Python Modules and Packages

1. Modules

Definition: A module is a file containing Python code (functions, classes, variables) that can be imported and used in other Python programs. Modules help organize code into logical units, promote code reusability, and avoid naming conflicts. Python files with .py extension are modules.

---

2. Importing Modules

Definition: Importing is the process of loading a module's code into your program so you can use its functions, classes, and variables. Python provides several ways to import modules using the import statement.

Method 1: Basic Import

Syntax:

python

```
import module_name
```

Example:

python

```
import math


# Using module functions with module_name.function_name
result = math.sqrt(16)
print(result)  # 4.0


area = math.pi * (5 ** 2)
print(f"Area of circle: {area}")  # 78.53981633974483
```

Method 2: Import with Alias

Definition: Importing a module with an alias gives it a shorter or different name for convenience.

Syntax:

python

```
import module_name as alias
```

Example:

python

```
import datetime as dt


# Using alias instead of full module name
current_time = dt.datetime.now()
print(current_time)


today = dt.date.today()
print(f"Today's date: {today}")
```

Method 3: Import Specific Items

Definition: Import only specific functions, classes, or variables from a module instead of the entire module.

Syntax:

python

```
from module_name import item1, item2
```

Example:

python

```
from math import sqrt, pi, pow


# Use directly without module name
```

print(sqrt(25)) *# 5.0*

print(pi) *# 3.141592653589793*

print(pow(2, 3)) *# 8.0*

Method 4: Import All

Definition: Import all items from a module (not recommended as it can cause naming conflicts).

Syntax:

python

from module_name import *

Example:

python

from random import *

 # All random module functions available directly

num = randint(1, 10)

print(num)

choice_item = choice(['apple', 'banana', 'cherry'])

print(choice_item)

---

3. Using Standard Libraries

Definition: Standard libraries are pre-installed modules that come with Python. They provide ready-to-use functions for common tasks like mathematical operations, random number generation, date/time handling, file operations, etc.

math Module

Definition: The math module provides mathematical functions and constants for advanced mathematical operations beyond basic arithmetic.

Syntax:

python

import math

Example:

python

```
import math

# Mathematical constants
print(f"Pi: {math.pi}")  # 3.141592653589793
print(f"Euler's number: {math.e}")  # 2.718281828459045

# Mathematical functions
print(f"Square root of 16: {math.sqrt(16)}")  # 4.0
print(f"Power: {math.pow(2, 3)}")  # 8.0
print(f"Factorial of 5: {math.factorial(5)}")  # 120

# Trigonometric functions
print(f"Sin(90°): {math.sin(math.radians(90))}")  # 1.0
print(f"Cos(0°): {math.cos(math.radians(0))}")  # 1.0

# Rounding functions
print(f"Ceil of 4.3: {math.ceil(4.3)}")  # 5
print(f"Floor of 4.7: {math.floor(4.7)}")  # 4

# Logarithmic functions
print(f"Log base 10 of 100: {math.log10(100)}")  # 2.0
print(f"Natural log of e: {math.log(math.e)}")  # 1.0
```

random Module

Definition: The random module provides functions to generate random numbers, make random selections, and shuffle sequences.

Syntax:

python

```
import random
```

Example:

python

```
import random


# Random integer in range
dice = random.randint(1, 6)
print(f"Dice roll: {dice}")


# Random float between 0 and 1
rand_float = random.random()
print(f"Random float: {rand_float}")


# Random float in range
temperature = random.uniform(20.0, 30.0)
print(f"Temperature: {temperature:.2f}")


# Random choice from list
fruits = ['apple', 'banana', 'cherry', 'mango']
selected = random.choice(fruits)
print(f"Selected fruit: {selected}")
```

 # Random sample (multiple items)

winners = random.sample(fruits, 2)

print(f"Winners: {winners}")

 # Shuffle list

numbers = [1, 2, 3, 4, 5]

random.shuffle(numbers)

print(f"Shuffled: {numbers}")

datetime Module

Definition: The datetime module provides classes for manipulating dates and times, including formatting, parsing, and arithmetic operations on dates.

Syntax:

python

import datetime

Example:

python

import datetime

 # Current date and time

now = datetime.datetime.now()

print(f"Current datetime: {now}")

 # Current date only

today = datetime.date.today()

print(f"Today's date: {today}")

```
# Current time only
current_time = datetime.datetime.now().time()
print(f"Current time: {current_time}")

# Creating specific date
birthday = datetime.date(1995, 5, 15)
print(f"Birthday: {birthday}")

# Creating specific datetime
event = datetime.datetime(2025, 12, 31, 23, 59, 59)
print(f"New Year Event: {event}")

# Formatting dates
formatted = now.strftime("%d-%m-%Y %H:%M:%S")
print(f"Formatted: {formatted}")

# Date arithmetic
tomorrow = today + datetime.timedelta(days=1)
print(f"Tomorrow: {tomorrow}")

week_ago = today - datetime.timedelta(weeks=1)
print(f"Week ago: {week_ago}")

# Components of date
print(f"Year: {today.year}")
```

```
print(f"Month: {today.month}")

print(f"Day: {today.day}")
```

os Module

Definition: The os module provides functions to interact with the operating system, including file and directory operations, environment variables, and process management.

Syntax:

```
python

import os
```

Example:

```
python

import os


# Current working directory

cwd = os.getcwd()

print(f"Current directory: {cwd}")


# List files in directory

files = os.listdir('.')

print(f"Files: {files}")


# Check if path exists

exists = os.path.exists('sample.txt')

print(f"File exists: {exists}")


# Check if it's a file or directory

is_file = os.path.isfile('sample.txt')
```

```
is_dir = os.path.isdir('my_folder')
print(f"Is file: {is_file}, Is directory: {is_dir}")

# Get file size
if os.path.exists('sample.txt'):
    size = os.path.getsize('sample.txt')
    print(f"File size: {size} bytes")

# Create directory
# os.mkdir('new_folder')

# Join paths (OS-independent)
file_path = os.path.join('folder', 'subfolder', 'file.txt')
print(f"Path: {file_path}")
```

sys Module

Definition: The sys module provides access to system-specific parameters and functions, including command-line arguments, Python version, and exit functions.

Syntax:

```
python
import sys
```

Example:

```
python
import sys

# Python version
print(f"Python version: {sys.version}")
```

```
# Platform information
print(f"Platform: {sys.platform}")

# Command line arguments
print(f"Script name: {sys.argv[0]}")
print(f"Arguments: {sys.argv}")

# Maximum integer size
print(f"Max int: {sys.maxsize}")

# Exit program
# sys.exit("Exiting program")

# Python path
print("Python path:")
for path in sys.path[:3]:  # Show first 3 paths
    print(path)
```

---

4. Creating Custom Modules

Definition: Custom modules are Python files you create containing your own functions, classes, and variables that can be imported and reused in other programs. They help organize large projects into manageable, reusable components.

Creating a Module

Step 1: Create a module file (mymodule.py)

Example:

python

```
# mymodule.py - Save this as a separate file

"""
This is a custom module for basic calculations
"""

# Module-level variable
PI = 3.14159

# Functions
def add(a, b):
    """Add two numbers"""
    return a + b

def subtract(a, b):
    """Subtract two numbers"""
    return a - b

def multiply(a, b):
    """Multiply two numbers"""
    return a * b

def divide(a, b):
    """Divide two numbers"""
    if b == 0:
        return "Error: Division by zero"
```

```
    return a / b


def greet(name):
    """Greet a person"""
    return f"Hello, {name}!"


# Class
class Calculator:
    """A simple calculator class"""


    def __init__(self):
        self.result = 0


    def add(self, num):
        self.result += num
        return self.result


    def reset(self):
        self.result = 0


# Code that runs only when module is executed directly
if __name__ == "__main__":
    print("This is mymodule")
    print(f"PI value: {PI}")
```

Using Custom Module

Step 2: Import and use the custom module

Example:

python

```
# main.py - Your main program file

# Method 1: Import entire module
import mymodule

print(mymodule.add(10, 5))  # 15
print(mymodule.multiply(4, 3))  # 12
print(mymodule.greet("Alice"))  # Hello, Alice!
print(f"PI from module: {mymodule.PI}")

# Using class from module
calc = mymodule.Calculator()
calc.add(10)
calc.add(5)
print(f"Calculator result: {calc.result}")  # 15

# Method 2: Import specific items
from mymodule import subtract, divide, PI

print(subtract(20, 8))  # 12
print(divide(100, 5))  # 20.0
print(f"PI value: {PI}")

# Method 3: Import with alias
```

```
import mymodule as mm


print(mm.add(7, 3))  # 10
print(mm.multiply(5, 5))  # 25
```

The dir() and help() Functions

Definition: dir() returns a list of all attributes and methods of a module/object. help() displays documentation for a module, function, or class.

Example:

```
python
import math


# List all functions in math module
print(dir(math))


# Get help on specific function
help(math.sqrt)


# For custom module
import mymodule
print(dir(mymodule))
help(mymodule.add)
```