**B.P.H.E. Society’s**

**Institute of Management Studies Career Development and Research**

**Ahmednagar – 414001**

**MCA (Sem.-II) (2020 Pattern)**

**IT-21L: Python Practical (Apr-June-2023)**

**INDEX**

**Roll No. : 83 Name of Student: Sonawane Shravani Vinod**

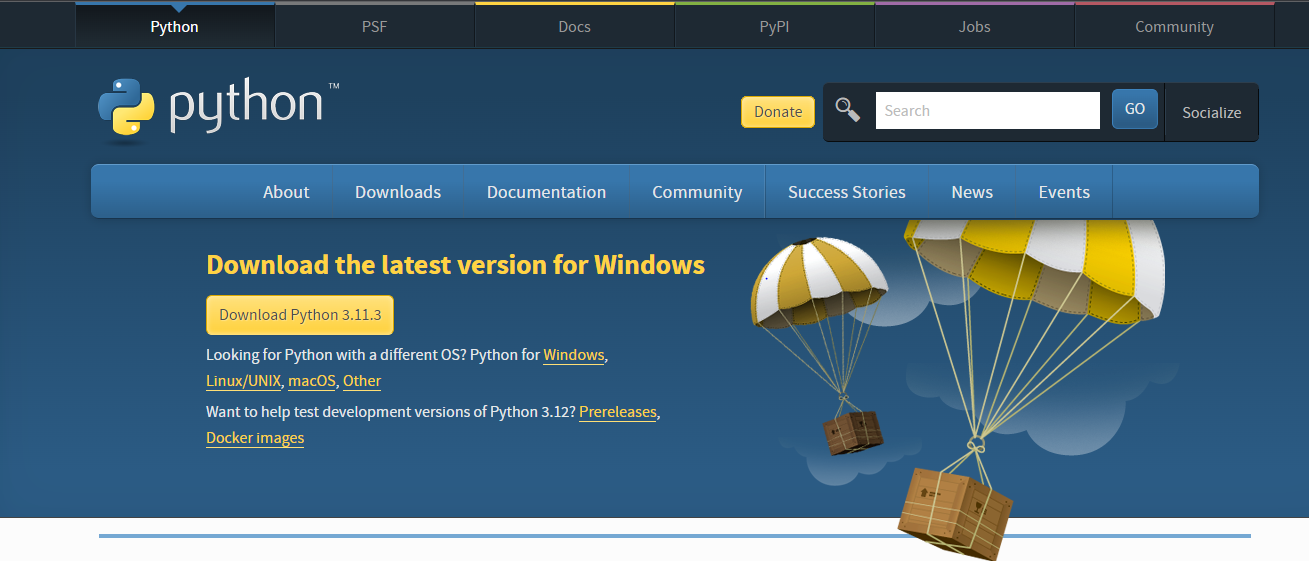
|  |  |  |  |
| --- | --- | --- | --- |
| **Sr.** | **Program Details** | **Page No.** | **Signature** |
| 1 | Python installation and configuration with windows and Linux |  |  |
| 2 | Programs for understanding the data types, control flow  statements, blocks and loops |  |  |
| 3 | Programs for understanding functions, use of built in functions,  user defined functions |  |  |
| 4 | Programs to use existing modules, packages and creating modules,  packages |  |  |
| 5 | Programs for implementations of all object-oriented concepts like  class, method, inheritance, polymorphism etc. |  |  |
| 6 | Programs for parsing of data, validations like Password, email,  URL, etc. |  |  |
| 7 | Programs for Pattern finding should be covered. |  |  |
| 8 | Programs covering all the aspects of Exception handling, user  defined exception, Multithreading should be covered. |  |  |
| 9 | Programs demonstrating the IO operations like reading from file,  writing into file from different file types like data/binary file, etc. |  |  |
| 10 | Programs to perform searching, adding, updating the content from  the file. |  |  |
| 11 | Program for performing CRUD operation with MongoDB and  Python |  |  |
| 12 | Basic programs with NumPy as Array, Searching and Sorting, date  & time and String handling |  |  |
| 13 | Programs for series and data frames should be covered. |  |  |
| 14 | Programs to demonstrate data pre-processing and data handling  with data frame |  |  |
| 15 | Program for data visualization should be covered. |  |  |

**Dr. Sanjay P. Bhakkad Dr. U.H. Nagarkar Faculty – In charge Head – IT Dept.**

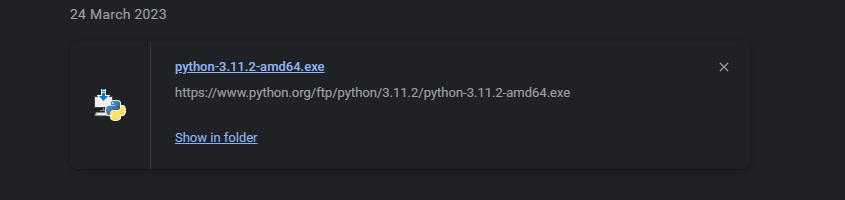
**1. Python Installation and Configuration with Windows and Linux**

a) Write steps followed during installation. If possible prepare word file along with the required screen shots captured during the installation process.

**Go to Official Website of Python.**

****

**Please Select the latest Verion of the Python and Click on the Download Button .**

****

**Run the installation file: Double-click on the installation file to start the installation process. In some cases, you might need to right-click on the file and select "Run as Administrator" to initiate the installation with administrative privileges.**

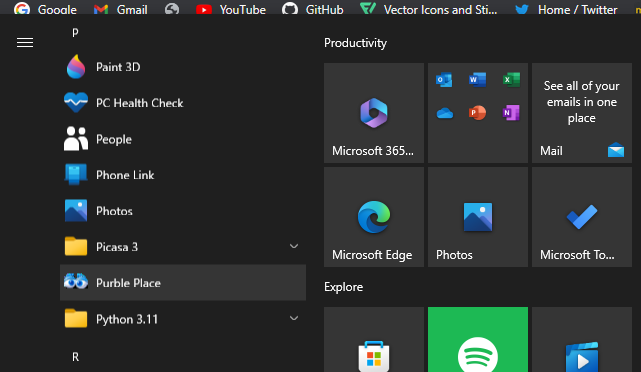
**Choose the installation language: If the software supports multiple languages, you may be prompted to select your preferred language for the installation process. Choose the desired language and click "Next" or "OK."**

**Read and accept the End User License Agreement (EULA): Most software installations include an EULA that outlines the terms and conditions of using the software. Read the agreement and if you agree to the terms, select the checkbox or click "Agree" to proceed.**

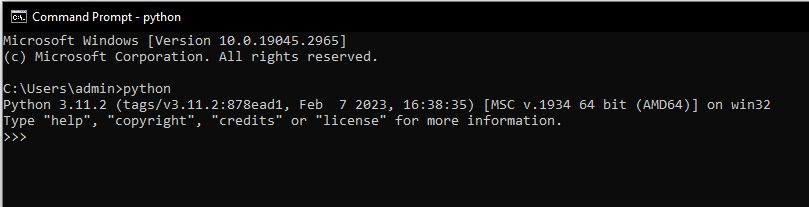
**Select the installation location: The installer might ask you to choose the destination folder where the software will be installed. The default location is usually recommended, but you can change it if necessary. Click "Next" or "Install" to proceed.**

**Wait for the installation to complete: Once you have made all the necessary selections and configurations, the installer will begin the installation process. Progress bars or indicators may be displayed to show the installation status. Wait for the process to finish.**

**Complete the installation: After the installation is complete, you may be presented with a confirmation message. Some installers also provide options to launch the software immediately after installation. Click "Finish" or "Close" to exit the installer.**

****

**Check the installation is install or not in the System using Command prompt.**

****

**Python is successfully install in the system.**

**2. Programs for understanding the data types, control flow statements, blocks and loops .**

a) Write a program to check if the input year is leap year or not.

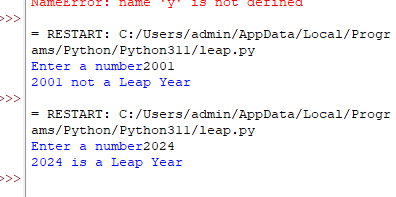
num = int(input("Enter a number"))

if (num%4) == 0 and num%100 != 0:

print(num,"is a Leap Year")

else:

print(num,"not a Leap Year")



**b) Write a program to print all prime numbers in an interval.**

lowerval = int(input ("Please, Enter the Lowest Range Value: "))

upperval = int(input ("Please, Enter the Upper Range Value: "))

print ("The Prime Numbers in the range are: ")

for number in range (lowerval, upperval + 1):

if number > 1:

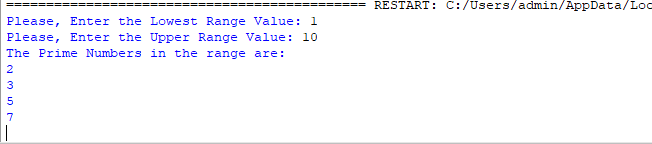
for i in range (2, number):

if (number % i) == 0:

break

else:

print (number)



**c) Write a program to find all Armstrong number(s) in an interval.**

lowerval = int(input ("Please, Enter the Lowest Range Value: "))

upperval = int(input ("Please, Enter the Upper Range Value: "))

print ("The Prime Numbers in the range are: ")

for number in range (lowerval, upperval + 1):

if number > 1:

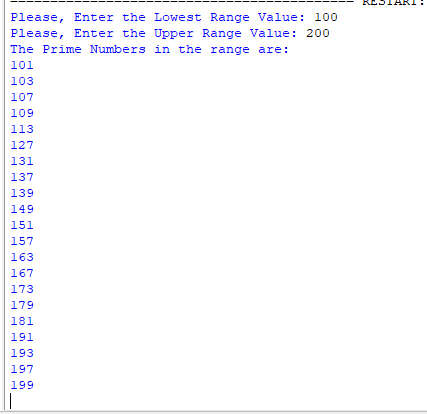
for i in range (2, number):

if (number % i) == 0:

break

else:

print (number)



**d) Write a program to print fibbonacci series up to n-th term**.

num = 14

n1, n2 = 0, 1

print("Fibonacci Series:", n1, n2, end=" ")

for i in range(2, num):

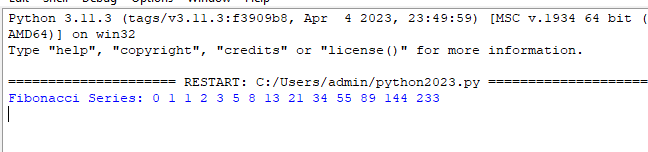
n3 = n1 + n2

n1 = n2

n2 = n3

print(n3, end=" ")

print()



**e) Write a python program to do summation of diagonal elements of a matrix.**

mtrx = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

rows = len(mtrx)

cols = len(mtrx[0])

diagonal\_sum = 0

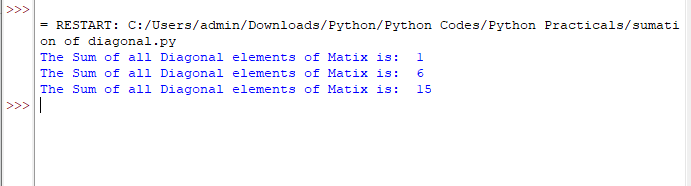
for m in range(rows):

for n in range(cols):

if m == n:

diagonal\_sum += mtrx[m][n]

print("The Sum of all Diagonal elements of Matix is: ", diagonal\_sum)



**f) Write a python program to add two matrices.**

X = [[1,7,3],[4 ,5,6],[7 ,8,9]]

Y = [[5,8,1],[6,7,3],[4,5,9]]

result = [[0,0,0],[0,0,0],[0,0,0]]

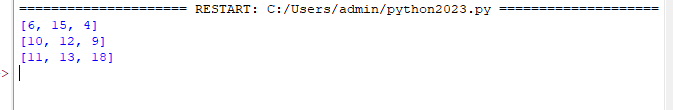
for i in range(len(X)):

for j in range(len(X[0])):

result[i][j] = X[i][j] + Y[i][j]

for r in result:

print(r)



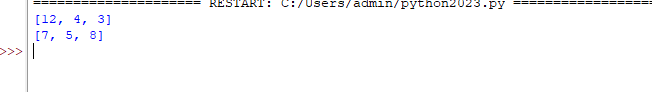
**g) Write a python program to transpose the matrix.**

X = [[12,7],[4 ,5],[3 ,8]]

result = [[X[j][i] for j in range(len(X))] for i in range(len(X[0]))]

for r in result:

print(r)



**h) Write a program to multiply two matrices.**

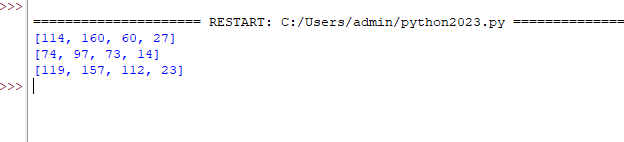
X = [[12,7,3],[4 ,5,6],[7 ,8,9]]

Y = [[5,8,1,2],[6,7,3,0],[4,5,9,1]]

result = [[sum(a\*b for a,b in zip(X\_row,Y\_col)) for Y\_col in zip(\*Y)] for X\_row in X]

for r in result:

print(r)



**i) Write programs which swap every odd-even position character in the string.** Input: “badcfe” output: “abcdef”

input= “1234567” output: “2143657”

def odd\_even\_swap(string):

result = ''

for i in range(0, len(string), 2):

if i + 1 < len(string):

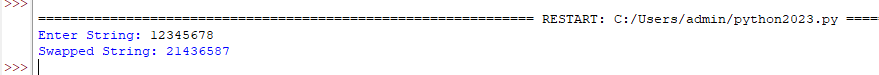
result += string[i + 1]

result += string[i]

print('Swapped String:', result)

string = input("Enter String: ")

odd\_even\_swap(string)



**j) Write a program to remove all the duplicate items from a list.**

Input List = [10,20,30,20,10,50,60,40,80,50,40]

Output List = [10, 20, 30, 50, 60, 40, 80]

my\_list = [10, 20, 30, 20, 10, 50, 60, 40, 80, 50, 40]

print("List Before:", my\_list)

temp\_list = []

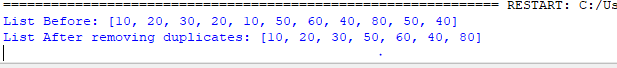
for i in my\_list:

if i not in temp\_list:

temp\_list.append(i)

my\_list = temp\_list

print("List After removing duplicates:", my\_list)



**3. Programs for understanding functions, use of built in functions, user defined functions**

**a) Write the following programs by making use of built-in functions:**

**a. To print string in reverse order by words**

def reverse\_string\_by\_words(string):

words = string.split()

reversed\_words = reversed(words)

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

return reversed\_string

# Example usage:

string = "Hello World, I am here"

reversed\_string = reverse\_string\_by\_words(string)

print(reversed\_string)



**b. To find all indices of a sub string in a given string**

def find\_substring\_indices(string, substring):

indices = []

start = 0

while start < len(string):

index = string.find(substring, start)

if index == -1:

break

indices.append(index)

start = index + 1

return indices

# Example usage:

string = "Hello World, Hello"

substring = "Hello"

indices = find\_substring\_indices(string, substring)

print(indices)



**b) Write a user defined function**

**a. average() which will return the average of the numbers inputted by the user.**

num = int(input('How many numbers: '))

total\_sum = 0

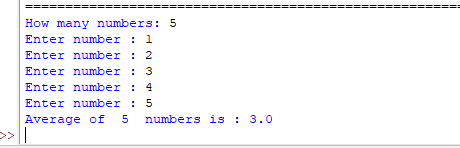
for n in range(num):

numbers = float(input('Enter number : '))

total\_sum += numbers

avg = total\_sum/num

print('Average of ', num, ' numbers is :', avg)



**b. nMax() which will return the maximum between the entered numbers.**

def max( x, y ):

if x > y:

return x

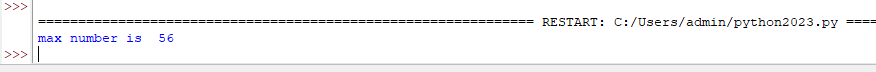
else:

return y

def maxx( x, y, z ):

return max( x, max( y, z ) )

print("max number is ",maxx(3, 45, 56))



**c) Write a program to find the factorial of a number using recursion.**

def recurFact(n):

if n == 1 or n == 0:

return 1

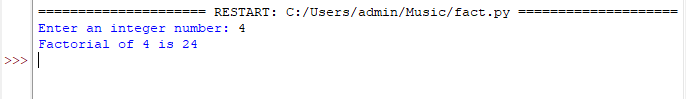
else:

return n \* recurFact(n - 1)

num = int(input("Enter an integer number: "))

result = recurFact(num)

print("Factorial of", num, "is", result)



**d) Write a program to generate a fibonacci sequence using recursion.**

n = int(input("Enter length of Fibonacci series: "))

num1 = 0

num2 = 1

next\_number = 0

count = 1

while(count <= n):

print(next\_number, end=" ")

count += 1

num1 = num2

num2 = next\_number

next\_number = num1 + num2

t\_number = num1 + num2



**e) Write a program to print the sum of natural numbers using recursion.**

**def sum\_of\_natural\_numbers(n):**

if n == 1:

return 1

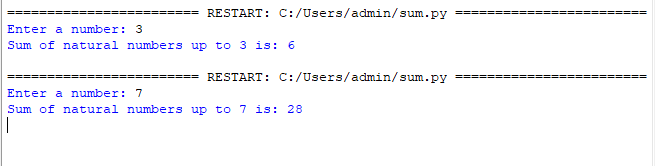
else:

return n + sum\_of\_natural\_numbers(n - 1)

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

result = sum\_of\_natural\_numbers(num)

print("Sum of natural numbers up to", num, "is:", result)



**f) Write a lambda functions for the following:**

**a. Lambda function which will return the square root of a given number.**

import math

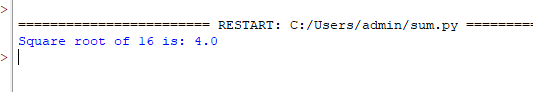
square\_root = lambda x: math.sqrt(x)

# Example usage:

number = 16

result = square\_root(number)

print("Square root of", number, "is:", result)



**b. Lambda function which will return the maximum between two numbers.**

maximum = lambda x, y: x if x > y else y

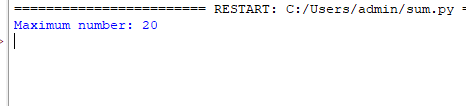
# Example usage:

num1 = 10

num2 = 20

result = maximum(num1, num2)

print("Maximum number:", result)



**c. Lambda function which will return True if the number is even, False otherwise.**

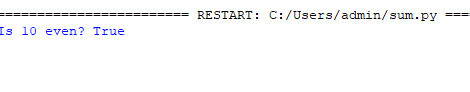
is\_even = lambda x: x % 2 == 0

# Example usage:

number = 10

result = is\_even(number)

print("Is", number, "even?", result)



**g) Write a message encoder - decoder program using a nested function. Define outer function as encodeMsg() which will accept a message to encode and a key to encode with. Define the inner function as encoder() which will encode the message with a given key (how to encode, that is up to you). The outer function then returns this encoded message to the main program. Similarly, do the task of decoding the encoded message, by writing outer function as decodeMsg() and inner function as decoder().**

**def encodeMsg(message, key):**

def encoder(message, key):

encoded\_message = ''

for char in message:

encoded\_char = chr(ord(char) + key)

encoded\_message += encoded\_char

return encoded\_message

return encoder(message, key)

def decodeMsg(encoded\_message, key):

def decoder(encoded\_message, key):

decoded\_message = ''

for char in encoded\_message:

decoded\_char = chr(ord(char) - key)

decoded\_message += decoded\_char

return decoded\_message

return decoder(encoded\_message, key)

message = input("Enter a message to encode: ")

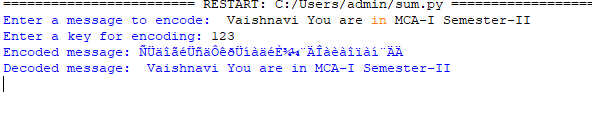
key = int(input("Enter a key for encoding: "))

encoded\_message = encodeMsg(message, key)

print("Encoded message:", encoded\_message)

decoded\_message = decodeMsg(encoded\_message, key)

print("Decoded message:", decoded\_message)



**h) Implement a generator called “squares” to yield the square of all numbers from (a) to (b). Test it with a “for” loop and print each of the yielded values.**

def squares(a, b):

for num in range(a, b + 1):

yield num \*\* 2

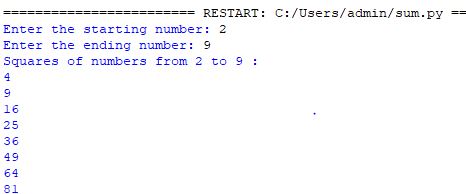
a = int(input("Enter the starting number: "))

b = int(input("Enter the ending number: "))

print("Squares of numbers from", a, "to", b, ":")

for square in squares(a, b):

print(square)



**i) Create a generator which will generate the next even number from the number passed. For example if the number passed is 3 then the next even number is 4, if the number passed is 6 then the next even number is 8.**

def next\_even(number):

if number % 2 == 0:

number += 2

else:

number += 1

while True:

yield number

number += 2

# Example usage:

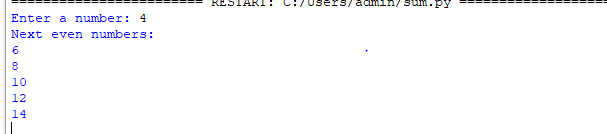
num = int(input("Enter a number: "))

generator = next\_even(num)

print("Next even numbers:")

for \_ in range(5):

print(next(generator))



**j) Create a decorator “smart\_div” which is used to decorate the regular function “div”. The decorator contains an “inner()” function which will do the division operation only if the denominator is not zero. (i.e. the smart division).**

def smart\_div(func):

def inner(a, b):

if b == 0:

print("Error: Division by zero!")

return None

else:

return func(a, b)

return inner

@smart\_div

def div(a, b):

return a / b

# Example usage:

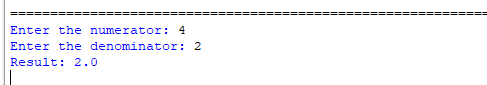
num1 = int(input("Enter the numerator: "))

num2 = int(input("Enter the denominator: "))

result = div(num1, num2)

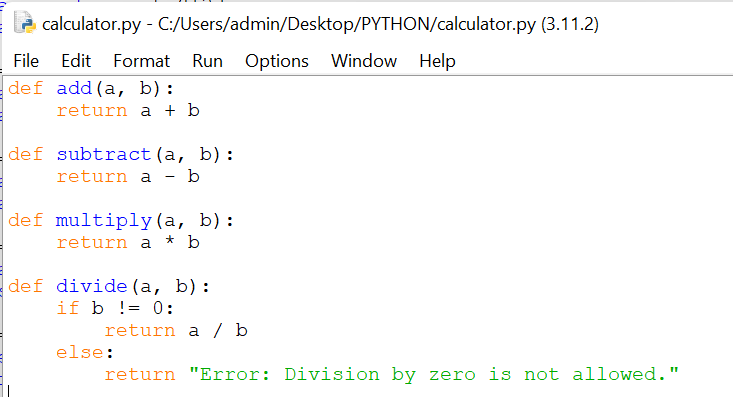
if result is not None:

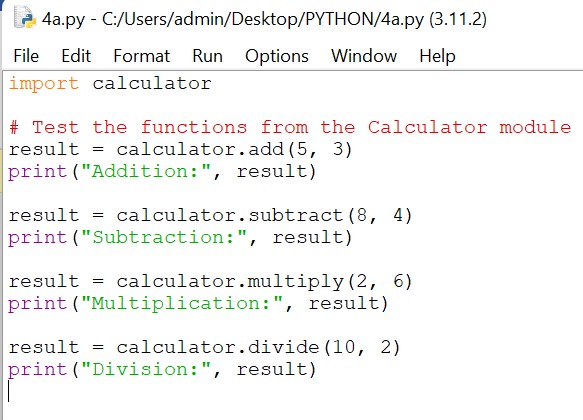
print("Result:", result)



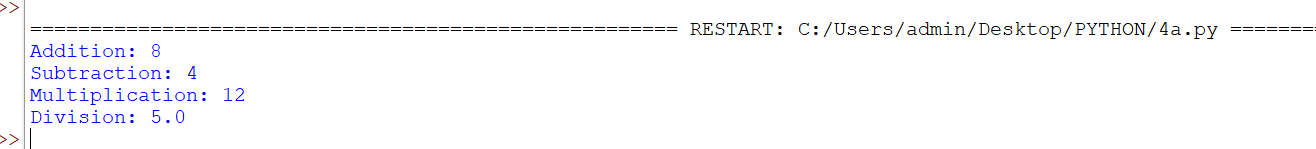
**4. Programs to use existing modules, packages and creating modules, packages**

**a) Create a module “Calculator” which contains the functions for addition, subtraction, multiplication and division. Write a main program to demonstrate the use of this module.**

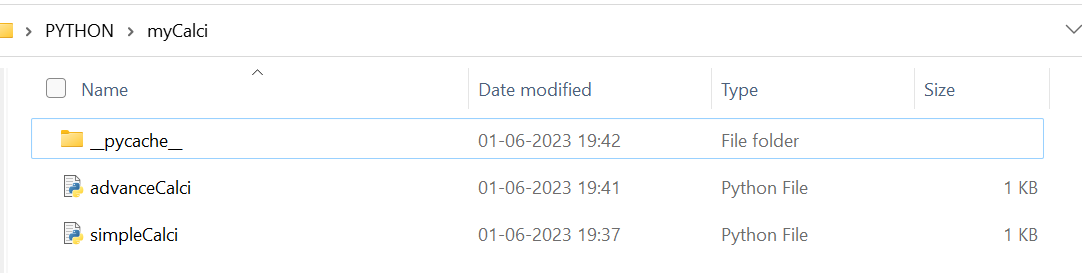


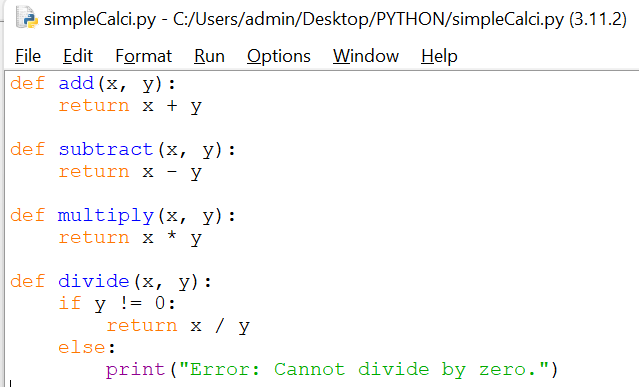


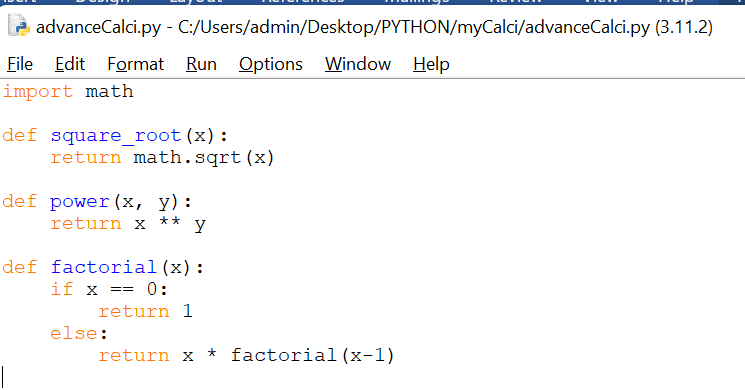
**Output:**

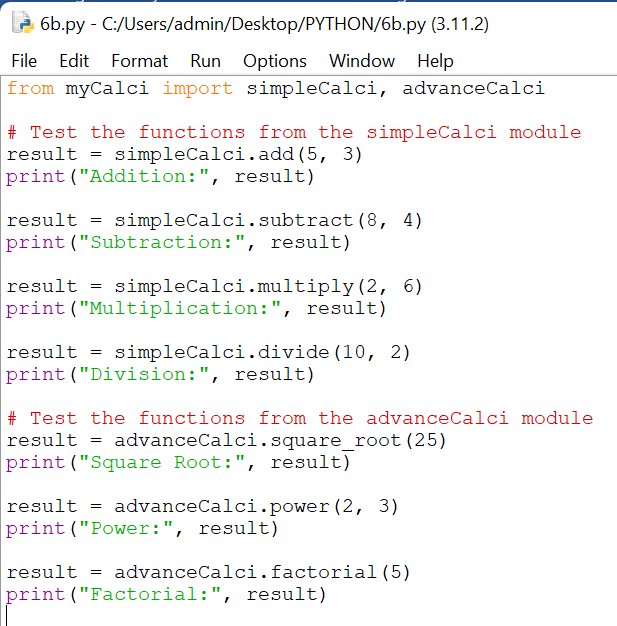


**b) Create a package “myCalci” which contains the modules “simpleCalci” and “advanceCalci”. Write the code for above two modules as per your choice. Write a main program to demonstrate the use of this package.**

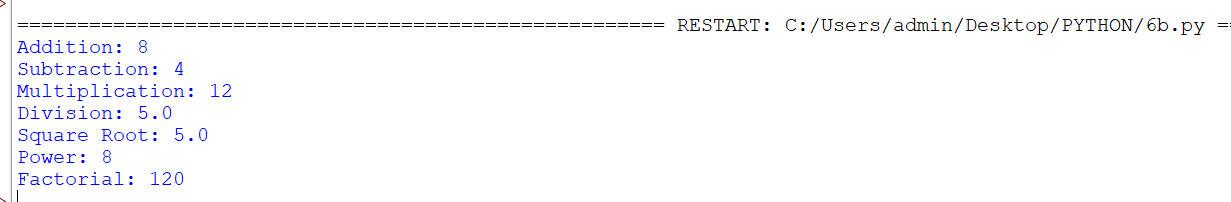








**Output:**



**c) Write a program to demonstrate the use of datetime moduleas:**

**a. Display date in different format**

import calculator

# Test the functions from the Calculator module

result = calculator.add(5, 3)

print("Addition:", result)

result = calculator.subtract(8, 4)

print("Subtraction:", result)

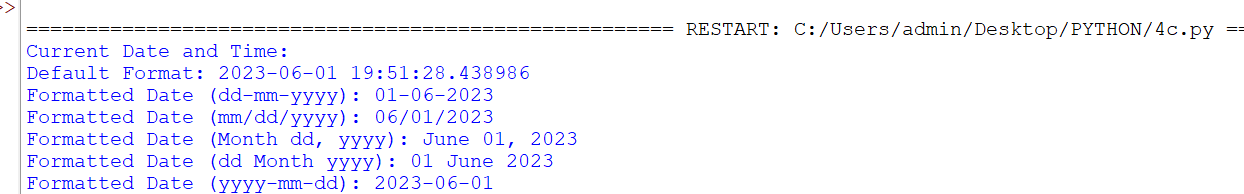
result = calculator.multiply(2, 6)

print("Multiplication:", result)

result = calculator.divide(10, 2)

print("Division:", result)

**Output:**



**b. Date arithmetic like display date after one year, date before 7 days etc.**

import datetime

# Get the current date and time

current\_datetime = datetime.datetime.now()

# Display the date in different formats

print("Current Date and Time:")

print("Default Format:", current\_datetime)

print("Formatted Date (dd-mm-yyyy):", current\_datetime.strftime("%d-%m-%Y"))

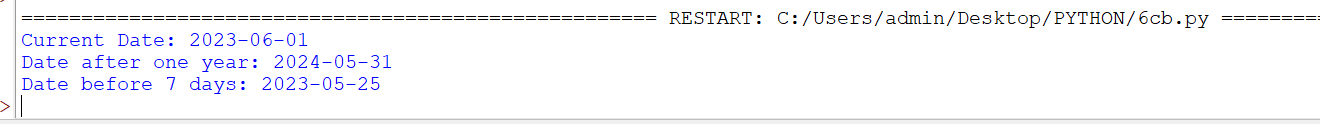
print("Formatted Date (mm/dd/yyyy):", current\_datetime.strftime("%m/%d/%Y"))

print("Formatted Date (Month dd, yyyy):", current\_datetime.strftime("%B %d, %Y"))

print("Formatted Date (dd Month yyyy):", current\_datetime.strftime("%d %B %Y"))

print("Formatted Date (yyyy-mm-dd):", current\_datetime.strftime("%Y-%m-%d"))

**Output:**



**d) Write a program to guess a number generated randomly.**

import random

# Generate a random number between 1 and 100

secret\_number = random.randint(1, 100)

# Initialize the guess variable

guess = None

# Start the guessing loop

while guess != secret\_number:

# Ask the user to input a guess

guess = int(input("Guess the number (between 1 and 100): "))

# Compare the guess with the secret number

if guess < secret\_number:

print("Too low! Try again.")

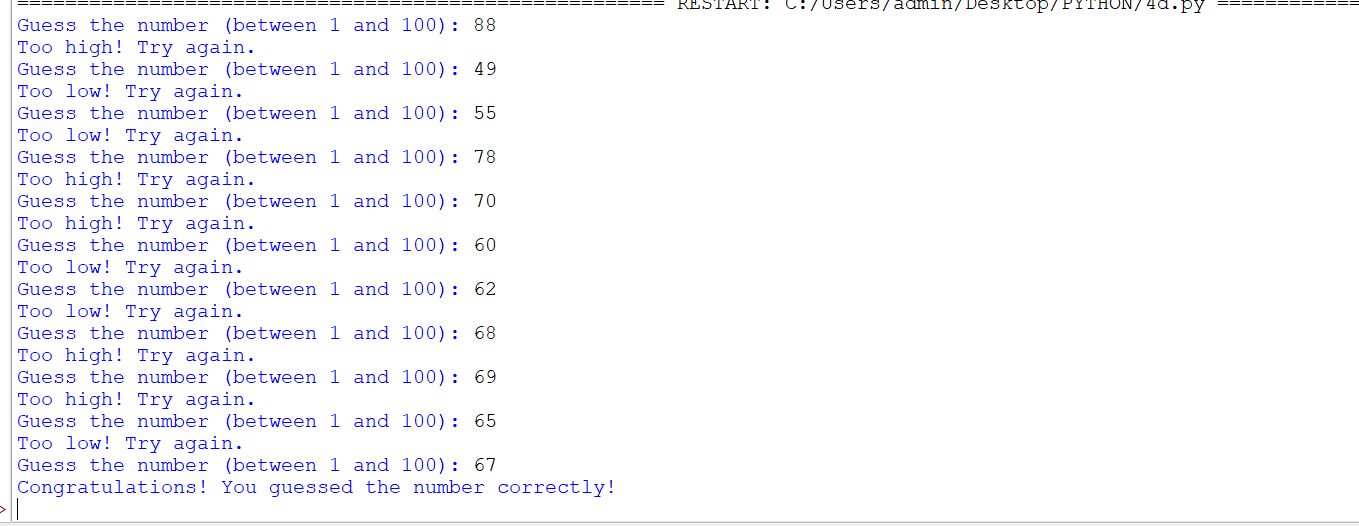
elif guess > secret\_number:

print("Too high! Try again.")

else:

print("Congratulations! You guessed the number correctly!")

Output:



**e) Write a program to implement Rock, Paper, Scissor game between human and computer. Make use of random module.**

import random

def play\_game():

# Define the valid choices

choices = ["rock", "paper", "scissors"]

# Get the user's choice

user\_choice = input("Enter your choice (rock, paper, scissors): ").lower()

# Validate the user's choice

while user\_choice not in choices:

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

user\_choice = input("Enter your choice (rock, paper, scissors): ").lower()

# Generate the computer's choice

computer\_choice = random.choice(choices)

# Display the choices

print("You chose:", user\_choice)

print("Computer chose:", computer\_choice)

# Determine the winner

if user\_choice == computer\_choice:

print("It's a tie!")

elif (

(user\_choice == "rock" and computer\_choice == "scissors") or

(user\_choice == "paper" and computer\_choice == "rock") or

(user\_choice == "scissors" and computer\_choice == "paper")

):

print("You win!")

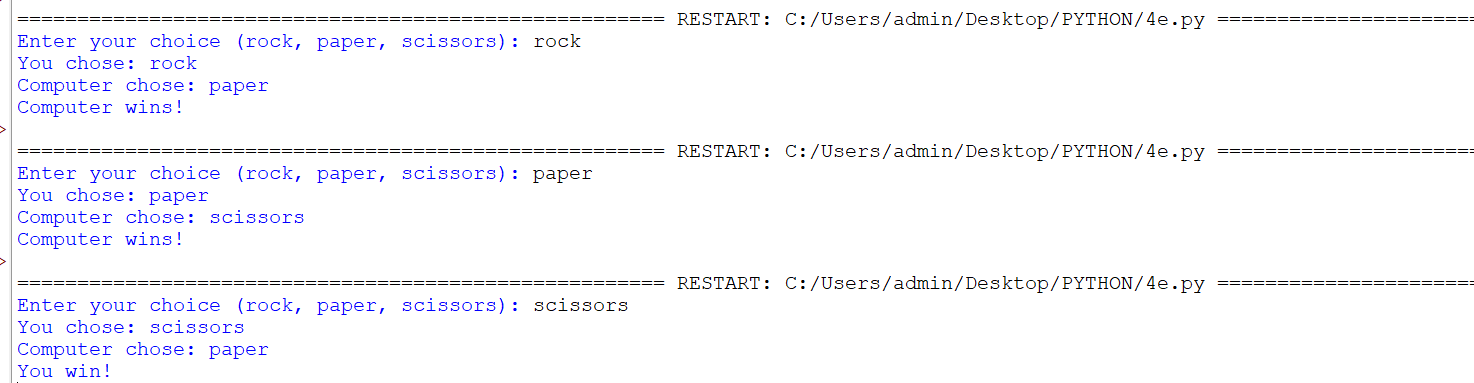
else:

print("Computer wins!")

# Play the game

play\_game()

**Output:**



**5. Programs for implementations of all object-oriented concepts like class, method, inheritance, polymorphism etc. (Real life examples must be covered for the implementation of OOP concepts)**

a) Create a class “Bank” having attributes “Bank Name”, “Branch”, “City”, “Manager Name” and

methods “Change of Manager Name” and “Display Details”. Write a constructor to initialize the instance variables. Write a main program to demonstrate the use of Bank class.

class Bank:

def \_\_init\_\_(self, bank\_name, branch, city, manager\_name):

self.bank\_name = bank\_name

self.branch = branch

self.city = city

self.manager\_name = manager\_name

def change\_manager\_name(self, new\_manager\_name):

self.manager\_name = new\_manager\_name

print("Manager name changed successfully!")

def display\_details(self):

print("Bank Name:", self.bank\_name)

print("Branch:", self.branch)

print("City:", self.city)

print("Manager Name:", self.manager\_name)

# Example usage:

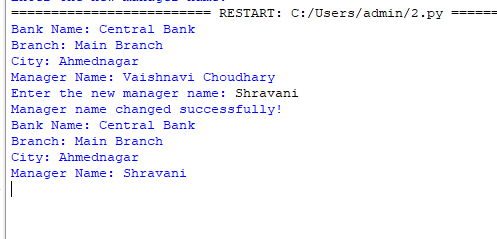
bank = Bank("Central Bank", "Main Branch", "Ahmednagar", "Vaishnavi Choudhary")

bank.display\_details()

new\_manager = input("Enter the new manager name: ")

bank.change\_manager\_name(new\_manager)

bank.display\_details()



**b) Create a class “Bank Account”, which will inherit the above “Bank” class. The Bank Account class contains the attributes “First Name”, “Last Name”, “Date of Birth”, “Profession”, “Address” and methods “Change of address”, “Change of Profession”, “Display Account Information”. Write a main program to demonstrate the use of Bank Account class. (*Hint: Program to implement single inheritance*)**

**class Bank:**

def \_\_init\_\_(self, bank\_name, branch, city, manager\_name):

self.bank\_name = bank\_name

self.branch = branch

self.city = city

self.manager\_name = manager\_name

def change\_manager\_name(self, new\_manager\_name):

self.manager\_name = new\_manager\_name

print("Manager name changed successfully!")

def display\_details(self):

print("Bank Name:", self.bank\_name)

print("Branch:", self.branch)

print("City:", self.city)

print("Manager Name:", self.manager\_name)

class BankAccount(Bank):

def \_\_init\_\_(self, bank\_name, branch, city, manager\_name, first\_name, last\_name, dob, profession, address):

super().\_\_init\_\_(bank\_name, branch, city, manager\_name)

self.first\_name = first\_name

self.last\_name = last\_name

self.dob = dob

self.profession = profession

self.address = address

def change\_address(self, new\_address):

self.address = new\_address

print("Address changed successfully!")

def change\_profession(self, new\_profession):

self.profession = new\_profession

print("Profession changed successfully!")

def display\_account\_info(self):

self.display\_details()

print("First Name:", self.first\_name)

print("Last Name:", self.last\_name)

print("Date of Birth:", self.dob)

print("Profession:", self.profession)

print("Address:", self.address)

# Example usage:

bank\_account = BankAccount("Central Bank", "Main Branch", "Ahmednagar", "Vaishnavi Choudhary", "Vaishnavi", "Choudhary", "2001-02-03", "Engineer", "123 Main St")

bank\_account.display\_account\_info()

new\_address = input("Enter the new address: ")

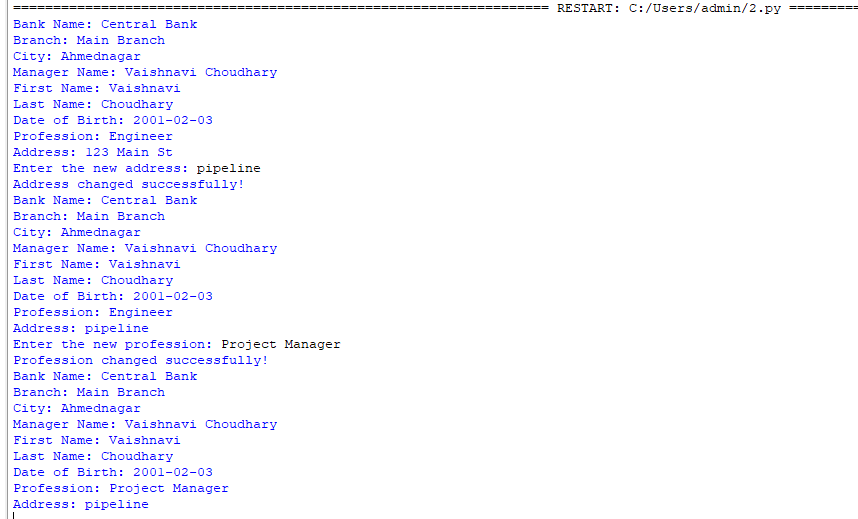
bank\_account.change\_address(new\_address)

bank\_account.display\_account\_info()

new\_profession = input("Enter the new profession: ")

bank\_account.change\_profession(new\_profession)

bank\_account.display\_account\_info()



**c) Create a class “Bank Operations” which inherits the above “Bank Account” class. The Bank Operations class contains the attribute “Balance Amount” and methods “Deposit Amount”,**

**“Withdraw Amount”, “Display Balance”. Write a main program to demonstrate the use of Bank Operations class. (*Hint: Program to implement multilevel inheritance*)**

class BankOperations(BankAccount):

def \_\_init\_\_(self, bank\_name, branch, city, manager\_name, first\_name, last\_name, dob, profession, address, balance):

super().\_\_init\_\_(bank\_name, branch, city, manager\_name, first\_name, last\_name, dob, profession, address)

self.balance = balance

def deposit\_amount(self, amount):

self.balance += amount

print("Amount deposited successfully!")

def withdraw\_amount(self, amount):

if amount <= self.balance:

self.balance -= amount

print("Amount withdrawn successfully!")

else:

print("Insufficient balance!")

def display\_balance(self):

print("Balance:", self.balance)

# Example usage:

bank\_operations = BankOperations("Central Bank", "Main Branch", "Ahmednagar", "Vaishnavi Choudhary", "Vaishnavi", "Choudhary", "2001-02-03", "Engineer", "123 Main St", 10000)

bank\_operations.display\_account\_info()

bank\_operations.display\_balance()

deposit\_amount = float(input("Enter the amount to deposit: "))

bank\_operations.deposit\_amount(deposit\_amount)

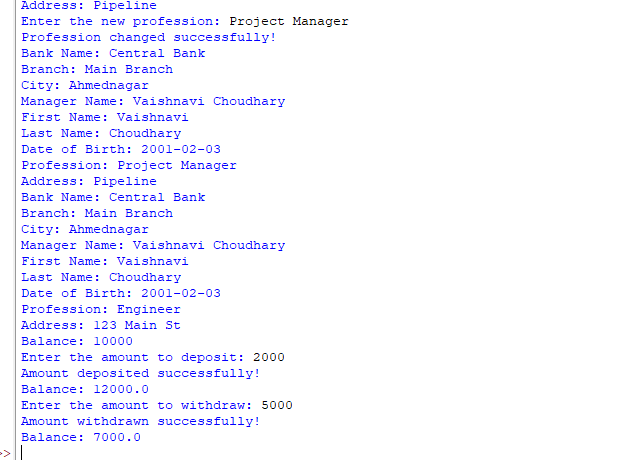
bank\_operations.display\_balance()

withdraw\_amount = float(input("Enter the amount to withdraw: "))

bank\_operations.withdraw\_amount(withdraw\_amount)

bank\_operations.display\_balance()

5



**d) Create a class “Person” having attributes “First Name”, “Last Name”, “Qualification” and methods “Update Qualification”, “Display Person Details”. Write a main program to demonstrate the use of Person class.**

**class Person:**

def \_\_init\_\_(self, first\_name, last\_name, qualification):

self.first\_name = first\_name

self.last\_name = last\_name

self.qualification = qualification

def update\_qualification(self, new\_qualification):

self.qualification = new\_qualification

print("Qualification updated successfully!")

def display\_person\_details(self):

print("First Name:", self.first\_name)

print("Last Name:", self.last\_name)

print("Qualification:", self.qualification)

# Create a Person object

person = Person("Vaishnavi", "Choudhary", "Bachelor's Degree")

# Display the initial person details

person.display\_person\_details()

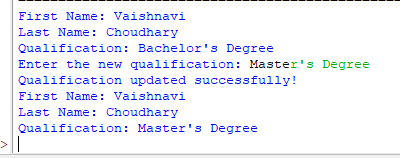
# Update the qualification

new\_qualification = input("Enter the new qualification: ")

person.update\_qualification(new\_qualification)

# Display the updated person details

person.display\_person\_details()



**e) Create a class “Employee” which inherits “Person” and “Bank” class. *(i.e. multiple inheritance)*. The Employee class contains attributes “Years of experience”, “Date of Joining”, ‘Skills” and methods “Update years of experience”, “Update Skills”, “Display Employee Details”. Write a main program to demonstrate the use of Employee class.**

class Employee(Person, Bank):

def \_\_init\_\_(self, first\_name, last\_name, qualification, bank\_name, branch, city, manager\_name, years\_of\_experience, date\_of\_joining, skills):

Person.\_\_init\_\_(self, first\_name, last\_name, qualification)

Bank.\_\_init\_\_(self, bank\_name, branch, city, manager\_name)

self.years\_of\_experience = years\_of\_experience

self.date\_of\_joining = date\_of\_joining

self.skills = skills

def update\_years\_of\_experience(self, new\_experience):

self.years\_of\_experience = new\_experience

print("Years of experience updated successfully!")

def update\_skills(self, new\_skills):

self.skills = new\_skills

print("Skills updated successfully!")

def display\_employee\_details(self):

self.display\_details()

self.display\_person\_details()

print("Years of Experience:", self.years\_of\_experience)

print("Date of Joining:", self.date\_of\_joining)

print("Skills:", self.skills)

# Create an Employee object

employee = Employee("Vaishnavi", "Choudhary", "Bachelor's Degree", "Central Bank", "Main Branch", "Ahmednagar", "Vaishnavi Choudhary", 5, "2020-01-01", "Programming")

# Display the employee details

employee.display\_employee\_details()

# Update the years of experience

new\_experience = input("Enter the new years of experience: ")

employee.update\_years\_of\_experience(new\_experience)

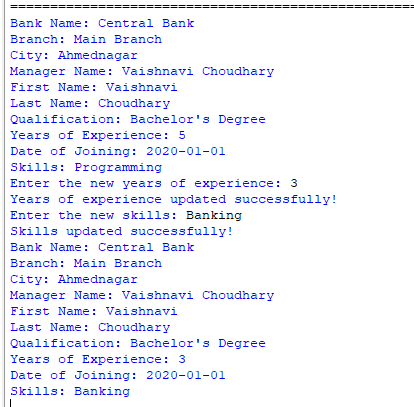
# Update the skills

new\_skills = input("Enter the new skills: ")

employee.update\_skills(new\_skills)

# Display the updated employee details

employee.display\_employee\_details()



**6. Programs for parsing of data, validations like Password, email, URL, etc.**

a) Write a program to validate Password using regular expression. The conditions to be met for valid strong password are:

a. Should have at least one number.

import re

def validate\_password(password):

pattern = r"^(?=.\*\d)(?=.\*[a-z])(?=.\*[A-Z])(?=.\*[@#$%^&+=])(?=.\*[a-zA-Z0-9@#$%^&+=]).{8,20}$"

if re.match(pattern, password):

print("Valid password")

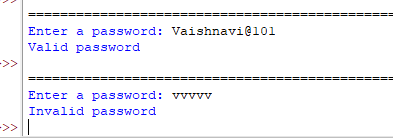
else:

print("Invalid password")

# Example usage:

password = input("Enter a password: ")

validate\_password(password)



**b. Should have at least one uppercase and one lowercase character**.

import re

def has\_upper\_and\_lower(string):

# Define the regex pattern

pattern = r'^(?=.\*[a-z])(?=.\*[A-Z]).+$'

# Use the re.search() function to check if the pattern matches the string

match = re.search(pattern, string)

# Return True if the pattern is found, indicating the presence of both uppercase and lowercase characters

return match is not None

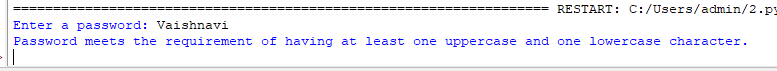
password = input("Enter a password: ")

if has\_upper\_and\_lower(password):

print("Password meets the requirement of having at least one uppercase and one lowercase character.")

else:

print("Password does not meet the requirement of having at least one uppercase and one lowercase character.")



**c. Should have at least one special symbol and**

**d. Should be between 8 to 20 characters long.**

import re

def has\_valid\_password(string):

# Define the regex pattern

pattern = r'^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\W)(?!.\*\s).{8,20}$'

# Use the re.search() function to check if the pattern matches the string

match = re.search(pattern, string)

# Return True if the pattern is found, indicating the password is valid

return match is not None

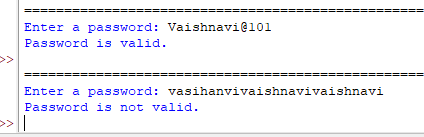
password = input("Enter a password: ")

if has\_valid\_password(password):

print("Password is valid.")

else:

print("Password is not valid.")



**b) Write a program to validate email address using regular expression.**

import re

def validate\_email(email):

pattern = r'^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'

if re.match(pattern, email):

print("Valid email address")

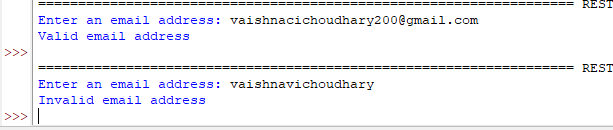
else:

print("Invalid email address")

# Example usage:

email = input("Enter an email address: ")

validate\_email(email)



**c) Write a program to validate URL using regular expression.**

import re

def validate\_url(url):

pattern = r'^(https?|ftp)://[^\s/$.?#].[^\s]\*$'

if re.match(pattern, url):

print("Valid URL")

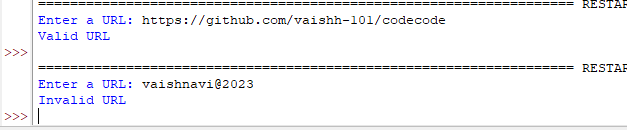
else:

print("Invalid URL")

# Example usage:

url = input("Enter a URL: ")

validate\_url(url)



**d) Write a program to validate PAN card number.**

import re

def validate\_pan\_card(pan):

pattern = r'^[A-Z]{5}[0-9]{4}[A-Z]$'

if re.match(pattern, pan):

print("Valid PAN card number")

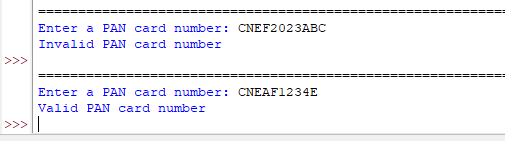
else:

print("Invalid PAN card number")

# Example usage:

pan\_number = input("Enter a PAN card number: ")

validate\_pan\_card(pan\_number)



**7. Programs for Pattern finding should be covered.**

e) Write a program for pattern finding using regular expression. The pattern you need to find is the area code from the given statement. Area code is of 2 to 5 digits, written in parenthesis and followed by few digits.

∙ Input: 'My phone number is (415)5554242 and (023)220404 also'

∙ Output: (415), (023)

import re

def find\_area\_codes(statement):

pattern = r"\(\d{2,5}\)" # Regular expression pattern to match area codes in parentheses

area\_codes = re.findall(pattern, statement)

return area\_codes

# Input statement

statement = 'My phone number is (415)5554242 and (023)220404 also'

# Find area codes

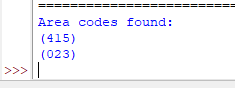
area\_codes = find\_area\_codes(statement)

# Output the area codes

print("Area codes found:")

for code in area\_codes:

print(code)



**f) Extract the user id, domain name and suffix from the following email addresses. ∙ emails = """zuck26@facebook.com**

**page33@google.com**

**jeff42@amazon.com"""**

**∙ desired\_output = [('zuck26', 'facebook', 'com'),**

**('page33', 'google', 'com'),**

**('jeff42', 'amazon', 'com')]**

import re

def extract\_email\_parts(emails):

pattern = r"(\w+)@(\w+).(\w+)"

email\_parts = re.findall(pattern, emails)

return email\_parts

# Input email addresses

emails = """zuck26@facebook.com

page33@google.com

jeff42@amazon.com"""

# Extract email parts

email\_parts = extract\_email\_parts(emails)

# Output the extracted parts

desired\_output = [(user, domain, suffix) for (user, domain, suffix) in email\_parts]

print(desired\_output)



**8. Programs covering all the aspects of Exception handling, user defined exception, Multithreading.**

**a) Write user defined exception program in python which will find the factorial of a number. If number is less than zero it should raise the exception as ‘Invalid Input’.**

class InvalidInputError(Exception):

pass

def factorial(n):

if n < 0:

raise InvalidInputError("Invalid Input: Number cannot be negative")

if n == 0 or n == 1:

return 1

else:

return n \* factorial(n-1)

# Test the factorial function

try:

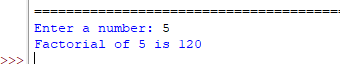
number = int(input("Enter a number: "))

result = factorial(number)

print("Factorial of", number, "is", result)

except InvalidInputError as e:

print(e)



**b) Write a multithread program, where one thread prints square of a number and another thread prints cube of numbers. Also display the total time taken for execution.**

import threading

import time

def calculate\_square(number):

print("Calculating square of", number)

time.sleep(1) # Simulate some computation time

print("Square of", number, "is", number\*\*2)

def calculate\_cube(number):

print("\nCalculating cube of", number)

time.sleep(1) # Simulate some computation time

print("Cube of", number, "is", number\*\*3)

# Start the timer

start\_time = time.time()

# Create and start the threads

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

square\_thread = threading.Thread(target=calculate\_square, args=(number,))

cube\_thread = threading.Thread(target=calculate\_cube, args=(number,))

square\_thread.start()

cube\_thread.start()

# Wait for the threads to finish

square\_thread.join()

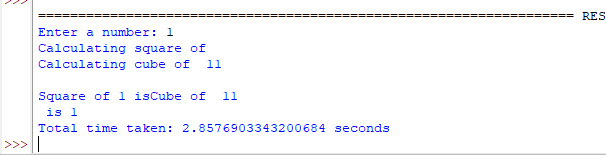
cube\_thread.join()

# Calculate and display the total time taken

end\_time = time.time()

total\_time = end\_time - start\_time

print("Total time taken:", total\_time, "seconds")



**c) Write a program for synchronization of threads using RLock. Accept the two numbers from user and calculate factorial of both numbers simultaneously.**

import threading

import time

def print\_square(number):

for num in number:

square = num \*\* 2

print(f"Square of {num}: {square}")

time.sleep(0.5) # Sleep for 0.5 seconds

def print\_cube(number):

for num in number:

cube = num \*\* 3

print(f"Cube of {num}: {cube}")

time.sleep(0.5) # Sleep for 0.5 seconds

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

numbers = [1, 2, 3, 4, 5]

# Create two threads: one for printing squares and another for printing cubes

square\_thread = threading.Thread(target=print\_square, args=(numbers,))

cube\_thread = threading.Thread(target=print\_cube, args=(numbers,))

# Start the threads

start\_time = time.time()

square\_thread.start()

cube\_thread.start()

# Wait for both threads to finish

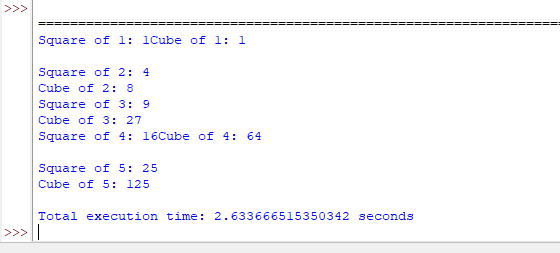
square\_thread.join()

cube\_thread.join()

end\_time = time.time()

total\_time = end\_time - start\_time

print(f"\nTotal execution time: {total\_time} seconds")



**d) You need to implement Hall ticket issue system for MCA CET examination. The system contains three tasks: Verify students details (8 sec), issue hall ticket (4 sec), show rules (2 sec). Task like issue hall ticket and show rules can be done concurrently but only after verifying the student’s details. Write a python program using concept of thread synchronization for the same**.

import threading

import time

def verify\_details():

print("Verifying student's details...")

time.sleep(8) # Simulate verification time

print("Student's details verified.")

def issue\_hall\_ticket():

print("Issuing hall ticket...")

time.sleep(4) # Simulate hall ticket issuing time

print("Hall ticket issued.")

def show\_rules():

print("Showing rules...")

time.sleep(2) # Simulate showing rules time

print("Rules displayed.")

# Create the lock

lock = threading.Lock()

# Create and start the threads

verify\_thread = threading.Thread(target=verify\_details)

issue\_thread = threading.Thread(target=issue\_hall\_ticket)

rules\_thread = threading.Thread(target=show\_rules)

verify\_thread.start()

# Wait for the verification to complete before issuing the hall ticket and showing rules

verify\_thread.join()

# Acquire the lock before proceeding with issuing hall ticket and showing rules

lock.acquire()

issue\_thread.start()

rules\_thread.start()

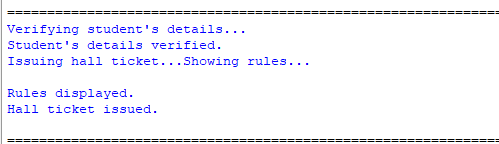
# Release the lock after issuing the hall ticket and showing rules

lock.release()

# Wait for the threads to finish

issue\_thread.join()

rules\_thread.join()



**9. Programs demonstrating the IO operations like reading from file, writing into file from different file types like data file, binary file, etc.**

a) Write a program which will continuously accept some input from the user and write it into the file line by line until user type the word 'exit'.

filename = input("Enter the filename to write: ")

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

while True:

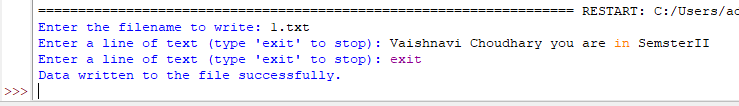
line = input("Enter a line of text (type 'exit' to stop): ")

if line == 'exit':

break

file.write(line + '\n')

print("Data written to the file successfully.")



**b) Write a program to which will read the above file and copy its content into another file.**

source\_filename = input("Enter the source filename: ")

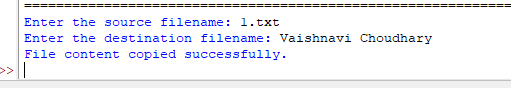
destination\_filename = input("Enter the destination filename: ")

with open(source\_filename, 'r') as source\_file, open(destination\_filename, 'w') as destination\_file:

for line in source\_file:

destination\_file.write(line)

print("File content copied successfully.")



**c) Write a program to write and read a binary file.**

**i**mport struct

filename = input("Enter the filename: ")

# Writing to the binary file

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

data = struct.pack('iidd', 10, 20, 3.14, 5.67)

file.write(data)

print("Binary data written to the file successfully.")

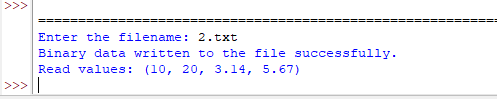
# Reading from the binary file

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

data = file.read()

values = struct.unpack('iidd', data)

print("Read values:", values)



**d) Write a program to create/write and read a csv file.**

import csv

# Create/write a CSV file

filename = input("Enter the filename to create/write: ")

with open(filename, 'w', newline='') as file:

writer = csv.writer(file)

writer.writerow(['Name', 'Age', 'City'])

while True:

name = input("Enter name (type 'exit' to stop): ")

if name == 'exit':

break

age = input("Enter age: ")

city = input("Enter city: ")

writer.writerow([name, age, city])

print("CSV file created/written successfully.")

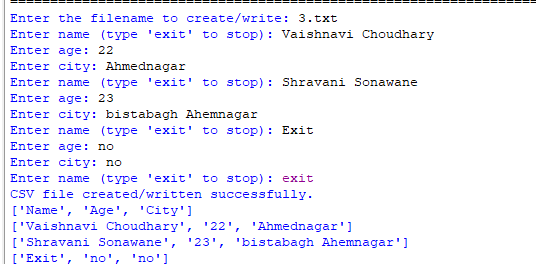
# Read from the CSV file

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

reader = csv.reader(file)

for row in reader:

print(row)



**e) Write a program to create and read a config file.**

import configparser

# Create and write to a config file

config = configparser.ConfigParser()

config['Settings'] = {

'Username': 'Vaishnavi200',

'Password': 'unknown',

'Server': 'vaishnavi200@gmail.com'

}

filename = input("Enter the config filename to create/write: ")

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

config.write(file)

print("Config file created/written successfully.")

# Read from the config file

config = configparser.ConfigParser()

config.read(filename)

username = config['Settings']['Username']

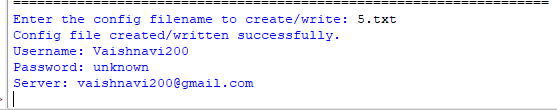
password = config['Settings']['Password']

server = config['Settings']['Server']

print("Username:", username)

print("Password:", password)

print("Server:", server)



**10. Programs to perform searching, adding, updating the content from the file.**

a) Write a program to count the number of lines, words and characters in a given file. Accept the file name from user. Also handle the necessary exceptions.

filename = input("Enter the filename: ")

try:

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

lines = 0

words = 0

characters = 0

for line in file:

lines += 1

words\_list = line.split()

words += len(words\_list)

characters += len(line)

print("Number of lines:", lines)

print("Number of words:", words)

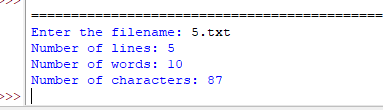
print("Number of characters:", characters)

except FileNotFoundError:

print("File not found.")

except IOError:

print("Error reading the file.")



**b) Write a python program for creating a list of all the longest words in a file. Write all those words, one on each line in another file.**

input\_filename = input("Enter the input filename: ")

output\_filename = input("Enter the output filename: ")

try:

with open(input\_filename, 'r') as input\_file, open(output\_filename, 'w') as output\_file:

longest\_words = []

max\_length = 0

for line in input\_file:

words = line.split()

for word in words:

word\_length = len(word)

if word\_length > max\_length:

max\_length = word\_length

longest\_words = [word]

elif word\_length == max\_length:

longest\_words.append(word)

for word in longest\_words:

output\_file.write(word + '\n')

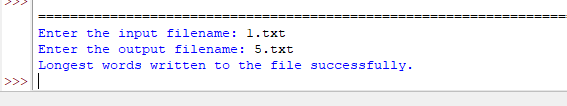
print("Longest words written to the file successfully.")

except FileNotFoundError:

print("Input file not found.")

except IOError:

print("Error reading/writing the file.")



**c) Write a program to update a config file.**

import configparser

filename = input("Enter the config filename to update: ")

config = configparser.ConfigParser()

config.read(filename)

# Update the config values

config['Settings']['Username'] = 'newuser'

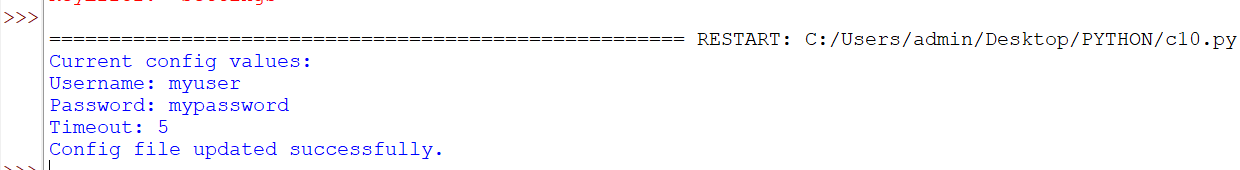
config['Settings']['Password'] = 'newpassword'

# Write the updated config to the file

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

config.write(file)

print("Config file updated successfully.")



**11. Program for performing CRUD operation with MongoDB and Python.**

**a) Write a program to perform following operations on MongoDB database:**

1. **Create a collection “EMPLOYEE” with fields (Id, Name, Department, Salary, Age, Phone, City)**

from pymongo import MongoClient

# Connect to MongoDB

client = MongoClient('mongodb://localhost:27017/')

database = client['company']

collection = database['EMPLOYEE']

# Define the schema for the collection

schema = {

"Id": str,

"Name": str,

"Department": str,

"Salary": float,

"Age": int,

"Phone": str,

"City": str

}

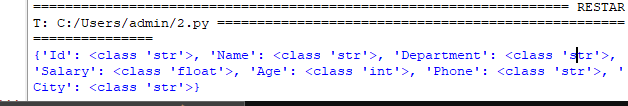
print(schema)

# Create the collection with the defined schema

collection.create\_index("Id", unique=True) # Set Id field as unique index

# Close the connection

client.close()



1. **Insert 10 documents**

**from pymongo import MongoClient**

# Connect to MongoDB

client = MongoClient('mongodb://localhost:27017/')

database = client['company']

collection = database['EMPLOYEE']

# Define the schema for the collection

schema = {

"Id": str,

"Name": str,

"Department": str,

"Salary": float,

"Age": int,

"Phone": str,

"City": str

}

# Create the collection with the defined schema

collection.create\_index("Id", unique=True) # Set Id field as unique index

# Sample records to insert

records = [

{

"Id": "111",

"Name": "Vaishnavi C",

"Department": "Accounts",

"Salary": 5000.0,

"Age": 22,

"Phone": "9988007890",

"City": "India"

},

{

"Id": "122",

"Name": "Vaishnavi Girme",

"Department": "Sales",

"Salary": 50000.0,

"Age": 23,

"Phone": "9923451340",

"City": "India" },

{

"Id": "123",

"Name": "Shravani S",

"Department": "Sales",

"Salary": 9000.0,

"Age": 24,

"Phone": "9973459340",

"City": "India" },

{

"Id": "124",

"Name": "Prasad C",

"Department": "Sales",

"Salary": 4000.0,

"Age": 20,

"Phone": "978456230",

"City": "India"},

{

"Id": "125",

"Name": "Pruthvi C",

"Department": "Purchase",

"Salary": 12000.0,

"Age": 14,

"Phone": "9128769230",

"City": "India"},

{

"Id": "126",

"Name": "Shweta C",

"Department": "Accounts",

"Salary": 6000.0,

"Age": 23,

"Phone": "9700556230",

"City": "India"},

{

"Id": "127",

"Name": "Panda",

"Department": "Sales",

"Salary": 10000.0,

"Age": 2,

"Phone": "978458998",

"City": "India"}

]

print("Records added successfully...")

# Insert the records into the collection

collection.insert\_many(records)

# Close the connection

client.close()

1. **Display all employees in ‘Accounts’ department**

**from pymongo import MongoClient**

# Connect to MongoDB

client = MongoClient('mongodb://localhost:27017/')

database = client['company']

collection = database['EMPLOYEE']

# Find all employees in 'Accounts' department

employees = collection.find({"Department": "Accounts"})

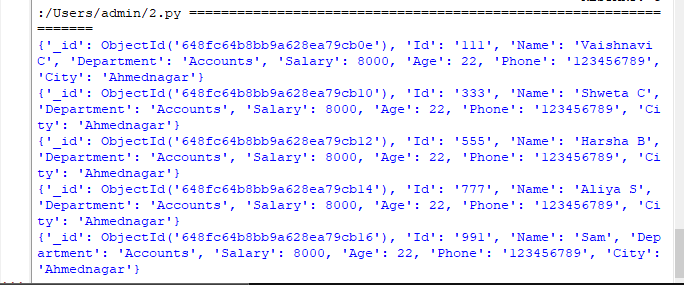
# Display the records

for employee in employees:

print(employee)

# Close the connection

client.close()



d.**Find the employees getting salary between 5000 to 10000**

from pymongo import MongoClient

# Connect to MongoDB

client = MongoClient('mongodb://localhost:27017/')

database = client['company']

collection = database['EMPLOYEE']

# Find employees with salary between 5000 and 10000

employees = collection.find({"Salary": {"$gte": 5000, "$lte": 10000}})

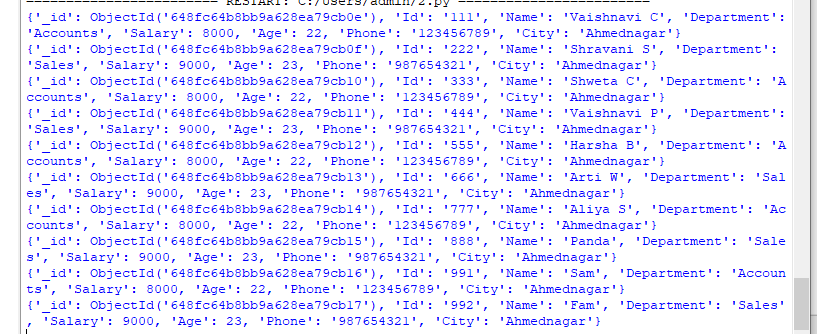
# Display the records

for employee in employees:

print(employee)

# Close the connection

client.close()



**e. Display all the employees in the reverse order of Age**

**from pymongo import MongoClient**

# Connect to MongoDB

client = MongoClient('mongodb://localhost:27017/')

database = client['company']

collection = database['EMPLOYEE']

# Find all employees and sort by Age in descending order

employees = collection.find().sort("Age", -1)

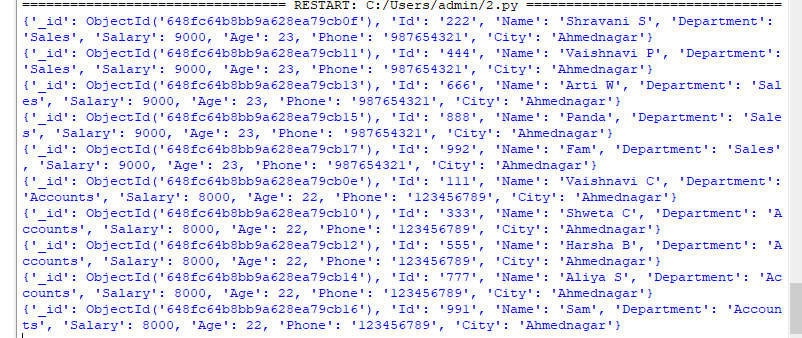
# Display the records in reverse order of Age

for employee in employees:

print(employee)

# Close the connection

client.close()



1. **Update phone number of employee whose ‘Name’ is ‘Panda’**

**from pymongo import MongoClient**

# Connect to MongoDB

client = MongoClient('mongodb://localhost:27017/')

database = client['company']

collection = database['EMPLOYEE']

# Update the phone number of employee with Name 'Sanjay'

collection.update\_one(

{"Name": "Panda"},

{"$set": {"Phone": "9156877859"}}

)

# Print a message indicating the update was successful

print("Phone number updated successfully.")

# Close the connection

client.close()



1. **Delete employee whose ‘Id’ is ‘222’**

**from pymongo import MongoClient**

# Connect to MongoDB

client = MongoClient('mongodb://localhost:27017/')

database = client['company']

collection = database['EMPLOYEE']

# Delete the employee with Id '222'

collection.delete\_one({"Id": "222"})

# Print a message indicating the deletion was successful

print("Employee deleted successfully.")

# Close the connection

client.close()



1. **Display all the employees from ‘Accounts’ department having salary less than 9000.**

from pymongo import MongoClient

# Connect to MongoDB

client = MongoClient('mongodb://localhost:27017/')

database = client['company']

collection = database['EMPLOYEE']

# Query for employees from 'Accounts' department with salary less than 9000

query = {"Department": "Accounts", "Salary": {"$lt": 9000}}

employees = collection.find(query)

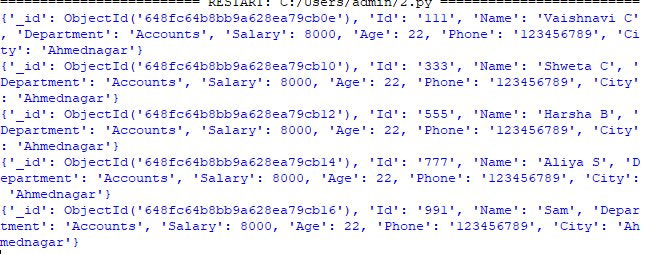
# Display the employees

for employee in employees:

print(employee)

# Close the connection

client.close()



1. **Increase the salary of employee whose id is ‘111’ by 10%**

**from pymongo import MongoClient**

# Connect to MongoDB

client = MongoClient('mongodb://localhost:27017/')

database = client['company']

collection = database['EMPLOYEE']

# Increase the salary of employee with ID '111' by 10%

query = {"Id": "111"}

update = {"$mul": {"Salary": 1.1}} # Multiply the current salary by 1.1 (10% increase)

collection.update\_one(query, update)

# Retrieve the updated employee

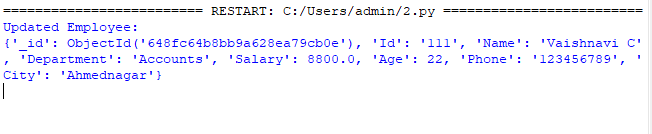
updated\_employee = collection.find\_one(query)

print("Updated Employee:")

print(updated\_employee)

# Close the connection

client.close()



1. **Display all the documents**

**from pymongo import MongoClient**

# Connect to MongoDB

client = MongoClient('mongodb://localhost:27017/')

database = client['company']

collection = database['EMPLOYEE']

# Find and display all documents in the collection

documents = collection.find()

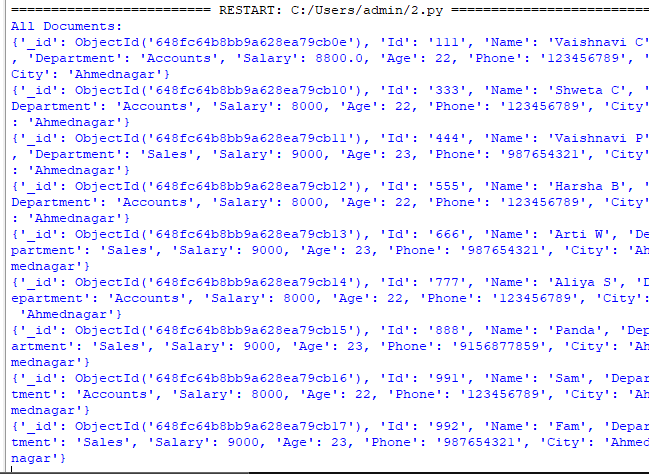
print("All Documents:")

for document in documents:

print(document)

# Close the connection

client.close()



# 12. Basic programs with NumPy as Array, Searching and Sorting, date & time and String handling.

1. **Create a 1D numpy array containing numbers from 1 to 5.**

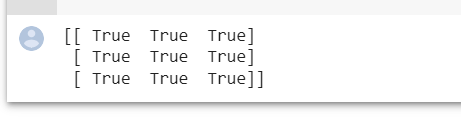
import numpy as np

x = np.arange(1, 6) print(x)



# Create a 3×3 numpy Boolean array of all True’s import numpy as np

x = np.ones((3, 3), dtype=bool) print(x)



# Create two 2x3 numpy array having values in the range of 1 to 6. import numpy as np

# Create the first array

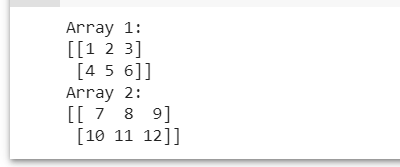
array1 = np.arange(1, 7).reshape(2, 3)

# Create the second array

array2 = np.arange(7, 13).reshape(2, 3)

print("Array 1:") print(array1)

print("Array 2:") print(array2)



# Perform element wise arithmetic operations on the arrays created above in Q3.

import numpy as np

# Arrays from Question 3

array1 = np.arange(1, 7).reshape(2, 3) array2 = np.arange(7, 13).reshape(2, 3)

# Element-wise addition addition = array1 + array2

print("Element-wise addition:") print(addition)

# Element-wise subtraction subtraction = array1 - array2

print("Element-wise subtraction:") print(subtraction)

# Element-wise multiplication multiplication = array1 \* array2

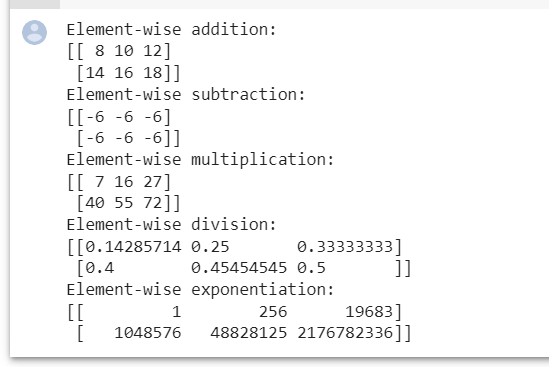
print("Element-wise multiplication:") print(multiplication)

# Element-wise division division = array1 / array2

print("Element-wise division:") print(division)

# Element-wise exponentiation exponentiation = array1 \*\* array2

print("Element-wise exponentiation:")

print(exponentiation)

# Convert any 1D array to a 2D array with exactly 2 rows.

import numpy as np # 1D array

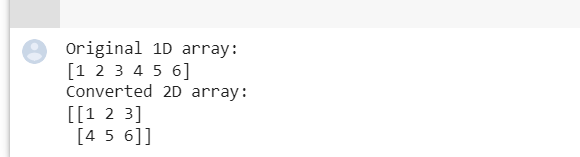
array\_1d = np.array([1, 2, 3, 4, 5, 6])

# Convert to 2D array with 2 rows

array\_2d = np.reshape(array\_1d, (2, -1))

print("Original 1D array:") print(array\_1d)

print("Converted 2D array:") print(array\_2d)



# 6.Convert any 1D array to a 2D array with exactly 2 columns.

**import numpy as np**

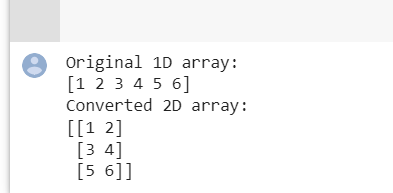
# 1D array

array\_1d = np.array([1, 2, 3, 4, 5, 6])

# Convert to 2D array with 2 columns array\_2d = np.reshape(array\_1d, (-1, 2))

print("Original 1D array:") print(array\_1d)

print("Converted 2D array:") print(array\_2d)

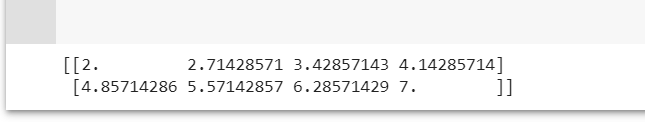


# Create a 2D array containng 8 equally spaced floating point values starting from 2 to 7.

import numpy as np

# Create a 2D array with 8 equally spaced values array\_2d = np.linspace(2, 7, 8).reshape(2, 4)

print(array\_2d)

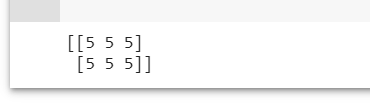


# Create a matrix (numpy array) of dimension 2x3 with each and every value equal to 5.

import numpy as np

# Create a 2x3 matrix with all values equal to 5 matrix = np.full((2, 3), 5)

print(matrix)

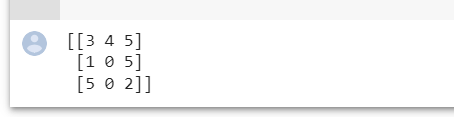


# Create a 3x3 array of random integers between 0 and 5.

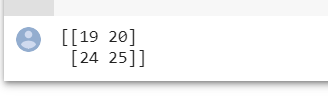
import numpy as np

# Create a 3x3 array of random integers between 0 and 5 array = np.random.randint(0, 6, (3, 3))

print(array)



# Create a 5x5 2D array, and retrieve bottom right corner 2x2 array from it.



import numpy as np

# Create a 5x5 2D array

array\_2d = np.arange(1, 26).reshape(5, 5)

# Retrieve the bottom right corner 2x2 array bottom\_right\_corner = array\_2d[3:5, 3:5]

print(bottom\_right\_corner)

1. **Create a 5x5 2D array, and retrieve middle 3x3 array from it.**

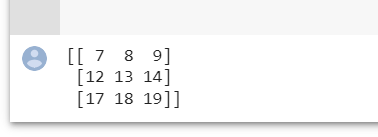
import numpy as np

# Create a 5x5 2D array

array\_2d = np.arange(1, 26).reshape(5, 5)

# Retrieve the middle 3x3 array middle\_3x3 = array\_2d[1:4, 1:4]

print(middle\_3x3**)**



1. **Find the column-wise mean of a two dimensional array of size 4x3.**

import numpy as np

|  |  |  |  |
| --- | --- | --- | --- |
| # Create a  array\_2d = | 4x3 2D array  np.array([[1, | 2, | 3], |
|  | [4, | 5, | 6], |
|  | [7, | 8, | 9], |

[10, 11, 12]])

# Find the column-wise mean

column\_means = np.mean(array\_2d, axis=0) print(column\_means)

1. **Create a 2D array of size 5x5 of random integers. Find the row-wise maximum from this array.**

import numpy as np

# Create a 5x5 2D array of random integers

array\_2d = np.random.randint(low=1, high=100, size=(5, 5))

# Find the row-wise maximum row\_maxima = np.max(array\_2d, axis=1)

print(row\_maxima)



1. **Create a 5x5 2D array, and find the square root of last row and first column.**

import numpy as np

# Create a 5x5 2D array

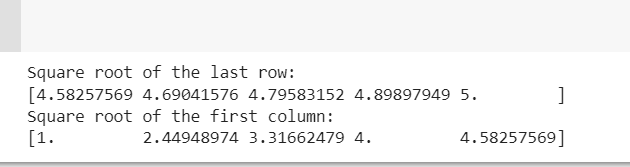
array\_2d = np.arange(1, 26).reshape(5, 5)

# Find the square root of the last row last\_row\_sqrt = np.sqrt(array\_2d[-1, :])

# Find the square root of the first column first\_column\_sqrt = np.sqrt(array\_2d[:, 0])

print("Square root of the last row:") print(last\_row\_sqrt)

print("Square root of the first column:") print(first\_column\_sqrt)



1. **Create an Numpy Array containing elements from 5 to**

**30 but at equal interval of 2. And**

**select the elements which are less than 10.**

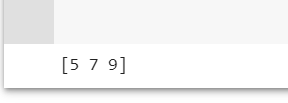
import numpy as np

# Create a NumPy array with elements from 5 to 30 at an interval of 2

array = np.arange(5, 31, 2)

# Select elements less than 10 selected\_elements = array[array < 10]

print(selected\_elements)



1. **Use array of Q15 and select the elements which are divisible by 3**

import numpy as np

# Create a NumPy array with elements from 5 to 30 at an interval of 2

array = np.arange(5, 31, 2)

# Select elements divisible by 3 selected\_elements = array[array % 3 == 0]

print(selected\_elements)



1. **Use array of Q15 and select elements which are greater than 10 and less than 20**

import numpy as np

# Create a NumPy array with elements from 5 to 30 at an interval of 2

array = np.arange(5, 31, 2)

# Select elements greater than 10 and less than 20 selected\_elements = array[(array > 10) & (array < 20)]

print(selected\_elements**) output:**



1. **Use array of Q15 and create a new array by replacing all the elements which are not**

**divisible by 3 with -1**

import numpy as np

# Create a NumPy array with elements from 5 to 30 at an interval of 2

array = np.arange(5, 31, 2)

# Create a new array by replacing elements not divisible by

3 with -1

new\_array = np.where(array % 3 == 0, array, -1)

**print(new\_array)**



1. **Add a new column of ones to the 2D numpy array.**

import numpy as np

|  |  |  |  |
| --- | --- | --- | --- |
| # Existing  array\_2d = | 2D array  np.array([[1, | 2, | 3], |
|  | [4, | 5, | 6], |
|  | [7, | 8, | 9]]) |

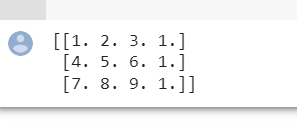
# Create a column of ones with the same number of rows as the existing array

column\_ones = np.ones((array\_2d.shape[0], 1))

# Add the column of ones to the existing array array\_2d\_with\_ones = np.hstack((array\_2d, column\_ones))

print(array\_2d\_with\_ones)

**Outout:**



1. **How to find the position of missing values in numpy array? And also find the number of**

**missing values in numpy array?**

import numpy as np

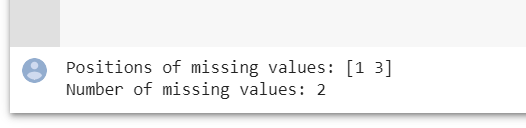
# Example NumPy array with missing values array = np.array([1, np.nan, 3, np.nan, 5])

# Find the position of missing values missing\_value\_positions = np.where(np.isnan(array))[0]

# Find the number of missing values num\_missing\_values = np.isnan(array).sum()

print("Positions of missing values:", missing\_value\_positions)

print("Number of missing values:", num\_missing\_values)



# 13. Programs for series and data frames should be covered.

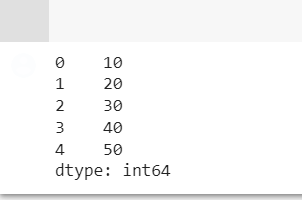
1. How to create a series from a list, numpy array and dict?

import pandas as pd

my\_list = [10, 20, 30, 40, 50]

series\_from\_list = pd.Series(my\_list)

print(series\_from\_list)



|  |  |  |
| --- | --- | --- |
| 2. How | to create a series | with a labled index? Also |
| Assign | name to the series | as well as its index. |
| import | pandas as pd |  |
| data = | [10, 20, 30, 40, 50] | |

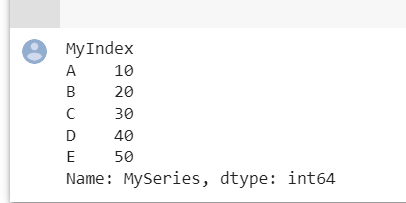
index = ['A', 'B', 'C', 'D', 'E']

# Create a series with labeled index

series = pd.Series(data, index=index, name='MySeries')

# Assign a name to the index series.index.name = 'MyIndex'

print(series)



1. **How to apply arithmetic operators on each and every element of a series?**

import pandas as pd

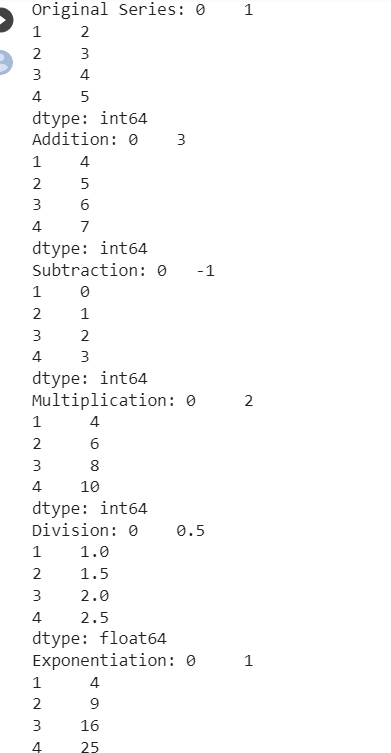
# Example series

series = pd.Series([1, 2, 3, 4, 5])

# Applying arithmetic operators

|  |  |  |
| --- | --- | --- |
| series\_add = series + 2 | # | Addition |
| series\_subtract = series | - | 2 # Subtraction |
| series\_multiply = series | \* | 2 # Multiplication |
| series\_divide = series / | 2 | # Division |
| series\_power = series \*\* | 2 | # Exponentiation |

print("Original Series:", series) print("Addition:", series\_add) print("Subtraction:", series\_subtract) print("Multiplication:", series\_multiply) print("Division:", series\_divide) print("Exponentiation:", series\_power)



1. **How to convert the index of a series into a column of a dataframe?**

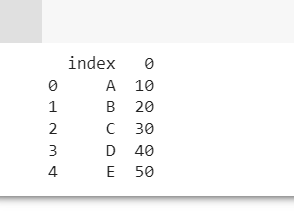
import pandas as pd

# Example series

series = pd.Series([10, 20, 30, 40, 50], index=['A', 'B', 'C', 'D', 'E'])

# Convert index to a column of a DataFrame df = series.reset\_index()

print(df)



1. How to get the items common to both series A and series B?
2. **How to get frequency counts of unique items of a series?**

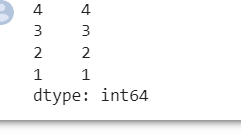
import pandas as pd

# Example Series

series = pd.Series([1, 2, 2, 3, 3, 3, 4, 4, 4, 4])

# Get frequency counts of unique items using value\_counts()

frequency\_counts = series.value\_counts() print(frequency\_counts)



1. **How to convert a pandas series to a dataframe of given shape?**

**Ex.: Reshape the series ser into a dataframe with 7 rows and 5 columns**

import pandas as pd

# Example Series

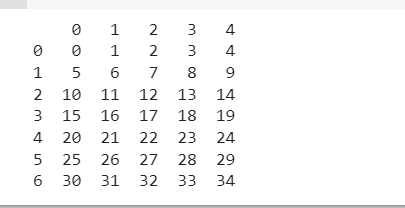
ser = pd.Series(range(35))

# Reshape the series into a DataFrame with 7 rows and

5 columns

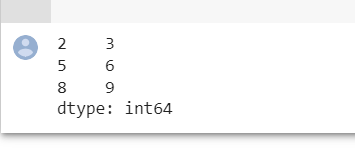
df = pd.DataFrame(ser.values.reshape(7, 5))

print(df)



|  |  |  |
| --- | --- | --- |
| 9. How | to find the numbers that are multiples | of 3 |
| from a  import | series?  pandas as pd |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # Example Series |  | | | |
| series = pd.Series([1, 2, 3, 4, 5, 6, | 7, | 8, | 9, | 10]) |
| # Find the numbers that are multiples multiples\_of\_3 = series[series % 3 == | of 0] | 3 |  |  |
| print(multiples\_of\_3) |  |  |  |  |



* 1. How to find the positions of numbers that are multiples of 3 from a series?

import pandas as pd import numpy as np

# Example Series

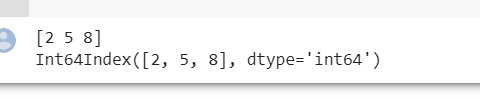
series = pd.Series([1, 2, 3, 4, 5, 6, 7, 8, 9,

10])

# Find the positions of numbers that are multiples of 3 using np.where() positions\_np = np.where(series % 3 == 0)[0]

# Find the positions of numbers that are multiples of 3 using index attribute positions\_index = series[series % 3 == 0].index

print(positions\_np) print(positions\_index)



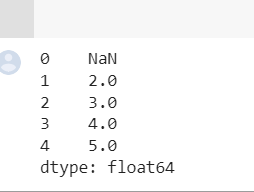
* 1. How to find the difference between the consecutive numbers of a series?

import pandas as pd # Example Series

series = pd.Series([1, 3, 6, 10, 15])

# Find the difference between consecutive numbers diff = series.diff()

print(diff)



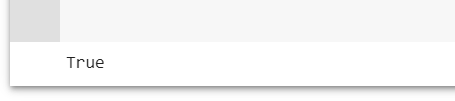
* 1. How to check whether the series contains any missing value?

import pandas as pd # Example Series

series = pd.Series([1, 2, 3, np.nan, 5])

# Check if the series contains any missing value has\_missing = series.isnull().any()

print(has\_missing)



1. Count the number of missing values in a series of Q.12.

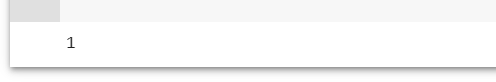
import pandas as pd import numpy as np

# Example Series

series = pd.Series([1, 2, 3, np.nan, 5])

# Count the number of missing values in the series

missing\_count = series.isnull().sum() print(missing\_count)



1. How to replace the missing values in a series with their averages?

import pandas as pd import numpy as np

# Example Series

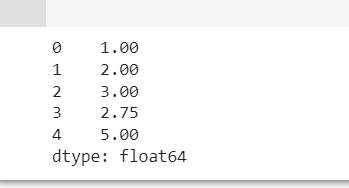
series = pd.Series([1, 2, 3, np.nan, 5])

# Replace missing values with their average

average = series.mean()

series\_filled = series.fillna(average)

print(series\_filled)



1. Make a Pandas DataFrame with two-dimensional list.

import pandas as pd

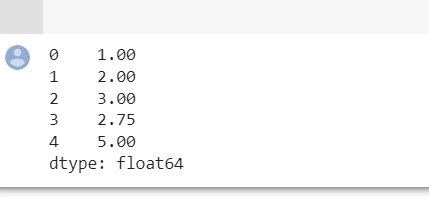
# Example two-dimensional list data = [['John', 25, 'USA'],

['Emily', 30, 'Canada'],

['Michael', 35, 'Australia']]

# Create a DataFrame from the two-dimensional list

df = pd.DataFrame(data) print(df)



* 1. **Programs to demonstrate data pre-processing and data handling**

**with dataframe.**

**We will do this lab assignment as a case study.**

**In this lab assignment we are going to analyze the Customer Churn Data, which will coveralmost all the functions and methods you are likely to use in a typical data analysis process.**

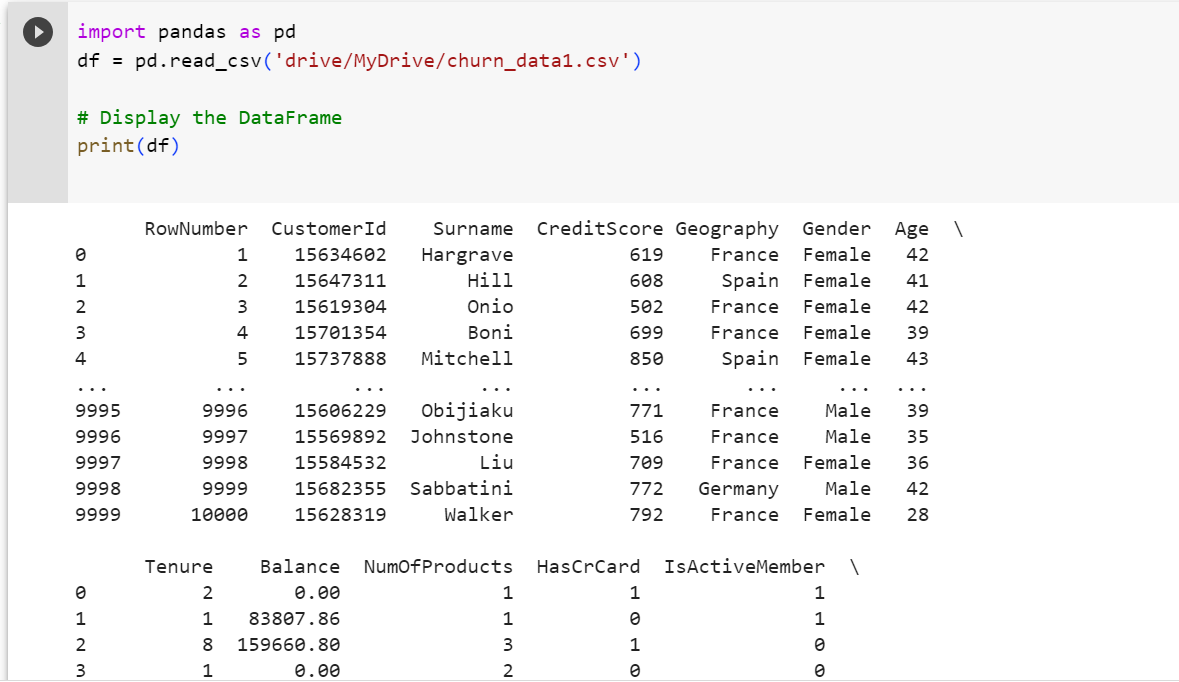
**The csv file to be used for this example is available in Google Classroom by the name:churn\_data1.csv**

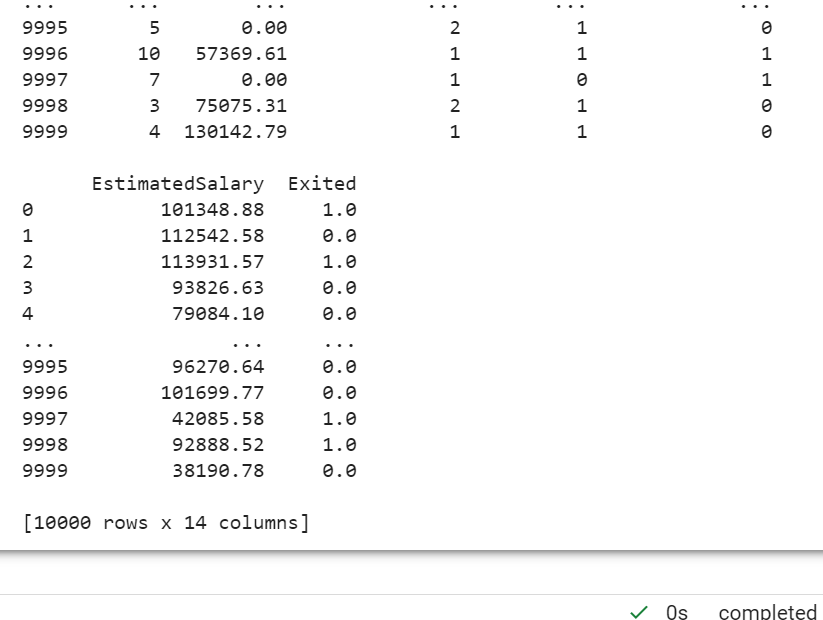
**Customer attrition (a.k.a customer churn) is one of the biggest expenditures of any**

**organization. If we could figure out why a customer leaves and when they leave withreasonable accuracy, it would immensely help the organization to strategize their retention initiatives in multiple ways.**

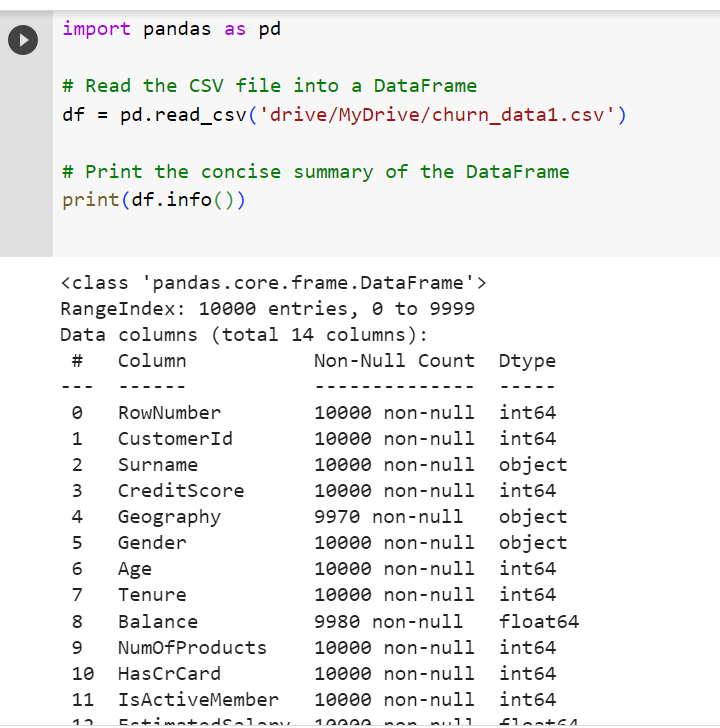
**As a part of the data pre-processing and data handling, you are asked to solve the followingquestions:**

**Read the customer churn data from a churn\_data1.csv file and create a dataframe.**

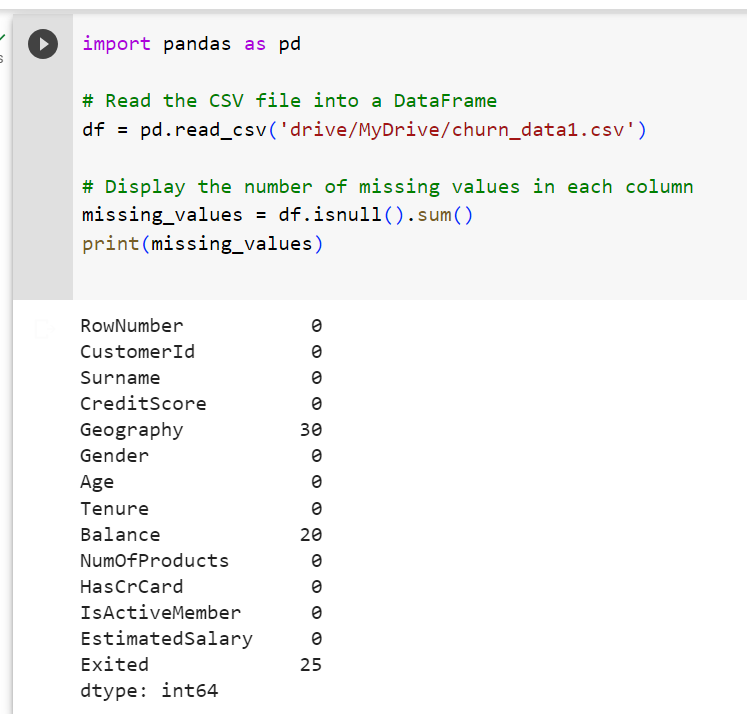




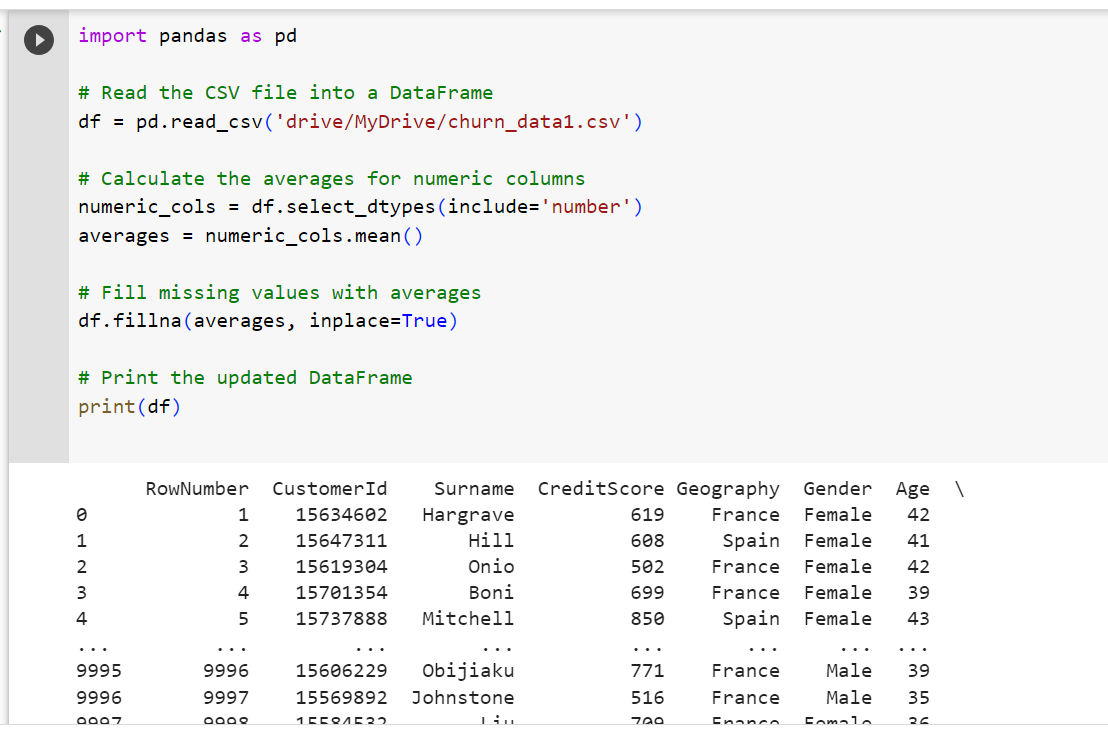
**2.Print a concise summary of a DataFrame.**

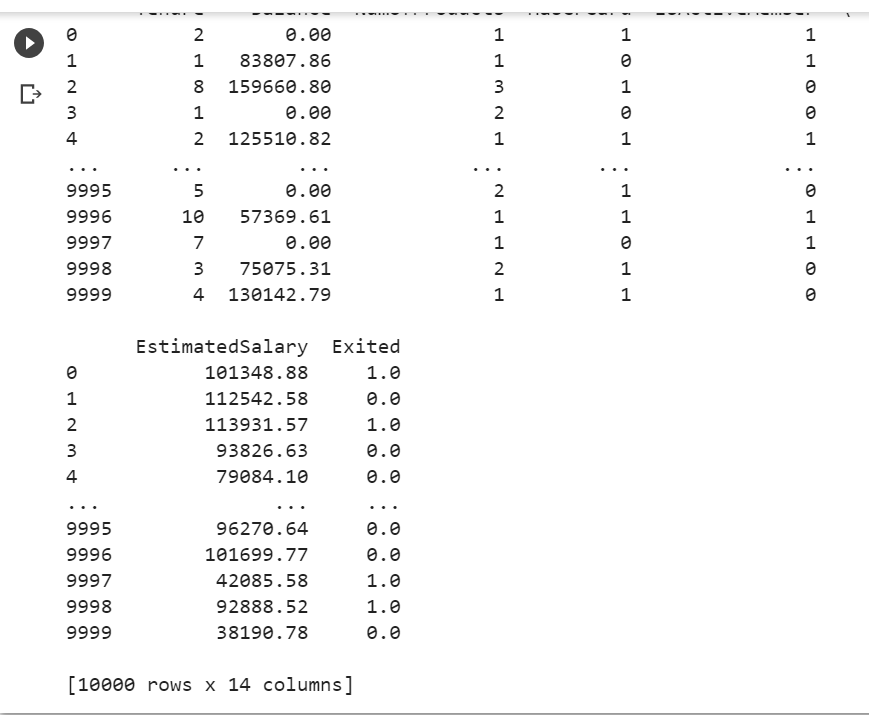


**3.Display the number of missing values in each column.**

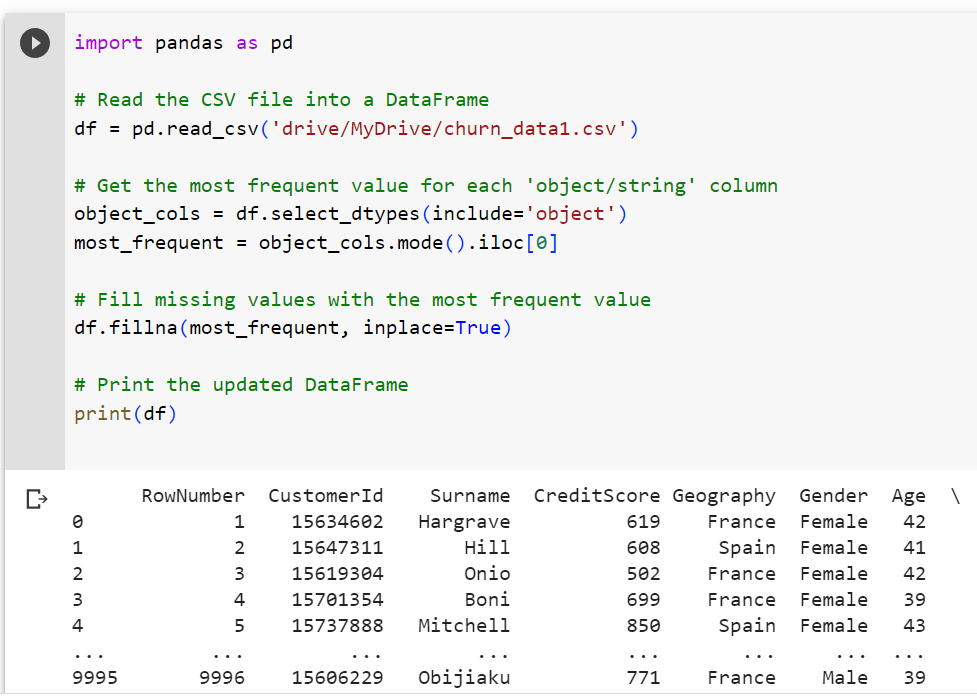


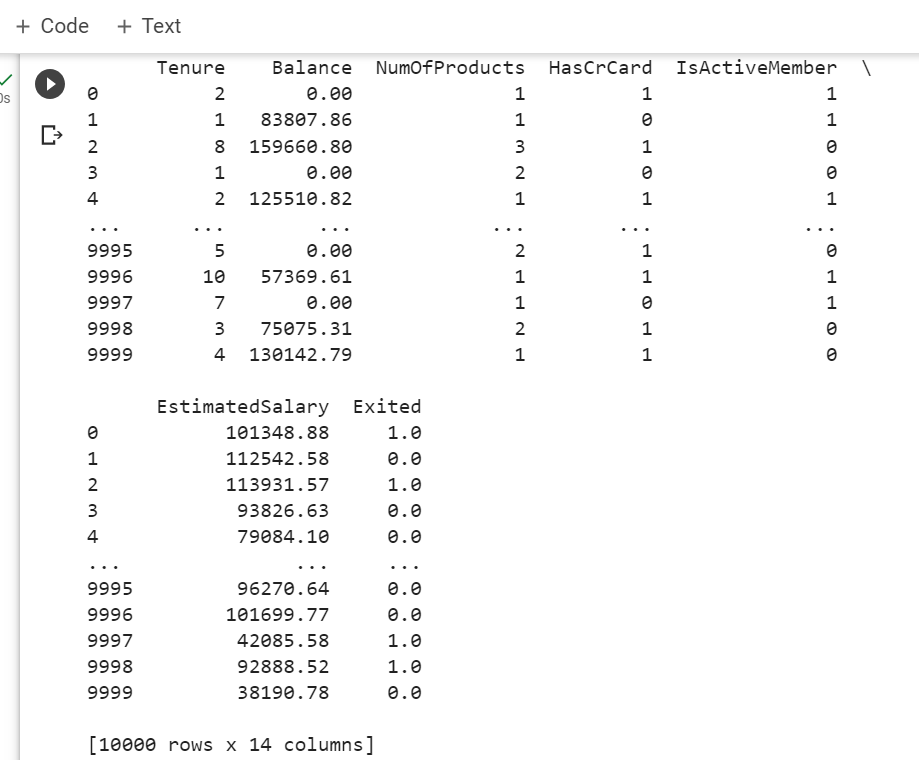
**4. Fill missing values with averages for the numeric columns.**



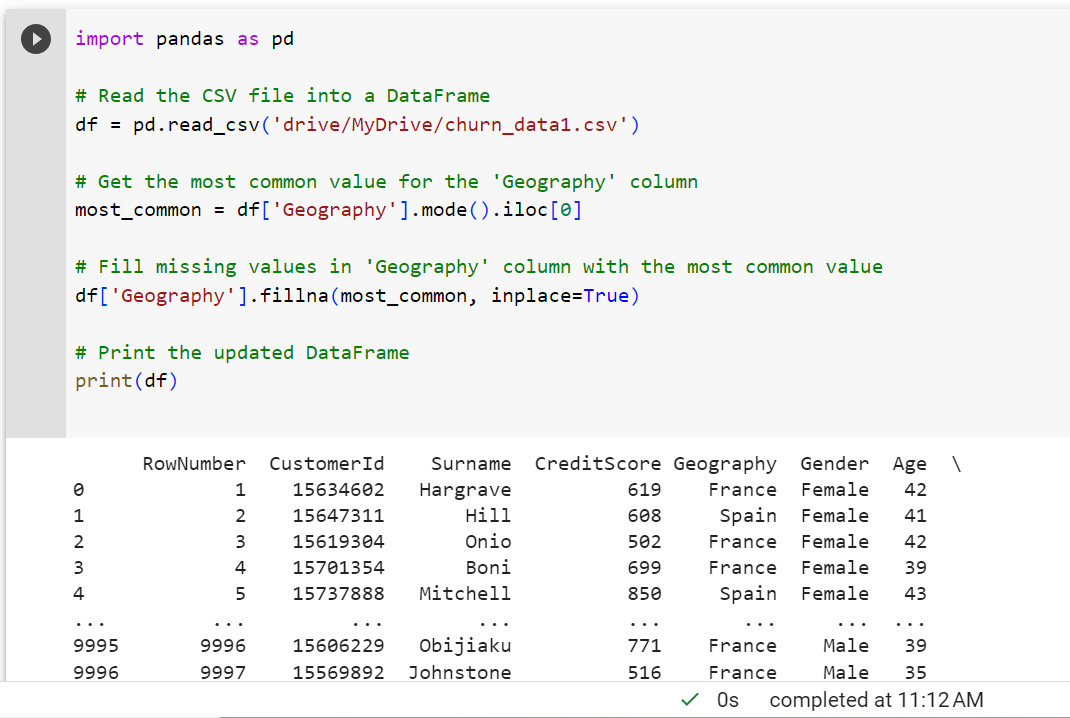


**5. Fill missing values with the most frequent value for 'object/string' columns.**

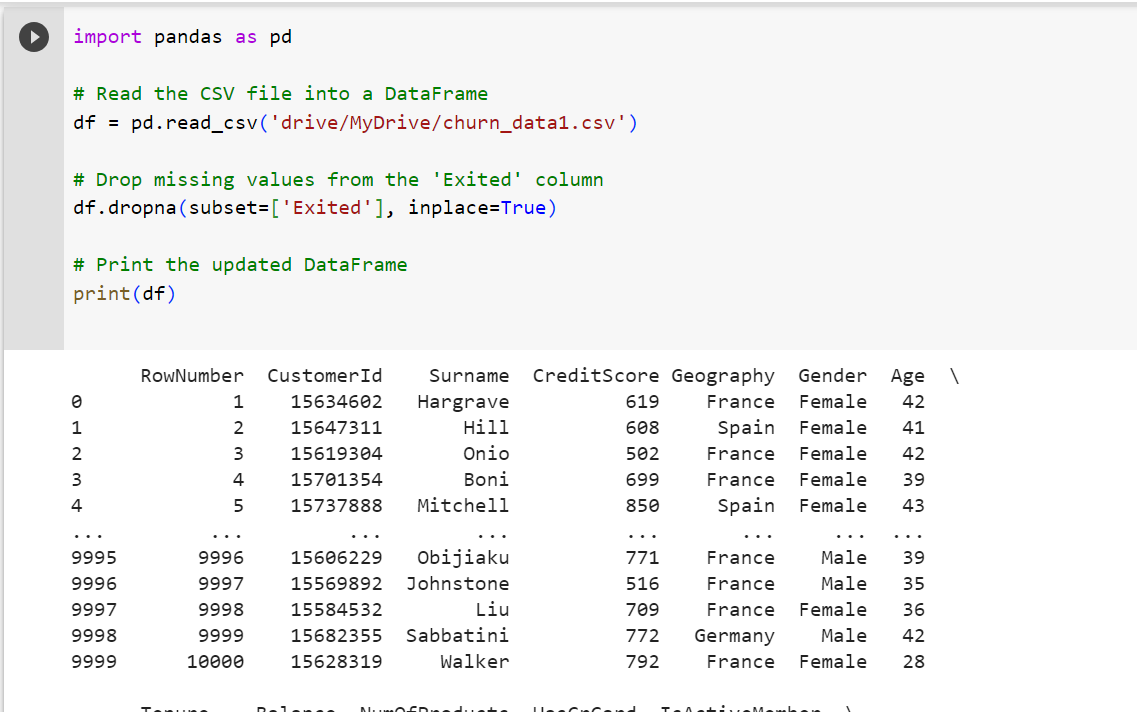




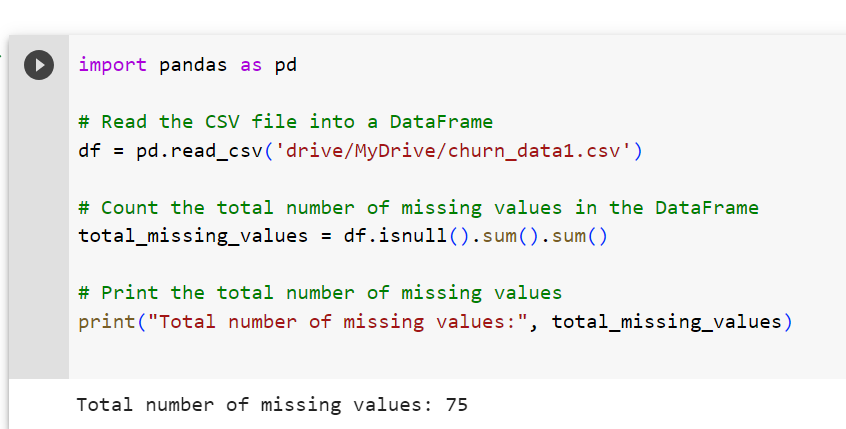
**6. Filling with most common value for ‘category’ like columns. (e.g. Geography column)**



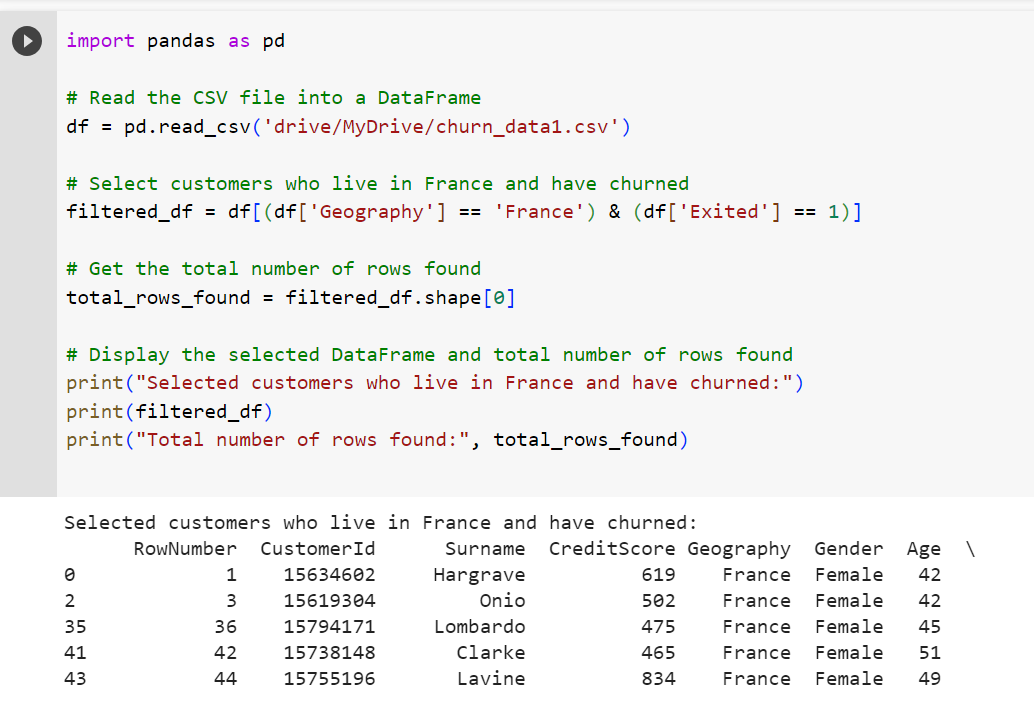
1. **Dropping missing values from a ‘Exited’ column.**

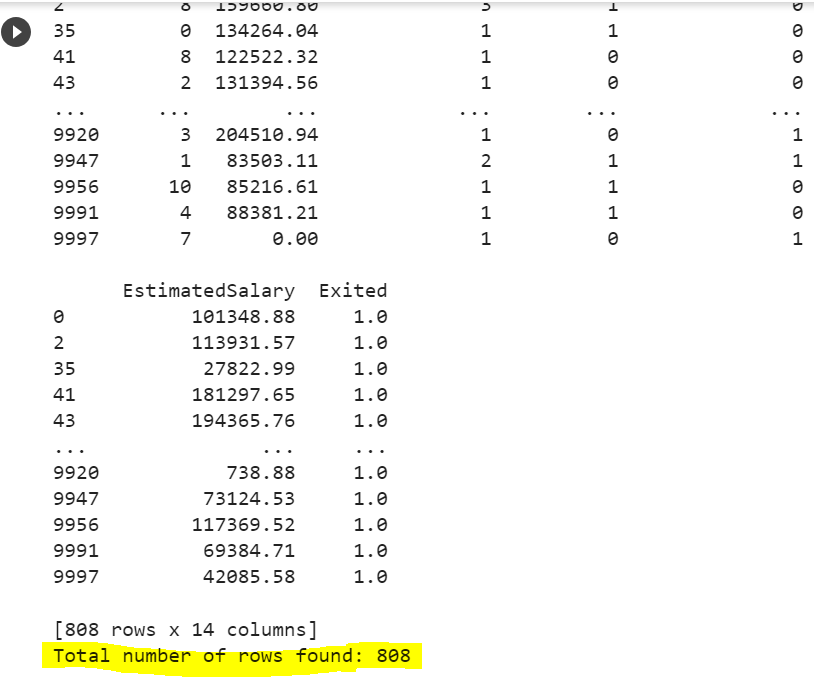


1. **Checking entire dataframe for any missing value. (Display total number of missing values)**

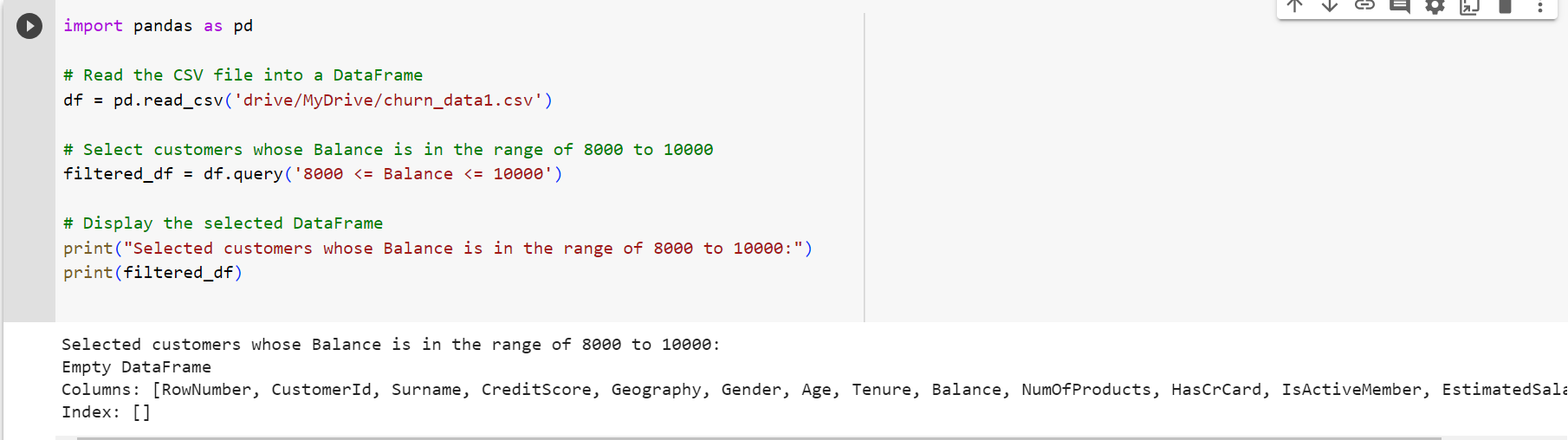


1. **Select customers who live in France and have churned (means Exited = 1). Also displaytotal number of rows found.**

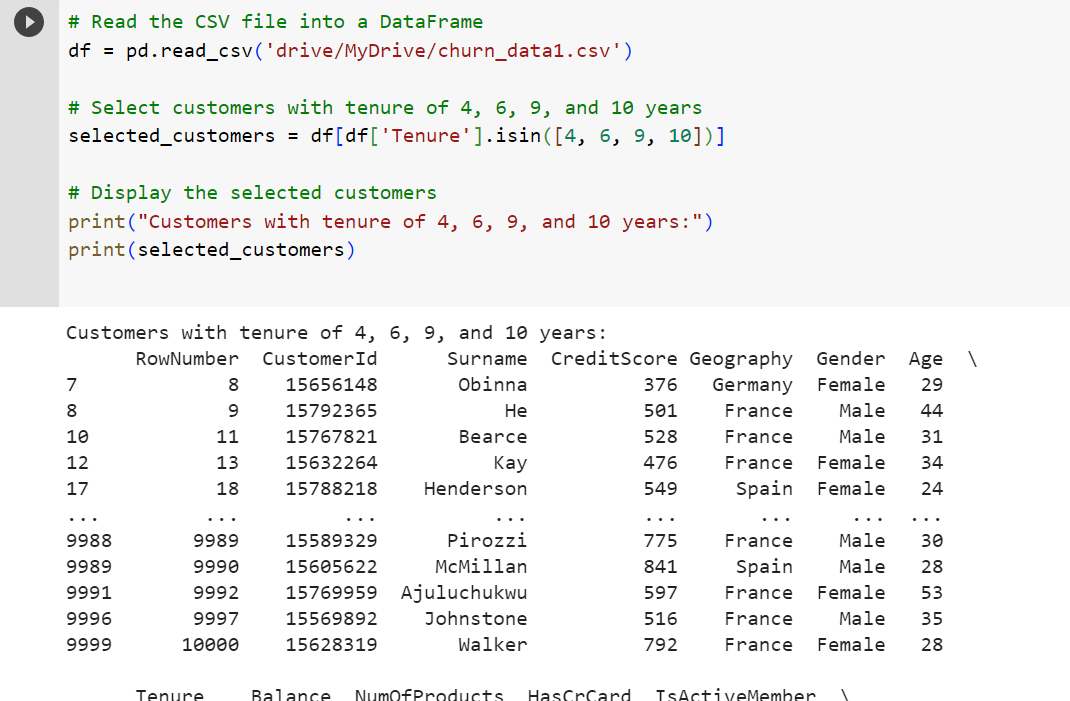




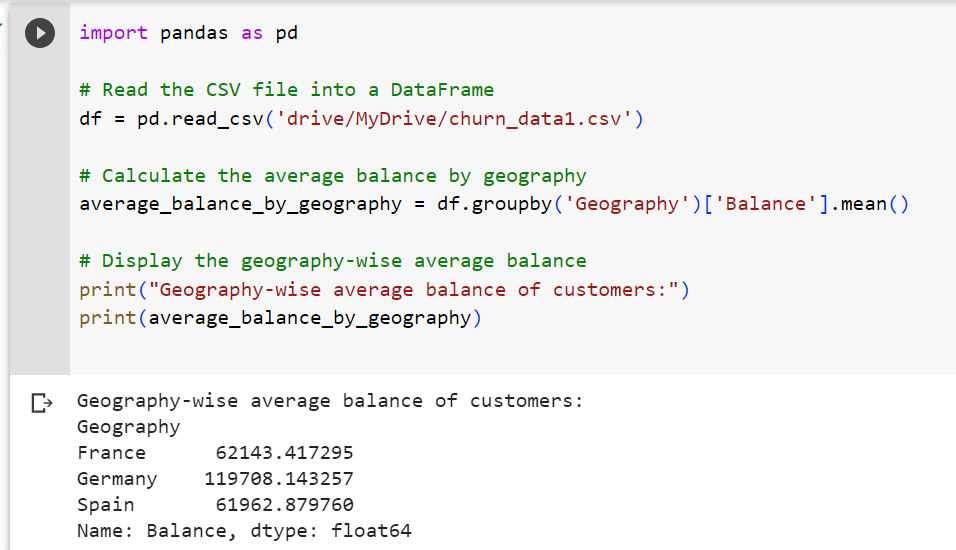
1. **Select customers whose Balance is in the range of 8000 to 10000 using query function.**



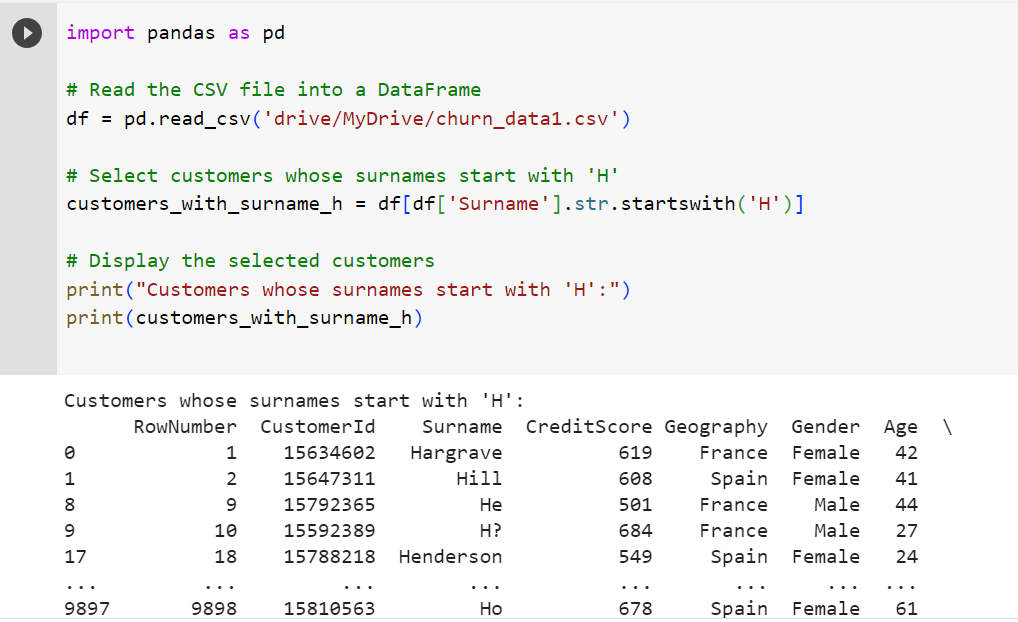
1. **Find the customers opted for tenure of 4, 6, 9, and 10 years only.**

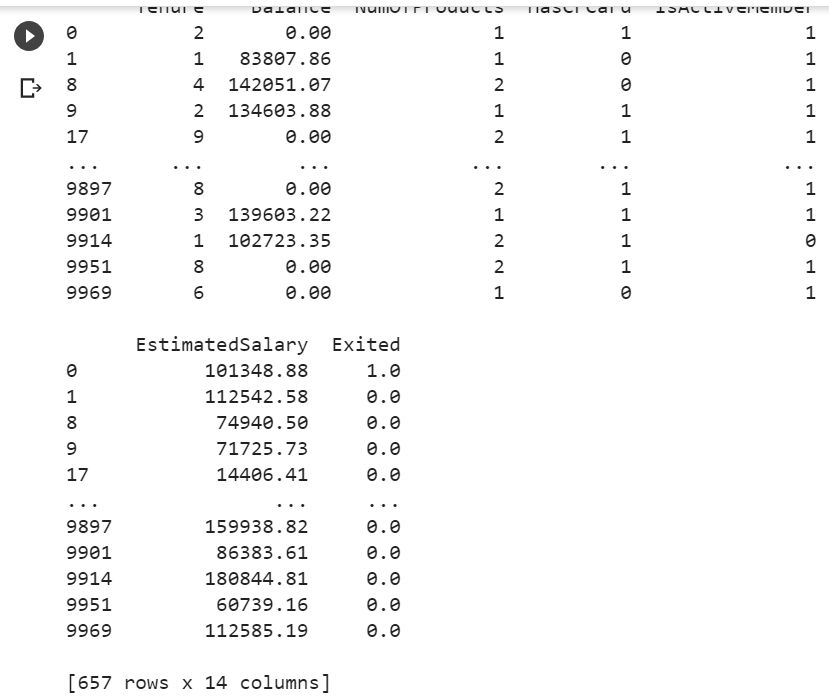


1. **Find the geography-wise average balance of customers**

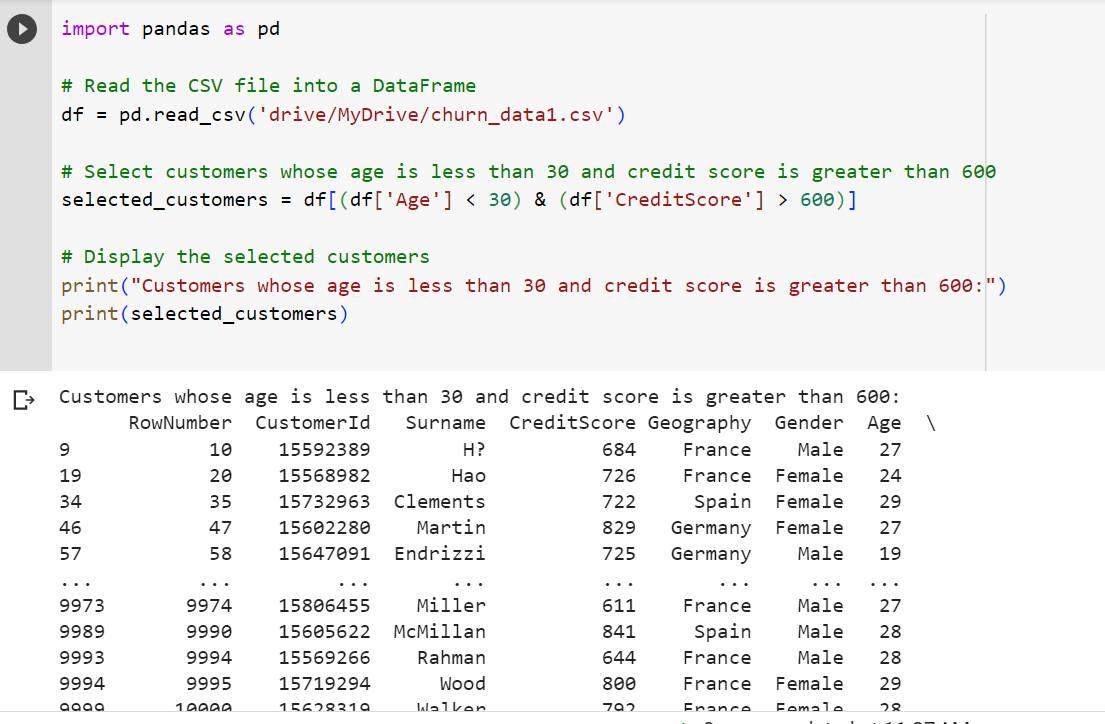


1. **Display customers whose surnames starting with 'H'.**

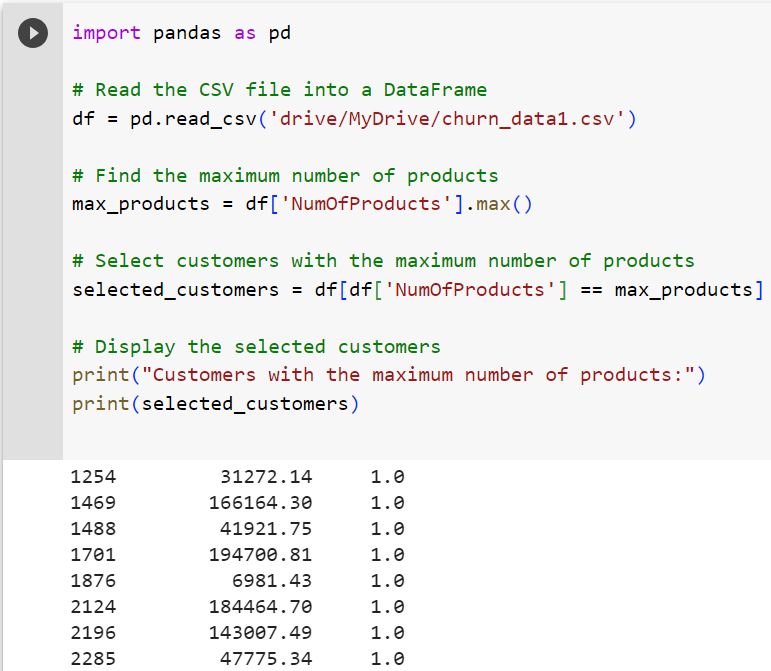


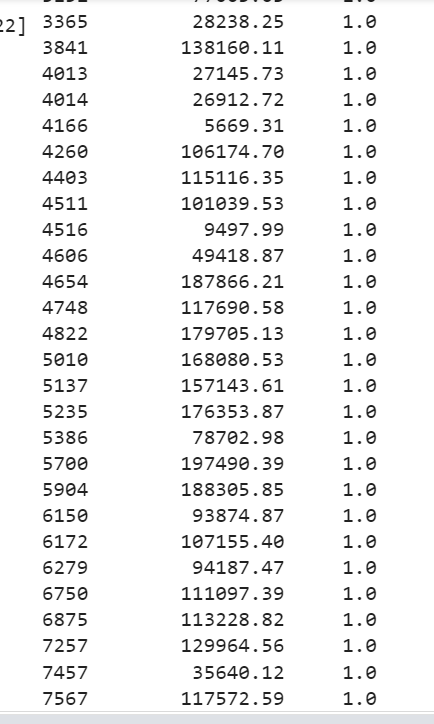


1. **Display customers whose age is < 30 years and credit score is> 600.**



1. **Display customers having maximum number of products.**





**15. Program for data visualization should be covered.**

Draw the following graphs using some sample data (series/dataframe) and decorate them  by configuring various graph elements.

1. Draw a histogram.

import matplotlib.pyplot as plt

import numpy as np

# Generate sample data

data = np.random.normal(0, 1, 1000) # Randomly generated data with a normal distribution

# Create histogram

plt.hist(data, bins=30, color='steelblue', edgecolor='black')

# Configure graph elements

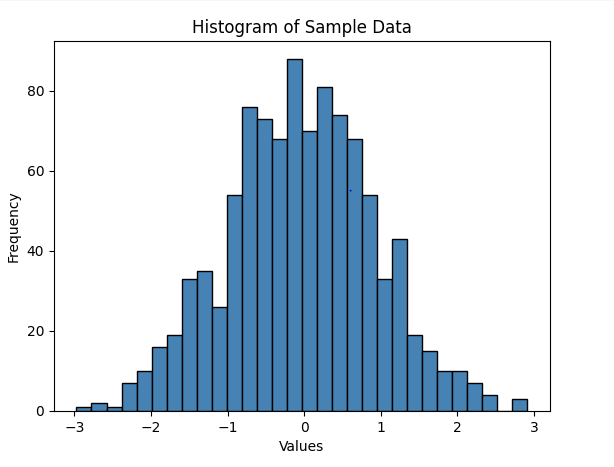
plt.title("Histogram of Sample Data")

plt.xlabel("Values")

plt.ylabel("Frequency")

# Display the histogram

plt.show()



1. Draw a scatter plot.

import matplotlib.pyplot as plt

import numpy as np

# Generate sample data

x = np.random.normal(0, 1, 100) # Randomly generated x-coordinates

y = np.random.normal(0, 1, 100) # Randomly generated y-coordinates

# Create scatter plot

plt.scatter(x, y, color='steelblue', edgecolor='black')

# Configure graph elements

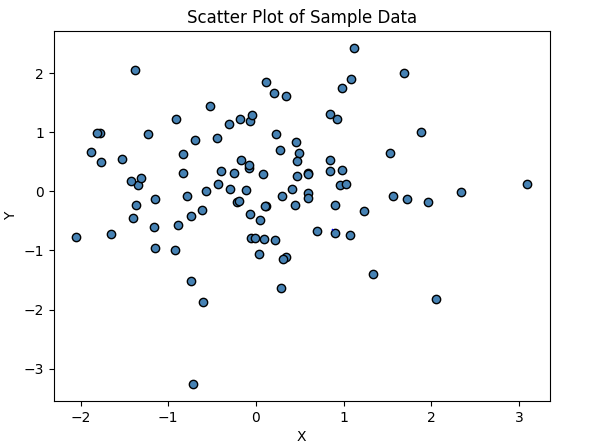
plt.title("Scatter Plot of Sample Data")

plt.xlabel("X")

plt.ylabel("Y")

# Display the scatter plot

plt.show()



1. Draw a Pie chart.

import matplotlib.pyplot as plt

# Sample data

labels = ['Apple', 'Banana', 'Orange', 'Mango']

sizes = [30, 20, 15, 35]

colors = ['red', 'yellow', 'orange', 'green']

explode = [0, 0, 0.1, 0] # Explode the third slice (Orange) by 0.1

# Create pie chart

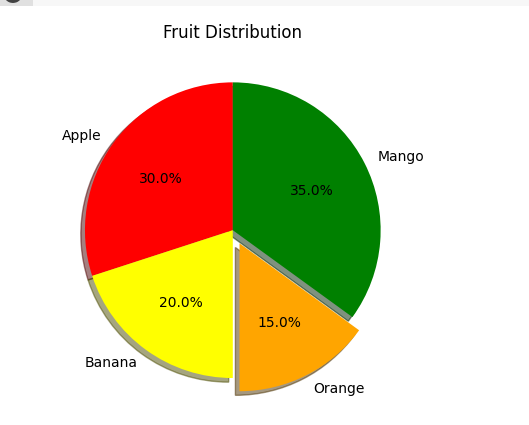
plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%', shadow=True, startangle=90)

# Configure graph elements

plt.title("Fruit Distribution")

# Display the pie chart

plt.show()



1. Draw a multi-series bar graph

 import matplotlib.pyplot as plt

import numpy as np

# Sample data

categories = ['Category A', 'Category B', 'Category C']

series1 = [20, 35, 30]

series2 = [25, 32, 28]

series3 = [18, 29, 36]

# Define the bar width

bar\_width = 0.2

# Set the positions of the bars on the x-axis

x = np.arange(len(categories))

# Create the bar graph

plt.bar(x - bar\_width, series1, width=bar\_width, label='Series 1')

plt.bar(x, series2, width=bar\_width, label='Series 2')

plt.bar(x + bar\_width, series3, width=bar\_width, label='Series 3')

# Configure graph elements

plt.xlabel('Categories')

plt.ylabel('Values')

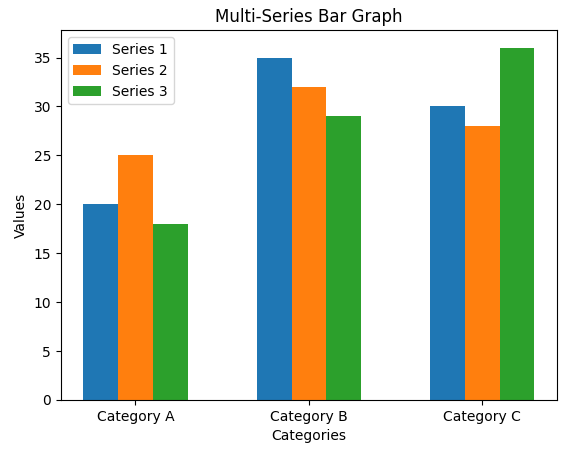
plt.title('Multi-Series Bar Graph')

plt.xticks(x, categories)

plt.legend()

# Display the bar graph

plt.show()



1. Draw a line graph.

import matplotlib.pyplot as plt

import numpy as np

# Sample data

x = np.linspace(0, 10, 100) # X-axis values

y = np.sin(x) # Y-axis values (sine function)

# Create line graph

plt.plot(x, y, color='steelblue', linestyle='-', linewidth=2, marker='o', markersize=4)

# Configure graph elements

plt.title("Line Graph of Sine Function")

plt.xlabel("X")

plt.ylabel("Y")

# Display the line graph

plt.show()

