Advance Python Assignment

1. Printing on Screen

Theory:

1• Introduction to the print() function in Python.

Ans : Python print() function prints the message to the screen or any other standard output device. In this article, we will cover about print() function in Python as well as it's various operations.

2• Formatting outputs using f-strings and format().

Ans : Python offers a powerful feature called f-strings (formatted string literals) to simplify string formatting and interpolation. f-strings is introduced in Python 3.6 it provides a concise and intuitive way to embed expressions and variables directly into strings. The idea behind f-strings is to make string interpolation simpler.

Lab: • Write a Python program to print a formatted string using print() and f-string.

Solution :

english = 78

maths = 56

hindi = 85

print(f"Ram got total marks {english + maths + hindi} out of 300")

Output : Ram got total marks 219 out of 300

Practical Example: 1. Write a Python program to print “Hello, World!” on the screen.

Solution :

a="Hello World"

print(a)

Output : Hello World

2. Reading Data from Keyboard Theory:

• Using the input() function to read user input from the keyboard. • Converting user input into different data types (e.g., int, float, etc.).

Ans :

1 String type :

name = input("Enter your name: ")

print("Hello", name)

2 Integer type :

age = int(input("Enter your age: "))

print("Next year, you will be", age)

3.Float type :

price = float(input("Enter the price: "))

print("Price entered is", price)

Lab:

• Write a Python program to read a name and age from the user and print a formatted output.

Solution :

name = input("Enter Your Name :")

age  = int(input("Enter Your Age :"))

print(f"Hello, {name}! You are {age} years old.")

Practical Example: 2) Write a Python program to read a string, an integer, and a float from the keyboard and display them.

Solution :

name = input("Enter A Name :")

age  = int(input("Enter Your Age :"))

price = float(input("Enter Price in float number :"))

print("Display All the things")

print("Your name is :",name)

print("Your age will be :",age)

print("The price is :",price)

3 Theory: • Opening files in different modes ('r', 'w', 'a', 'r+', 'w+').

• Using the open() function to create and access files. • Closing files using close().

Ans :

| Mode   | Description                                                               |

| `'r'`  | Read mode (default). Opens file for reading. Error if file doesn't exist. |

| `'w'`  | Write mode. Creates new file or overwrites if file exists.                 |

| `'a'`  | Append mode. Creates file if not exists, adds data at the end.             |

| `'r+'` | Read and write. File must exist.                                          |

| `'w+'` | Write and read. Overwrites existing file or creates a new one.            |

Lab:

• Write a Python program to open a file in write mode, write some text, and then close it.

Solution :

# Open the file in write mode

file = open("myfile.txt", "w")

# Write text to the file

file.write("Hello, this is a file handling lab example.\n")

file.write("We are writing data to a file using Python.\n")

# Close the file

file.close()

print("Data written successfully to myfile.txt")

Practical Example:

3) Write a Python program to create a file and write a string into it.

Solution :

# Open or create the file in write mode

file = open("output.txt", "w")

# Write a string into the file

file.write("Python makes file handling easy!\nThis is a practical example.")

# Close the file

file.close()

print("File 'output.txt' created and string written successfully.")

4. Reading and Writing Files Theory: • Reading from a file using read(), readline(), readlines(). • Writing to a file using write() and writelines().

Ans:

1.read()

f = open("sample.txt", "r")

content = f.read()

print(content)

f.close()

2.readline()

f = open("sample.txt", "r")

line1 = f.readline()

line2 = f.readline()

print(line1)

print(line2)

f.close()

3.readlines()

f = open("sample.txt", "r")

lines = f.readlines()

print(lines)

f.close()

Writing a File:

1.write()

f = open("sample.txt", "w")

f.write("Hello, world!\n")

f.write("This is a new line.")

f.close()

2. writelines()

f = open("sample.txt", "w")

lines = ["Line 1\n", "Line 2\n", "Line 3\n"]

f.writelines(lines)

f.close()

Lab: • Write a Python program to read the contents of a file and print them on the console. • Write a Python program to write multiple strings into a file.

Solution :

1.

# Open the file in read mode

file = open("sample.txt", "r")

# Read the entire content

content = file.read()

# Print content to console

print("File Contents:\n")

print(content)

# Close the file

file.close()

2.

# Open the file in write mode

file = open("output.txt", "w")

# List of strings to write

lines = [

"This is line 1.\n",

"This is line 2.\n",

"This is line 3.\n"

]

# Write the list of strings to the file

file.writelines(lines)

# Close the file

file.close()

print("Multiple lines written to output.txt successfully.")

Practical Examples: 4) Write a Python program to create a file and print the string into the file. 5) Write a Python program to read a file and print the data on the console. 6) Write a Python program to check the current position of the file cursor using tell().

Solution :

4.

# Open (or create) the file in write mode

file = open("example4.txt", "w")

# Write a string to the file

file.write("This is Practical Example 4.\nPython file handling is easy!")

# Close the file

file.close()

print("Data written to example4.txt successfully.")

5.

# Open the file in read mode

file = open("example4.txt", "r")

# Read the content

data = file.read()

# Print the data

print("Content of example4.txt:")

print(data)

# Close the file

file.close()

6.

# Open the file in read mode

file = open("example4.txt", "r")

# Read some content

text = file.read(10)

# Print the text read

print("Read text:", text)

# Check the cursor position

position = file.tell()

print("Current cursor position:", position)

# Close the file

file.close()

5.Exception Handling

Theory: •Introduction to exceptions and how to handle them using try, except, and finally. • Understanding multiple exceptions and custom exceptions.

Ans :

**1. Introduction to Exceptions**

* **Exceptions** are errors that occur during the execution of a program.
* When an exception occurs, Python stops normal execution and looks for a way to handle the error.
* If unhandled, the program crashes and shows a traceback (error message).

try:

    n1=int(input("Enter a number 1:"))

    print(n1)

except ValueError as e:

    print(e)

else:

    print("Try Executed !!")

finally:

   print("Finally")

2.

try:

    n=int(input("Enter a number 1:"))

    n1=int(input("Enter a number 2:"))

    print("divison :",n/n1)

except:

    print("Invalid Input !!!")

Lab: • Write a Python program to handle exceptions in a simple calculator (division by zero, invalid input). • Write a Python program to demonstrate handling multiple exceptions.

Solution :

try:

num1 = float(input("Enter numerator: "))

num2 = float(input("Enter denominator: "))

result = num1 / num2

except ZeroDivisionError:

print("Error: Division by zero is not allowed.")

except ValueError:

print("Error: Invalid input! Please enter numeric values only.")

else:

print(f"Result: {result}")

finally:

print("Calculator execution completed.")

Practical Examples: 7) Write a Python program to handle exceptions in a calculator. 8) Write a Python program to handle multiple exceptions (e.g., file not found, division by zero). 9) Write a Python program to handle file exceptions and use the finally block for closing the file. 10) Write a Python program to print custom exceptions.

Solution :

7.

try:

num1 = float(input("Enter first number: "))

num2 = float(input("Enter second number: "))

operation = input("Choose operation (+, -, \*, /): ")

if operation == '+':

result = num1 + num2

elif operation == '-':

result = num1 - num2

elif operation == '\*':

result = num1 \* num2

elif operation == '/':

result = num1 / num2

else:

print("Invalid operation!")

result = None

if result is not None:

print(f"Result: {result}")

except ValueError:

print("Error: Please enter valid numbers.")

except ZeroDivisionError:

print("Error: Division by zero is not allowed.")

8.

try:

filename = input("Enter filename to read: ")

with open(filename, 'r') as file:

content = file.read()

print("File content:\n", content)

num = int(input("Enter a number to divide 100 by: "))

result = 100 / num

print("Result:", result)

except FileNotFoundError:

print("Error: File not found.")

except ZeroDivisionError:

print("Error: Cannot divide by zero.")

except ValueError:

print("Error: Invalid input! Please enter a number.")

9.

try:

file = open("testfile.txt", "r")

data = file.read()

print("File data:\n", data)

except FileNotFoundError:

print("Error: File not found.")

finally:

# Close the file if it was opened

try:

file.close()

print("File closed successfully.")

except NameError:

print("File was never opened.")

10.

# Define a custom exception class

class NegativeNumberError(Exception):

def \_\_init\_\_(self, message="Negative numbers are not allowed"):

self.message = message

super().\_\_init\_\_(self.message)

def check\_positive(number):

if number < 0:

raise NegativeNumberError

try:

num = int(input("Enter a positive number: "))

check\_positive(num)

print(f"You entered: {num}")

except NegativeNumberError as e:

print("Custom Exception Caught:", e)

except ValueError:

print("Error: Invalid input! Please enter an integer.")

6.Class And Objects (OOPs)

Theory: • Understanding the concepts of classes, objects, attributes, and methods in Python. • Difference between local and global variables.

1.

Class : Class is a collection of data member and member function.

Object : All the property of a class can access through the object, if we create an class we have to write the object of the class

Attributes : Variables that hold data related to the object.

Defined inside the \_\_init\_\_ method using self.

Methods : Functions defined inside a class.

Can access and modify object data using self.

2.Local and Global Variable

| **Feature** | **Local Variable** | **Global Variable** |
| --- | --- | --- |
| Scope | Inside a function or block | Outside all functions (global scope) |
| Access | Only within the function | Accessible anywhere in the code |
| Lifetime | Exists until the function ends | Exists until the program ends |

Lab: • Write a Python program to create a class and access its properties using an object.

Solution :

class Myclass:

    def fun1(self):

        n=int(input("Enter a number :"))

        fac=1

        i=1

        for i in range(1,n+1):

            fac= fac\*i

            i=i+1

        print("Factorial of a number is :",fac)

obj=Myclass()

obj.fun1()

Practical Examples: 11) Write a Python program to create a class and access the properties of the class using an object. 12) Write a Python program to demonstrate the use of local and global variables in a class.

Solution :

11.class Myclass:

   def myfun1(self):

       print("This is Function 1 !!!")

   def myfun2(self):

       print("This is function 2 !!!")

obj = Myclass()

obj.myfun1()

obj.myfun2()

12.

# Global variable

school\_name = "Green Valley School"

class Student:

def \_\_init\_\_(self, name):

self.name = name

def show\_info(self):

# Local variable

age = 16

print(f"Student Name: {self.name}")

print(f"Student Age (Local Variable): {age}")

print(f"School Name (Global Variable): {school\_name}")

# Create an object

s1 = Student("Vinay")

# Call method to demonstrate local and global variables

s1.show\_info()

7. Inheritance Theory: • Single, Multilevel, Multiple, Hierarchical, and Hybrid inheritance in Python. • Using the super() function to access properties of the parent class.

Ans :

Inheritance allows one class (**child/derived class**) to acquire the properties and behaviors (methods and attributes) of another class (**parent/base class**). This promotes **code reuse** and **logical hierarchy**.

1.Single level:

class Parent:

def display(self):

print("Parent class")

class Child(Parent):

def show(self):

print("Child class")

obj = Child()

obj.display()

obj.show()

2.Multilevel :

class Grandparent:

def grand\_method(self):

print("Grandparent class")

class Parent(Grandparent):

def parent\_method(self):

print("Parent class")

class Child(Parent):

def child\_method(self):

print("Child class")

obj = Child()

obj.grand\_method()

obj.parent\_method()

obj.child\_method()

3.Multiple:

class Father:

def father\_method(self):

print("Father class")

class Mother:

def mother\_method(self):

print("Mother class")

class Child(Father, Mother):

def child\_method(self):

print("Child class")

obj = Child()

obj.father\_method()

obj.mother\_method()

obj.child\_method()

4.Hierarchi :

class Parent:

def show(self):

print("Parent class")

class Child1(Parent):

def child1\_method(self):

print("Child1 class")

class Child2(Parent):

def child2\_method(self):

print("Child2 class")

obj1 = Child1()

obj2 = Child2()

obj1.show()

obj2.show()

5. Hybrid:

class A:

def method\_a(self):

print("Class A")

class B(A):

def method\_b(self):

print("Class B")

class C:

def method\_c(self):

print("Class C")

class D(B, C): # Hybrid: Multilevel + Multiple

def method\_d(self):

print("Class D")

obj = D()

obj.method\_a()

obj.method\_b()

obj.method\_c()

obj.method\_d()

2nd.

class Animal:

def \_\_init\_\_(self, name):

self.name = name

def speak(self):

print(f"{self.name} makes a sound")

class Dog(Animal):

def \_\_init\_\_(self, name, breed):

super().\_\_init\_\_(name) # Call parent constructor

self.breed = breed

def speak(self):

super().speak() # Call parent method

print(f"{self.name} barks")

d = Dog("Tommy", "Bulldog")

d.speak()

Lab: • Write Python programs to demonstrate different types of inheritance (single, multiple, multilevel, etc.).

Solution :

# Multilevel Inheritance

class MyclassA:

     def myfun1(self):

          print("This is Function 1 !!!")

class MyclassB(MyclassA):

     def myfun2(self):

       print("This is Function 2 !!! ")

class MyclassC(MyclassB):

     def myfun3(self):

         print("This is Function 3 !!!")

obj=MyclassC()

obj.myfun1()

obj.myfun2()

obj.myfun3()

#Hierarchi inheritance = single parent with multiple child

class MyclassA:

    def myfun1(self):

        print("This is method 1 !!!")

class MyclassB(MyclassA):

    def myfun2(self):

        print("This is method 2 !!!")

class MyclassC(MyclassA):

    def myfun3(self):

        print("This is method 3 !!!")

print("Child class B access :")

obj = MyclassB()

obj.myfun1()

obj.myfun2()

print("Child class C access :")

obj = MyclassC()

obj.myfun1()

obj.myfun3()

#Hybrid inheritance = merge single level and multiple

class MyclassA:

    def myfun1(self):

       print("This is method 1 !!!")

class MyclassB(MyclassA):

    def myfun2(self):

        print("This is method 2 !!!")

class MyclassC:

    def myfun3(self):

        print("This is method 3 !!!")

class MyclassD(MyclassB,MyclassC):

    def myfun4(self):

        print("This is Method 4 !!!")

obj=MyclassD()

obj.myfun1()

obj.myfun2()

obj.myfun3()

obj.myfun4()

Single Level :

# Parent class

class Animal:

def speak(self):

print("Animal speaks")

# Child class inheriting from Animal

class Dog(Animal):

def bark(self):

print("Dog barks")

# Creating object of child class

d = Dog()

# Accessing methods

d.speak() # Inherited from Animal

d.bark() # Defined in Dog

5.Multiple:

# First parent class

class Father:

def show\_father(self):

print("Father's properties")

# Second parent class

class Mother:

def show\_mother(self):

print("Mother's properties")

# Child class inherits from both Father and Mother

class Child(Father, Mother):

def show\_child(self):

print("Child's own properties")

# Create object of Child

c = Child()

# Accessing all parent and child methods

c.show\_father()

c.show\_mother()

c.show\_child()

Practical Examples: 13) Write a Python program to show single inheritance. 14) Write a Python program to show multilevel inheritance. 15) Write a Python program to show multiple inheritance. 16) Write a Python program to show hierarchical inheritance. 17) Write a Python program to show hybrid inheritance. 18) Write a Python program to demonstrate the use of super() in inheritance.

Solution :

1.Single Level :

class MyclassA:

    def myfun1(self):

       print("This is method 1 !!!")

obj=MyclassA()

obj.myfun1()

2.Multilevel :

# Multilevel Inheritance

class MyclassA:

     def myfun1(self):

          print("This is Function 1 !!!")

class MyclassB(MyclassA):

     def myfun2(self):

       print("This is Function 2 !!! ")

class MyclassC(MyclassB):

     def myfun3(self):

         print("This is Function 3 !!!")

obj=MyclassC()

obj.myfun1()

obj.myfun2()

obj.myfun3()

3.Multiple :

# Multiple Inheritance

class MyclassA:

    def myfun1(self):

          print("This is Function 1 !!!")

class MyclassB():

    def myfun2(self):

       print("This is Function 2 !!! ")

class MyclassC(MyclassB,MyclassA):

    def myfun3(self):

         print("This is Function 3 !!!")

obj=MyclassC()

obj.myfun1()

obj.myfun2()

obj.myfun3()

4.Hierarchi :

#Hierarchi inheritance = single parent with multiple child

class MyclassA:

    def myfun1(self):

        print("This is method 1 !!!")

class MyclassB(MyclassA):

    def myfun2(self):

        print("This is method 2 !!!")

class MyclassC(MyclassA):

    def myfun3(self):

        print("This is method 3 !!!")

print("Child class B access :")

obj = MyclassB()

obj.myfun1()

obj.myfun2()

print("Child class C access :")

obj = MyclassC()

obj.myfun1()

obj.myfun3()

5.Hybrid :

#Hybrid inheritance = merge single level and multiple

class MyclassA:

    def myfun1(self):

       print("This is method 1 !!!")

class MyclassB(MyclassA):

    def myfun2(self):

        print("This is method 2 !!!")

class MyclassC:

    def myfun3(self):

        print("This is method 3 !!!")

class MyclassD(MyclassB,MyclassC):

    def myfun4(self):

        print("THis is Method 4 !!!")

obj=MyclassA()

obj.myfun1()

obj.myfun2()

obj.myfun3()

obj.myfun4()

Super() Function:

class Parent:

def \_\_init\_\_(self):

print("Parent constructor")

def show(self):

print("Method in Parent")

class Child(Parent):

def \_\_init\_\_(self):

super().\_\_init\_\_() # Call parent constructor

print("Child constructor")

def show(self):

super().show() # Call parent method

print("Method in Child")

# Create object

c = Child()

c.show()

8. Method Overloading and Overriding Theory: • Method overloading: defining multiple methods with the same name but different parameters. • Method overriding: redefining a parent class method in the child class.

Ans:

* **Method Overloading** means defining **multiple methods with the same name** but **different number/type of parameters**.
* Python **does not support true method overloading** like Java/C++.
* But we can **simulate** it using **default arguments** or \*args.

Example :

class Dog(Animal):

def speak(self):

super().speak() # Calls parent class method

print("Dog also barks")

d = Dog()

d.speak()

Example:

class Calculator:

def add(self, a=None, b=None, c=None):

if a is not None and b is not None and c is not None:

return a + b + c

elif a is not None and b is not None:

return a + b

elif a is not None:

return a

else:

return 0

# Create object

calc = Calculator()

print(calc.add(5, 10)) # 2 arguments

print(calc.add(1, 2, 3)) # 3 arguments

print(calc.add(7)) # 1 argument

Lab: • Write Python programs to demonstrate method overloading and method overriding.

Solution:

1. class Animal:

def sound(self):

print("Animals make sounds")

class Dog(Animal):

def sound(self):

super().sound() # Call the parent class method

print("Dog barks loudly")

# Create object of Dog

dog = Dog()

dog.sound()

1. # Method Overloading using default arguments

class Calculator:

def add(self, a=None, b=None, c=None):

if a is not None and b is not None and c is not None:

print("Sum of three numbers:", a + b + c)

elif a is not None and b is not None:

print("Sum of two numbers:", a + b)

elif a is not None:

print("Single number:", a)

else:

print("No arguments")

# Create object

calc = Calculator()

# Call add method with different number of arguments

calc.add(10, 20)

calc.add(5, 15, 25)

calc.add(100)

calc.add()

Practical Examples: 19) Write a Python program to show method overloading. 20) Write a Python program to show method overriding.

Solution :

19.

# Program to show Method Overloading

class AreaCalculator:

def area(self, a=None, b=None):

if a is not None and b is not None:

print("Area of Rectangle:", a \* b)

elif a is not None:

print("Area of Square:", a \* a)

else:

print("No dimensions provided")

# Create object

calc = AreaCalculator()

# Different calls simulate overloading

calc.area() # No argument

calc.area(5) # One argument (square)

calc.area(4, 6) # Two arguments (rectangle)

20.

# Program to show Method Overriding

class Vehicle:

def start(self):

print("Vehicle is starting...")

class Car(Vehicle):

def start(self):

print("Car is starting with a key...")

# Create object of Car

c = Car()

c.start() # Overridden method is called

Theory: • Introduction to SQLite3 and PyMySQL for database connectivity. • Creating and executing SQL queries from Python using these connectors.

Ans : import sqlite3

# Connect to SQLite database (or create it if it doesn't exist)

conn = sqlite3.connect('example.db')

cursor = conn.cursor()

# Create table

cursor.execute('''CREATE TABLE IF NOT EXISTS users (id INTEGER PRIMARY KEY, name TEXT, age INTEGER)''')

# Insert data

cursor.execute("INSERT INTO users (name, age) VALUES (?, ?)", ("Alice", 30))

# Query data

cursor.execute("SELECT \* FROM users")

rows = cursor.fetchall()

for row in rows:

print(row)

# Commit changes and close

conn.commit()

conn.close()

import pymysql

# Connect to MySQL database

conn = pymysql.connect(

host='localhost',

user='your\_username',

password='your\_password',

database='your\_db\_name'

)

cursor = conn.cursor()

# Create table

cursor.execute("CREATE TABLE IF NOT EXISTS users (id INT AUTO\_INCREMENT PRIMARY KEY, name VARCHAR(100), age INT)")

# Insert data

cursor.execute("INSERT INTO users (name, age) VALUES (%s, %s)", ("Bob", 25))

# Query data

cursor.execute("SELECT \* FROM users")

rows = cursor.fetchall()

for row in rows:

print(row)

# Commit changes and close

conn.commit()

conn.close()

Lab: • Write a Python program to connect to an SQLite3 database, create a table, insert data, and fetch data.

Solution : import sqlite3

# Step 1: Connect to SQLite3 database (creates file if it doesn't exist)

conn = sqlite3.connect('students.db')

cursor = conn.cursor()

# Step 2: Create a table

cursor.execute('''

CREATE TABLE IF NOT EXISTS students (

id INTEGER PRIMARY KEY AUTOINCREMENT,

name TEXT NOT NULL,

age INTEGER,

grade TEX

Student Records:

(1, 'Alice', 20, 'A')

(2, 'Bob', 22, 'B')

(3, 'Charlie', 21, 'A')

(4, 'Diana', 23, 'C')

Practical Examples: 21) Write a Python program to create a database and a table using SQLite3. 22) Write a Python program to insert data into an SQLite3 database and fetch it.

Solution : import sqlite3

# Connect to SQLite database (creates the file if it doesn't exist)

conn = sqlite3.connect('company.db')

cursor = conn.cursor()

# Create a table named 'employees'

cursor.execute('''

CREATE TABLE IF NOT EXISTS employees (

id INTEGER PRIMARY KEY AUTOINCREMENT,

name TEXT NOT NULL,

position TEXT NOT NULL,

salary REAL

)

''')

print("✅ Database and table 'employees' created successfully.")

# Commit and close the connection

conn.commit()

conn.close()

2nd :

import sqlite3

# Connect to the database

conn = sqlite3.connect('company.db')

cursor = conn.cursor()

# Insert data into the 'employees' table

employee\_data = [

('John Doe', 'Manager', 75000.00),

('Jane Smith', 'Developer', 65000.00),

('Emily Davis', 'Designer', 60000.00)

]

cursor.executemany("INSERT INTO employees (name, position, salary) VALUES (?, ?, ?)", employee\_data)

# Fetch all data from the table

cursor.execute("SELECT \* FROM employees")

records = cursor.fetchall()

# Display the records

print("📋 Employee Records:")

for row in records:

print(row)

# Commit and close the connection

conn.commit()

conn.close()

Employee Records:

(1, 'John Doe', 'Manager', 75000.0)

(2, 'Jane Smith', 'Developer', 65000.0)

(3, 'Emily Davis', 'Designer', 60000.0)

Theory: • Using re.search() and re.match() functions in Python’s re module for pattern matching. • Difference between search and match.

Ans : The re.match function in Python is part of the re module, which provides support for working with regular expressions. Regular expressions are a powerful tool for matching patterns in text.

**Definition and Usage**

The re.match function attempts to match a pattern at the **beginning** of a string. If the pattern is found at the start of the string, it returns a match object; otherwise, it returns None. This function is useful when you need to ensure that the pattern appears right at the start of the string

import re

pattern = r'string'

string1 = 'We are learning regex with geeksforgeeks'

string2 = 'string We are learning regex with geeksforgeeks'

# Use of re.match() Method

match1 = re.match(pattern, string1, re.IGNORECASE)

match2 = re.match(pattern, string2, re.IGNORECASE)

print(match1) # Output: None

print(match2) # Output: <re.Match object; span=(0, 6), match='string'>

**Key Points**

* **Position-Specific**: re.match only checks for a match at the beginning of the string. If the pattern appears elsewhere, it will not be matched[1](https://www.bing.com/ck/a?!&&p=3611146295b481fe6927c44372e09283bf82215e631449b93307e89ea7fdc4fcJmltdHM9MTc0OTg1OTIwMA&ptn=3&ver=2&hsh=4&fclid=0b42fb86-0169-6291-24d7-e96600f263ef&u=a1aHR0cHM6Ly93d3cuZ2Vla3Nmb3JnZWVrcy5vcmcvcHl0aG9uLXJlLXNlYXJjaC12cy1yZS1tYXRjaC8&ntb=1).
* **Match Object**: If a match is found, re.match returns a match object, which contains information about the match, such as the start and end positions and the matched text[2](https://www.bing.com/ck/a?!&&p=b83db8a40fd86cfc0dc25bd377251c5b88aa17ae2b708f2d7907984a0239f6daJmltdHM9MTc0OTg1OTIwMA&ptn=3&ver=2&hsh=4&fclid=0b42fb86-0169-6291-24d7-e96600f263ef&u=a1aHR0cHM6Ly93d3cuc2FuZm91bmRyeS5jb20vcHl0aG9uLXF1ZXN0aW9ucy1hbnN3ZXJzLXJlZ3VsYXItZXhwcmVzc2lvbnMv&ntb=1).
* **Flags**: You can use optional flags to modify the matching behavior. For example, re.IGNORECASE makes the match case-insensitive

2nd . While re.match checks for a match only at the beginning of the string, re.search scans the entire string for a match. This makes re.search more flexible when the pattern can appear anywhere in the string[1](https://www.bing.com/ck/a?!&&p=3611146295b481fe6927c44372e09283bf82215e631449b93307e89ea7fdc4fcJmltdHM9MTc0OTg1OTIwMA&ptn=3&ver=2&hsh=4&fclid=0b42fb86-0169-6291-24d7-e96600f263ef&u=a1aHR0cHM6Ly93d3cuZ2Vla3Nmb3JnZWVrcy5vcmcvcHl0aG9uLXJlLXNlYXJjaC12cy1yZS1tYXRjaC8&ntb=1)[3](https://www.bing.com/ck/a?!&&p=4deead46084a52155bb1d15b966d6a23587781aabba3a0f1b17c702762458379JmltdHM9MTc0OTg1OTIwMA&ptn=3&ver=2&hsh=4&fclid=0b42fb86-0169-6291-24d7-e96600f263ef&u=a1aHR0cHM6Ly93d3cuZ3VydTk5LmNvbS9weXRob24tcmVndWxhci1leHByZXNzaW9ucy1jb21wbGV0ZS10dXRvcmlhbC5odG1s&ntb=1).

import re

pattern = r'string'

string = 'We are learning regex with geeksforgeeks'

# Use of re.search() Method

search\_result = re.search(pattern, string, re.IGNORECASE)

print(search\_result) # Output: <re.Match object; span=(75, 81), match='string'>

Lab: • Write a Python program to search for a word in a string using re.search(). • Write a Python program to match a word in a string using re.match().

Solution : import re

# Input string and word to search

text = "Python is a powerful programming language."

word\_to\_search = "powerful"

# Search for the word in the string

if re.search(rf'\b{word\_to\_search}\b', text):

print(f"The word '{word\_to\_search}' was found in the string.")

else:

print(f"The word '{word\_to\_search}' was not found in the string.")

2nd .

import re

# Input string and word to match

text = "Python is a powerful programming language."

word\_to\_match = "Python"

# Match the word at the beginning of the string

if re.match(rf'\b{word\_to\_match}\b', text):

print(f"The word '{word\_to\_match}' matches the beginning of the string.")

else:

print(f"The word '{word\_to\_match}' does not match the beginning of the string.")

Practical Examples: 23) Write a Python program to search for a word in a string using re.search(). 24) Write a Python program to match a word in a string using re.match().

Solution :

import re

# Input string and word to search

text = "Python is a powerful programming language."

word = "powerful"

# Search for the word in the string

if re.search(rf'\b{word}\b', text):

print(f"The word '{word}' was found in the string.")

else:

print(f"The word '{word}' was not found in the string.")

2nd. import re

# Input string and word to match

text = "Python is a powerful programming language."

word = "Python"

# Match the word at the beginning of the string

if re.match(rf'\b{word}\b', text):

print(f"The word '{word}' matches the beginning of the string.")

else:

print(f"The word '{word}' does not match the beginning of the string.")