DAY – 12

1. Python Classes: An Introduction

A **class** is a blueprint for creating objects in Python. It defines the properties and behaviours that objects of that class will have. In simple terms, a class is like a template that defines the structure of an object.

Creating a Class

In Python, a class is created using the class keyword followed by the class name. A class can contain variables (attributes) and functions (methods).

```
# Creating a simple class

class Person:

    # Class Attribute

    species = "Homo sapiens"

    # Constructor Method (Initializer)

    def __init__(self, name, age):

        self.name = name  # Instance Attribute

        self.age = age    # Instance Attribute

    # Method to display person details

    def greet(self):

        return f"Hello, my name is {self.name} and I am {self.age} years old."

# Creating an object of the class

person1 = Person("Alice", 25)

# Calling the method

print(person1.greet())  # Output: Hello, my name is Alice and I am 25 years old.
```

Key Points:

- **Constructor Method (__init__)**: Initializes instance variables when a new object is created.
- **Instance Variables**: Defined inside the constructor (self.name, self.age).
- **Class Variables**: Defined outside the constructor and are shared across all instances (species).
- **Methods**: Functions defined inside a class that describe the behaviors of the object.

2. Python Inheritance

Inheritance allows a class (child class) to inherit attributes and methods from another class (parent class). It helps to avoid redundancy and allows for the reuse of code.

```
# Parent class
class Animal:
    def __init__(self, name):
        self.name = name
    def sound(self):
        return f"{self.name} makes a sound"
# Child class inheriting from Animal
class Dog(Animal):
    def sound(self):
        return f"{self.name} barks"
# Creating object of the child class
dog1 = Dog("Buddy")
print(dog1.sound())  # Output: Buddy barks
```

Key Points:

- A child class inherits methods and attributes from the parent class.
- The child class can override methods of the parent class.
- Inheritance helps to maintain DRY (Don't Repeat Yourself) principle.

3. Python Regex and the re Module

Regular expressions (regex) are sequences of characters that form search patterns. The re module in Python allows you to work with regular expressions.

Commonly Used re Module Functions

- re.match(pattern, string): Checks for a match at the start of the string.
- re.search(pattern, string): Searches for the pattern anywhere in the string.
- re.findall(pattern, string): Returns all occurrences of the pattern.
- re.sub(pattern, replacement, string): Replaces occurrences of the pattern with a replacement.

Example 1: Using re.match and re.search

```
import re
# Example using re.match
pattern = r"Hello"
text = "Hello, world!"
match_result = re.match(pattern, text)
```

```
if match_result:
    print("Match found:", match_result.group())
else:
    print("No match found")
# Example using re.search
search_result = re.search(pattern, text)
if search_result:
    print("Search found:", search_result.group())
else:
    print("No match found")
```

Output:

```
Match found: Hello
Search found: Hello
```

Example 2: Using re.findall

```
import re
# Find all numbers in a string
text = "There are 3 cats and 4 dogs."
pattern = r"\d+"  # \d matches digits
numbers = re.findall(pattern, text)
print("Numbers found:", numbers)
```

Output:

```
Numbers found: ['3', '4']
```

Example 3: Using re.sub for Replacing Text

```
import re
text = "I have a cat, and my cat is cute."
pattern = r"cat"
replacement = "dog"
modified_text = re.sub(pattern, replacement, text)
print("Modified Text:", modified_text)
```

Output:

Modified Text: I have a dog, and my dog is cute.

Key Points:

- Regex patterns can match specific character sequences in strings.
- Use \d for digits, \w for words, \s for spaces, and other metacharacters.
- The re module makes it easy to manipulate strings using patterns.

4. Importing Libraries and Modules in Python

Python allows you to import pre-written modules and libraries to extend its functionality. There are multiple ways to import modules and libraries into a Python program.

Importing Entire Module

You can import the entire module using the import keyword.

```
import math

# Using a function from the math module

result = math.sqrt(16)

print("Square Root of 16 is:", result)  # Output: 4.0
```

Importing Specific Functions from a Module

You can import specific functions from a module using the from ... import ... syntax.

```
from math import pow

# Using the pow function

result = pow(2, 3)

print("2 raised to the power 3 is:", result)  # Output: 8.0
```

Importing with Alias

You can assign an alias to a module to shorten its name using the as keyword.

```
import numpy as np

# Using numpy with alias np

array = np.array([1, 2, 3])

print("Numpy Array:", array)  # Output: [1 2 3]
```

Importing from a Custom Module

You can also import from your own Python files or modules. For example, if you have a custom file named my_module.py, you can import it like this:

```
# Assume my_module.py contains a function called greet()

from my_module import greet
```

```
greet()  # Call the greet function from my_module
```

5. Organizing Code with Packages

A **package** is a collection of modules. It allows for better organization of code into smaller, reusable components. You can create a package by placing your modules in a directory and adding an __init__.py file to that directory.

Example Directory Structure:

```
my_package/
    __init__.py
    module1.py
    module2.py
```

To import from a package:

```
from my_package import module1
module1.some_function()
```