**Code (Experiment1) :**

1. Data Types in Python :

a = "hello world"

print('\nStrings implemenation : ',a,type(a))

b = 2

print('\ninteger implemenation : ',b,type(b))

c = 29.87

print('\nfloat implemenation : ',c,type(c))

com = 15+98j

print('\nComplex numbers implemenation : ',com,type(com))

l = [1,'one',1.00,1+2j]

print('\nlist implemenation : ',l,type(l))

ran = range(10)

print('\nRange implemenation : ',ran,type(ran))

tup = ('abc',12,0.92)

print('\ntuple implemenation : ',tup,type(tup))

dic = {1:'hi',2:'you'}

print('\nDictionary implemenation : ')

print(dic,type(dic))

print('Dictionary values : ',dic.values())

print('Dictionary items : ',dic.items())

print('Dictionary keys : ',dic.keys())

s = {'hello','world','python'}

print('\nSet Implementation : ',s,type(s))

by = b'hi'

print('\nByte implemenation : ',by,type(by))

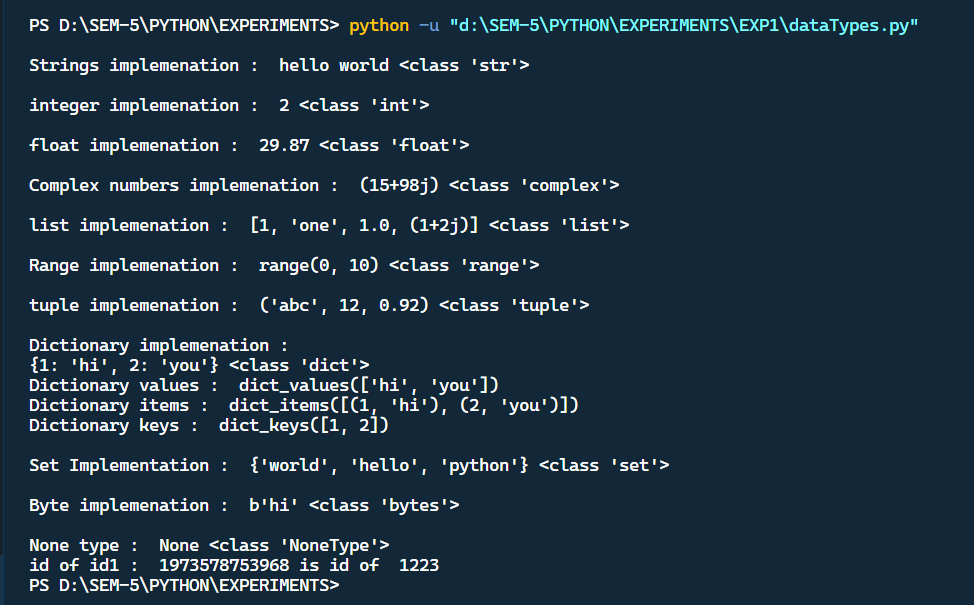
n = None

print('\nNone type : ',n,type(n))

id1 = 1223

print('id of id1 : ',id(id1),'is id of ',id1)

**Output for above code :**



1. Operators in Python :

a = 18

b = 3

print("\nArithmetic Operators :")

print(f'Addition of {a} and {b} is ',a+b)

print(f'subtraction of {a} and {b} is ',a-b)

print(f'Modulo of {a} and {b} is ',a%b)

print(f'Multiplication of {a} and {b} is ',a\*b)

print(f'Division of {a} and {b} is ',a/b)

print(f'Remainder of {a} by {b} is ',a//b)

print(f'{a} raise to {b} is ',a\*\*b)

c = 14

print("\nAssignment Operators :")

print("Inital value of c :")

print(c)

c+=6

print("Adding 6 to it gives ",c)

c-=6

print("subtracting 6 to it gives ",c)

c\*=6

print("multiplying 6 to it gives ",c)

c/=6

print("divding by 6 to it gives ",c)

c%=6

print("taking modulo gives ",c)

c//=6

print("c raised to 6 gives ",c)

c\*\*=6

d = 15

e = 10

print('\nComparison Operators :')

print(f"is {d} equal to {e}",d==e)

print(f"is {d} not equal to {e}",d!=e)

print(f"is {d} greater than or equal to {e}",d>=e)

print(f"is {d} smaller than or equal to {e}",d<=e)

print(f"is {d} greater than {e}",d>e)

print(f"is {d} smaller than {e}",d<e)

f = 10

g = 19

print('\nLogical Operators :')

print(f"if {f}>=10 and {g} <20 then : ", f>=10 and g<20)

print(f"if {f}>=10 or {g} >20 then : ", f>=10 or g>20)

h = 9

i = 19

print('\nidentity Operators :')

print(f"h is i ", h is i)

print(f"h is not i ", h is not i)

x = 'java'

y = 'javascript'

print("\nMembership Operators : ")

print(f'is {x} in {y} : ',x in y)

print(f'is {x} not in {y} : ',x not in y)

a = 5

b = 7

print("\nBitwise Operators : ")

print(f"{a} and {b} is : ",a&b)

print(f"{a} or {b} is : ",a|b)

print(f"{a} ^ {b} is : ",a^b)

print(f"{a} << 3 is : ",a<<3)

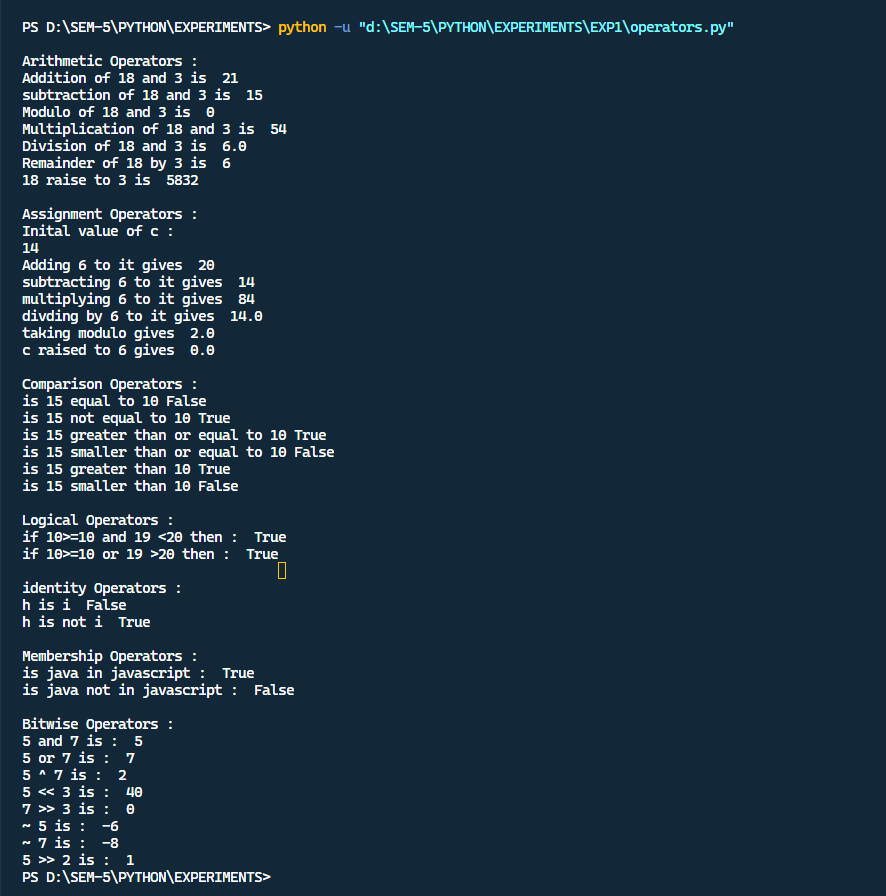
print(f"{b} >> 3 is : ",b>>3)

print(f"~ {a} is : ",~a)

print(f"~ {b} is : ",~b)

print(f"{a} >> 2 is : ",b>>2)

**Output for above code :**



**Code (Experiment 2) :**

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

print('square root of ', a, a\*\*0.5)

l, b = float(input("Enter the length : ")), float(input("Enter the breadth : "))

print("Area : ", l\*b, " ,", "Perimeter : ", 2 \* (l + b))

a,b = int(input("Enter 1st numbers : ")), int(input("Enter 2nd numbers : "))

c = a

a = b

b = c

print("Swapping with third variable : ",a,b)

a,b = int(input("Enter 1st numbers : ")), int(input("Enter 2nd numbers : "))

b,a = a,b

print("swapping without third variable : ", a,b)

*# another way*

a,b = int(input("Enter 1st numbers : ")), int(input("Enter 2nd numbers : "))

print("a :",a," b :",b)

a = a+b

b = a-b

a = a-b

print("After Swapping ")

print("a:",a,"b:",b)

*# adding elements to list , set , tuples*

n = int(input("Enter the size of list : "))

l = list()

for i in range(n):

    l.append(int(input(f"Enter the {i} element : ")))

print("list :",l)

n = int(input("Enter the size of list : "))

s = set()

for i in range(n):

    s.add(int(input(f"Enter the {i} element : ")))

print("set :", s)

n = int(input("Enter size "))

t = tuple()

l = list(t)

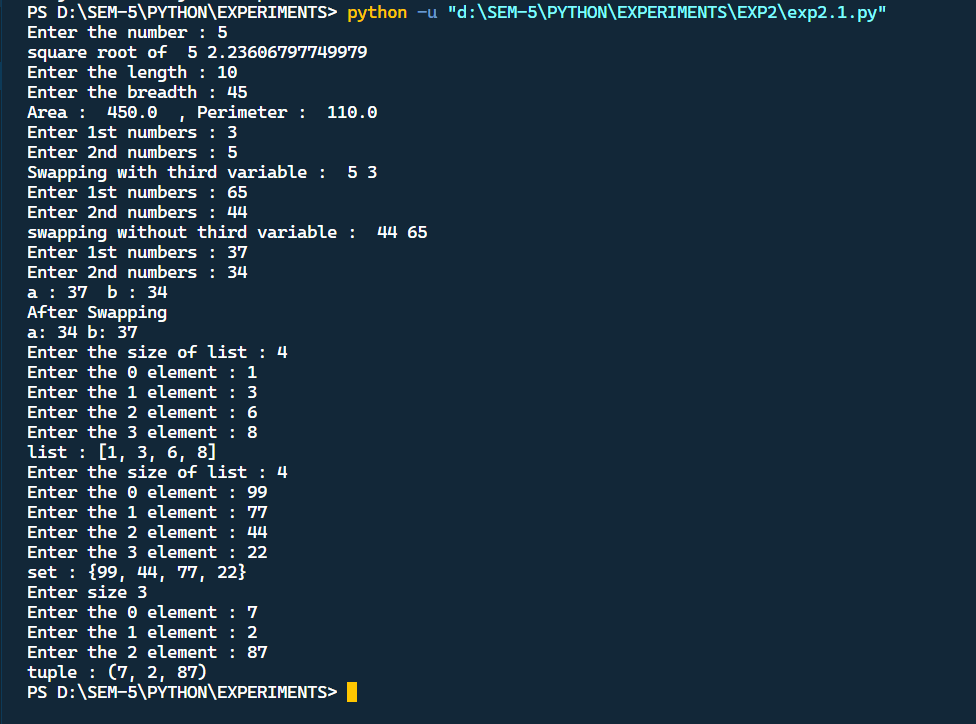
for i in range(n):

    l.append(int(input(f"Enter the {i} element : ")))

t = tuple(l)

print("tuple :", t)

**Output :**



**Code :**

n = int(input("Enter the number of rows : " ))

for i in range(1,n+1):

    for j in range(1,i+1):

        print(i, end=" ")

    print()

*# factorial*

print("\nFactorial example : ")

n = int(input("Enter the number for factorial : "))

fact = 1

if n<0 : print("factorial does not exist !!")

elif n==0: print("Factorial of 0 is 1 ")

else :

    for i in range(1,n+1):

        fact \*= i

print(f"Factorial of {n} with for loop : {fact}")

n = int(input("Enter the number for factorial : "))

fact = 1

i=1

if n<0 : print("factorial does not exist !!")

elif n==0: print("Factorial of 0 is 1 ")

else :

    while i <=n:

        fact \*=i

        i+=1

print(f"Factorial of {n} with while loop : {fact}")

*#Even odd with if else*

print("\nEven odd with if else example : ")

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

if (n%2)==0: print(f"{n} is even")

else: print(f"{n} is odd")

*# checking the number*

print("\nchecking the number with if elif else example : ")

number = 2+9j

if isinstance(number, int):

    print("Number is an integer.")

elif isinstance(number, list):

    print("Number is a list.")

elif isinstance(number, tuple):

    print("Number is a tuple.")

elif isinstance(number, set):

    print("Number is a set.")

elif isinstance(number, float):

    print("Number is a float data type.")

elif isinstance(number, str):

    print("Number is a string.")

else:

print("Number is complex.")

*# break continue pass*

print("\nbreak continue pass example : ")

for letter in 'python':

    if letter == 't' or letter == '0':

        continue

    print('current letter:',letter)

for letter in 'python':

    if letter == 'y':

        break

    print('current letter:',letter)

for letter in 'python':

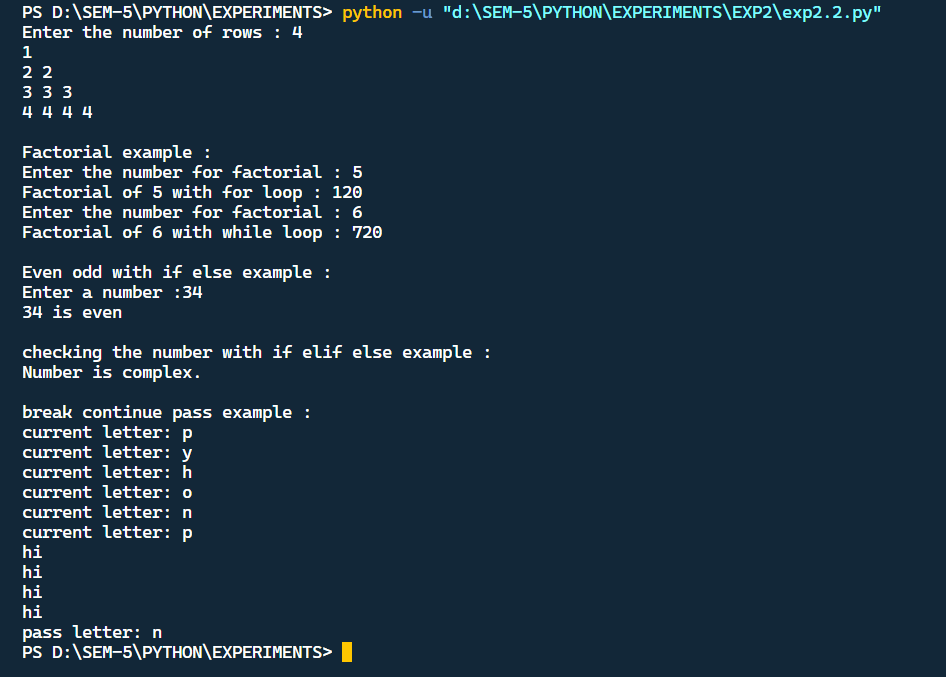
    if letter in ['p', 'y', 't', 'h']:

        print('hi')

        pass

print('pass letter:',letter)

**Output for above code :**



**Code (Experiment 3):**

**List in built function :**

*# append()-appends element to list*

print("append() function")

list1 = [10, 20, 30]

print("Original list:", list1)

list1.append(50)

print("List After Appending 50:", list1)

*# extend()- extends by adding on with new list*

print("extend() function")

list1 = [10, 20, 30]

print("Original list:", list1)

list2 = [222, 333]

list1.extend(list2)

print("After extending to list2, the original list is : ", list1)

*# insert() – inserts element at particular location*

print("insert() function")

list1 = [10, 20, 30]

print("Original list:", list1)

list1.insert(2, 15)

print("List After Appending 15:", list1)

*# pop()- pops element from list*

print("pop() function")

list1 = [10, 20, 30]

print("Original list:", list1)

list1.pop(2)

print("List after poping from index 2:", list1)

*# copy()-copies a list*

print("copy() function")

list1 = [10, 20, 30]

print("Original list:", list1)

list2 = list1.copy()

print("Copied list is:", list2)

*# clear()-clears the list*

print("Clear() function")

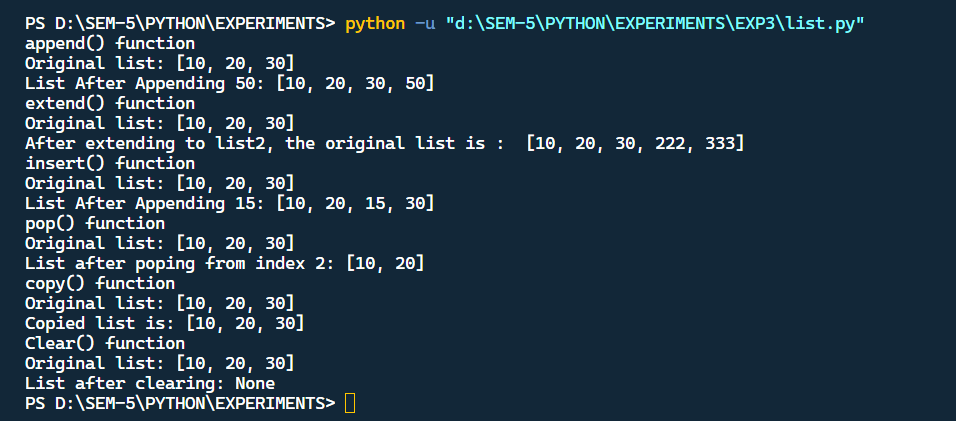
list1 = [10, 20, 30]

print("Original list:", list1)

list2 = list1.clear()

print("List after clearing:", list2)

**Output for above code :**



**Tuple in built function :**

tuple1 = (10, 20, 30, 50, 50, 70, 90, 80, 100)

print("Original tuple:", tuple1)

*# len()- length of tuple*

print("len() function")

print("The length of tuple is:", len(tuple1))

*# count()-repetation of element in tuple*

print("count() function")

print("The count of 50 in tuple is:", tuple1.count(50))

*# index()- gives index of element in tuple*

print("Index() function")

print("The index of 80 in tuple is:", tuple1.index(80))

*# sort()-sorts the elements in tuple*

print("sort() function")

print("The sorted tuple is:", sorted(tuple1))

*# min()- gives minimum element of tuple*

print("min() function")

print("The minimum element of tuple is:", min(tuple1))

*# max()- gives maximum element of tuple*

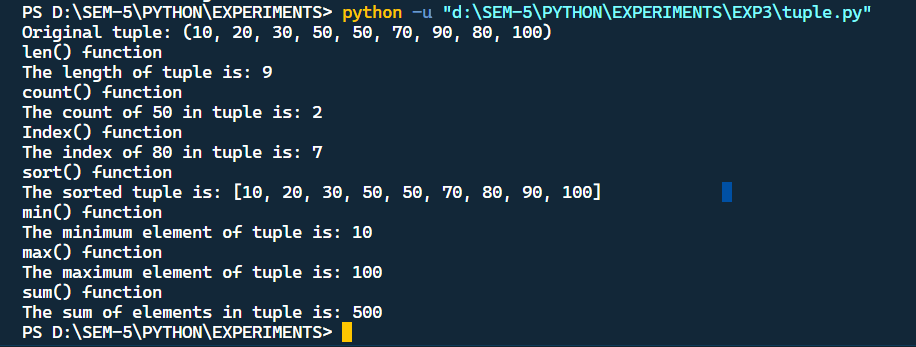
print("max() function")

print("The maximum element of tuple is:", max(tuple1))

*# sum()-gives sum of elements in tuple*

print("sum() function")

print("The sum of elements in tuple is:", sum(tuple1))



**Sets in built function :**

set1 = {'a','b', 'c', 'd', 'e'}

print("Original set is:", set1)

*# add()- adds element to set*

set1.add('f')

print("add() function")

print("Set after adding 'f'", set1)

*# discard() – discards element from set*

set1.discard('e')

print("discard() function")

print("Set after discarding/Updating 'e': ",set1)

*# remove() – removes element from set*

set1.remove('a')

print("remove() function")

print("Set after removing 'a':", set1)

*# pop()- pops element from the set*

set1.pop()

print("pop() function")

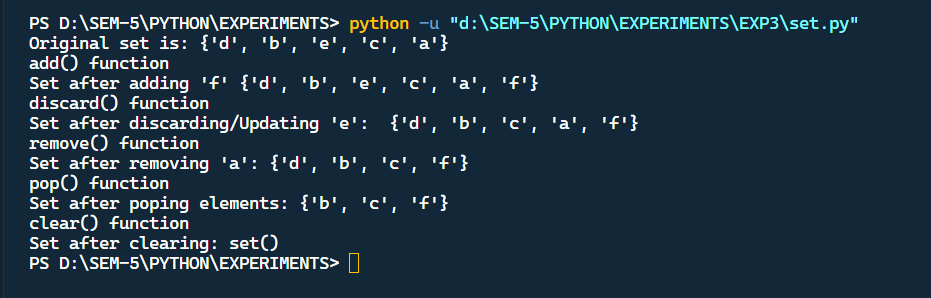
print("Set after poping elements:", set1)

*# clear()-clears the set*

set1.clear()

print("clear() function")

print("Set after clearing:",set1)



**Dictionary in built function :**

dict1 = {'1': 'One','2':'Two','3':'Three'}

print("Original dictionary:", dict1)

*# copy()- copies the dictionary*

dict2 = dict1.copy()

print("Copied Dictionary :",dict2)

*# fromkeys() – gives details from dictionary*

seq = ('1', '2', '3')

print("fromkeys() method")

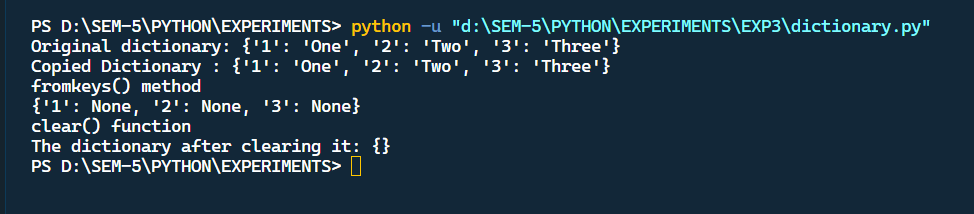
print(dict1.fromkeys(seq, None))

*# clear()- clears the dictionary*

dict1.clear()

print("clear() function")

print("The dictionary after clearing it:", dict1)



**User defined function :**

def histogram(input\_list :list) -> list[int]:

    sorted\_list = list(set(input\_list))

    hist = []

    for i in sorted\_list:

        hist.append((i,input\_list.count(i)))

    histogram = sorted(hist,key=lambda x:x[1])

    return histogram

h = histogram([13,7,12,7,11,13,14,13,7,11,13,14,12,14,14,7])

print(h)

def hanoi(disks, source, aux, target):

    if disks==1:

        print(f"Move 1 disk from peg{source} to peg{target}")

        return

    hanoi(disks-1,source,target,aux)

    print(f"Move disk {disks} from peg{source} to peg{target}")

hanoi(disks-1,aux,source,target)

disk\_count = int(input("Enter num of disks : "))

hanoi(disk\_count,'A','B','C')

def perfect\_number(num):

    """Checks if a number is perfect or not."""

    sum = 0

    for i in range(1,num//2+1):

        if num % i == 0: sum+= I

return True if sum==num else False

n = int(input("\nEnter a num : "))

isPerfect = perfect\_number(n)

print(f"{n} is a perfect number\n") if isPerfect else print(f"{n} is a not a perfect number\n")

*# code to print greatest of two number using lamda*

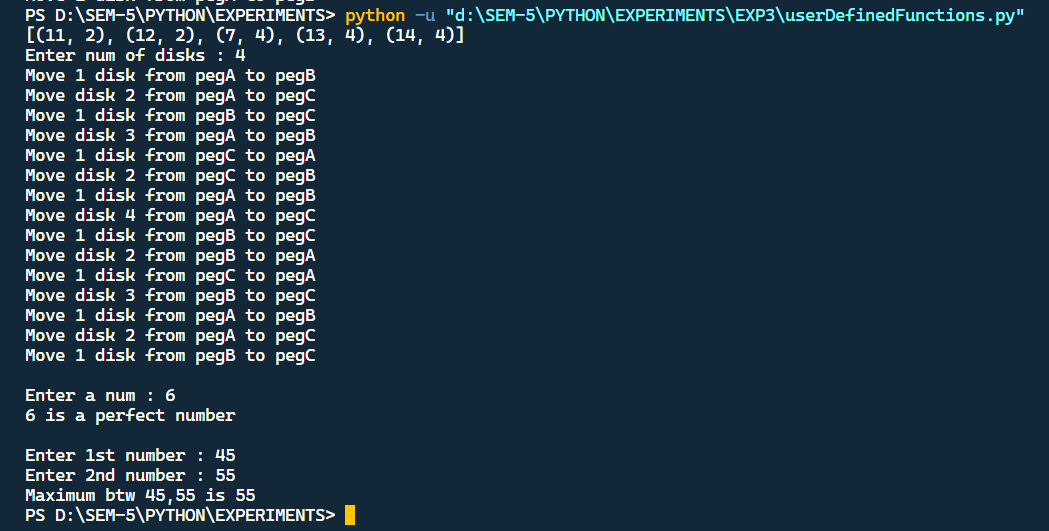
a = int(input("Enter 1st number : "))

b = int(input("Enter 2nd number : "))

maximum = lambda a,b : a if a>b else b

print(f"Maximum btw {a},{b} is {maximum(a,b)}")

**Output :**



**Code (Experiment 4):**

**OOPs concept in banking transaction :**

class BankAccount:

    def \_\_init\_\_(self, account\_number, holder\_name, balance=0):

        self.account\_number = account\_number

        self.holder\_name = holder\_name

        self.balance = balance

    def deposit(self, amount):

        if amount > 0:

            self.balance += amount

            print(f"Deposited ${amount} into account {self.account\_number}.")

        else:

            print("Invalid deposit amount.")

    def withdraw(self, amount):

        if 0 < amount <= self.balance:

            self.balance -= amount

            print(f"Withdrew ${amount} from account {self.account\_number}.")

        else:

            print("Insufficient funds or invalid withdrawal amount.")

    def display\_balance(self):

        print(f"Account {self.account\_number} balance: ${self.balance}")

class SavingsAccount(BankAccount):

    def \_\_init\_\_(self, account\_number, holder\_name, balance=0, interest\_rate=0.01):

        super().\_\_init\_\_(account\_number, holder\_name, balance)

        self.interest\_rate = interest\_rate

    def apply\_interest(self):

        interest = self.balance \* self.interest\_rate

        self.balance += interest

        print(f"Interest of ${interest} applied to account {self.account\_number}.")

class CheckingAccount(BankAccount):

    def \_\_init\_\_(self, account\_number, holder\_name, balance=0, overdraft\_limit=100):

        super().\_\_init\_\_(account\_number, holder\_name, balance)

        self.overdraft\_limit = overdraft\_limit

    def withdraw(self, amount):

        if 0 < amount <= (self.balance + self.overdraft\_limit):

            self.balance -= amount

            print(f"Withdrew ${amount} from account {self.account\_number}.")

        else:

            print("Insufficient funds or invalid withdrawal amount.")

class Bank:

    def \_\_init\_\_(self):

        self.accounts = {}

    def add\_account(self, account):

        self.accounts[account.account\_number] = account

    def get\_account(self, account\_number):

        return self.accounts.get(account\_number)

*# Create instances of BankAccount, SavingsAccount, and CheckingAccount*

account1 = BankAccount("001", "Alice", 1000)

savings\_account1 = SavingsAccount("002", "Bob", 5000, 0.02)

checking\_account1 = CheckingAccount("003", "Charlie", 2000, 500)

*# Create a Bank and add accounts to it*

bank = Bank()

bank.add\_account(account1)

bank.add\_account(savings\_account1)

bank.add\_account(checking\_account1)

*# Perform transactions*

account1.deposit(500)

account1.withdraw(200)

account1.display\_balance()

savings\_account1.deposit(1000)

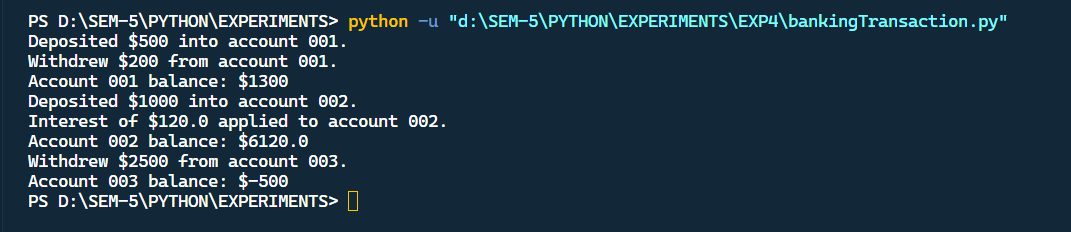
savings\_account1.apply\_interest()

savings\_account1.display\_balance()

checking\_account1.withdraw(2500)

checking\_account1.display\_balance()

**Output :**



**OOPs concept in inventory management :**

class InventoryItem:

    def \_\_init\_\_(self, item\_id, description, price):

        self.item\_id = item\_id

        self.description = description

        self.price = price

    def \_\_str\_\_(self):

        return f"{self.item\_id}: {self.description}, Price: ${self.price:.2f}"

class Electronics(InventoryItem):

    def \_\_init\_\_(self, item\_id, description, price, brand, power\_consumption):

        super().\_\_init\_\_(item\_id, description, price)

        self.brand = brand

        self.power\_consumption = power\_consumption

    def \_\_str\_\_(self):

        return super().\_\_str\_\_() + f", Brand: {self.brand}, Power Consumption: {self.power\_consumption}W"

class Clothing(InventoryItem):

    def \_\_init\_\_(self, item\_id, description, price, size, color):

        super().\_\_init\_\_(item\_id, description, price)

        self.size = size

        self.color = color

    def \_\_str\_\_(self):

        return super().\_\_str\_\_() + f", Size: {self.size}, Color: {self.color}"

class Furniture(InventoryItem):

    def \_\_init\_\_(self, item\_id, description, price, material, dimensions):

        super().\_\_init\_\_(item\_id, description, price)

        self.material = material

        self.dimensions = dimensions

    def \_\_str\_\_(self):

        return super().\_\_str\_\_() + f", Material: {self.material}, Dimensions: {self.dimensions}"

class StoreInventory:

    def \_\_init\_\_(self):

        self.inventory = []

    def add\_item(self, item):

        self.inventory.append(item)

    def remove\_item(self, item\_id):

        for item in self.inventory:

            if item.item\_id == item\_id:

                self.inventory.remove(item)

                return True

        return False

    def display\_inventory(self):

        if not self.inventory:

            print("Inventory is empty.")

        else:

            print("Inventory Items:")

            for item in self.inventory:

                print(item)

*# Usage example:*

store\_inventory = StoreInventory()

*# Add items to the inventory*

tv = Electronics("E001", "Smart TV", 799.99, "Sony", 120)

shirt = Clothing("C001", "Men's Shirt", 29.99, "M", "Blue")

table = Furniture("F001", "Coffee Table", 149.99, "Wood", "40x20x18 inches")

store\_inventory.add\_item(tv)

store\_inventory.add\_item(shirt)

store\_inventory.add\_item(table)

*# Display the inventory*

store\_inventory.display\_inventory()

*# Remove an item*

item\_id\_to\_remove = "C001"

if store\_inventory.remove\_item(item\_id\_to\_remove):

    print(f"Item {item\_id\_to\_remove} removed.")

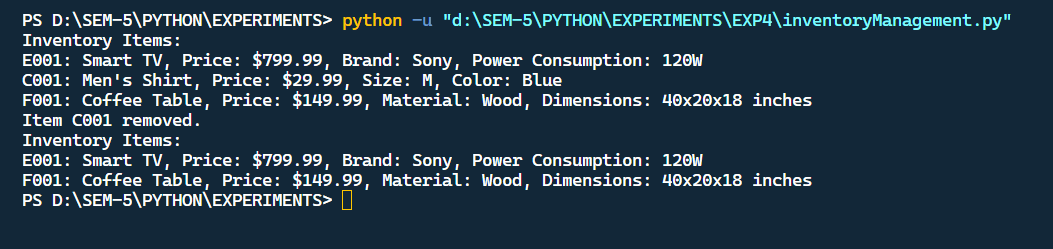
else:

print(f"Item {item\_id\_to\_remove} not found.")

*# Display the updated inventory*

store\_inventory.display\_inventory()

**Output :**



**OOPs concept in flight transaction :**

class Flight:

    def \_\_init\_\_(self, flight\_number, destination, departure\_time, capacity):

        self.flight\_number = flight\_number

        self.destination = destination

        self.departure\_time = departure\_time

        self.capacity = capacity

        self.passengers = []

    def display\_flight\_info(self):

        return f"Flight {self.flight\_number} to {self.destination} at {self.departure\_time}"

    def available\_seats(self):

        return self.capacity - len(self.passengers)

    def book\_ticket(self, passenger):

        if self.available\_seats() > 0:

            self.passengers.append(passenger)

            return True

        else:

            return False

    def reschedule\_flight(self, new\_departure\_time):

        self.departure\_time = new\_departure\_time

    def cancel\_reservation(self, passenger):

        if passenger in self.passengers:

            self.passengers.remove(passenger)

            return True

        else:

            return False

    def refund\_ticket(self, passenger):

        if passenger in self.passengers:

            self.passengers.remove(passenger)

            return True

        else:

            return False

    def prioritize\_reservations(self):

*# Sort passengers by a priority criterion (e.g., frequent flyer status)*

        self.passengers.sort(key=lambda passenger: passenger.priority, reverse=True)

class Passenger:

    def \_\_init\_\_(self, name, age, priority=0):

        self.name = name

        self.age = age

        self.priority = priority

    def \_\_str\_\_(self):

        return f"{self.age}, Age: {self.age}, Priority: {self.priority}"

class Reservation:

    def \_\_init\_\_(self, flight, passenger):

        self.flight = flight

        self.passenger = passenger

    def display\_reservation(self):

        return f"Reservation: {self.passenger.name} on {self.flight.display\_flight\_info()}"

*# Usage example:*

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

*# Create a Flight*

    flight1 = Flight("F001", "New York", "12:00 PM", 150)

*# Create Passengers*

    passenger1 = Passenger("Alice", 30, 5)

    passenger2 = Passenger("Bob", 25, 2)

    passenger3 = Passenger("Charlie", 45, 4)

*# Book Tickets*

    flight1.book\_ticket(passenger1)

    flight1.book\_ticket(passenger2)

    flight1.book\_ticket(passenger3)

*# Display Flight Info and Passengers*

    print(flight1.display\_flight\_info())

    for passenger in flight1.passengers:

        print(passenger)

*# Reschedule Flight*

    flight1.reschedule\_flight("2:00 PM")

    print("Rescheduled to:", flight1.departure\_time)

*# Cancel Reservation*

    if flight1.cancel\_reservation(passenger2):

        print(f"{passenger2.name}'s reservation has been canceled.")

*# Refund Ticket*

    if flight1.refund\_ticket(passenger3):

        print(f"Ticket refunded for {passenger3.name}.")

*# Display Passengers after changes*

    print("Passengers after changes:")

    for passenger in flight1.passengers:

        print(passenger)

*# Prioritize Reservations*

    flight1.prioritize\_reservations()

    print("Passengers after prioritizing:")

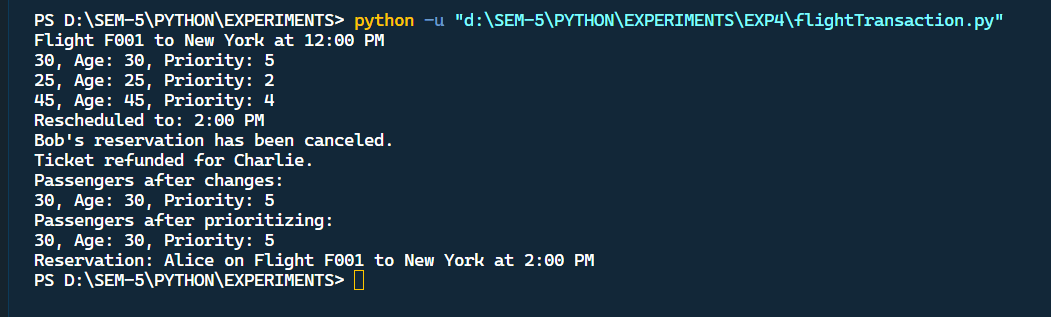
    for passenger in flight1.passengers:

        print(passenger)

*# Create Reservation*

    reservation = Reservation(flight1, passenger1)

    print(reservation.display\_reservation())



**Code (Experiment 5 ) :**

**Inbuild error and their handling :**

import math

import time

def catchKeyError():

    print("\nKey Error implemenation")

    try:

        dic = {'1':'params','2':'query'}

        print(dic['3'])

    except KeyError as ke:

        print('Key error exception : ',ke)

def catchImportError():

    print("\nimport error implemenation")

    try:

        import nonexistent\_module

    except ImportError:

        print("Import error: The module does not exist or cannot be imported.")

def catchZeroDivError():

    print("\nzero Division Error implemenation")

    try:

        print(9/0)

    except ZeroDivisionError as e:

        print€

def catchRangeError():

    print("\nout range error handling implemenation")

    try:

        l = []

        print(l[1])

    except Exception as e:

        print(e)

def catchKeyboardInterrupt():

    print('\nkeyboard interrupt ')

    try:

        print(input("Enter something : "))

        while True:

            print('print')

            time.sleep(1)

    except Exception as kbe:

    print(kbe)

*# create a custom exception*

catchKeyError()

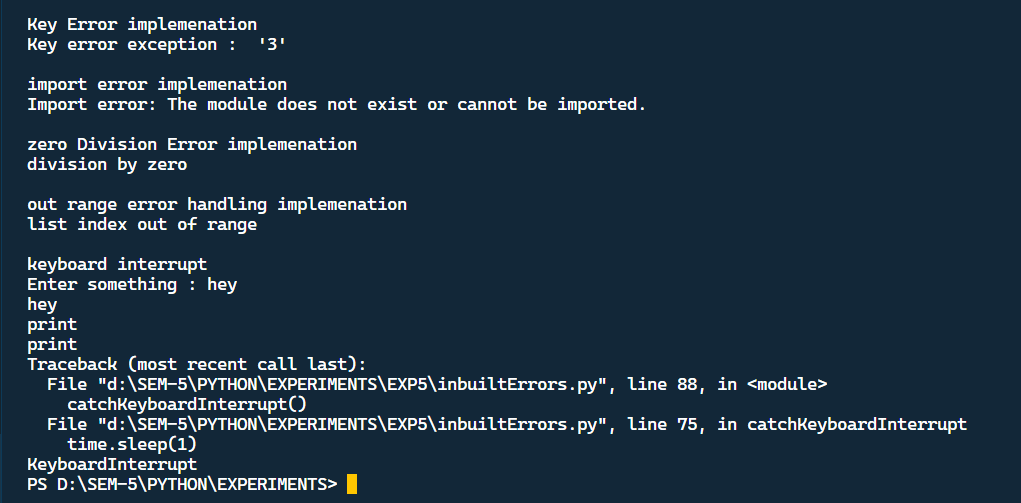
catchImportError()

catchZeroDivError()

catchRangeError()

catchKeyboardInterrupt()

**Output :**



**User defined Error Handling :**

class BaseError(Exception):pass

class HighValueError(Exception):pass

class LowValueError(Exception):pass

value = 81

while(1):

    try:

        n=int(input("Enter number:"))

        if n>value:

            raise HighValueError

        elif n < value:

            raise LowValueError

        else:

            print("Nice!Correct answer")

            break

    except LowValueError:

        print("Very Low Value, Give input again")

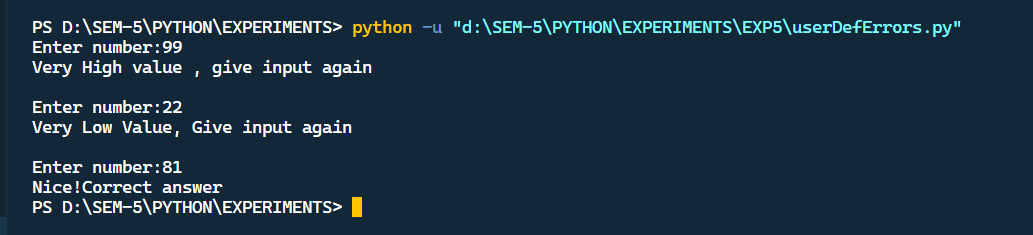
        print()

    except HighValueError:

        print("Very High value , give input again")

        print()

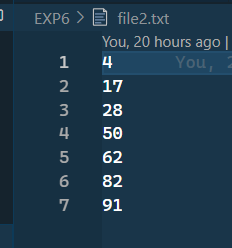
**Output :**



**Code (Experiment 6):**

File1.txt contains :   


File2.txt contains :



Sorting code :

def getValueAtIndex(content,index):

    return int(content[index].strip())

with open('file1.txt','r') as f, open('file2.txt','r') as g, open('sort.txt','w') as s:

    i = j = 0

k = 1

content1, content2 = f.readlines(), g.readlines()

    len1 = len(content1)

len2 = len(content2)

    while(i<len1 and j<len2):

        value1, value2 = int(content1[i].strip()), int(content2[j].strip())

        if value1 < value2:

            s.write(str(value1)+"\n")

            i+=1

        elif value1 > value2:

            s.write(str(value2)+"\n")

            j+=1

        else:

            s.write(str(value1)+"\n")

            s.write(str(value2)+"\n")

            i+=1

            j+=1

    while(i<len1):

        value1 = content1[i].strip()

        s.write(str(value1)+"\n")

        i+=1

    while(j<len2):

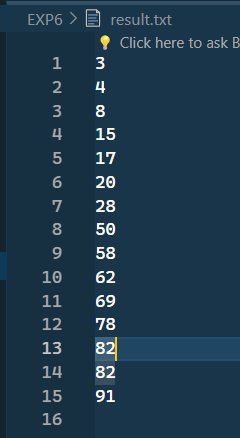
        value2 = content2[j].strip()

        s.write(str(value2)+"\n")

        j+=1

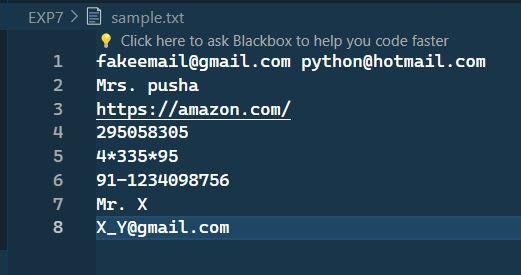
**Result :**

**A new file named as file3.txt is created and contains element as :**



**Code (Experiment 7):**

Sample text where re is to be applied :



Code for re :

import re

file = open('sample.txt')

text = file.read()

email\_pat = r'[a-zA-Z0-9\.\+\_]+@[a-zA-Z0-9\.\+]+.com'

mob\_pat = r'[0-9]+[#\-\*]\*[0-9]+[0-9]+[#\-\*]\*[0-9]'

url\_pat = r"*(?i)*\b((?:https?://|www\d{0,3}[.]|[a-z0-9.\-]+[.][a-z]{2,4}/)(?:[^\s()<>]+|\(([^\s()<>]+|(\([^\s()<>]+\)))\))+(?:\(([^\s()<>]+|(\([^\s()<>]+\)))\)|[^\s`!()\[\]{};:'\".,<>?«»“”‘’]))"

name\_pat = r'M(?:r\.|rs\.|s\.) [a-zA-Z]+'

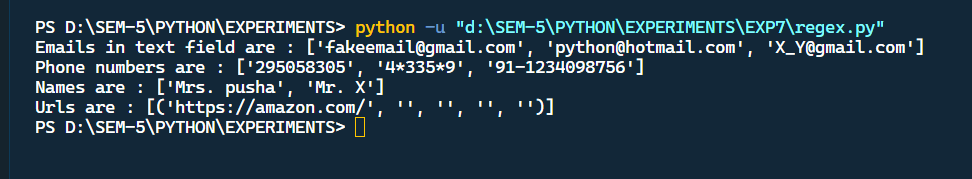
print(f"Emails in text field are : {re.findall(email\_pat,text)}")

print(f"Phone numbers are : {re.findall(mob\_pat,text)}")

print(f"Names are : {re.findall(name\_pat,text)}")

print(f"Urls are : {re.findall(url\_pat,text)}")

**Output :**



**Code (Experiment 8) :**

1. code for database creation and table creation:

import mysql.connector

mydb = mysql.connector.connect(

    host="localhost",

    user="root",

    password="xxx",

)

mycursor.execute("CREATE DATABASE electronics")

mycursor = mydb.cursor()

mycursor.execute("""

                 CREATE TABLE LAPTOP(

                    Id int(11) NOT NULL,

                    Name varchar(250) NOT NULL,

                    Price float NOT NULL,

                    Purchase\_date Date NOT NULL,

                    PRIMARY KEY(Id)

                 )

                 """)

mydb.commit()

1. code for insering values in table :

Single values query :

import mysql.connector

mydb = mysql.connector.connect(

    host="localhost",

    user="root",

    password="xxx",

    database="electronics"

)

mycursor.execute("""

                 INSERT INTO LAPTOPS(Id, Name, Price, Purchase\_date) VALUES(15,'Acer aspire 7','52000','2019-08-17')

                 """)

mydb.commit()

Multiple values query :

import mysql.connector

mydb = mysql.connector.connect(

    host="localhost",

    user="root",

    password="xxx",

    database="electronics"

)

mycursor = mydb.cursor()

sql = """

        INSERT INTO LAPTOP(Id, Name, Price, Purchase\_date, Rating) VALUES(%s, %s, %s, %s, %s)

                 """

val = [(31,'Microsoft Go','120000','2019-08-22',4.1),(29,'Macbook M2 pro','2370000','2019-08-27',4.7)]

mycursor.executemany(sql,val)

mydb.commit()

1. code for displaying rows in table :

import mysql.connector

mydb = mysql.connector.connect(

    host="localhost",

    user="root",

    password="xxx",

    database="electronics"

)

mycursor = mydb.cursor()

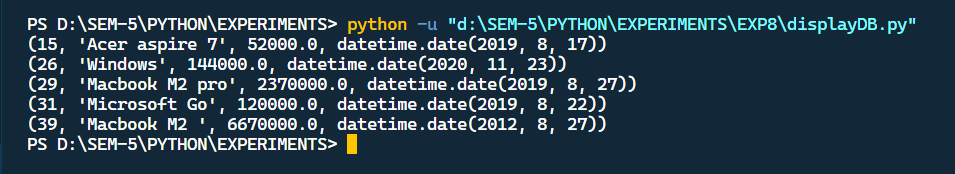
mycursor.execute("SELECT \* FROM Laptop")

res = mycursor.fetchall()

for x in res:

print(x)

**Output :**



1. code for deleting specific row in table :

import mysql.connector

mydb = mysql.connector.connect(

    host="localhost",

    user="root",

    password="xxx",

    database="electronics"

)

mycursor = mydb.cursor()

mycursor.execute("DELETE FROM Laptop WHERE Id = 17")

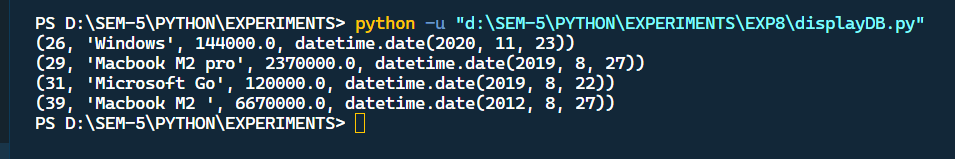
mydb.commit()

res = mycursor.fetchall()

for x in res:

    print(x)

Table after deletion :



1. code for upadting specific row in table :

import mysql.connector

mydb = mysql.connector.connect(

    host="localhost",

    user="root",

    password="xxx",

    database="electronics"

)

mycursor = mydb.cursor()

print("Eariler Records : ")

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

res = mycursor.fetchall()

for x in res:

    print(x)

mycursor.execute("UPDATE Laptop SET Price = 150000 WHERE Id = 16")

mydb.commit()

print("Updated Records : ")

mycursor = mydb.cursor()

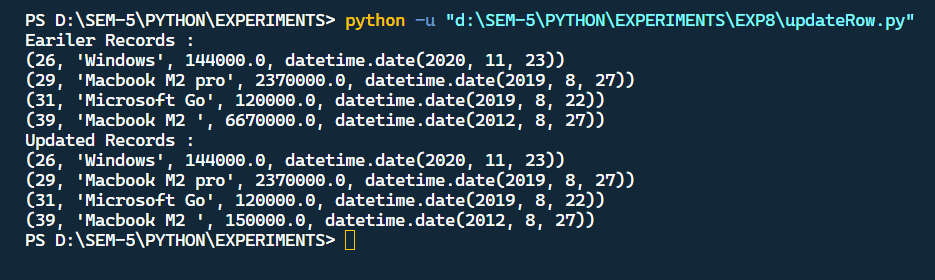
mycursor.execute("SELECT \* FROM Laptop")

res = mycursor.fetchall()

for x in res:

print(x)

Output :



1. code for searching specific row in table:

import mysql.connector

mydb = mysql.connector.connect(

    host="localhost",

    user="root",

    password="xxx",

    database="electronics"

)

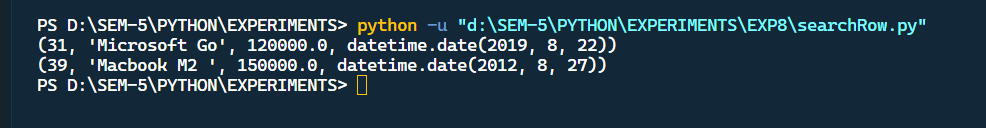
mycursor = mydb.cursor()

mycursor.execute("SELECT \* FROM Laptop WHERE Rating > 3")

res = mycursor.fetchall()

for x in res:

print(x)



1. code for altering table:

import mysql.connector

mydb = mysql.connector.connect(

    host="localhost",

    user="root",

    password="xxx",

    database="electronics"

)

mycursor = mydb.cursor()

mycursor.execute("ALTER TABLE Laptop ADD payment\_mode varchar(250)")

mydb.commit()

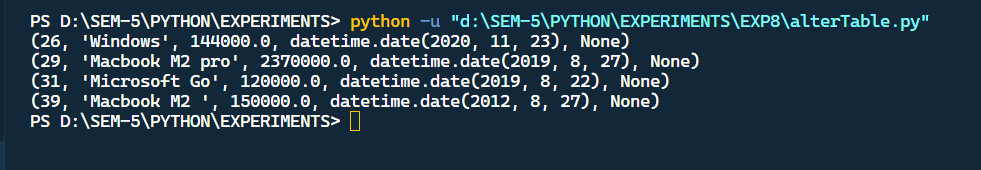
mycursor.execute("SELECT \* FROM Laptop")

res = mycursor.fetchall()

for x in res:

print(x)

**Output where a new column is added :**



1. code for deleting table and database :

import mysql.connector

mydb = mysql.connector.connect(

 host="localhost",

 user="root",

 password="xxx",

 database="electronics"

)

mycursor = mydb.cursor()

mycursor.execute("DROP TABLE LAPTOPS")

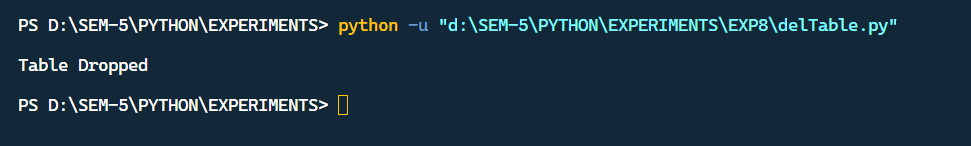
print('\nTable Dropped\n')

mydb.commit()

mycursor.execute("DROP DATABASE electronics")

mydb.close()

**Output :**



**Code (Experiment 9 ) :**

from tkinter import \*

class MyWindow:

    def \_\_init\_\_(self, win):

        self.lbl1 = Label(win, text='Select Shape:')

        self.lbl2 = Label(win, text='Select Calculation:')

        self.t1 = StringVar()

        self.t2 = StringVar()

        shapes = ['Circle', 'Rectangle', 'Sphere', 'Cone']

        calculations = ['Area', 'Perimeter', 'Curved Area', 'Volume', 'Total Surface Area']

        self.shape\_dropdown = OptionMenu(win, self.t1, \*shapes)

        self.calculation\_dropdown = OptionMenu(win, self.t2, \*calculations)

        self.shape\_dropdown.place(x=200, y=10)

        self.calculation\_dropdown.place(x=200, y=50)

        self.lbl1.place(x=100, y=10)

        self.lbl2.place(x=100, y=50)

        self.dimension\_label\_1 = Label(win, text='Enter 1st Dimension:')

        self.dimension\_label\_1.place(x=100, y=90)

        self.dimension\_entry\_1 = Entry(win)

        self.dimension\_entry\_1.place(x=200, y=90)

        self.dimension\_label\_2 = Label(win, text='Enter 2nd Dimension:')

        self.dimension\_label\_2.place(x=100, y=110)

        self.dimension\_entry\_2 = Entry(win)

        self.dimension\_entry\_2.place(x=200, y=110)

        self.dimension\_label\_3 = Label(win, text='Enter 3rd Dimension:')

        self.dimension\_label\_3.place(x=100, y=130)

        self.dimension\_entry\_3 = Entry(win)

        self.dimension\_entry\_3.place(x=200, y=130)

        self.result\_label = Label(win, text='')

        self.result\_label.place(x=200, y=170)

        self.btn = Button(text='Submit', command=self.calculate)

        self.btn.place(x=200, y=210)

    def calculate(self):

        selected\_shape = self.t1.get()

        selected\_calculation = self.t2.get()

        d1 = self.dimension\_entry\_1.get()

        d2 = self.dimension\_entry\_2.get()

        d3 = self.dimension\_entry\_3.get()

        result = "Result: "

        if selected\_shape == 'Circle':

            if selected\_calculation == 'Area':

                res = str(3.14 \* float(d1)\*float(d1))

                result += res

            elif selected\_calculation == 'Perimeter':

                result += "no Perimeter Calculation"

            elif selected\_calculation == 'Curved Area':

                result += "Circle Curved Area Calculation"

        elif selected\_shape == 'Rectangle':

            if selected\_calculation == 'Area':

                res = float(d1) \* float(d2)

                result += str(res)

            elif selected\_calculation == 'Perimeter':

                res = 2 \* (float(d1) + float(d2))

                result += str(res)

            elif selected\_calculation == 'Curved Area':

                result += "no Curved Area Calculation"

        elif selected\_shape == 'Sphere':

            if selected\_calculation == 'Volume':

                res = 1.33 \* 3.14 \* (float(d1) \* float(d1) \* float(d1))

                result += str(res)

            elif selected\_calculation == 'Perimeter':

                result += "no Perimeter Calculation"

            elif selected\_calculation == 'Area':

                result += "no Area Calculation"

            elif selected\_calculation == 'Total Surface Area':

                res = 4 \* 3.14 \* (float(d1) \* float(d1))

                result += str(res)

        elif selected\_shape == 'Cone':

            if selected\_calculation == 'Volume':

                res = 0.3333 \* 3.14 \* (float(d1) \* float(d1)) \* float(d2)

                result += str(res)

            elif selected\_calculation == 'Perimeter':

                result += "no Perimeter Calculation"

            elif selected\_calculation == 'Area':

                result += "no Area Calculation"

            elif selected\_calculation == 'Total Surface Area':

                res = 3.14 \* float(d1) \* (float(d1) + float(d2))

                result += str(res)

        self.result\_label.config(text=result)

window = Tk()

mywin = MyWindow(window)

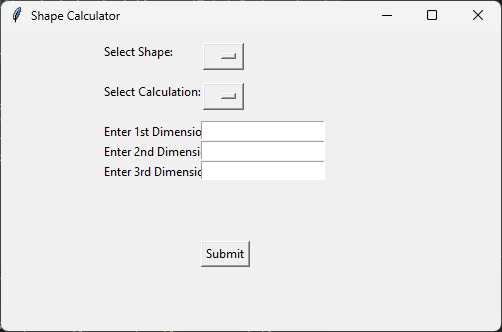
window.title('Shape Calculator')

window.geometry("500x300+10+10")

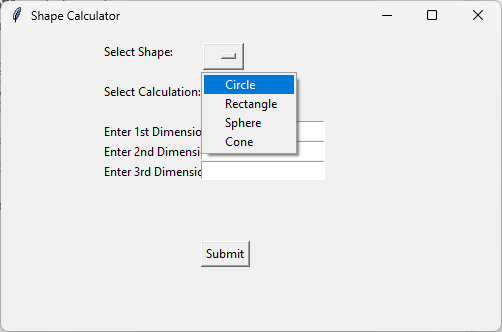
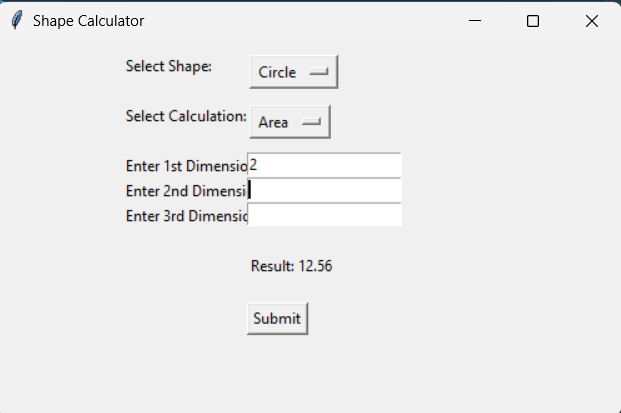
window.mainloop()

**Output Screen :**

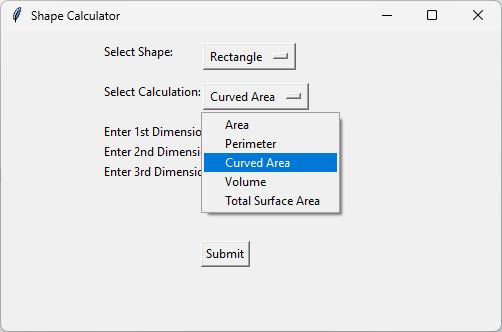
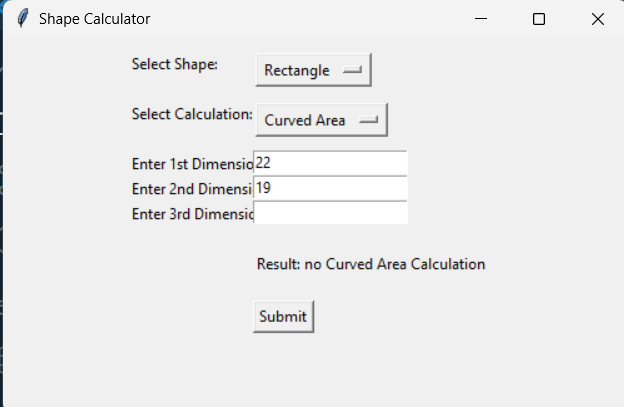
Intial screen :



selecting Shape , calculation type and entering values :

Selecting wrong configuration :

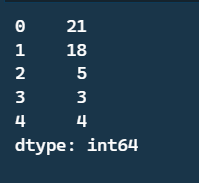
**Code (Experiment 10 ) :**

import pandas as pd

list1=[]

var1=pd.Series(list1)

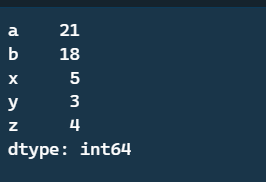
print(var1)



Giving custom indexes :

var1=pd.Series(list1,index=['a','b','x','y','z'])

print(var1)

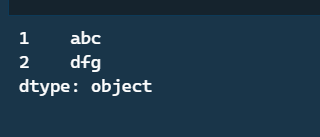


series for dictionary :

dist1={'1':'abc','2':'dfg'}

var1=pd.Series(dist1)

print(var1)



creating a DataFrame and printing value of index 0:

data={

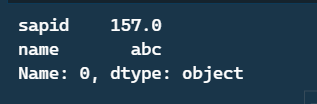
    'sapid':[157,159,161,None],

    'name':['abc','dfg','xyz',None]

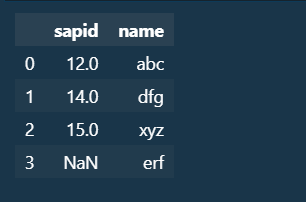
}

df=pd.DataFrame(data)

df.loc[0]

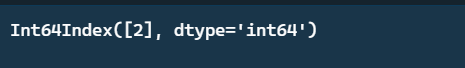


Finding the Index of Rows in a DataFrame Where the 'name' Column is 'abc :



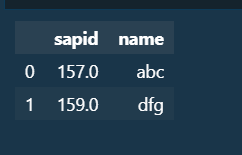
row=df.loc[df['name']=='xyz'].index

row

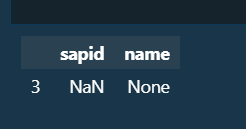


Use of head ,tail, info, finding null values in Dataframe :

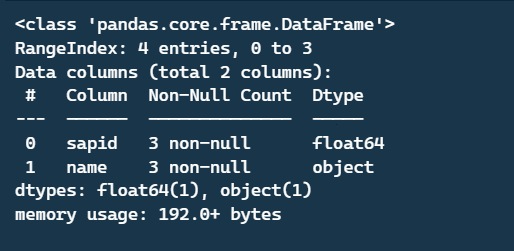
df.head(2)



df.tail(1)



df.info()



filling Null Values with mean :

data={

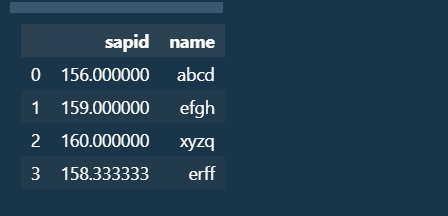
    'sapid':[156,159,160,None],

    'name':['abcd','efgh','xyzq','erff']

}

df=pd.DataFrame(data)

df.fillna(df.mean())



filling Null Values with median:

data={

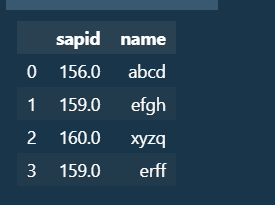
    'sapid':[156,159,160,None],

    'name':['abcd','efgh','xyzq','erff']

}

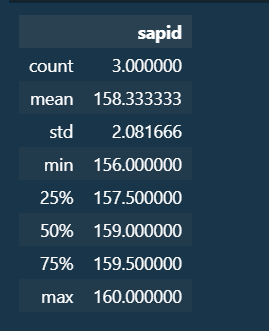
df=pd.DataFrame(data)

df.fillna(df.median())



Use of Describe method :

df.describe()



Plotting a graph :

import matplotlib.pyplot as plt

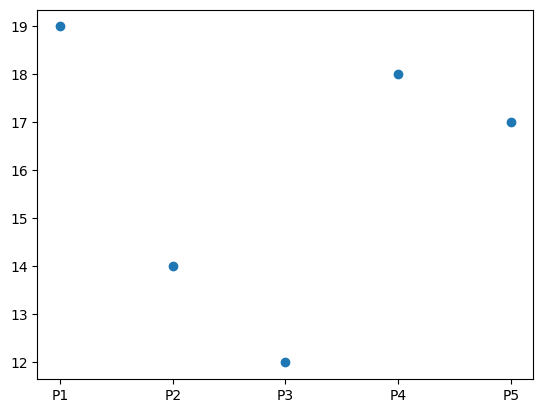
import numpy as np

x=np.array(['P1','P2','P3','P4','P5'])

y=np.array([19,14,12,18,17])

plt.scatter(x,y)

plt.show()



Plotting a Piechart :

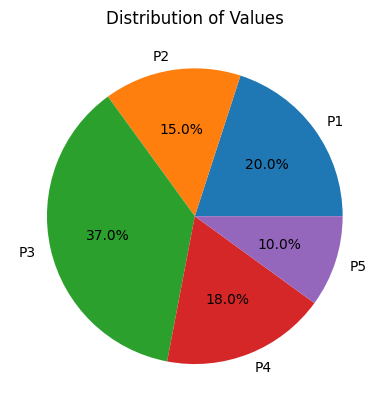
x = np.array(['P1', 'P2', 'P3', 'P4', 'P5'])

y = np.array([20, 15, 37, 18, 10])

plt.pie(y, labels=x, autopct='%1.1f%%')

plt.title('Distribution of Values')

plt.show()



Plotting a bargraph:

x = np.array(['P1', 'P2', 'P3', 'P4', 'P5'])

y = np.array([19, 14, 12, 18, 17])

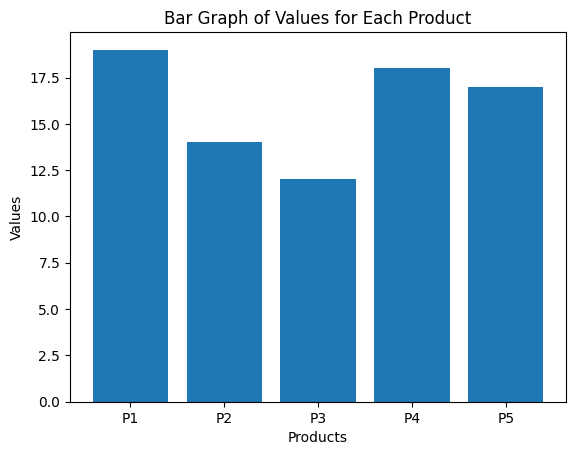
plt.bar(x, y)

plt.xlabel('Products')

plt.ylabel('Values')

plt.title('Bar Graph of Values for Each Product')

plt.show()



Plotting a histogram:

import matplotlib.pyplot as plt

import numpy as np

np.random.seed(42)

data = np.random.randn(1000)

plt.hist(data, bins=20, color='skyblue', edgecolor='black')

plt.xlabel('Random Values')

plt.ylabel('Frequency')

plt.title('Histogram of Random Values')

plt.show()

