PES UNIVERSITY

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Prof. Sowmya Shree P, Prof. Sindhu R Pai
Department of Computer Science
and Engineering

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Unit - 3: Modules – Import Mechanisms

Prof. Sowmya Shree P, Prof. Sindhu R Pai

Department of Computer Science and Engineering

Module

- In Python, a module is a file that contains code.
- It can include functions, classes, variables or any runnable code.
- Basically, a module contains code to perform specific task.
- We can use modules to separate codes in separate files as per their functionality.

Modules – Import mechanisms

Advantages of Modules

- Reusability - makes the code reusable
- Modularity – Organizing the code into modules logically
- Separate scopes – separate namespace is defined by the module
- Grouping - Python modules help us to organize and group the content by using files and folders

Modules – Import mechanisms

Need for Modules

- While working on coding, We need to use many classes, variables, functions.
- If we include everything in a single file, the program may become large.
- To reduce size of the code, we can group together some similar functions, classes into a collection.
- That collection is nothing but modules in python.

Packages and Namespace

- Modules in Python can be grouped together in packages.
- Packages organize the code into logical groups and provide a namespace to the modules so that they don’t conflict with modules with the same name in other packages.
- The import statement can also be used to import modules from a package, which allows to access the functions and classes defined in the package’s modules.

Packages vs. Modules vs. Libraries

- Modules – contain several functions, variables, classes etc.
- Packages – contain several modules.

  - folder that contains various modules as files.
- Library - a collection of packages and modules used to access **built-in functionality**

Types of Modules

1. **Built-in Modules**
   - Python's standard library comes bundled with a large number of modules.
   - They are called built-in modules.

2. **User-defined Modules**
   - Any file with .py extension and containing Python code is basically a module.
   - It can contain definitions of one or more functions, variables, constants ,classes.

Modules – Import mechanisms

Built-in Modules

Name	Description
os	provides a unified interface to a number of operating system functions
string	contains a number of functions for string processing
re	regular expression functionalities
math	a number of mathematical operations
cmath	a number of mathematical operations for complex numbers
datetime	functions to deal with dates and the time
gc	an interface to the built-in garbage collector
asyncio	functionality required for asynchronous processing
collections	advanced Container datatypes
functools	Higher-order functions and operations on callable objects

Modules – Import mechanisms

Built-in Modules

Name	Description
operator	Functions on the standard operators
pickle	Convert Python objects to streams of bytes and back
socket	Low-level networking interface
sqlite3	A DB-API 2.0 implementation using SQLite 3.x
statistics	Mathematical statistics functions
typing	Support for type hints
venv	Creation of virtual environments
json	Encode and decode the JSON format
unittest	Unit testing framework for Python
random	Generate pseudo-random numbers

Modules – Import mechanisms

User-defined Modules

- Any file with .py extension and containing Python code is a module.
- It can contain definitions of one or more functions, variables, constants as well as classes.
- Any object from a module can be made available to interpreter or another Python script by using import statement.

Creating a Module

- Crete a file with .py extension
- **Example**: Creating a module(module1.py)

```
a=10
print("Welcome to Module-1")
def f1():
        print("in f1")
def f2():
        print("in f2")
def _f3():
        print("in f3")
```

← **module1.py**

Import a Module

- import the functions, and classes defined in a module to another module
- When the interpreter encounters an import statement, it imports the module if the module is present. Otherwise, *ModuleNotFoundError* is thrown.
- Syntax - **import** *module_name*

  import – the keyword used to import the module

  module_name – name of the module to be imported

- To access the functions inside the module the dot(.) operator is used.

Modules – Import mechanisms

Import a Module

- **Example**: importing the **module1**(refer previous slide)

```
import module1 #All functions and variables of module1 are available

module1.f1()
module1.f2()
module1._f3()

print("value is", module1.a)
module1.a = module1.a + 2
print("value is", module1.a)
```

← **usingModule1.py**

Modules – Import mechanisms

Output

- Executing **usingModule1.py**

```
Welcome to Module-1
in f1
in f2
in f3
value is 10
value is 12
```

Import from a Module

- Import Specific Attributes from a module
- Syntax – **from** *module_name* **import** *specific_attributes*
- **Example 2**

```
def add(x, y):
   return (x+y)
def subtract(x, y):
   return (x-y)
def multiply(x, y):
   return (x*y)
def divide(x, y):
   return (x/y)          #Assuming 'y' is never zero
```

module2.py

Modules – Import mechanisms

Import from a Module

- Example 2 (Contd…)

```
from module2 import add,multiply
#importing only add and multiply functions from module2

print("Sum=",add(10,20))
print("Product=",multiply(25,10))
```

← **usingmodule2.py**

```
Output
Sum= 30
Product= 250
```

Import all Names from a Module

- * symbol with the import statement is used to import all the names from a module.
- Syntax - **from** *module_name* **import** ***

- **Example 3**

```
def add(x, y):
   return (x+y)
def subtract(x, y):
   return (x-y)
def multiply(x, y):
   return (x*y)
def divide(x, y):
   return (x/y)          #Assuming 'y' is never zero
```

← **module3.py**

Modules – Import mechanisms

Import all Names from a Module

- <u>Example 3 (Contd…)</u>

```
from module3 import *
#importing all the functions from module2

print("Sum=",add(10,20))
print("Product=",multiply(25,10))
```

usingmodule3.py

<u>Output</u>

```
Sum= 30
Product= 250
```

Note: If we know exactly which attribute to import from the module, it is not recommended to use import *

Renaming/Aliasing Python Modules

- We can rename the module while importing it.
- Syntax – **import** *module_name* **as** *alias_name*
- **Example 4**

```
import math as mt  #Renaming math module as 'mt’

print(mt.factorial(6))
```

Output

```
720
```

Renaming/Aliasing Python Modules

- Example 5

```
GRAVITY=9.8
print("Illustration of Renaming a Module")
```

module5.py

```
import module5 as m5
print("***********************************")
print("Acceleration due to gravity on earth=",m5.GRAVITY,"m/s\u00b2")
```

usingmodule5.py

Output

```
Illustration of Renaming a Module
***********************************

Acceleration due to gravity on earth= 9.8 m/s²
```

Locating Python Modules

- Python modules are located by interpreter in following steps.
  - First, it will check for the built-in module.
  - If not built-in module, Search for the Module in the current directory
  - If not found in current directory, Python then searches each directory in the shell variable PYTHONPATH (An environment variable, consisting of a list of directories).
  - If that also fails python checks the sys.path (A built-in variable within the sys module. It contains a installation-dependent list of directories configured during Python installation).

Locating Python Modules

- **To get the Directories List**

```
# importing sys module
import sys
# importing sys.path
print(sys.path)
```

Output:

```
['', 'C:\\Users\\SOWMYA SHREE P\\AppData\\Local\\Programs\\Python\\Python311\\Lib\\idlelib',
'C:\\Users\\SOWMYA SHREE P\\AppData\\Local\\Programs\\Python\\Python311\\python311.zip',
'C:\\Users\\SOWMYA SHREE P\\AppData\\Local\\Programs\\Python\\Python311\\DLLs',
'C:\\Users\\SOWMYA SHREE P\\AppData\\Local\\Programs\\Python\\Python311\\Lib',
'C:\\Users\\SOWMYA SHREE P\\AppData\\Local\\Programs\\Python\\Python311',
'C:\\Users\\SOWMYA SHREE P\\AppData\\Local\\Programs\\Python\\Python311\\Lib\\site-
packages’]
```

This is a list of directories that the interpreter will search for the required module.

Modules – Import mechanisms

Sys.path.append()

- a built-in function of sys module that can be used with path variable to add a specific path for interpreter to search.

```
import sys
sys.path.append( '/path/to/module’)
```

Sys.path.insert()

- a built-in function of sys module that can be used to insert a path at a specific position in sys.path.

```
import sys
sys.path.insert(0, '/path/to/module’)
```

0 indicates that the path should be inserted at the beginning of sys.path .

-1 to insert a path at the end of sys.path.

Modules – Import mechanisms

__doc__ variable

- In Python, each object(Class, Fucntion, variable,..) can be documented using Docstrings.
- Docstrings can be accessed using the __doc__ attribute.

```
def add():
   '''Performing addition of two numbers.'''
   a=10
   b=7
   print(a+b)

print("Using __doc__:")
print(add.__doc__)

print("Using help:")
help(add)
```

Output

```
Using __doc__:
Performing addition of two numbers.
Using help:
Help on function add in module __main__:

add()
   Performing addition of two numbers.
```

__name__ variable and Modules

- It is a special variable in Python.
- If the source file is executed as the main program, the interpreter sets the __name__ variable to the value “__main__”.
- If this file is being imported from another module, __name__ will be set to the module’s name.

Modules – Import mechanisms

```
print (“file1 __name__ = %s" %__name__)

if __name__ == "__main__":
    print (“file1 is executed directly")
else:
    print (“file1 is imported")
```

file1.py

Output

```
file1 __name__ = __main__
file1 is executed directly
```

After executing file1.py

Modules – Import mechanisms

```
import file1

print ("file2 __name__ = %s" %__name__)

if __name__ == "__main__":
   print ("file2 is executed directly")
else:
   print ("file2 is imported")
```

file2.py

Output

```
file1 __name__ = file1
file1 is imported
file2 __name__ = __main__
file2 is executed directly
```

After executing file2.py

Modules - Summary

- Python Module is a python script file that can contain variables, functions, and classes.
- Python modules help us in organizing our code and then referencing them in other classes or python scripts.
- Modular Programming is the practice of segmenting a single, complicated coding task into multiple, simpler, easier-to-manage sub-tasks. These sub-tasks are Modules.

PES UNIVERSITY

THANK YOU

Department of Computer Science and Engineering

Contact Email ID's:

sindhurpai@pes.edu

sowmyashree@pes.edu