**1. Write a program to demonstrate the following:**

(a)Different numeric data types

(b)To perform different arithmetic operations

**Aim :** The aim of this program is find the different numeric data types in python and also to perform different arithmetic operations.

**Description:** Python supports several numeric data types, including integers, floating-point numbers, and complex numbers. This program demonstrates the different numeric data types and performs various arithmetic operations.

**Numeric Data types:**

**Integers**

Integers are whole numbers, positive or negative, without decimals, of unlimited length.

Example: 5, -10, 150

**Floating-Point Numbers**

Floating-point numbers are real numbers with a floating point representation. They are specified with a decimal point.

Example: 5.0, -10.5, 150.0

**Complex Numbers**

Complex numbers are numbers with a real and imaginary component represented as a + bj, where a is the real part and b is the imaginary part.

Example: 3 + 4j, -5 + 2j

**Arithmetic Operations**

1. Addition :

Example : 5 + 6

(2)Subtraction : Subtracts one number from another.

Example : 10 - 3

(3)Multiplication : Multiplies two numbers together.

Example : 3 \* 2

(4) Division : Divides one number by another.

Example : 10 / 5

(5)Modulus : Returns the remainder of a division operation.

Example : 5 % 6

**Program Implementation:**

#Program to print different numeric data typesin python and to perform arithmetic operations

a = 10

b = 15.5

c = 5 + 4j

print("\*\* Different Numeric data types are \*\*")

print(a,"is Type of",type(a))

print(b,"is Type of",type(b))

print(c,"is Type of",type(c))

print("\n\*\* Arithmetic Operations \*\*")

num1 = int(input("Enter num1 value : "))

num2 = int(input("Enter num2 value : "))

print("Addition : num1 + num2 = ",num1 + num2)

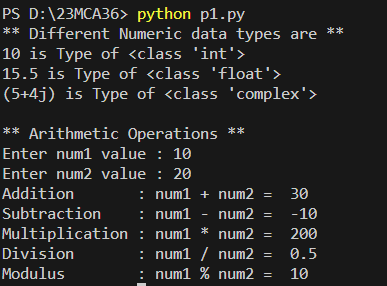
print("Subtraction : num1 - num2 = ",num1 - num2)

print("Multiplication : num1 \* num2 = ",num1 \* num2)

print("Division : num1 / num2 = ",num1 / num2)

print("Modulus : num1 % num2 = ",num1 % num2)

**Result:**

****

**2. Write a program to find a factorial of a number using recursion**

**Aim:** The aim of these program is to find factorial of number using recursion function

**Description:** The factorial of a number is the product of all positive integers up to that number.

For example, the factorial of 5 is 1\*2\*3\*4\*5 = 120.

**Recursion** is a programming technique where a function calls itself to solve a problem by breaking it down into smaller sub-problems. In this program, we use recursion to find the factorial of a number.

In this program, we used recursion to find the factorial of a number. The factorial function takes an integer n as input and returns its factorial. The base case is when n is 0, in which case the function returns 1. Otherwise, the function calls itself with the argument n-1 and multiplies the result by n.

**Program Implementation:**

def recursion\_fact(n):

if n == 1:

return n

else:

return n \* recursion\_fact(n-1)

num = int(input("Enter a num value : "))

if(num < 0):

print(f"Factorial of a given number {num} is not possible")

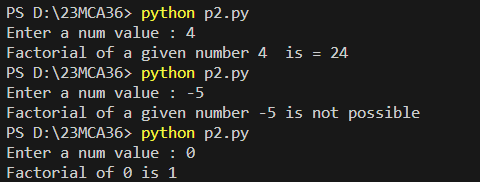
elif(num == 0):

print("Factorial of 0 is 1")

else:

print(f"Factorial of a given number {num} is = {recursion\_fact(num)} ")

**Result:**



**3. Write a python program to check whether a given number is Armstrong number or not**

**Aim:** The aim of this program is to find whether a given input number id Armstrong number or not

**Description:** An Armstrong number in a given number base b is a number that is the sum of its own digits each raised to the power of the number of digits. To put it simply, if I have a 3-digit number then each of the digits is raised to the power of three and added to obtain a number. If the number obtained equals the original number then, we call that Armstrong number. The unique property related to the Armstrong numbers is that it can belong to any base in the number system.

For example, in the decimal number system, 153 is an Armstrong number.

1^3+5^3+3^3=153

**Code Implementation:**

num = int(input("Enter the num value : "))

sum = 0

temp = num

while temp > 0:

digit = temp % 10

sum += digit \*\* 3

temp //= 10

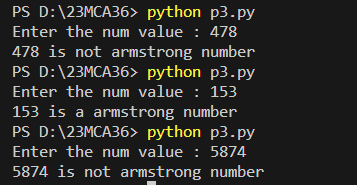
if num == sum:

print(num,"is a armstrong number")

else:

print(num,"is not armstrong number")

**Output:**

****

**4. Write a program to create classes and objects**

**Aim:** The aim of this program is to create class and accessing methods and variables by creating class objects.

**Description:** Python is an object oriented programming language almost everything in Python is an object, with its properties and methods.

A Class is like an object constructor, or a "blueprint" for creating objects. To create a class we class keyword

E.g.: class Student:

Roll no = 11

An object is a real time entity.

E.g. s1 = Student ()

In this program we going to create a Student class and to print his/her name, roll no, 5 subject marks and to calculate total and average and display the result

**Code Implementation:**

class Student:

def s\_info(self):

self.name = input("Enter a Student name : ")

self.reg\_no = int(input("Enter reg\_no : "))

self.sem = int(input("Enter semester : "))

self.section = input("Enter a section : ")

def cal(self):

self.m1 = int(input("Enter maths marks : "))

self.m2 = int(input("Enter stat marks : "))

self.m3 = int(input("Enter java marks : "))

self.m4 = int(input("Enter python marks : "))

self.total = self.m1+self.m2+self.m3+self.m4

self.res = self.total/4

#displaying student information

def display(self):

self.s\_info()

self.cal()

print("\*\* STUDENT INFORMATION \*\*")

print("Name = ",self.name)

print("Reg\_No = ",self.reg\_no)

print("Semester = ",self.sem)

print("Section = ",self.section)

print(" Marks Scored in each subject :")

print("Maths = ",self.m1)

print("Stat = ",self.m2)

print("Java = ",self.m3)

print("Python = ",self.m4)

print("Total marks = ",self.total)

print("Percentage = ",self.res)

if self.res >= 85:

print("Grade : A")

elif self.res >= 75:

print("Grade :B")

elif self.res >= 65:

print("Grade :C")

elif self.res >= 55:

print("Grade :D")

elif self.res >= 45:

print("Grade :E")

else:

print("Grade :Failed")

#creating objects

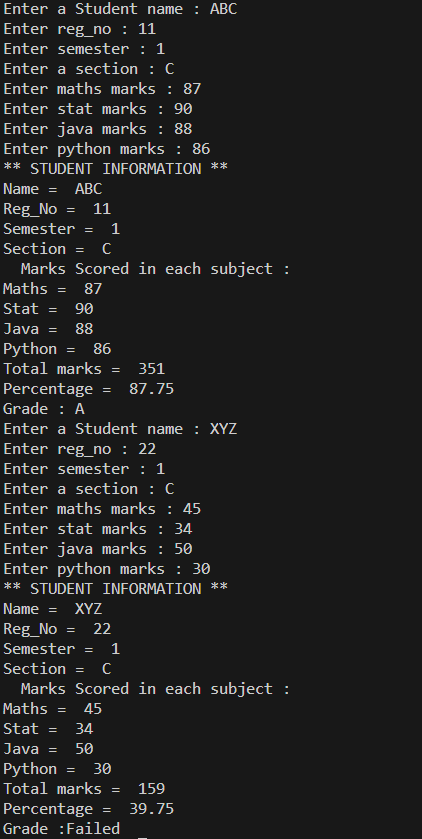
s1 = Student()

s2 = Student()

s1.display()

s2.display()

**Result:**

****

**5. Write a program to reverse a string word by word**

**Aim:** The aim of this program is to reverse a given string word by word

**Description:**

**Code:**

class StringReverser:

def reverse\_words(self,input\_string):

words = input\_string.split()

reversed\_words = words[::-1]

reversed\_string = "".join(reversed\_words)

return reversed\_string

if \_\_name\_\_ == "\_\_main\_\_":

input\_string = "Hello World"

reverser = StringReverser()

reversed\_result = reverser.reverse\_words(input\_string)

print("Original String : ",input\_string)

print("Reversed String : ",reversed\_result)

**Output:**

**6.Write a python program to create append and remove elements from lists.**

**Aim:** To create a python program and perform list operations like append and remove elements from lists.

**Description:** In Python, lists are foundational built-in data structures that facilitate the storage and manipulation of collections of items. They are immensely versatile and serve as one of the most commonly used data structures in Python. Lists find applications across a wide spectrum of tasks, including data storage, data collection, implementing various data structures, and data processing.

To interact with lists in Python, we typically prompt the user to specify the desired action, whether it's creating a new list, appending elements to an existing list, or removing elements from it. Based on the user's input, we proceed accordingly. For instance, we can create a new list using the syntax list\_name = [], add elements to the list using the append () method, remove elements from the list using the remove () method, and display the updated list using print ().

**Code:**

def create\_list():

global my\_list

my\_list = []

print("New empty list created.")

#return my\_list

def append\_to\_list(element):

my\_list.append(element)

print(f'Appended {element} to the list.')

def remove\_from\_list(element):

try:

my\_list.remove(element)

print(f'Removed {element} from the list')

except ValueError:

print(f'{element} not found in list')

def display\_list():

print("Current List",my\_list)

while True:

print("\nChoose an action")

print("1.Create a new list")

print("2.Append an element to the list")

print("3.Remove an element from the list")

print("4.Display the current list")

print("5.Exit")

choice = input("Enter your choice(1/2/3/4/5) :")

if choice == '1':

create\_list()

elif choice =='2':

element = input("Eneter an element to append : ")

append\_to\_list(element)

elif choice == '3':

element = input("Enter an element to remove :")

remove\_from\_list(element)

elif choice == '4':

display\_list()

elif choice == '5':

break

else:

print("Invalid choice. Please try again")

**Output:**

PS D:\23MCA36> python p6.py

Choose an action

1.Create a new list

2.Append an element to the list

3.Remove an element from the list

4.Display the current list

5.Exit

Enter your choice(1/2/3/4/5) :1

New empty list created.

Choose an action

1.Create a new list

2.Append an element to the list

3.Remove an element from the list

4.Display the current list

5.Exit

Enter your choice(1/2/3/4/5) :4

Current List []

Choose an action

1.Create a new list

2.Append an element to the list

3.Remove an element from the list

4.Display the current list

5.Exit

Enter your choice(1/2/3/4/5) :2

Eneter an element to append : 10

Appended 10 to the list.

Choose an action

1.Create a new list

2.Append an element to the list

3.Remove an element from the list

4.Display the current list

5.Exit

Enter your choice(1/2/3/4/5) :2

Eneter an element to append : 20

Appended 20 to the list.

Choose an action

1.Create a new list

2.Append an element to the list

3.Remove an element from the list

4.Display the current list

5.Exit

Enter your choice(1/2/3/4/5) :2

Eneter an element to append : 30

Appended 30 to the list.

Choose an action

1.Create a new list

2.Append an element to the list

3.Remove an element from the list

4.Display the current list

5.Exit

Enter your choice(1/2/3/4/5) :2

Eneter an element to append : 40

Appended 40 to the list.

Choose an action

1.Create a new list

2.Append an element to the list

3.Remove an element from the list

4.Display the current list

5.Exit

Enter your choice(1/2/3/4/5) :4

Current List ['10', '20', '30', '40']

Choose an action

1.Create a new list

2.Append an element to the list

3.Remove an element from the list

4.Display the current list

5.Exit

Enter your choice(1/2/3/4/5) :3

Enter an element to remove :20

Removed 20 from the list

Choose an action

1.Create a new list

2.Append an element to the list

3.Remove an element from the list

4.Display the current list

5.Exit

Enter your choice(1/2/3/4/5) :4

Current List ['10', '30', '40']

Choose an action

1.Create a new list

2.Append an element to the list

3.Remove an element from the list

4.Display the current list

5.Exit

Enter your choice(1/2/3/4/5) :5

**7. Write a python program to demonstrate the working of tuple.**

**Aim:** To generate a python program to demonstrate the working of tuples.

**Description:** In Python, a 'tuple' is an ordered immutable collection of elements, distinct from lists, which are mutable and use square brackets '[]'. Tuples are denoted by parentheses '()'. They are particularly useful when we need to ensure that the data remains unchanged and is treated as a single unit.

To work with tuples in Python, we typically determine the action the user wants to perform, such as adding elements to the tuple, accessing elements within the tuple, removing elements from the tuple, or displaying the tuple. Depending on the user's choice, we carry out the corresponding operation.

**Code:**

def create\_tuple():

my\_tuple=()

print("Empty tuple created.")

return my\_tuple

def add\_to\_tuple(my\_tuple,element):

my\_tuple+=(element,)

print(f"Added '{element}' to the tuple.")

return my\_tuple

def access\_tuple(my\_tuple,index):

if 0<=index<len(my\_tuple):

element=my\_tuple[index]

print(f"Element at index {index}:{element}")

else:

print("Invalid index.")

def remove\_from\_tuple(my\_tuple,element):

if element in my\_tuple:

my\_list=list(my\_tuple)

my\_list.remove(element)

my\_tuple=tuple(my\_list)

print(f"Removed '{element}' from the tuple.")

return my\_tuple

else:

print(f"'{element}' not found in the tuple.")

def display\_tuple(my\_tuple):

print("Current Tuple:",my\_tuple)

my\_tuple=create\_tuple()

while True:

print("\n Choose an action:")

print("1.Add an element to the tuple")

print("2.Access an element in the tuple")

print("3.Remove an element from the tuple")

print("4.Display the current tuple")

print("5.Exit")

choice=input("enter your choice (1/2/3/4/5):")

if choice=='1':

element=input("Enter a element to add:")

my\_tuple=add\_to\_tuple(my\_tuple,element)

elif choice=='2':

index=int(input("Enter the index to access:"))

access\_tuple(my\_tuple,index)

elif choice=='3':

element=input("Enter the element to be remove:")

my\_tuple=remove\_from\_tuple(my\_tuple,element)

elif choice=='4':

display\_tuple(my\_tuple)

elif choice=='5':

break

else:

print("Invalid choice.Please try again:")

**Output:**

PS D:\23MCA36> python p7.py

Empty tuple is created

Choose an action :

1.Add an element to tuple :

2.Access an element in the tuple :

3.Remove an element from the tuple :

4.Display the current tuple

5.Exit

Enter you choice (1/2/3/4/5) : 4

Cuurent tuple : ()

Choose an action :

1.Add an element to tuple :

2.Access an element in the tuple :

3.Remove an element from the tuple :

4.Display the current tuple

5.Exit

Enter you choice (1/2/3/4/5) : 1

Enter an elements to add : 10

Added 10 to the tuple

Choose an action :

1.Add an element to tuple :

2.Access an element in the tuple :

3.Remove an element from the tuple :

4.Display the current tuple

5.Exit

Enter you choice (1/2/3/4/5) : 1

Enter an elements to add : 20

Added 20 to the tuple

Choose an action :

1.Add an element to tuple :

2.Access an element in the tuple :

3.Remove an element from the tuple :

4.Display the current tuple

5.Exit

Enter you choice (1/2/3/4/5) : 1

Enter an elements to add : 30

Added 30 to the tuple

Choose an action :

1.Add an element to tuple :

2.Access an element in the tuple :

3.Remove an element from the tuple :

4.Display the current tuple

5.Exit

Enter you choice (1/2/3/4/5) : 4

Cuurent tuple : ('10', '20', '30')

Choose an action :

1.Add an element to tuple :

2.Access an element in the tuple :

3.Remove an element from the tuple :

4.Display the current tuple

5.Exit

Enter you choice (1/2/3/4/5) : 2

Enter the index to access : 0

Element at index 0 : 10

Choose an action :

1.Add an element to tuple :

2.Access an element in the tuple :

3.Remove an element from the tuple :

4.Display the current tuple

5.Exit

Enter you choice (1/2/3/4/5) : 3

Enter an element to remove : 20

Removed 20 from tuple.

Choose an action :

1.Add an element to tuple :

2.Access an element in the tuple :

3.Remove an element from the tuple :

4.Display the current tuple

5.Exit

Enter you choice (1/2/3/4/5) : 4

Cuurent tuple : ('10', '30')

Choose an action :

1.Add an element to tuple :

2.Access an element in the tuple :

3.Remove an element from the tuple :

4.Display the current tuple

5.Exit

Enter you choice (1/2/3/4/5) : 5

**8. Program to create classes and objects in python.**

**Aim:** To create a python program to demonstrate the concepts of classes and objects.

**Description:** Python's classes and objects are foundational to object-oriented programming (OOP) and play a crucial role in creating structured and scalable code. They allow for the creation of custom datatypes and their associated behaviors, facilitating code organization and management in large projects.

Classes serve as blueprints for creating objects, each representing an instance of a specific datatype. Objects encapsulate both data (attributes) and behaviors (methods) related to that datatype.

Here's a rephrased description of using classes and objects in Python, with an example of a Dog class:

Python's classes and objects enable the creation of sophisticated and organized code structures. These concepts, fundamental to Object-Oriented Programming (OOP), empower developers to design reusable and structured code components, essential for managing complexity in large-scale projects.

Classes define custom datatypes, while objects represent instances of those datatypes, each possessing its distinct state and behavior. To create a class, the class keyword followed by the class name is used, while objects are instantiated by invoking the class constructor.

For instance, let's consider a Dog class. We initialize instances of this class with attributes such as name and breed, and define a method named bark to simulate a dog's barking behavior. We then create two objects (Dog1 and Dog2) of the Dog class, each assigned specific names and breeds.

Finally, we access and print the attributes (name and breed) of each dog instance and invoke the bark method for each, causing them to bark.

**Code:**

class Dog:

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

self.name=name

self.breed=breed

def bark(self):

print(f"{self.name}is barking!")

dog1=Dog("Buddy","Golden Retiever")

dog2=Dog("Max","German Shepherd")

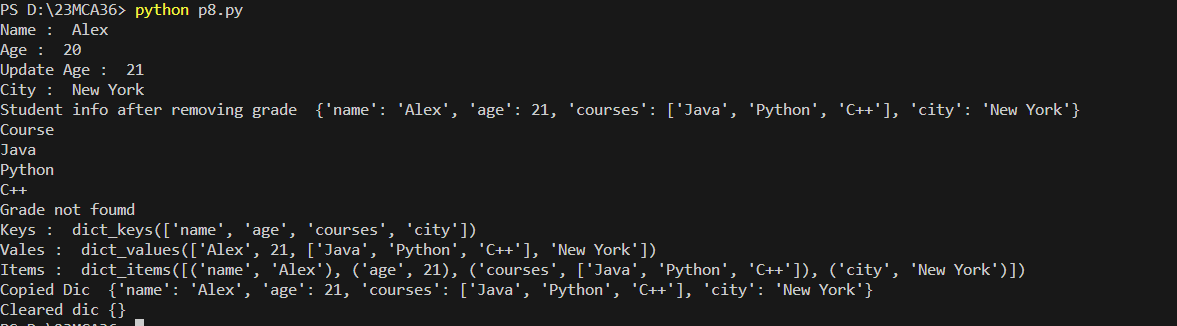
print(f"{dog1.name} is a {dog1.breed}")

print(f"{dog2.name} is a {dog2.breed}")

dog1.bark()

dog2.bark()

Output:



**9. Write a python program to implement functions.**

**Aim:** To create a python program to implement functions.

**Description:** In python, the functions are created using the def keyword. The functions can accept () and more parameter/arguments within parentheses. And functions can return values using the return statement. To execute functions we call it by its name followed by parentheses.

In this program we have demonstrated working of functions in python by creating the following functions:

Calculate \_circle\_area ()

Greet

Is\_even\_or\_odd ()

Whose values are given as arguments during runtime. These functions calculate the area of a rectangle by area=length\*width and area for circle by area of a rectangle by area=pi\*radius^2 and greets the user by printf function and determines weather the number is odd or even using the if-else block statement respectively, and all the resultant values are pointed.

**Code:**

def calcualte\_rectangle\_area(length,width):

area=length\*width

return area

def calculate\_circle\_area(radius):

import math

area=math.pi\*radius\*\*2

return area

def greet(name):

print(f"Hello,{name}!")

def is\_even\_or\_odd(number):

if number%2==0:

return "Even"

else:

return "Odd"

if \_\_name\_\_=="\_\_main\_\_":

length=5

width=3

rectangle\_area=calcualte\_rectangle\_area(length,width)

print(f"Rectangle Area:{rectangle\_area}")

radius=4

circle\_area=calculate\_circle\_area(radius)

print(f"Circle Area:{circle\_area}")

name="Your Name"

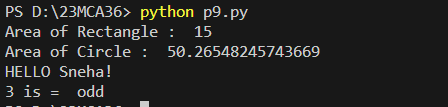
greet(name)

number=7

result=is\_even\_or\_odd(number)

print(f" {number} is {result}")

**Output:**

****