## **Module 8**

**(Advance Python Programming)**

1. **Printing on Screen**

**Theory:**

**1.1 Introduction to the print() function in Python.**

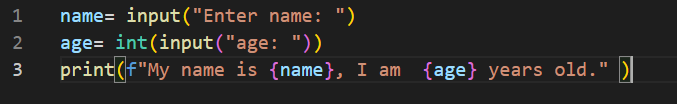
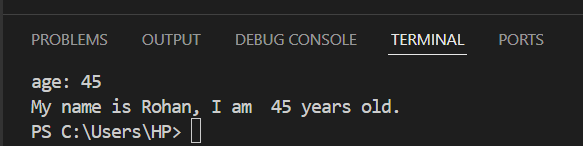
The print() function in python is one of the most widely used functions for displaying output on the screen. It allows programmers to show text, variables or any computed values during the execution of a program, making it an essential tool for debugging, logging and interacting with users.  
**key features of the print() function**

* Display output
* Accepts multiple arguments
* Customized separator
* Customizable end character
* Supports formatting
* Redirecting output
  1. **Formatting outputs using f-strings and format()**

Display output in well structured way by using f-string and formate(). it helps to formatting the output.

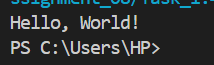
**Lab:**

* 1. **Write a Python program to print a formatted string using print() and f-string.**

 Output:

**Practical Example:**

**1.4 Write a Python program to print “Hello, World!” on the screen.**

Output:

1. **Reading Data from Keyboard**

**Theory:**

**2.1 Using the input() function to read user input from the keyboard.**

The input() function in Python is used to take input from the user by using keyboard. It always returns the input as a string, even if the user enters the numbers.

* 1. **Converting user input into different data types (e.g., int, float, etc.).**

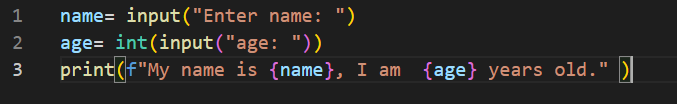
By default, the input() function returns user input as a string. If user want output in different data types, it can be possible by take input as int() or as float().

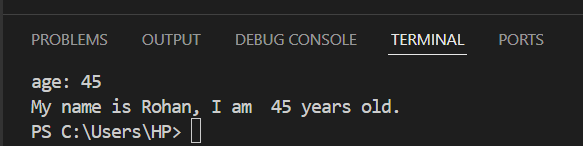
X= input(“name: “) ……it will be take i/p as string by\_default

Y= int(input(“y: “) …….it will be take i/p as into integer types.

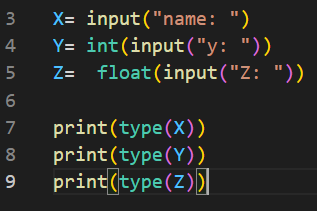
Z= float(input(“Z: “) …it will be take i/p as into float data types.

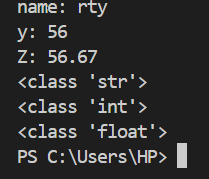
**Lab:**

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

Output:

**Practical Example:**

**2.4 Write a Python program to read a string, an integer, and a float from the keyboard and display them.**

**** Output:

1. **Opening and Closing Files**

**Theory:**

**3.1 Opening files in different modes ('r', 'w', 'a', 'r+', 'w+').**

**“r”: read**

It is used to read the existing file

If file doesn’t exist. It raise the error “File not found”

**“w”: write**

itis used to write something into existing file. If file doesn’t exist, it creates

a new file.

**“a”: append**

itis used to append or add the data into existing file, if file doesn’t exist. By default It creates a new file.

This mode doesn’t affect to the existing data.

**“r+”: read and write**

It is also use to read and write into existing file.

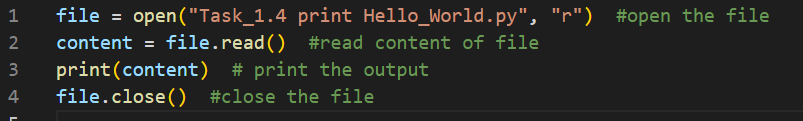
if file not fount it raises the “file not found” error. In this mode it doesn’t create new file.

**“w+”: write and read**

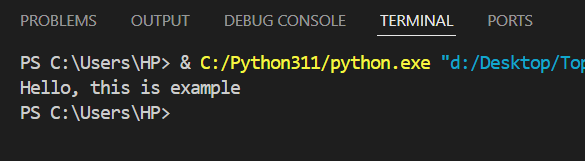
It is used to write and read the existing file. If file exist this method is used to overwrite into the existing file.

If file not fount, it doesn’t raise the “file not found” error, because it create the new one.

* 1. **Using the open() function to create and access files.**
* The open() function in python is used to open files that stored internally. It returns the file content as python objects.
* By using open() function, we can open a file in the current directory as well as the file located in a specified location with the help of its path.
* It allows us to create, read, write and modifies the files.
* The open() function returns a file object that can be used to perform operations on the file.



Output:

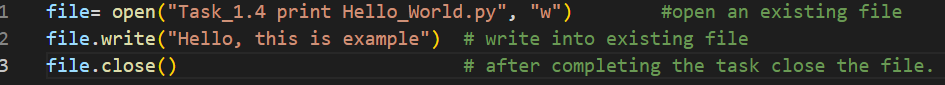


* 1. **Closing files using close().**
* The **close()** method closes an open file.
* It is important to close a file after performing operations like reading or writing to free up system resources and ensure data integrity.
* We should always close our files, in some cases, due to buffering, changes made to a file may not show until we doesn’t close the file.

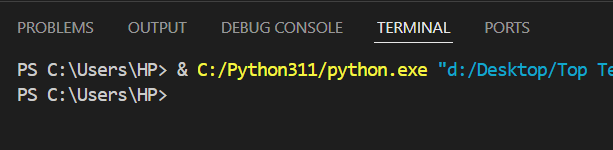
**File.close()**

**WHY SHOULD WE CLOSE A FILE**

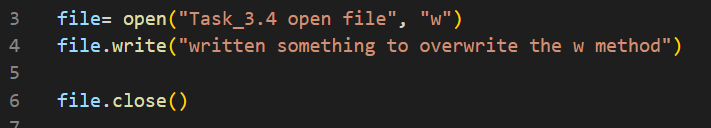
* if a file remains open, changes may not be saved immediately due to buffering.
* Open files consume memory, closing them to improves performance.
* If a program crashes or an error occurs before closing the file, data might be lost or corrupted.



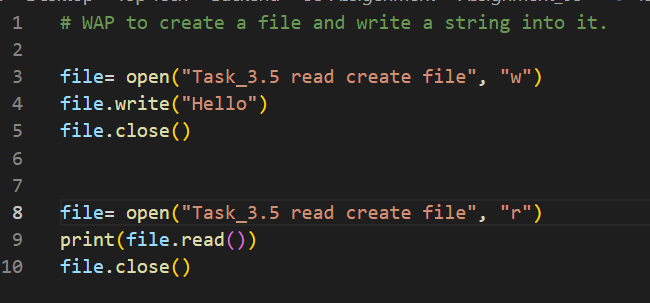
Output:

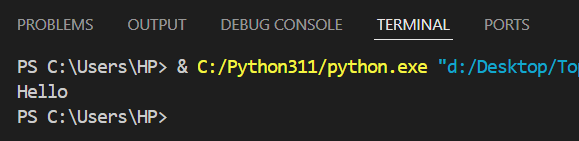


**Lab:**

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

**Practical Example:**

**3.5 Write a Python program to create a file and write a string into it.**

Output:

1. **Reading and Writing Files**

**Theory:**

**4.1 Reading from a file using read(), readline(), readlines().**

In python, we can read from a file by using different methods as read(), readline() and readlines(). Every method has a different use case depending on how much data we need to read at a time.

1. **read() :-** This method reads the entire file (or a specified number of bytes if size is given.
   * If size is specified, it reads only that how many bytes.
   * If user do not specify a value for the size parameter in read method as read(size), python will read the entire file by default
   * If no argument is given to read(), it will read the entire content of the file.
2. **readline() :-** This readline() method reads one line at a time.

* It stop reading at the newline character.
* Calling readline() again reads the next line.

1. **Readlines() :-** This method reads all lines and returns them as a list.

* Each line is an element in the list.
* Useful for processing lines in a loop.

|  |  |  |
| --- | --- | --- |
| **Method** | **Description** | **Output** |
| **read(size)** | * **Reads the whole file** * **Reads the file size into bytes** | **String** |
| **readline()** | **Reads one line at a time** | **String** |
| **readlines()** | **Reads all lines at once** | **List of string** |

* 1. **Writing to a file using write() and writelines().**

In python we can write data into a file by using write() and writelines() method.

**Write() :-** The write() method allows us to write a string into a file.

* The newline character “\n” must be included to move to a new line.
* If the file doesn’t exist, by default python will create the file.
* If the file already exists. It will be overwritten.

(also, we can use append method to add content into existing table.)

**Writelines() :-** The writelines() method is used to write multiple lines at once. It takes a list of string and writes them into the file by sequence.

* The wrtelines() method writes each string in the provided list to the file.
* It doesn’t add newlines character automatically.

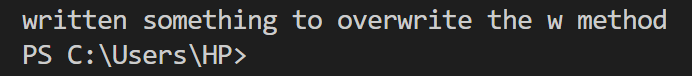
**Lab:**

* 1. **Write a Python program to read the contents of a file and print them on the console.**

file= open("Task\_3.4 open file", "r")

content= file.read()

print(content)

 Output:

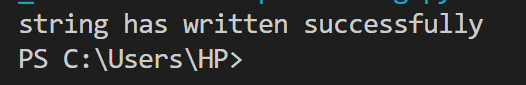
**4.4 Write a Python program to write multiple strings into a file.**

file= open("Task\_3.4 open file", "a")

multi\_strings= ["Hello", "\n I am a python developer", "\n currently working as backend developer", "\n thank you."]

file.writelines(multi\_strings)

print("string has written successfully")

Output:

**Practical Examples:**

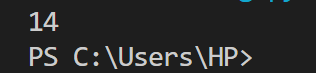
**4.5 Write a Python program to create a file and print the string into the file.**

file= open("Task\_4.5\_create\_file.py", "w")

num\_char = file.write("this is string")

file.close()

print(num\_char)

Output:

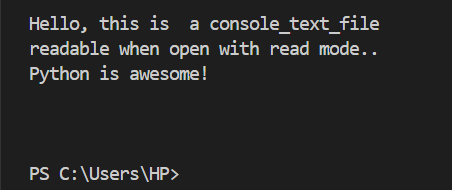
**4.6 Write a Python program to read a file and print the data on the console.**

file= open("D:\Desktop\Top Tech\Backend\Se-Assigenment\Assignment\_08\Task\_4.6\_

read\_text\_console.txt", "r")

print(file.read())

file.close()

Output:

**4.7 Write a Python program to check the current position of the file cursor using tell().**

file= open("D:\Desktop\Top Tech\Backend\Se-Assigenment\Assignment\_08\Task\_4.6\_ read\_text\_console.txt", "r")

print("\ncursor position before reading file: ", file.tell())

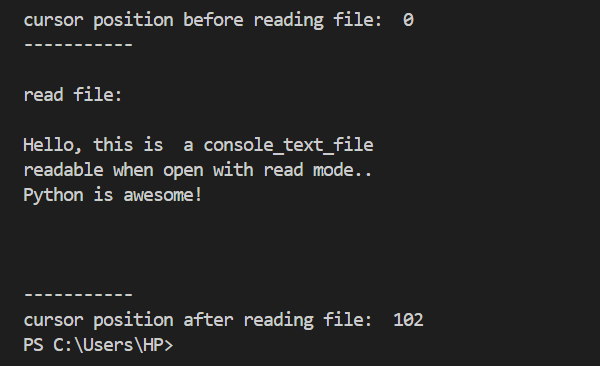
print("-----------")

print("\nread file:", file.read())

print("-----------")

print("cursor position after reading file: ", file.tell())

file.close()

Output:

1. **Exception Handling**

**Theory:**

**5.1 Introduction to exceptions and how to handle them using try, except, and finally.**

When a program encounters an **unexpected error** at runtime, it is known as an **exception**. If the error is not handled by the user, the program may **crash**. The process of handling such errors is known as **exception handling**. In Python, the try-except block is used to handle exceptions.  
 **Ex:**

a = 10

b = 0

print(a / b) # ZeroDivisionError: division by zero

(This will raise a **ZeroDivisionError** because division by zero is not allowed in Python.)

**There are four blocks of exception handling**

* **try:**

 The code that might cause an error is written inside the try block.

 If an error occurs, the control moves to the except block.

* **except:**

 The except block handles the exception and prevents the program from crashing.

 If an error occurs in the try block, the code inside except executes.

* **else:** *(optional)*

 If no error occurs in the try block, then the else block executes.

 The else block is **only executed when there is no exception**.

* **finally:**

 The finally block is used for cleanup operations like **closing files, terminating database connections, etc.**

 It **always executes**, regardless of whether an error occurs or not.

* 1. **Understanding multiple exceptions and custom exceptions.**

The except block handles the exception and prevents the program from crashing.

* **ZeroDivisionError:** when any number divide by zero. It raises an error.
* **ValueError:** when any value input by user as into the wrong format, it raises the ValueEroor.
* **except Exception as e:** sometimes user/developer unknown by the error or not able to understand the error, in this situation this use the generic exception handling. By using this all errors can handle and display the error message.

**Lab:**

* 1. **Write a Python program to handle exceptions in a simple calculator (division by zero, invalid input).**

class calc:

        def add(a, b):

                sum= a+b

                return sum

        def sub(a, b):

            subtract= a-b

            return subtract

        def mul(a, b):

            multiply = a\*b

            return multiply

        def div(a, b):

            division= a/b

            return division

menu= """

            press\_1 for addition

            press\_2 for subtract

            press\_3 for multiplication

            press\_4 for division

            """

print(menu)

try:

    choice= int(input("Enter number between 1 to 4: "))

    a= int(input("a= "))

    b= int(input("b= "))

    if(choice==1):

                    result= calc.add(a, b)

                    print("addition is:", result )

    elif(choice==2):

                    result= calc.sub(a, b)

                    print("subtraction is: ", result)

    elif(choice==3):

                    result= calc.mul(a, b)

                    print("multiplication is: ", result)

    elif(choice==4):

                    result= calc.div(a, b)

                    print("division is: ", result)

    else:

                print("please, enter correct choice!!")

except ZeroDivisionError as e:

        print("\*\*\*\*\*\*\*\*\*", e)

except Exception as e:

        print("invalid input", e)

finally:

print("thank you")

**5.4 Write a Python program to demonstrate handling multiple exceptions.**

class calc:

        def add(a, b):

                return a+b

        def sub(a, b):

            return a-b

        def mul(a, b):

            return a\*b

        def div(a, b):

            return a/b

menu= """

            press\_1 for addition

            press\_2 for subtract

            press\_3 for multiplication

            press\_4 for division

            """

print(menu)

try:

    choice= int(input("Enter number between 1 to 4: "))

    a= int(input("a= "))

    b= int(input("b= "))

    if(choice==1):

                    result= calc.add(a, b)

                    print("addition is:", result )

    elif(choice==2):

                    result= calc.sub(a, b)

                    print("subtraction is: ", result)

    elif(choice==3):

                    result= calc.mul(a, b)

                    print("multiplication is: ", result)

    elif(choice==4):

                    result= calc.div(a, b)

                    print("division is: ", result)

    else:

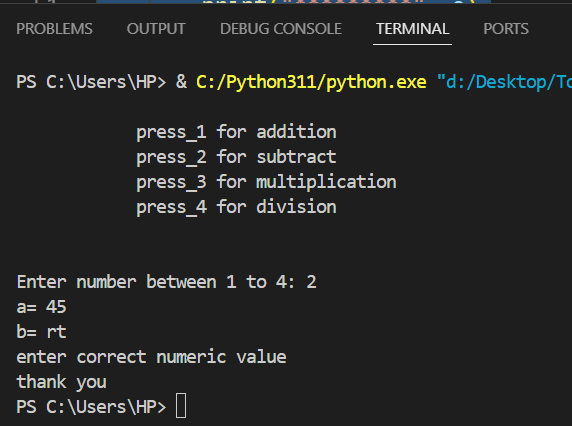
                print("please, enter correct choice!!")

except ZeroDivisionError as e:

        print("\*\*\*\*\*\*\*\*\*", e)

except ValueError:

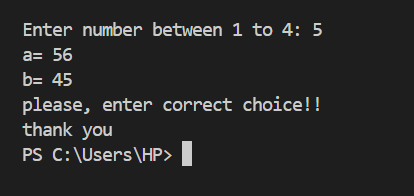
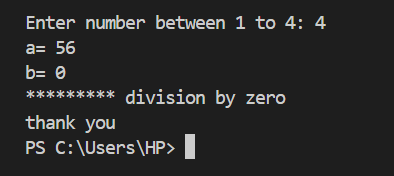
        print("enter correct numeric value")

except Exception as e:

        print("invalid input", e)

finally:

        print("thank you")

****Output:

**Practical Examples:**

**5.5 Write a Python program to handle exceptions in a calculator.**

class calc:                                                                              #class created "calc"

        def add(a, b):                                                                 # functions with parameter

                return a+b

        def sub(a, b):

            return a-b

        def mul(a, b):

            return a\*b

        def div(a, b):

            return a/b

menu= """

            press\_1 for addition

            press\_2 for subtract

            press\_3 for multiplication

            press\_4 for division

            """

print(menu)

try:

    choice= int(input("Enter number between 1 to 4: "))

    a= int(input("a= "))

    b= int(input("b= "))

    if(choice==1):

                    result= calc.add(a, b)                  #function called and passed the argument

                    print("addition is:", result )

    elif(choice==2):

                    result= calc.sub(a, b)

                    print("subtraction is: ", result)

    elif(choice==3):

                    result= calc.mul(a, b)

                    print("multiplication is: ", result)

    elif(choice==4):

                    result= calc.div(a, b)

                    print("division is: ", result)

    else:

                print("please, enter correct choice!!")

except ZeroDivisionError as e:                      #it show when user divide  by zero

        print("\*\*\*\*\*\*\*\*\*", e)

except ValueError:                                  #it display when wrong data type input

        print("enter correct numeric value")

except Exception as e:                              # it display when other exceptions not handle the error

        print("invalid input", e)

finally:                                            # it display when overall executions done.

        print("thank you")

**5.6 Write a Python program to handle multiple exceptions (e.g., file not found, division by zero).**

class calc:              #class created "calc"

        def add(a, b):   # functions with parameter

            return a+b

        def sub(a, b):

            return a-b

        def mul(a, b):

            return a\*b

        def div(a, b):

            return a/b

menu= """

            press\_1 for addition

            press\_2 for subtract

            press\_3 for multiplication

            press\_4 for division

            """

print(menu)

try:

    choice= int(input("Enter number between 1 to 4: "))

    a= int(input("a= "))

    b= int(input("b= "))

    if(choice==1):

                    result= calc.add(a, b)                  #function called and passed the argument

                    print("addition is:", result )

    elif(choice==2):

                    result= calc.sub(a, b)

                    print("subtraction is: ", result)

    elif(choice==3):

                    result= calc.mul(a, b)

                    print("multiplication is: ", result)

    elif(choice==4):

                    result= calc.div(a, b)

                    print("division is: ", result)

    else:

                print("please, enter correct choice!!")

except ZeroDivisionError as e:                      #it show when user divide  by zero

        print("\*\*\*\*\*\*\*\*\*", e)

except ValueError:                                  #it display when wrong data type input

        print("enter correct numeric value")

except Exception as e:                              # it display when other exceptions not handle the error

        print("invalid input", e)

finally:                                            # it display when overall executions done.

        print("thank you")

**5.7 Write a Python program to handle file exceptions and use the finally block for closing the file.**

class calc:                                                                              #class created "calc"

        def add(a, b):                                                                 # functions with parameter

                return a+b

        def sub(a, b):

            return a-b

        def mul(a, b):

            return a\*b

        def div(a, b):

            return a/b

menu= """

            press\_1 for addition

            press\_2 for subtract

            press\_3 for multiplication

            press\_4 for division

            """

print(menu)

try:

    choice= int(input("Enter number between 1 to 4: "))

    a= int(input("a= "))

    b= int(input("b= "))

    if(choice==1):

                    result= calc.add(a, b)                  #function called and passed the argument

                    print("addition is:", result )

    elif(choice==2):

                    result= calc.sub(a, b)

                    print("subtraction is: ", result)

    elif(choice==3):

                    result= calc.mul(a, b)

                    print("multiplication is: ", result)

    elif(choice==4):

                    result= calc.div(a, b)

                    print("division is: ", result)

    else:

                print("please, enter correct choice!!")

except FileNotFoundError as e:                      #it show when file not found

        print("file doesn't exist")

except ValueError:                                  #it display when wrong data type input

        print("enter correct numeric value")

except Exception as e:                              # it display when other exceptions not handle the error

        print("invalid input", e)

finally:                                            # it display when overall executions done.

        print("thank you")

**5.8 Write a Python program to print custom exceptions.**

class calc:                                  #class created "calc"

        def add(a, b):              # functions with parameter

            return a+b

        def sub(a, b):

            return a-b

        def mul(a, b):

            return a\*b

        def div(a, b):

            return a/b

menu= """

            press\_1 for addition

            press\_2 for subtract

            press\_3 for multiplication

            press\_4 for division

            """

print(menu)

try:

    choice= int(input("Enter number between 1 to 4: "))

    a= int(input("a= "))

    b= int(input("b= "))

    if(choice==1):

                    result= calc.add(a, b)                  #function called and passed the argument

                    print("addition is:", result )

    elif(choice==2):

                    result= calc.sub(a, b)

                    print("subtraction is: ", result)

    elif(choice==3):

                    result= calc.mul(a, b)

                    print("multiplication is: ", result)

    elif(choice==4):

                    result= calc.div(a, b)

                    print("division is: ", result)

    else:

                print("please, enter correct choice!!")

except FileNotFoundError as e:                      #it show when file not found

        print("file doesn't exist")

except ValueError:                                  #it display when wrong data type input

        print("enter correct numeric value")

except NegativeNumberError as e:

        print("Entered Negative numbers", e)

except InvalidChoiceError as e:

        print("Invalid choice", e)

except Exception as e:                              # it display when other exceptions not handle the error

        print("invalid input", e)

finally:                                            # it display when overall executions done.

        print("thank you")

1. **Class and Object (OOP Concepts)**

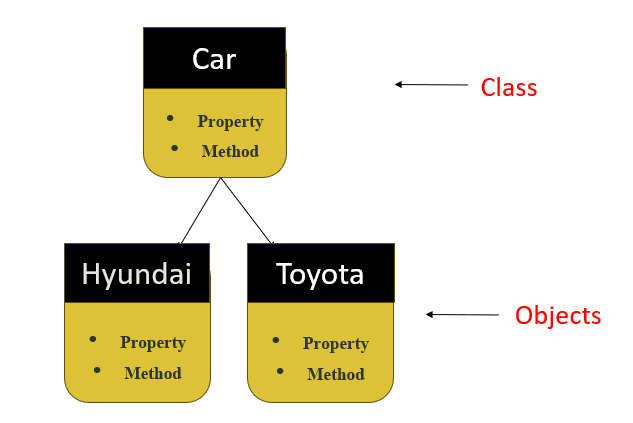
**Theory:**

* 1. **Understanding the concepts of classes, objects, attributes, and methods in Python.**

Python is a programming language that supports object-oriented programming. This makes it simple to create and use classes and objects. We can say that it revolves around the concepts of classes and objects.

**Let’s breakdown this concepts one by one:**

* **Classes:** A class is a blueprint or template for creating objects. It is a user-defined data type that contains both the data itself and the method that may be used to manipulate it.
* A class is a group of objects that has mutual methods. It can be considered as the blueprint using which objects are created.
* It can be a User Defined Data Type. Inside a class, we define variables, constants, member functions and some other functionality.
* It binds data and functions together in a single unit.
* It does not consume memory at runtime.
* All classes have a function called \_\_init\_\_(), which is always executed when the class is being initiated.



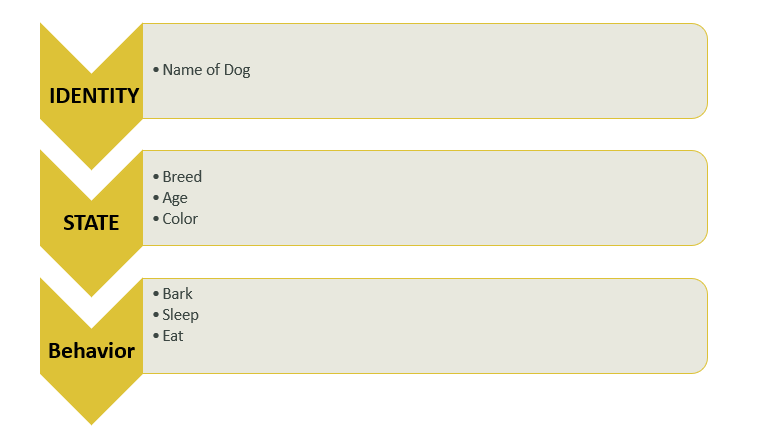
The \_\_init\_\_() function is called automatically every time the class is being used to create a new object.

* **Object:** An object is an instance of a class with unique characteristics and functions.
* By using the class constructor, you may create an object of a class in python.
* It represents a specific implementation of the class and holds its own data.

**An object consists of:**

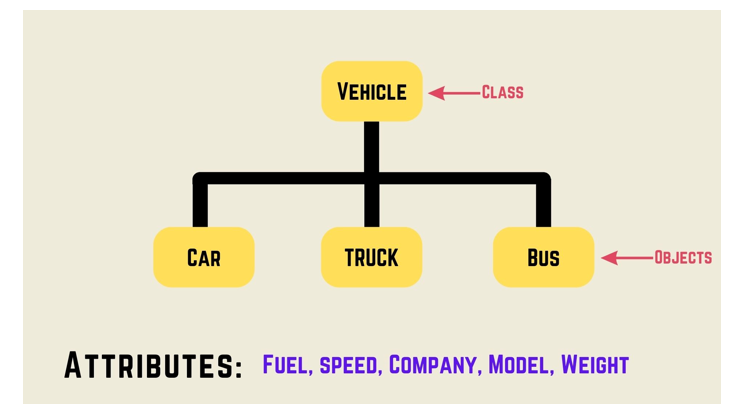
**State:** It is represented by attributes of an object. It also reflects the properties of an object.

**Behaviour:** It is represented by method of an object.

**Identity:** It gives a unique name to an object.

Class​

Objects

* **Attribute:** Attributes are variables that store data for an object. They are defined inside a class and each object can have different attributes values.
* **Method:** Methods are functions inside a class that define behaviours of an object.

**There are two method used:**

**Instance method:** Instance methods operate on individual objects and can access instance attributes. It is an object created from class. Each instance has its own unique data.

**Calling method:** A method is a function inside a class. We call a method using the instance of the class.

**Ex.**

class Person:

    def \_\_init\_\_(self, name):

        self.name= name   #attribute

    def greet(self):  #Method

        print("Good Morning!!")

obj= Person("Mania")   #objct created

obj.greet()

**Output:**

**6.2 Difference between local and global variables.**

**Local variable:** This are the variables that are declared inside of a function. This variables exist only inside the function.

#using local variables

def fun():

    string = "python is awesome!"

    print(string)

fun()

**Global variable:** This are the variables that are created outside of a function are known as global variables. Global variables can be used by everyone, both inside of functions and outside.

#using global variables

string = "python is awesome!"

def fun():

    print(string)

fun()

**Lab:**

* 1. **Write a Python program to create a class and access its properties using an object.**

class A:

    def x(self):

        print("A - x")

    def y(self):

        print("A - y")

class B:

    def p(self):

        print("B - P")

    def q(self):

        print("B - q")

obj = A()

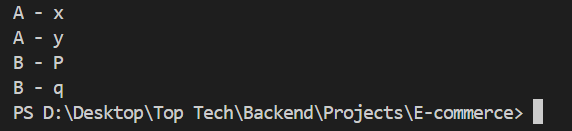
obj.x()

obj.y()

obj\_9= B()

obj\_9.p()

obj\_9.q()

Output:

**Practical Examples:**

* 1. **Write a Python program to create a class and access the properties of the class using an object.**

class A:

    def x(self, num\_1, num\_2):

        return num\_1 + num\_2

class B:

    def p(self):

        print("B - P")

    def q(self):

        print("B - q")

obj = A()

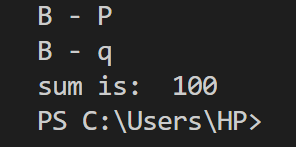
result= obj.x(45, 55)

obj\_9= B()

obj\_9.p()

obj\_9.q()

print("sum is: ", result)

 Output:

**6.5 Write a Python program to demonstrate the use of local and global variables in a class.**

str\_1= "this is global variable"

class x:

    def a(self):

     print(str\_1)

class y:

    def b(self):

        str\_2 = "this is local variable"

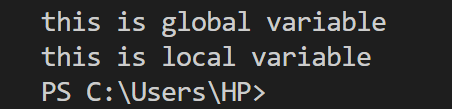
        print(str\_2)

obj = x()

obj.a()

obj\_1= y()

obj\_1.b()

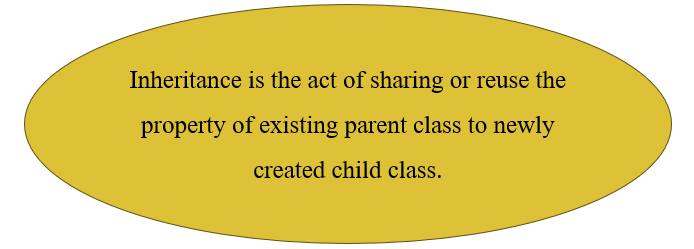
Output:

1. **Inheritance**

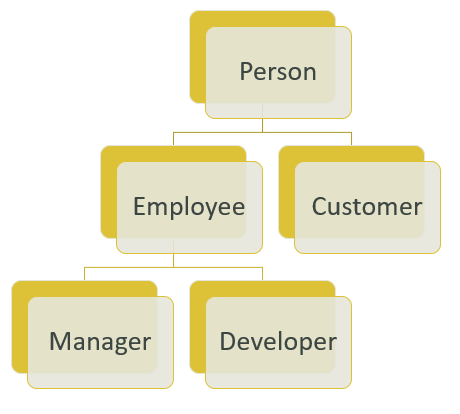
**Theory:**

**7.1 Single, Multilevel, Multiple, Hierarchical, and Hybrid inheritance in Python.**

Inheritance is a fundamental concept in OOPs, that allows us to define a class that inherits all the method and properties from another class

* The concept allows us to inherit the properties of parent class into a child class.
* It provides code reusability.
* When an object acquire the properties and behaviour of the parent object.

**Inheritance example and syntax:**

****class Parent:

    s= "this is parent class"

class child(Parent):

    s\_1= "this is child class"

obj= child()

print(obj.s)

print(obj.s\_1)

**Single Inheritance:** When a child class inherits the properties and behaviour of a single parent class, It is called single inheritance.

Parent Class

│

▼

Child Class

**Multiple Inheritance:** When a child class inherits properties and behaviours from more than one parent class, it is known as multiple inheritance.

Parent Class 1 Parent Class 2

│ │

|

▼

Child Class

**Multilevel Inheritance:** When a class inherits from another class and then a third class inherits from the second class, forming a chain of inheritance is called multilevel inheritance.

Parent Class

│

▼

Intermediate Class

│

▼

Child Class

**Hierarchical Inheritance:** When multiple child classes inherit from a single parent class, it is called hierarchical inheritance.

Parent Class

┌───────┴────────┐

▼ ▼

Child Class 1 Child Class 2

**Hybrid Inheritance:** hybrid inheritance is a combination of two or more types of inheritance like single, multiple, multilevel or hierarchical in single program. It is used when we need complex relationships between classes.

Parent Class

│

Child Class 1 Child Class 2

│ │

Child Class

**7.2 Using the super() function to access properties of the parent class.**

In object- oriented programming, the super() function is used to access the properties and methods of a parent class inside a child class. This is useful when we want to override methods or using same name or functions into different classes, while still retaining the functionality of the parent class.

**Lab:**

**7.3 Write Python programs to demonstrate different types of inheritance (single, multiple, multilevel, etc.).**

***# Single Inheritance***

class A:

    def a(self):

        print("this is class A of a")

    def b(self):

        print("this is class A of b")

class B(A):

    def c(self):

        print("this is class B of c")

    def d(self):

        print("this is class B of d")

obj= B()

obj.a()

obj.b()

obj.c()

obj.d()

***# Multiple Inheritance***

class A:

    def a(self):

        print("This is multiple Inheritance")

    def b(self):

        print("this is class A of a & b")

class B:

    def c(self):

        print("this is class B of c")

    def d(self):

        print("this is class B of d")

class C(B, A):

    def e(self):

        print("this is class C of e")

    def f(self):

        print("this is class C of f")

obj= C()

obj.a()

obj.b()

obj.c()

obj.d()

obj.e()

obj.f()

***# Multilevel Inheritance***

class A:

    def a(self):

        print("This is multiple Inheritance")

    def b(self):

        print("this is class A of a & b")

class B(A):

    def c(self):

        print("this is class B of c")

    def d(self):

        print("this is class B of d")

class C(B):

    def e(self):

        print("this is class C of e")

    def f(self):

        print("this is class C of f")

obj= C()

obj.a()

obj.b()

obj.c()

obj.d()

obj.e()

obj.f()

***# Hierarchical Inheritance***

class A:

    def a(self):

        print("This is multiple Inheritance")

    def b(self):

        print("this is class A of a & b")

class B(A):

    def c(self):

        print("this is class B of c")

    def d(self):

        print("this is class B of d")

class C(A):

    def e(self):

        print("this is class C of e")

    def f(self):

        print("this is class C of f")

obj= B()

obj.a()

obj.b()

obj.c()

obj.d()

obj\_1= C()

obj\_1.e()

obj\_1.f()

***# Hybrid Inheritance***

class A:

    def a(self):

        print("This is multiple Inheritance")

    def b(self):

        print("This is class A of a & b")

class B(A):  # Inheriting from A

    def c(self):

        print("This is class B of c")

    def d(self):

        print("This is class B of d")

class C(A):  # Inheriting from A

    def e(self):

        print("This is class C of e")

    def f(self):

        print("This is class C of f")

# Hybrid Inheritance: Class D inherits from both B and C

class D(B, C):

    def g(self):

        print("This is class D of g")

# Creating object of Class D

obj = D()

obj.a()

obj.b()

obj.c()

obj.d()

obj.e()

obj.f()

obj.g()

**Practical Examples:**

**7.4 Write a Python program to show single inheritance.**

class A:

    def add(self, x, y):

        return x+y

class B(A):

    def sub(self, x, y):

        return x-y

obj = B()

obj.add(50, 50)

obj.sub(45, 5)

**7.5 Write a Python program to show multilevel inheritance.**

class A:

    def fact(self):

        n= int(input("enter number: "))

        fact= 1

        for i in range (1, n+1):

            fact= fact\*i

        print("Factorial is: ", fact)

class B(A):

    def mul(self, x, y):

        return x\*y

class C(B):

    def div(self, x, y):

        return x/y

obj = C()

obj.fact()

x= int(input("X= "))

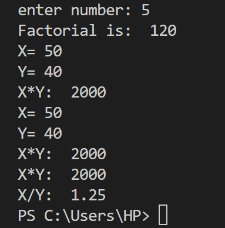
y= int(input("Y= "))

multiply= obj.mul(x, y)

divide= obj.div(x, y)

print("X\*Y: ", multiply)

print("X/Y: ", divide)

Output:

**7.6 Write a Python program to show multiple inheritance.**

class A:

    def prime(self):

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

        prime= 0

        for i in range(1, n+1):

            if(n%i==0):

                prime+=1

        if (prime==2):

         print(n, "is prime\n")

        else:

            print(n, "is not prime\n")

class B:

   def fact(self):

        n= int(input("enter number: "))

        fact= 1

        for i in range (1, n+1):

            fact= fact\*i

        print("Factorial is: ", fact)

class C(A, B):

    def pattern(self):

        print("\n star pattern:")

        for i in range(1, 6):

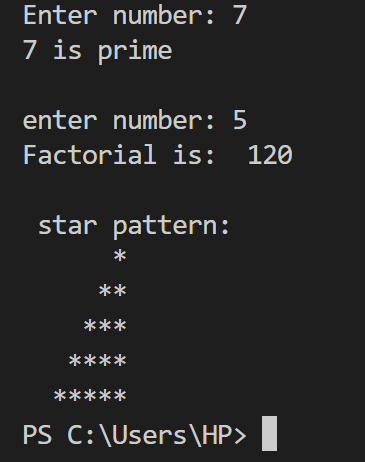
         print(" "\*(6-i), "\*"\*i)

obj = C()

obj.prime()

obj.fact()

obj.pattern()

Output:

**7.7 Write a Python program to show hierarchical inheritance.**

class A:

    def fun(self):

        print("This is fun\_1")

class B(A):

    def fun\_1(self):

        print("This is fun\_2\n")

class C(A):

    def fun\_2(self):

        print("This is fun\_3")

obj= B()

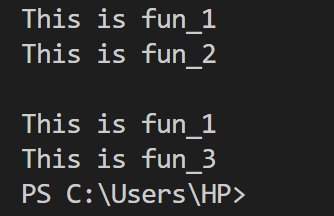
obj.fun()

obj.fun\_1()

obj\_1= C()

obj\_1.fun()

obj\_1.fun\_2()

Output:

**7.8 Write a Python program to show hybrid inheritance.**

class A:

    def P(self):

        print("This is P")

class B(A):

    def Q(self):

        print("This is Q")

class C(B):

    def R(self):

        print("This is R")

class D(B):

    def S(self):

        print("This is S")

obj= C()

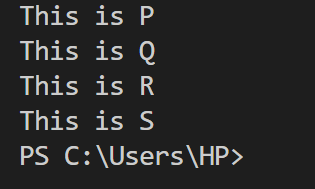
obj.P()

obj.Q()

obj.R()

obj\_1= D()

obj\_1.S()

Output:

**7.9 Write a Python program to demonstrate the use of super() in inheritance.**

class A:

    def P(self):

        print("This is P")

class B(A):  # Inheriting from A

    def Q(self):

        super().P()  # Calling P() from A

        print("This is Q")

class C(A):  # Inheriting from A

    def R(self):

        print("This is R")

class D(B, C):  # Multiple Inheritance

    def S(self):

        super().Q()  # Calls Q() from B, which also calls P() from A

        super().R()  # Calls R() from C, which also calls P() from A

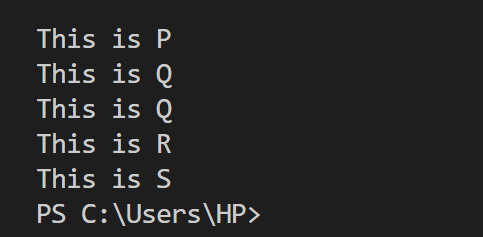
        print("This is S")

# Creating an object of class D

obj = D()

obj.S()  # Calls methods from A, B, and C using super()

Output:



1. **Method Overloading and Overriding**

**Theory:**

* 1. **Method overloading: defining multiple methods with the same name but different parameters.**

Method overloading refers to defining multiple methods with the same name of function with different parameters. In many programming languages like Java and C++, method overloading is directly not supported, it means multiple methods with the same name but different parameter list can coexist in a class.

However, Python doesnot support method overloading.

If multiple methods with the same name are defined in a class, the last defined method overrides the previous ones.

**Ways to Achieve Method Overloading in Python**

Python doesnot support method overloading directly, we can achieve similar behaviour using:

1. Default Parameter
2. Variable length arguments (\*args, \*\*kwargs)
3. Function Dispatching (functools, singledispatch)

**Default Parameter:** By setting default values for parameters, we can allow a method to handle multiple argument cases.

**Using \*args and \*\*kwargs: \***args allows passing a variable number of arguments and \*\*kwargs handles named parameters dynamically.

**Using functools.singledispatch (**Type-Based Overloading**):** Python’s functools.singledispatch enables function overloading based on argument type.

**Keypoints:** If multiple methods with the same name of functions are defined in a class, the last one overrides the previous ones.

* 1. **Method overriding: redefining a parent class method in the child class.**

Method overriding in python is When a child class redefines a method from its parent class with the same name of function and with parameters but with different behaviour. This allows the child class to provide a specific implementation of the method.

**Lab:**

* 1. **Write Python programs to demonstrate method overloading and method overriding.**

# overiding

class Parent:

    def fun(self):

        print("this is parent class")

class Child(Parent):

    def fun(self):

        super().fun()

        print("this is child class")

# overloading

class calc:

    def add(self, a, b= None):

        if b is not None:

            return a+b

        else:

            return a

obj= Child()

obj.fun()

obj\_1= calc()

a= int(input("a: "))

b= input("b (optional): ")

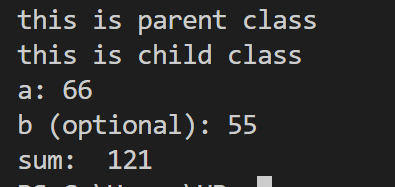
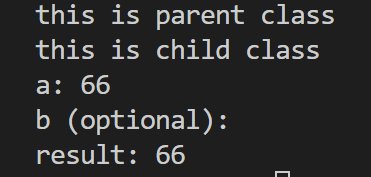
if b:

    print("sum: ", obj\_1.add(a, int(b)))

else:

    print("result:", obj\_1.add(a))

Output:



**Practical Examples:**

* 1. **Write a Python program to show method overloading.**

class Parent:

    def fun(self, a, b=None):

        if b is not None:

            return a+b

        else:

            return a\*a

obj= Parent()

a= int(input("A: "))

b= input("B:(can be skip the value) ")

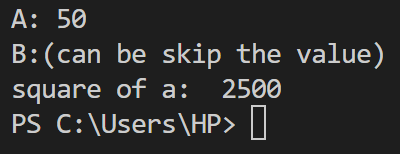
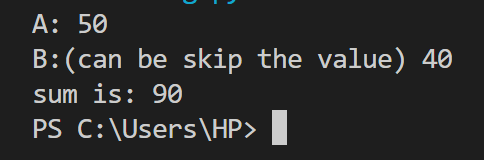
if b:

    print("sum is:", obj.fun(a, int(b)))

else:

    print("square of a: ", obj.fun(a))

Output:

**8.5 Write a Python program to show method overriding.**

class Parent:

    def fun(self):

        print("This is parent class")

class Child(Parent):

    def fun(show):

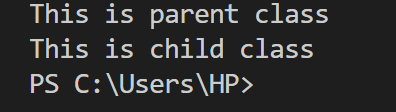
        super().fun()

        print("This is child class")

obj = Child()

obj.fun()

Output:



1. **SQLite3 and PyMySQL (Database Connectors)**

**Theory:**

* 1. **Introduction to SQLite3 and PyMySQL for database connectivity.**

Python provides multiple ways to connect and interact with databases. There are two popular python libraries for database connectivity:

**SQLite3**

**PyMySQL**

**SQLite3:** A light weight, serveless, self-contained database that comes into built-in python library. It deals for small applications that do not require a full database server.

**Advantage of SQLite3:**

* **Built-in Python Library:** No need to install separately.
* Simple and easy to use for small projects.
* **Single File Storage:** Uses a single file (.sqlite or .db) to store data.
* Fast and efficient for local applications.

**PyMySQL:**  PyMySQL is a Python library that allows connection and interaction with a MySQL database. It is used when working with MySQL server, which is more suitable for large-scale applications requiring multi-user access and complex queries.

* **Use SQLite3** if user need a simple, lightweight database without a separate server.
* **PyMySQL** if user need to connect Python to a MySQL server for larger applications.
  1. **Creating and executing SQL queries from Python using these connectors.**

In Python, you can create and execute SQL queries using various database connectors. The most commonly used connectors depend on the database we are working with.

**SQLite3:** This is built into python, so we not need to install it separately.

* No installation required.
* Query:

import sqlite3

connect\_1= sqlite3.connect("database.db")

cursor=connect\_1.cursor()

cursor.execute('''CREATE TABLE IF NOT EXISTS user\_1 (

      ID integer primary key autoincrement,

      NAME text,

      SUBJECT text,

      CLASS text)''')

cursor.execute("INSERT INTO user\_1 (ID, NAME, SUBJECT, CLASS) VALUES(?, ?, ?, ?)", (1, 'AB', "Physics", "v"))

connect\_1.commit()

cursor.execute("select \* from user\_1")

rows= cursor.fetchall()

**PyMySQL:** This is used for MySQL Database Connection.

* Unlike SQLite, MySQL requires installation and a running MySQL server.

**Installation Command:**

**Pip install pymysql**

**Executing query:**

import pymysql

mydb= pymysql.connect(host="localhost", user="root", password="")

mycursor= mydb.cursor()

print("DB connection successful..\n")

mycursor.execute("CREATE DATABASE IF NOT EXISTS pydb\_0")

mydb.commit()

print("DB successfully created..\n")

mydb= pymysql.connect(host="localhost", user="root", password="", database= "pydb\_0")

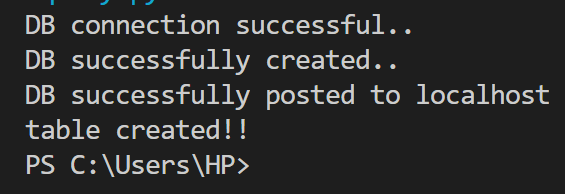
mycursor= mydb.cursor()

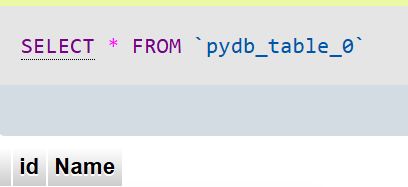
print("DB successfully posted to localhost")

mycursor.execute('''CREATE TABLE IF NOT EXISTS pydb\_table\_0(id INT NOT NULL PRIMARY KEY AUTO\_INCREMENT, Name VARCHAR(40))''')

mydb.commit()

print("table created!!")

**** Output:

****

**Lab:**

**9.3 Write a Python program to connect to an SQLite3 database, create a table, insert data, and fetch data.**

import sqlite3

# using database configuration to connect localhost with python

mydb = sqlite3.connect("sqlite.db")   #database name "sqlite"

mycursor= mydb.cursor()    #calling cursor which is in-built method to post data into sql

print("connection successful")  # created a cursor object to execute queries

# Create table if it doesn't exist

#"sqlite\_table" is table name

mycursor.execute('''CREATE TABLE IF NOT EXISTS user

                (

                    id INTEGER PRIMARY KEY AUTOINCREMENT,

                    Name TEXT,

                    Age INTEGER

                )'''

)

print("database created successfully")

#Insert data into table

Data= [("x", 44), ("y", 45), ("z", 46),]

mycursor.executemany("INSERT INTO user(Name, Age) VALUES(?, ?)", Data)

print("created database successfully posted to local host")

# commit changes and close the connection

mydb.commit()

print("create table successfully")

# to fetch all the data

mycursor.execute("SELECT \* FROM user")

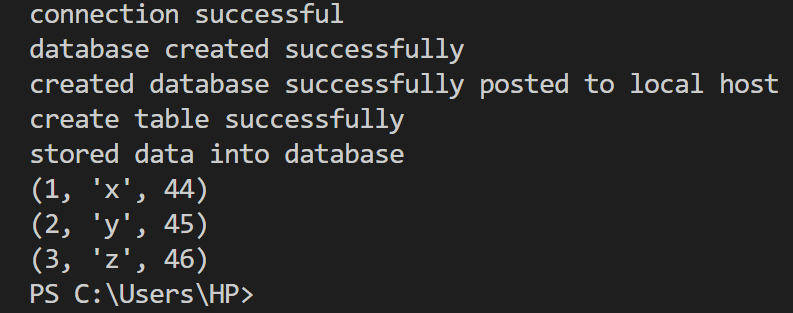
rows= mycursor.fetchall()

print("stored data into database")

for row in rows:

    print(row)

mydb.close()

Output:

**Practical Examples:**

* 1. **Write a Python program to create a database and a table using SQLite3.**

import sqlite3

mydb= sqlite3.connect("sqdb\_1.db")

mycursor= mydb.cursor()

mycursor.execute('''CREATE TABLE IF NOT EXISTS sq\_user\_00(

                     id INTEGER PRIMARY KEY AUTOINCREMENT,

                     Employee\_Name TEXT,

                     Salary INT

                   )'''

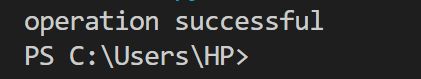
                )

mydb.commit()

print("operation successful")

mydb.close()

Output:



**9.5 Write a Python program to insert data into an SQLite3 database and fetch it.**

import sqlite3

mydb= sqlite3.connect("sqdb.db")

mycursor= mydb.cursor()

mycursor.execute('''CREATE TABLE IF NOT EXISTS sq\_user (

                     Roll\_No INTEGER PRIMARY KEY AUTOINCREMENT,

                     Name TEXT,

                     Course TEXT

                   )'''

                )

mycursor.execute("INSERT INTO sq\_user (Name, Course) VALUES(?, ?)", ('XY', 'Backend Development'))

mydb.commit()

mycursor.execute("select \* from sq\_user")

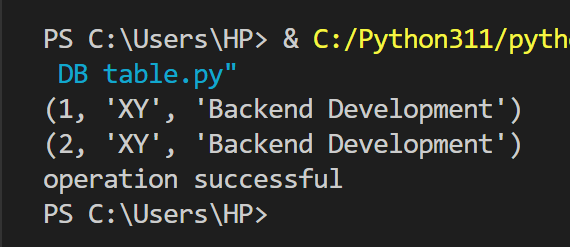
rows= mycursor.fetchall()

for row in rows:

    print(row)

print("operation successful")

mydb.close()

Output:

1. **Search and Match Functions**

**Theory:**

**10.1 Using re.search() and re.match() functions in Python’s re module for pattern matching.**

Python’s **re**  module provides powerful tools for working with regular expressions. Two commonly used functions for pattern matching are:

**re.match** (pattern, string)

**re.search** (pattern, string)

**re.match():** The re.match function checks only at the beginning of the string for a match. If the pattern is found at the start, it returns a match object, otherwise it returns **None.**

**re.match(patttern, string, flag=0)**

**Pattern:** The regex pattern to match.

**String:** The string to be checked.

**Flags:** Optional flags like **re.IGNORECASE** for case-insensitive matching.

**Re.search():** The re.search() function searches the entire string for a match and returns the first occurrence. Unlike re.match(), the match does not have to be at the start of the string.

**re.search(pattern, string, flags=0)**

**10.2 Difference between search and match.**

|  |  |  |
| --- | --- | --- |
| **Feature** | **re.match()** | **re.search()** |
| |  |  |  | | --- | --- | --- | | **Search Scope** |  |  | | Only at the start of the string | Searches anywhere in the string |
| **Returns** | Match object if found at the beginning | Match object if found anywhere |
| **Best Used For** | Checking if a string starts with a pattern | Finding patterns anywhere in a string |

**Lab:**

**10.3 Write a Python program to search for a word in a string using re.search().**

import re

pattern = r"Hello"

string= "Hello world"

string= input("Enter a string: ")

word= input("enter the word to search: ")

pattern= rf"\b{word}\b"

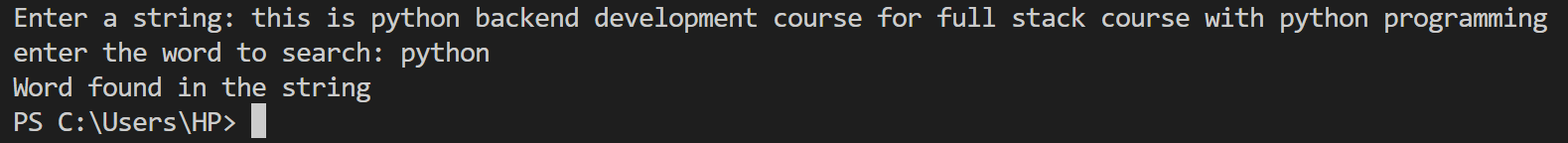
search= re.search(pattern, word, re.IGNORECASE)  #checks for a match at the beginning of the string

if search:

    print("Word found in the string")

else:

    print("word not found.")

Output:

**10.4 Write a Python program to match a word in a string using re.match().**

import re

string= input("Enter string: ")

pattern = r"Hello"

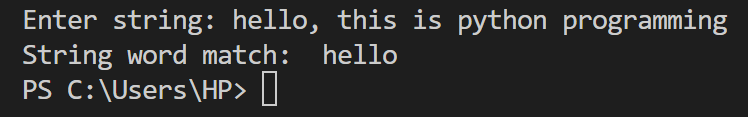
match= re.match(pattern, string, re.IGNORECASE)  #checks for a match at the beginning of the string

if match:

    print("String word match: ", match.group())

else:

    print("Not matching any of string words")

Output:

**Practical Examples:**

10.5 Write a Python program to search for a word in a string using re.search().

import re

string= input("Enter a string: ")

word= input("enter the word to search: ")

pattern= rf"\b{word}\b"

search= re.search(pattern, word, re.IGNORECASE)  #checks for a match at the beginning of the string

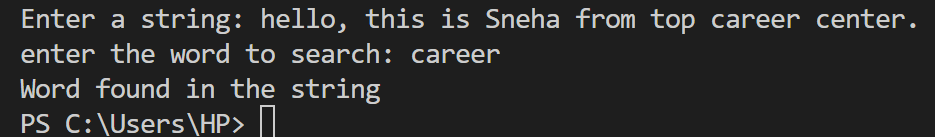
if search:

    print("Word found in the string")

else:

    print("word not found.")

Output:



10.6 Write a Python program to match a word in a string using re.match().

import re

string= input("Enter string: ")

pattern = input("Enter word to match: ")

match= re.match(pattern, string, re.IGNORECASE)  #checks for a match at the beginning of the string

if match:

    print("String word match: ", match.group())

else:

    print("Not matching any of string words")

Output:

