PYTHON MODULES

Go through video before you start reading the resources mentioned below.

YoutubeVideo Link: https://youtu.be/1RuMJ53CKds

If you quit from the Python interpreter and enter it again, the definitions you have made (functions and variables) are lost. Therefore, if you want to write a somewhat longer program, you are better off using a text editor to prepare the input for the interpreter and running it with that file as input instead. This is known as creating a **script**. As your program gets longer, you may want to split it into several files for easier maintenance. You may also want to use a handy function that you've written in several programs without copying its definition into each program.

To support this, Python has a way to put definitions in a file and use them in a script or in an interactive instance of the interpreter. Such a file is called a **module**; definitions from a module can be imported into other modules or into the main module (the collection of variables that you have access to in a script executed at the top level and in calculator mode).

Program code making use of a given module is called a **client** of the module.

1. What is a Module?

A module is a file containing Python definitions and statements. A module can define functions, classes and variables. A module can also include runnable code. Grouping related code into a module makes the code easier to understand and use.

1.1. Modules vs. Functions

Function: it's a block of code that you can use/reuse by calling it with a keyword. Eg. print() is a function.

Module: it's a .py file that contains a list of functions (it can also contain variables and classes).

1.2. Relation Between Python Libraries, Module and Package

1. A module is a file containing python definition, functions, variables, classes and statements. The extension of this file is ".py".
2. While Python package, is directory(folder) of python modules.
3. A library is collection of many packages in python. Generally there is no difference between python package and python library.

1.3. Types of Modules

1. Python provides us with some built-in modules, which can be imported by using the "import" keyword.
2. Python also allows us to create your own modules and use them in your programs.

2. Usages of Modules

- Modules are used to reduce the redundant statements in a program.
- It saves time and increases readability as well as productivity.
- They are also used to extend the functionality of python and allows different developers around the world to work in a coordinated manner.

  For example, Google developed Tensorflow, which contains functions for deep learning and is open for contributions from past many years. It is an open source module so that various people from different parts of the world could participate and improve the scope of deep learning applications.

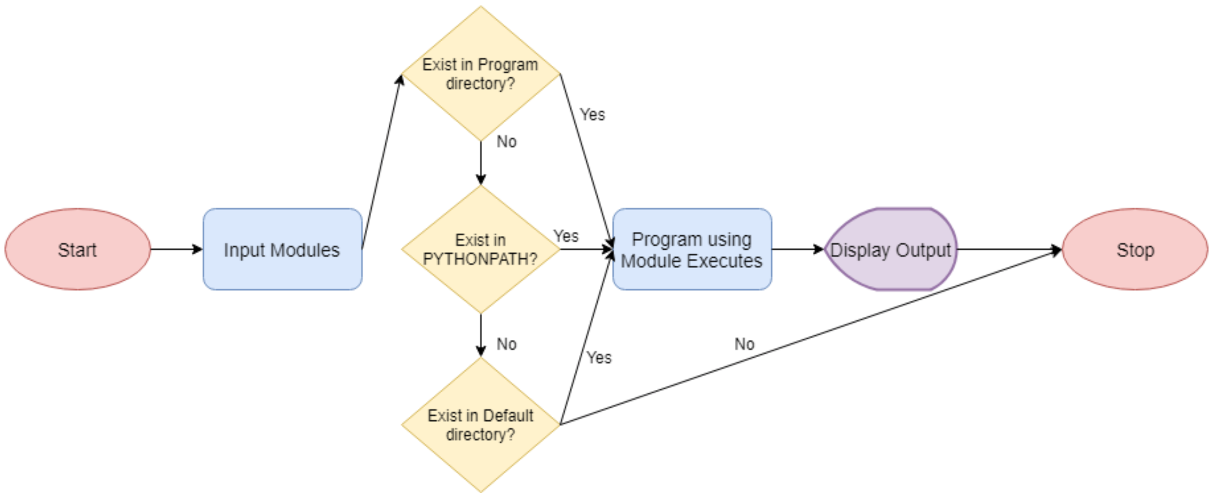
3. Python Module : Mechanism

When we import modules, the python interpreter locates them from three locations:

1. The directory from the program is getting executed
2. The directory specified in the PYTHONPATH variable (A shell variable or an environment variable)
3. The default directory (It depends on the OS distribution.)

If the module exist in any of the 3 then program using module is executed, otherwise the program is stopped there only.

The Flowchart for the above mechanism is as follows:


Start
Input Modules
Exist in Program directory?
Yes
No
Exist in PYTHONPATH?
Yes
No
Exist in Default directory?
Yes
No
Program using Module Executes
Display Output
Stop

4. Creating a Module

- To create a module, you first create a Python file and save it with the name of the module.
- So if you want to create a module called mymodule, then you would first create a file called mycodes.py.
- Inside this file you can then write the functions or other codes you want to reuse.

Example 1:

Save this code in a file named **mymodule.py or mymodule.ipynb**

```
def greeting(name):
  print("Hello, " + name)
```

5. Use a Module

5.1. The *import* statement

- We can use any Python source file as a module by executing an import statement in some other Python client file.
- When interpreter encounters an import statement, it imports the module if the module is present in the search path.
- A search path is a list of directories that the interpreter searches for importing a module.
- You can import more than one module.

Example 2:

For this, create a new script (new jupyter notebook) in which first statement is import statement .

Import the module named mymodule, and call the greeting function:

```
import mymodule

mymodule.greeting("Jonathan")
```

Output:

Hello Jonathan

Note: *When using a function from a module, use the syntax: module_name.function_name.*

5.1.1 The *import* statement for jupyter notebook

You need to create 2 Jupyter Notebook –

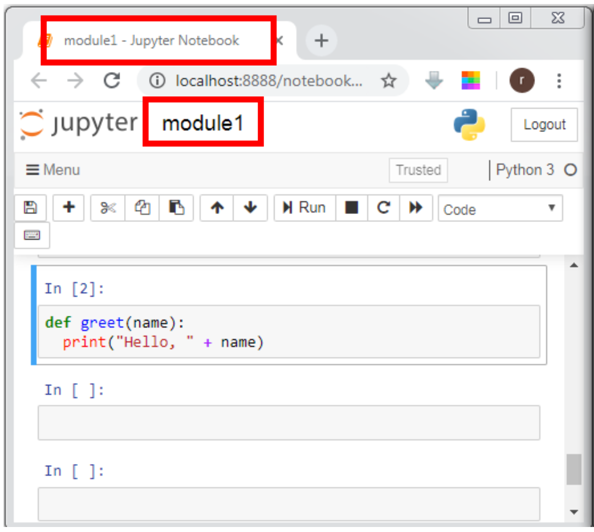
1. Jupyter notebook containing module
2. **Client** Jupyter notebook - making use of a given module

- To create a module, you first create a Jupyter Notebook containing functions and save it with the name of the module1 (refer figure 1).
- There is need to install a library named "import_pynb" in client jupyter notebook using following command as shown in figure 2.

```
!pip install import_ipynb
```

(or use !pip3 install import_ipynb in case above command does not work.)

- After installing the library, we first need to import the library(module1 jupyter notebook) then the above procedure explained in example 2 can be followed.

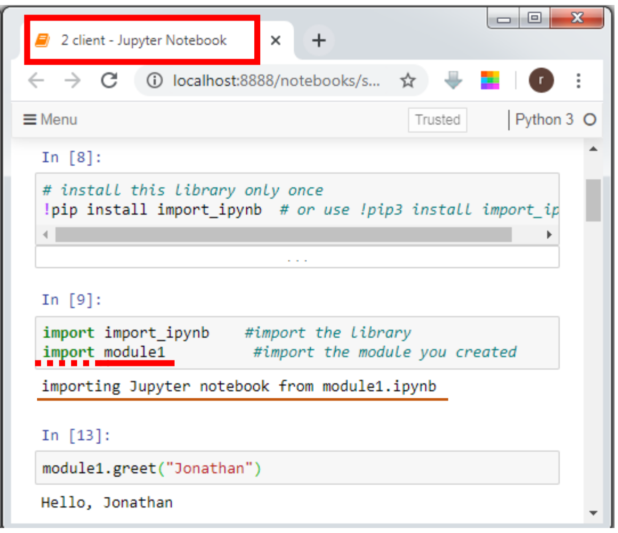


```
In [2]:
def greet(name):
  print("Hello, " + name)

In [ ]:

In [ ]:
```

Figure 1: ***Module*** *Jupyter Notebook*



```
In [8]:
# install this library only once
!pip install import_ipynb  # or use !pip3 install import_ip

In [9]:
import import_ipynb     #import the library
import module1           #import the module you created

importing Jupyter notebook from module1.ipynb

In [13]:
module1.greet("Jonathan")

Hello, Jonathan
```

Figure 2: ***Client*** *Jupyter Notebook*

5.2. The *from import* statement

- Let's say a module contains 100 functions, and you want to use only one of them.
- You don't have to import the whole module.
- Python's *from* statement lets you import specific attributes from a module.
- You can choose to import only parts from a module, by using the from keyword.

Example 3:

The module named mymodule has one function and one dictionary:

```
def greeting(name):
  print("Hello, " + name)

person1 = {
  "name": "John",
  "age": 36,
  "country": "Norway"
}
```

So if you have a module called mymodule and the name of the variable you want to import is person1, you can use the syntax given below

```
from mymodule import person1

print (person1["age"])
```

***Note**: When importing using the from keyword, do not use the module name when referring to elements in the module. Example: person1["age"], not mymodule.person1["age"]*

- It is also possible to import all names from a mymodule into the current namespace by using the following import statement
  `from mymodule import *`
- This provides an easy way to import all the items from a module into the current namespace.

6. Built-in Modules

There are several built-in modules in Python, which you can import whenever you like.

For example, the math module that contains useful functions for performing arithmetic.

The random module for working with random numbers and sequences, and lots more.

Example 4:

Importing built-in module random

In the below code randint in a function present inside the module random

```
import random

print (random.randint(1,20)) #print a random value from integers1-20
```

7. The dir( ) function

- There is a built-in function to list all the function names (or variable names) in a module.
- The dir( ) built-in function returns a sorted list of strings containing the names defined by a module.
- The list contains the names of all the modules, variables and functions that are defined in a module.

Example 5:

```
import math
content = dir(math)
print (content)
```

```
This would produce following result:
['__doc__', '__name__', '__package__', 'acos', 'acosh', 'asin',
'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos',
'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs',
'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'hypot',
'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p',
'modf', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan',
'tanh', 'trunc']
```

8. The reload( ) function

- If you import a module into your program, then the code in the top-level part of the module is executed just once.
- This means that if changes are made to the module, you would not have the updated version of the module.
- To solve this you used the reload() function. This function is used to re-execute the top-level code in a module.
- So you can have the update module. It runs the import statement again.

Example 6:

Suppose you made changes in the module named as mymodule then the following can be used to reload it

```
reload(mymodule)
```

Note: *But for python version above 3.4 the above statement will give error. So for this we need to import other library ie.* ***importlib*** *where the reload function is present inside the library named importlib*

```
import importlib

importlib.reload(mymodule)
```

9. Re-naming a Module

You can create an alias when you import a module, by using the **as** keyword:

Example 7:

Create an alias for mymodule called mx:

```
import mymodule as mx

a = mx.person1["age"]
print(a)
```

Note: *The above process will not follow when you are using Jupyter notebook. When using jupyter notebook we need to install a library first and then the above procedure is done.*

10. Example covering all these Sections

For better understanding let's create a python module to explore it completely. First create a file named **printNumbers.py** with the following contents.

```
def printForward(n):

    #print 1 to n
    for i in range(n):
        print(i+1)


def printBackwards(n):

    #print n to 1
    for i in range(n):
        print(n-i)
```

10.1. Import statement

```
#this will import your module and we can access every function
present in it

import printNumbers

# This statement will call the function printforward

printNumber.printForward(10)
```

10.2. Aliasing

Since the name of module is to long it is not feasible everytime to write its name while accessing its functions.

So we can use the concept of aliasing

```
#this allows us to use np in place of printNumbers
import printNumbers as pn


# so while calling the function we can use np

np.printForward(10)
```

10.3. from ………import statement

Suppose if the module contains 100 function and if want to access only printBackward function then this can be achieved through from……import statement

```
#this statement will only import printBackward function unlike
the import statement where the whole module in imported and we
need to access each and every function using dot operator

from printNumbers import printBackward

# in this case the syntax of calling the function also changes
there is no need of using dot operator

printBackward(10)
```

Watch the following process to understand process of accessing Modules using in Jupyter Notebook

References:

1. https://docs.python.org/3/tutorial/modules.html
2. https://www.geeksforgeeks.org/python-modules/
3. https://www.tutorialspoint.com/python/python_modules.htm
4. https://www.w3schools.com/python/python_modules.asp
5. https://pypi.org/project/import-ipynb/