as len(), min(), max(), print(), input().

   ---

3. **Define function in Python.(2M)**
   Functions are **independent and reusable** blocks of instructions. Each function is a separate, complete and reusable software component.
   ---

4. **State a user defined function with its syntax.(2M)**

We can create our own functions are known as **user defined functions**.

Syntax of Function without arguments:

```
def function_name() :
    statement_1
    statement_2
    ....
```

Syntax of Function with arguments:

```
def function_name(argument1, argument2, ...) :
    statement_1
    statement_2
    ....
    return expression

function_name(arg1, arg2)
```

In Python the return statement (the word return followed by an expression.) is used to return a value from a function, return statement without an expression argument returns none.

---

5. **Explain a user defined function with its syntax(4M)**

- We can create our own functions are known as **user defined functions**.
- **Syntax of Function without arguments:**

```
def function_name() :
    statement_1
    statement_2
    ....
```

Syntax of Function with arguments:

```
def function_name(argument1, argument2, ...) :
    statement_1
    statement_2
    ....
    return expression

function_name(arg1, arg2)
```

In Python the return statement (the word return followed by an expression.) is used to return a value from a function, return statement without an expression argument returns none.

- We use **def** keyword to define a new function, this is followed by a suitable identifier for the function. The identifier is followed by round brackets. If you want, you can add parameters within these brackets. And finally statement is ended with a colon.

- With a start a new indented block in next line. In this block, the first statement is an explanatory string which describes the functionality. It is called **docs string** and it is optional. It is somewhat similar to comment.

- The statement that follow which perform a certain task form the body of the function. The last statement in the function block has the keyword **return.** It senses the execution back to the calling environment.

```
def SayHello():
   "This will display a greeting message"
   print ("Welcome to Python Learning"
   return
```

Output:

```
>>> SayHello()
Welcome to Python Learning
```

---

6. Explain Function Arguments(4M)

- Functions should also be flexible and allow you to do more than just one thing. Otherwise, you need to create a lot of functions that vary by the data they have used rather than the functionality they provide.
- It is possible to define a function to receive data using parameters also called arguments which function can use when performing the task.
- We have two ways for specifying the arguments
  - **Simply include argument in parenthesis while calling the function**

```
def SayHello(name):
   "This will display a greeting message"
   print("Hello {}.Welcome to Python Learning.\n".format(name))
   return
```

Output:

```
>>> SayHello("Srikar")
Hello Srikar.Welcome to Python Learning.
```

  - **The other way is to use a variables.When we call the function, It takes the variable as an argument**

```
def SayHello(name):
    "This will display a greeting message"
    print("Hello {}.Welcome to Python Learning.\n".format(name))
    return
friend=input("Enter your name:")
SayHello(friend)
```

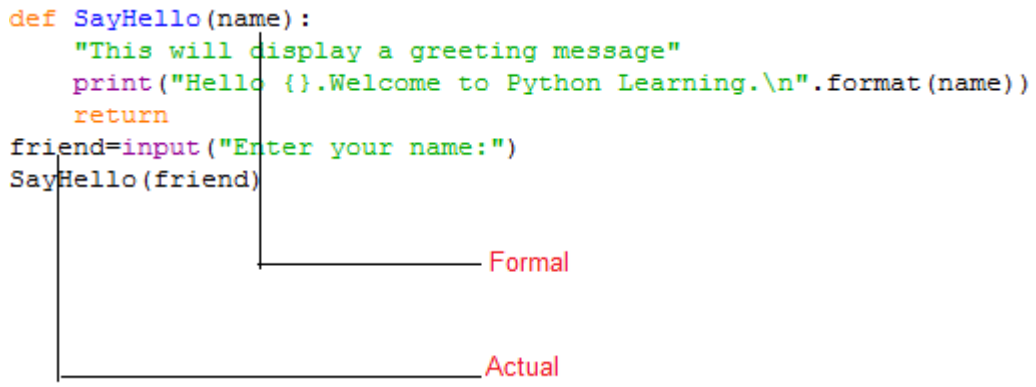
Output:

```
Enter your name:Srikar
Hello Srikar.Welcome to Python Learning.
```

- **There are two kinds of arguments**

  Formal: The arguments which are used in function definition are formal arguments.(Just like plug points)

  Actual- Arguments used when calling the functions.(As plugs)

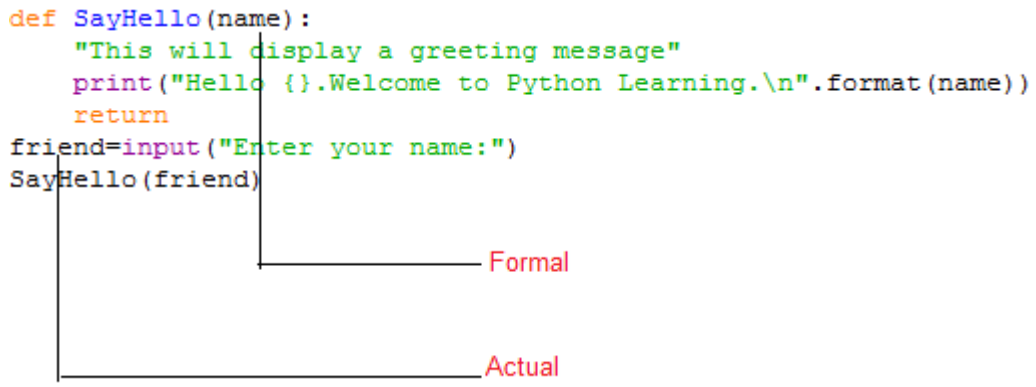


---

7. **State Formal and Actual function arguments.(2M)**
   - Functions should also be flexible and allow you to do more than just one thing. It is possible to define a function to receive data using parameters also called arguments which function can use when performing the task.
   - **There are two kinds of arguments**

     Formal: The arguments which are used in function definition are formal argments.(Just like plug points)

     Actual- Arguments used when calling the functions.(As plugs)



---

8. State the function of Positional arguments.(4M)

- When we call function with arguments, we had to make sure that we provided the same number of actual arguments and also have a matching data types.Such arguments are called positional arguments.
- If there is a mismatch in either the number or type of argument, we get an error.

```
def Myphone(brand,RAM):
    print("I wish I had a {:s} phone with {:d} GB RAM".format(brand,RAM))
    return
```

output:

>>> Myphone("iphone",6)

I wish I had a iphone phone with 6 GB RAM

output:

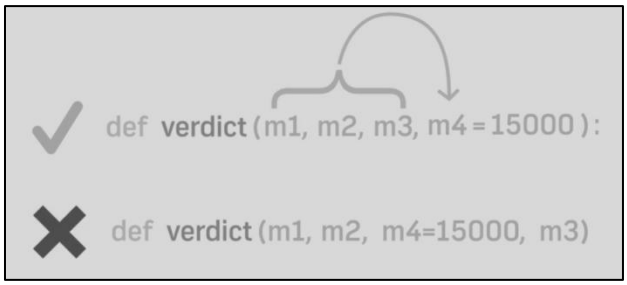
>>> Myphone("iphone")

TypeError: Myphone() missing 1 required positional argument: 'RAM'

>>> Myphone(6,"iphone")

ValueError: Unknown format code 's' for object of type 'int'

- In python, default argument must come always after the required arguments.So in the example we used m4 must be written after m1, m2, m3 because it has default value specified. If this is not followed, the function will throw an error during execution.



---

9. State the features of function in python.(4M)

Following are the features of Python Functions:

- It is used to avoid repetitions of code.
- Using the function, we can divide a group of code into smaller modules.
- It helps to hide the code and create clarity to understand the modules.
- It allows code to be reusable, thus saving memory.
- Statements written inside a function can only be executed with a function name.
- Python function starts with **def** and then a colon (**:**) followed by the function name.

---

10. Explain the following w.r.t function in python(6M)

i. creating a function
ii. calling a function
iii. function with return value

- **Create a function in Python:**

To create a function, we need to use a **def** keyword to declare or write a function in Python. Here is the syntax for creating a function:

Syntax:

```
def function_name():
Statement to be executed
return statement
```

Example:

Let's create a Myfun.py function program in Python.

```
def myFun(): # define function name
   print(" Welcome to Python Programming")
myFun() # call to print the statement
```

Output:

Welcome to Python Programming

- **Calling a function:**

  Once a function is created in Python, we can call it by writing **function_name()** itself or another function/ nested function. Following is the syntax for calling a function.

Syntax:

```
def function_name():
     Statement1
function_name() # directly call the function
```

Example:

```
def MyFun():
   print("Hello World")
   print(" Welcome to Python Programming ")

MyFun() # Call Function to print the message.
```

Output:

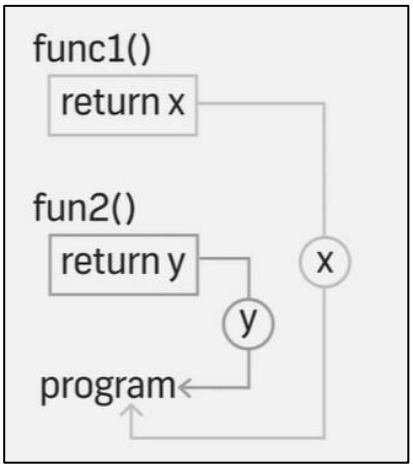
Hello World

Welcome to Python Programming

- **Function with return value:**

  A return statement is used to end the execution of the function call and "returns" the result (value of the expression following the return keyword) to the caller. The

statements after the return statements are not executed. If the return statement is without any expression, then the special value None is returned.
Note: Return statement can not be used outside the function.



Syntax:

```
def fun():
   statements
   .
   .
   return [expression]
```

Example:

```
def add(a, b):

   # returning sum of a and b
   return a + b
# calling function
res = add(2, 3)
print("Result of add function is {}".format(res))
```

Output:
Result of add function is 5

---

11. State local and global variable.(2M/4M)

Local Variables

Local variables are those which are initialized inside a function and belongs only to that particular function. It cannot be accessed anywhere outside the function.

```
def a():
   tag="AO"
   print(tag)
   return
a()
print(tag)
```

Output:

AO

NameError: name 'tag' is not defined

Global Variables

The global variables are those which are defined outside any function and which are accessible throughout the program i.e. inside and outside of every function

```
age=7
def a():
   age=12
   print("Inside the Function-Age is:",age)
   return
a()
print("Outside the Function-Age is:",age)
```

Output:

Inside the Function-Age is: 12

Outside the Function-Age is: 7

---

12. Compare local and global variable.(4M)

Parameter	Local	Global
Scope	It is declared inside a function.	It is declared outside the function.
Lifetime	It is created when the function starts execution and lost when the functions terminate.	It is created before the program's global execution starts and lost when the program terminates.
Data sharing	Data sharing is not possible as data of the local variable can be accessed by only one function.	Data sharing is possible as multiple functions can access the same global variable.
Modification of variable value	When the value of the local variable is modified in one function, the changes are not visible in another function.	When the value of the global variable is modified in one function changes are visible in the rest of the program.
Memory storage	It is stored on the stack unless specified.	It is stored on a fixed location decided by the compiler.

---

13. Same name can be used for local ang global variable,Justify.(4M)

- This is very much possible with built in function **globals().**

- Look at this example. We start with global variable age. Then we modify the age inside function globals(). Let's also create a local variable called age. Notice that we wouldn't prefix with global. So it becomes a global variable.
- Example:

```
age=7
def a():
    print("Global variable 'age':",globals()['age'])
    #Now modifying the GLOBAL variable 'age'Inside the Function.
    globals()['age']=27
    #Now creating a Local variable,'age' I nside the function.
    age=11
    print("Local variable'age':",age)
    return
a()
print("Checking global variable Outside the Function-Age is:",age)
```

Output:

```
Global variable 'age': 7
Local variable'age': 11
Checking global variable Outside the Function-Age is: 27
```

---

14. Explain passing arguments in python.(4M)

- Python's argument passing model is **neither "Pass by Value" nor "Pass by Reference" but it is "Pass by Object Reference".**
- The id() of a list object before and after passing to a function shows an identical value.
- **Example:**

```
def myfunction(newlist):
print("List accessed in function: ", "id:", id(newlist))
return

mylist=[10,20,30,40,50]
print("List before passing to function: ", "id:", id(mylist))
```

Output:

```
List before passing to function:  id: 55345960
List accessed in function:  id: 55345960
```

- If we modify the list object inside the function and display its contents after the function is completed, changes are reflected outside the function as well.

```
def myfunction(newList):
    newList.append(60)
    print("modified list inside function:", newList)
    return

mylist=[10,20,30,40,50]
```

```
print("list before passing to function:", mylist)

myfunction(mylist)

print(  print("list after passing to function:", mylist)
```

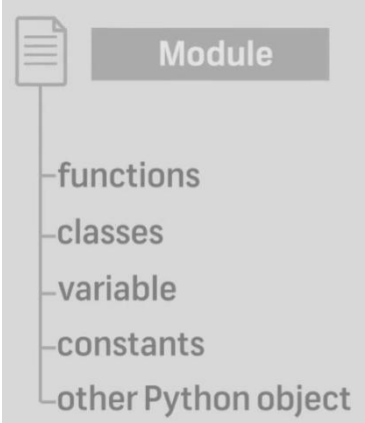
Output:

```
list before passing to function: [10, 20, 30, 40, 50]
modified list inside function: [10, 20, 30, 40, 50, 60]
list after passing to function: [10, 20, 30, 40, 50, 60]
```

---

15. Define module.(2M)

Python offers a simple solution to organize codes which is called module. A module is a file containing functions, classes, variables, constants and or any other python object.



Built in modules are modules merged with python interpreter, they are written in C language, are automatically loaded when the python interpreter starts.

---

16. Write the syntax to import a module and list any two built in python modules.(4M)

- When the import is used, it searches for the module initially in the local scope by calling __import__() function. The value returned by the function is then reflected in the output of the initial code.
- We use the import keyword to do this. To import our previously defined module example, we type the following in the Python prompt.

```
>>> import example
# import statement example
# to import standard module math

import math
print("The value of pi is", math.pi)
```

Output:

```
The value of pi is 3.141592653589793
```

- **Built in python modules**
  1. os module.

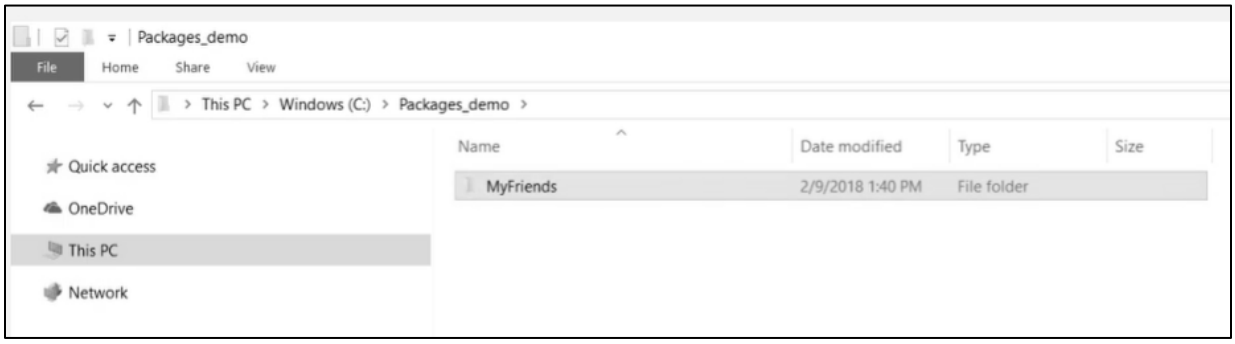
2. random module.
3. math module.
4. time module.
5. sys module.
6. collections module.
7. statistics module.

---

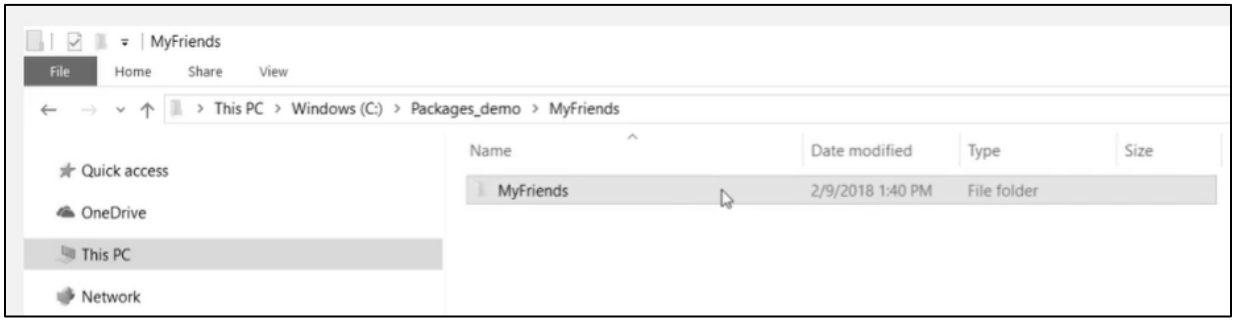
17. **Define package. State the procedure to create a package in python with an example.(6M)**
    One or more relevant modules can be put into a package. A package is a folder containing one or more modules.

Write/Create a package:

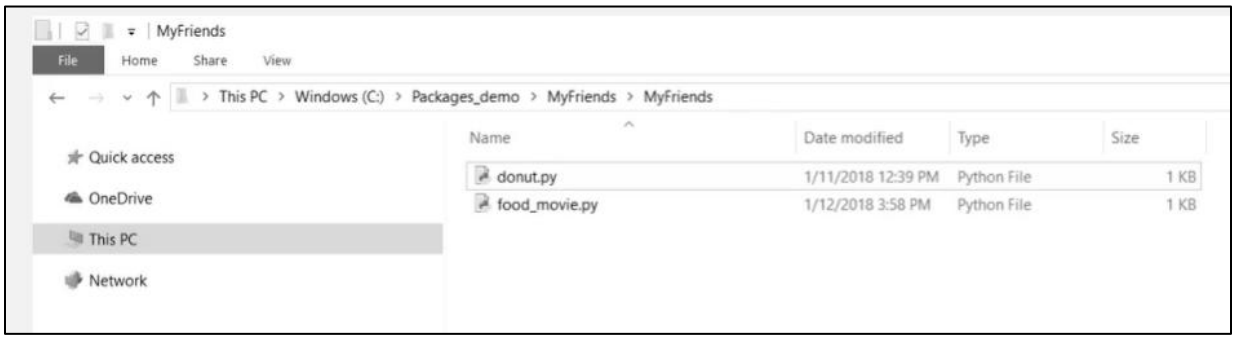
Let us see how to create a package called Myfriends. First we create a folder called Myfriends



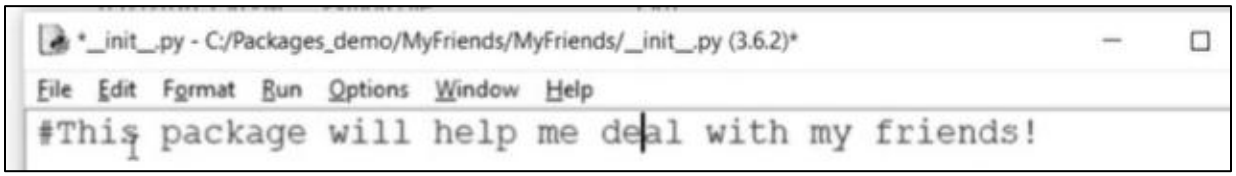
then we create a sub folder with the same name Myfriends.



We now placed different modules inside the sub folder.



Now let's create the __init__.py file simply open IDLE in the scripting mode and save an empty file with the file name double underscore init double underscore .py. You can also add some comments or define functions from individual module to be made available conveniently.



Finally we need to create the set up file which will allow us to install the package.

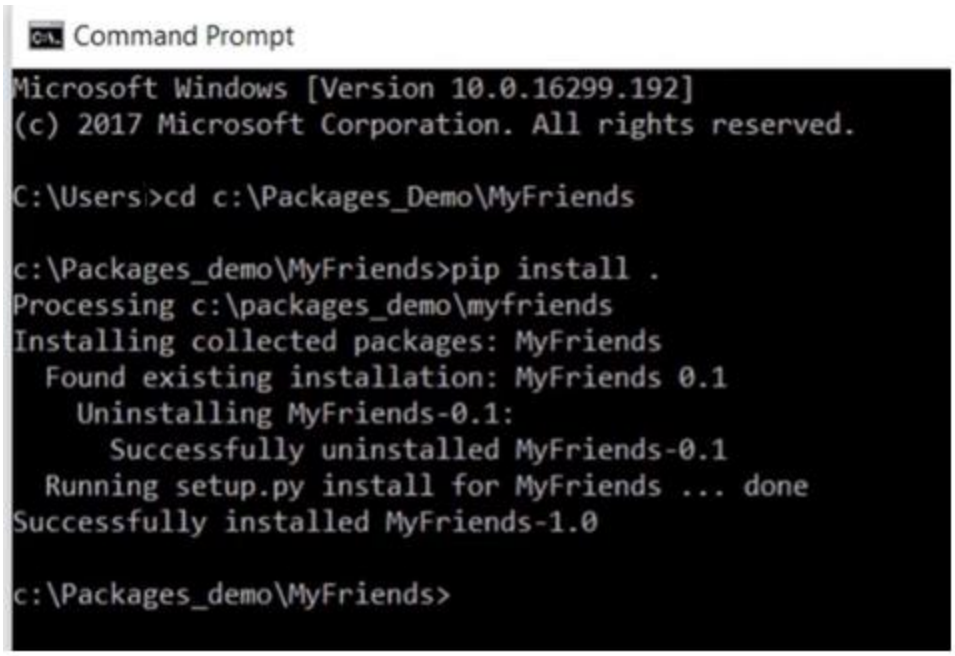
Note that this file must be placed in the parent folder. Basically this script calls the setup function from setuptools module. The setup function takes several argument such as name, version, author etc. The argument zip safe determines whether the package is installed in compressed mode or regular mode.

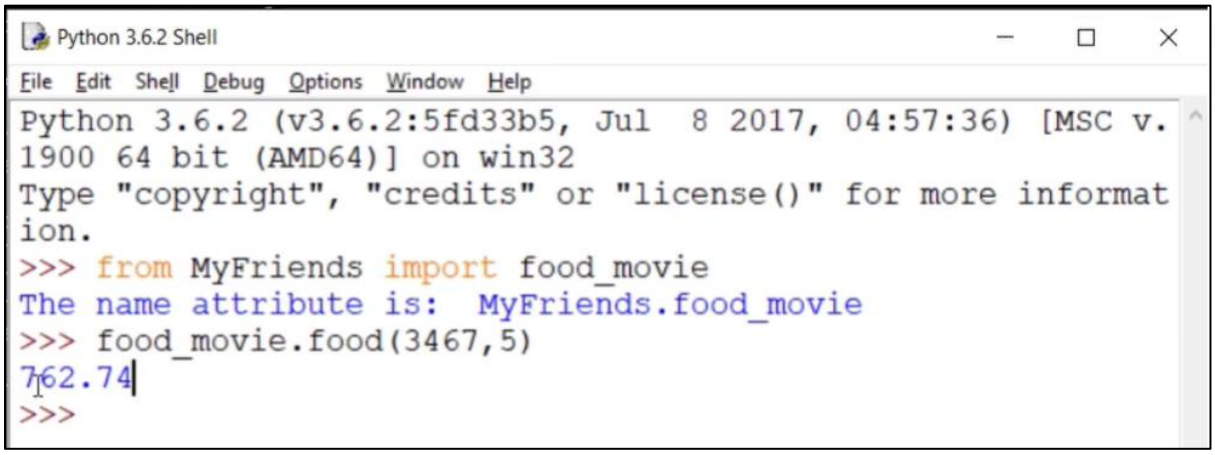
- **Creating a Package Setup File**

```
from setuptools import setup

setup(name='MyFriends', version='1.0', description='A package to calculate expenses when you hang-out with friends.',
url='#',
      author='V.E.S.P',
author_email='V.E.S.P@VES.AC.IN',
      license='MIT',
      packages=['MyFriends'],
zip_safe=False)
```

Now let's test if we can install the Myfriends package for this we first launch the command prompt then we change to the parent directory Myfriends.We execute command **pip install.** To install the Myfriends package using the utility Myfriends package is now available for use in the system and can be imported from any script. We can try using some of the functions from it to verify this. let say we import the food function It works.





---

18. Explain the procedure to build a user defined package.(6M)

To Create a package in Python, we need to follow these three simple steps:

- First, we create a directory and give it a package name, preferably related to its operation.
- Then we put the classes and the required functions in it.
- Finally we create an __init__.py file inside the directory, to let Python know that the directory is a package.

Let's create a package named Cars and build three modules in it namely, Bmw, Audi and Nissan.

1. **First we create a directory and name it Cars.**
2. **Then we need to create modules**. To do this we need to create a file with the name Bmw.py and create its content by putting this code into it.

```
# Python code to illustrate the Modules
class Bmw:
    # First we create a constructor for this class
    # and add members to it, here models
    def __init__(self):
        self.models = ['i8', 'x1', 'x5', 'x6']

    # A normal print function
    def outModels(self):
        print('These are the available models for BMW')
        for model in self.models:
            print('\t%s ' % model)
```

3. Then we create another file with the name Audi.py and add the similar type of code to it with different members.

```
# Python code to illustrate the Module
class Audi:
    # First we create a constructor for this class
    # and add members to it, here models
    def __init__(self):
        self.models = ['q7', 'a6', 'a8', 'a3']

    # A normal print function
    def outModels(self):
        print('These are the available models for Audi')
        for model in self.models:
            print('\t%s ' % model)
```

4. Then we create another file with the name Nissan.py and add the similar type of code to it with different members.

```
# Python code to illustrate the Module
class Nissan:
    # First we create a constructor for this class
    # and add members to it, here models
    def __init__(self):
        self.models = ['altima', '370z', 'cube', 'rogue']

    # A normal print function
    def outModels(self):
        print('These are the available models for Nissan')
        for model in self.models:
```

```
        print('\t%s ' % model)
```

5. **Finally we create the __init__.py file.** This file will be placed inside Cars directory and can be left blank or we can put this initialisation code into it.

```
from Bmw import Bmw
from Audi import Audi
from Nissan import Nissan
```

6. Now, let's use the package that we created. To do this make a sample.py file in the same directory where Cars package is located and add the following code to it:

```
# Import classes from your brand new package
from Cars import Bmw
from Cars import Audi
from Cars import Nissan

# Create an object of Bmw class & call its method
ModBMW = Bmw()
ModBMW.outModels()

# Create an object of Audi class & call its method
                                        ModAudi = Audi()
ModAudi.outModels()

# Create an object of Nissan class & call its method
ModNissan = Nissan()
ModNissan.outModels()
```

---

19. List and explain standard python packages.

- **Matplotlib:** This library is responsible for plotting numerical data. And that's why it is used in data analysis. It is also an open-source library and plots high-defined figures like pie charts, histograms, scatterplots, graphs, etc.
- Pandas: Pandas are an important library for data scientists. It is an open-source machine learning library that provides flexible high-level data structures and a variety of analysis tools. It eases data analysis, data manipulation, and cleaning of data. Pandas support operations like Sorting, Re-indexing, Iteration, Concatenation, Conversion of data, Visualizations, Aggregations, etc.
- **Numpy:** The name "Numpy" stands for "Numerical Python". It is the commonly used library. It is a popular machine learning library that supports large matrices and multi-dimensional data. It consists of in-built mathematical functions for easy computations.

Even libraries like TensorFlow use Numpy internally to perform several operations on tensors. Array Interface is one of the key features of this library.

- **SciPy:** The name "SciPy" stands for "Scientific Python". It is an open-source library used for high-level scientific computations. This library is built over an extension of Numpy. It works with Numpy to handle complex computations. While Numpy allows sorting and indexing of array data, the numerical data code is stored in SciPy. It is also widely used by application developers and engineers.

---

20. Plot sine wave using matplotlib.

plot(x, y)

```
import numpy as np
import matplotlib.pyplot as plot
# Get x values of the sine wave
time       = np.arange(0, 10, 0.1);
# Amplitude of the sine wave is sine of a variable like time
amplitude   = np.sin(time)
# Plot a sine wave using time and amplitude obtained for the sine wave
plot.plot(time, amplitude)
# Give a title for the sine wave plot
plot.title('Sine wave')
# Give x axis label for the sine wave plot
plot.xlabel('Time')
# Give y axis label for the sine wave plot
plot.ylabel('Amplitude = sin(time)')
plot.grid(True, which='both')
plot.axhline(y=0, color='k')
plot.show()
# Display the sine wave
plot.show()
```