UE25CS151A

Department of Computer Science and Engineering

PES University, Bangalore, India

Lecture Notes

Python for Computational Problem Solving - UE25CS151A

Lecture 70

Introduction to Modules

Prepared by,

Prof. Kundhavai,
Assistant Professor
Dept. of CSE, PESU

Python Modules and PIP

Introduction

- A module in Python is simply a file containing Python code, functions, classes, or variables that you can reuse in other programs.
- Modules help organize large programs into smaller, manageable, and reusable pieces.
- Python provides many built-in modules like **math, os, random, and datetime** that simplify common programming tasks.
- You can also **create your own modules** by saving functions in a .py file and importing them into another program using the import statement.
- You can also include the **third-party modules** in the python code.

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 content by using files and folders

Types of Modules:

1. Built-in Modules

- You can use code from one module in another using the **import** statement.
- Example:

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

- **You can also import specific functions using:**

```
from math import sqrt
print(sqrt(16))
```

- **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 *

```
from math import *
```

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:

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

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

Output
720

Common 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

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

2. User-Defined Modules (Creating Your Own Modules)

- A user-defined module is a .py file you create to store your own functions, variables, or classes for reuse.
- This improves modularity and maintainability of code.

Example 1:

- **Create a file my_module.py:**

```
def greet(name):
    return f"Hello, {name}!"
```

- **Use it in another file:**

```
import my_module
print(my_module.greet("Good morning"))
```

Example 2:

Import Specific Attributes from a module

- **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

```
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

Example 3:

```
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

3. PIP – Third-Party Modules

- Python's Standard Library covers many common tasks, but for advanced features, you can use **third-party packages** created by other developers.
- These are collections of modules available online through **PyPI (Python Package Index).**

- **PIP (Pip Installs Packages)** is the standard command-line tool for installing and managing these external packages.

- **Common pip commands:**
  - **pip install package_name** → Install a package
  - **pip uninstall package_name** → Remove a package
  - **pip list** → View installed packages

Popular third-party modules:

- **numpy** – numerical computing
- **pandas** – data analysis
- **matplotlib** – data visualization
- **scikit-learn** (sklearn) – machine learning
- **pytmage** (or Pillow) – image processing
- **requests** – sending HTTP requests and working with APIs

- **Example**:

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print("Array:", arr)
print("Mean:", np.mean(arr))
```