HINDUSTAN
INSTITUTE OF TECHNOLOGY & SCIENCE
(DEEMED TO BE UNIVERSITY)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LABORATORY RECORD

STUDENT NAME	: ………………………………..
REGISTER NUMBER	: ………………………………..
SUBJECT CODE	: EAD51001
SUBJECT NAME	: Python for Data Science

HINDUSTAN
INSTITUTE OF TECHNOLOGY & SCIENCE
(DEEMED TO BE UNIVERSITY)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

REGISTRATION NUMBER:………………………………………….

Certified that this is a Bonafide record work done ……………………………………… of B.Tech Computer Science and Engineering in specialisation with in Artificial Intelligence and Data Science, for the course EAD51001 Python for Data Science Lab in the second semester during the period August 2023 to November 2023 at Hindustan Institute of Technology and Science, Chennai, 603103.

Faculty InCharge

This record is submitted for the practical examination held on……………….

Signature of the Internal Examiner Signature of the External Examiner

INDEX

Creating a New String from User Input

Exp. No. : 01

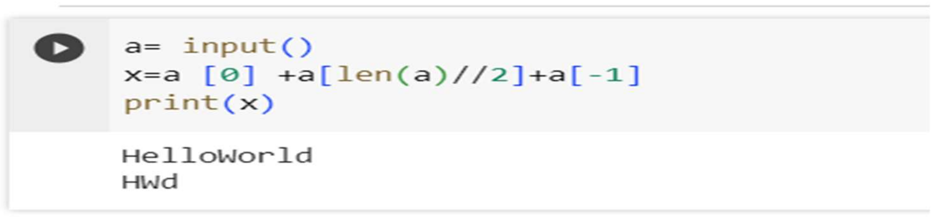
Date :

Aim : To Write a Python Program to Creating a New String from User Input.

Algorithm :

- Start the program.
- Accept user input and store it in the variable 'a'.
- Create a new string 'x' by selecting characters from the input string 'a':
    - The first character is selected using 'a[0]'.
    - The middle character is obtained by using 'a[len(a)//2]'. For strings with an even length, this selects the character to the left of the exact center.
    - The last character is selected using 'a[-1]'.
- Print the new string 'x', which is a combination of the first, middle, and last characters of the input.
- End the program.

Source Code & Output :



```
a= input()
x=a [0] +a[len(a)//2]+a[-1]
print(x)
```

```
HelloWorld
HWd
```

Result :

The Python program for Creating a New String from User Input is compiled & executed successfully.

Teacher's signature

Arrange the characters of a string so that all lowercase letters come first

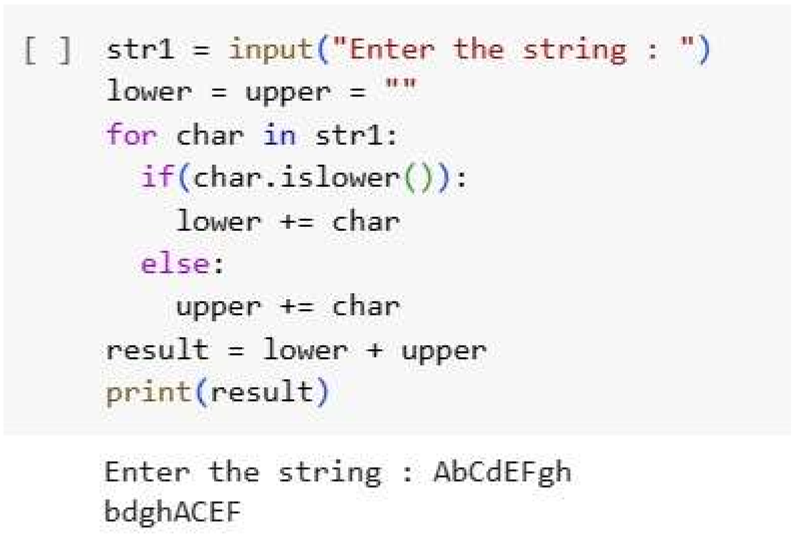
Exp. No. : 02

Aim : To write a program to arrange the characters of a string so that all lowercase letters come first

Algorithm :

- Start the Program
- Get string from user input
- Search for all lowercase letters and append to list ‘lower’
- Search for all uppercase letters and append to list ‘upper’
- Combine list lower and list upper as result
- Print result
- Stop the Program

Source Code & Output:



```
[ ]  str1 = input("Enter the string : ")
     lower = upper = ""
     for char in str1:
       if(char.islower()):
         lower += char
       else:
         upper += char
     result = lower + upper
     print(result)
```

```
Enter the string : AbCdEFgh
bdghACEF
```

Result :

The Python program for arranging the characters of a string so that all lowercase letters come first is compiled & executed successfully.

Teacher’s signature

Count all letters, digits and special characters from a given string

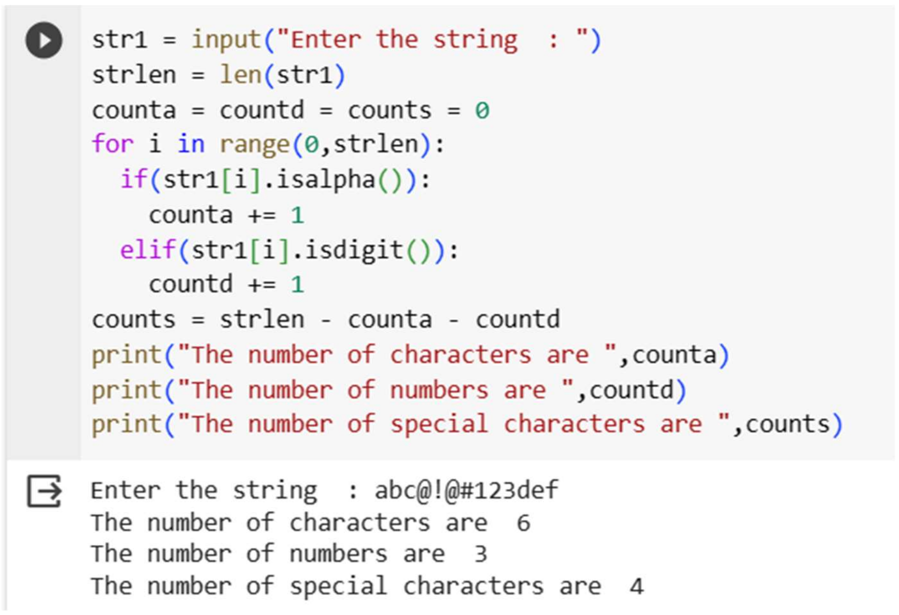
Exp. No. : 03

Aim : To write a program to count all letters, digits and special characters from a given string.

Algorithm :

- Start the Program
- Get string from user
- Count number of letters, digits and special characters
- Print each count
- Stop the Program

Source Code & Output:



```
str1 = input("Enter the string  : ")
strlen = len(str1)
counta = countd = counts = 0
for i in range(0,strlen):
  if(str1[i].isalpha()):
    counta += 1
  elif(str1[i].isdigit()):
    countd += 1
counts = strlen - counta - countd
print("The number of characters are ",counta)
print("The number of numbers are ",countd)
print("The number of special characters are ",counts)
```

```
Enter the string  : abc@!@#123def
The number of characters are  6
The number of numbers are  3
The number of special characters are  4
```

Result :

The Python program for counting all letters, digits and special characters from a given string is compiled & executed successfully.

Teacher's signature

Find all occurrences of “USA” in a given string, ignoring the case

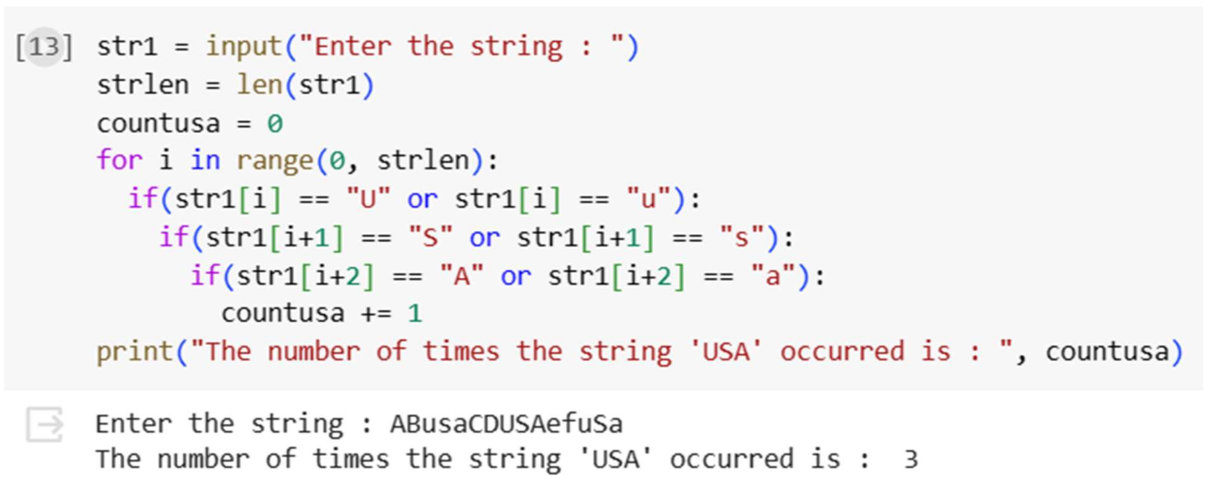
Exp. No. : 04

Aim : To write a program to find all occurrences of “USA” in a given string, ignoring the case.

Algorithm :

- Start the Program
- Get string input
- Check if USA is present in the string, in lower or upper case
- Count the number of occurrences
- Stop the Program

Source Code & Output:



```
[13] str1 = input("Enter the string : ")
     strlen = len(str1)
     countusa = 0
     for i in range(0, strlen):
       if(str1[i] == "U" or str1[i] == "u"):
         if(str1[i+1] == "S" or str1[i+1] == "s"):
           if(str1[i+2] == "A" or str1[i+2] == "a"):
             countusa += 1
     print("The number of times the string 'USA' occurred is : ", countusa)
```

```
Enter the string : ABusaCDUSAefuSa
The number of times the string 'USA' occurred is :  3
```

Result :

The Python program for finding all occurrences of “USA” in a given string is compiled & executed successfully.

Teacher’s signature

Sum and Average of Digits of a String

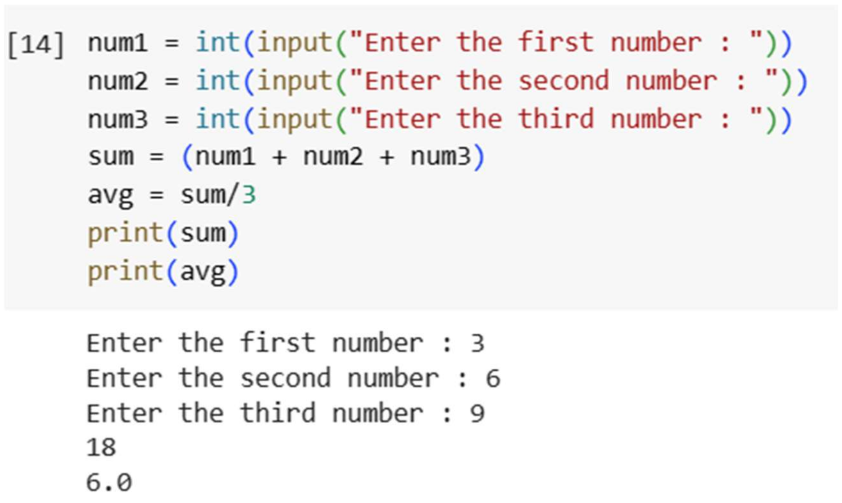
Exp. No. : 05

Aim : To write a program to find the sum and average of three numbers

Algorithm :

- Start the Program
- Get 3 numbers
- Find sum of the numbers
- Divide the sum by 3 to find average
- Print sum, average
- Stop the Program

Source Code & Output :



```
[14] num1 = int(input("Enter the first number : "))
     num2 = int(input("Enter the second number : "))
     num3 = int(input("Enter the third number : "))
     sum = (num1 + num2 + num3)
     avg = sum/3
     print(sum)
     print(avg)
```

```
Enter the first number : 3
Enter the second number : 6
Enter the third number : 9
18
6.0
```

Result :

The Python program for finding the sum and average of three numbers is compiled & executed successfully.

Teacher's signature

Combining Lists of Strings

Exp. No. : 06

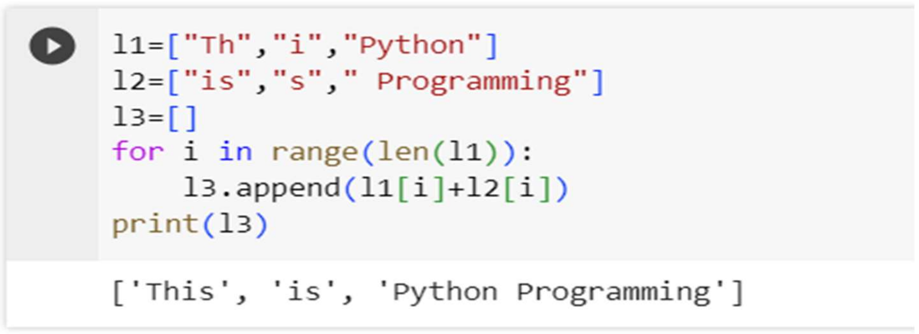
Date :

Aim : To Write a Python Program to Combining List of Strings.

Algorithm :

- Start the program.
- Create two lists, 'l1' and 'l2', where each list contains individual string elements.
- Initialize an empty list 'l3' to store the combined strings.
- Use a 'for' loop to iterate through the indices of the lists. For each index 'i' from 0 to the length of 'l1' (non-inclusive), do the following:
    - Concatenate the string elements at the same index from 'l1' and 'l2'.
    - Append the concatenated string to the 'l3' list.
- Print the 'l3' list, which contains the combined strings.
- End the program.

Source Code & Output :



```
l1=["Th","i","Python"]
l2=["is","s"," Programming"]
l3=[]
for i in range(len(l1)):
    l3.append(l1[i]+l2[i])
print(l3)
```

```
['This', 'is', 'Python Programming']
```

Result :

The Python program forCombining List of Strings is compiled & executed successfully.

Teacher's signature

Adding an Element to a Nested List

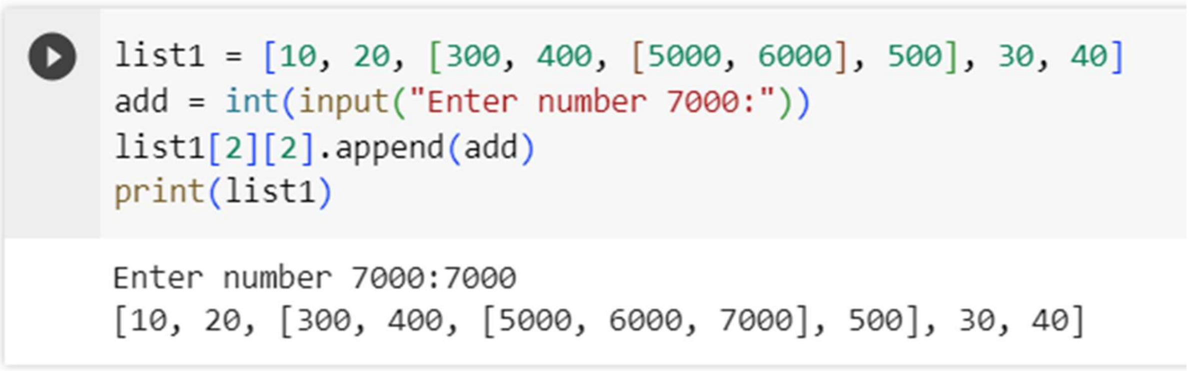
Exp. No. : 07

Aim : To write a program to Adding an Element to a Nested List.

Algorithm :

- Start the program.
- Define a nested list 'list1', which contains integers and sublists.
- Accept user input to enter a number, stored in the variable 'add'. The user is prompted to enter the number 7000.
- Append the value of 'add' to the innermost sublist in 'list1'. Specifically, append 'add' to the third element of 'list1' (index 2), the third element of the inner sublist (index 2), resulting in the addition of 'add' to the list [5000, 6000].
- Print the modified 'list1', which now includes the added value 'add'.
- End the program.

Source Code & Output :



```python
list1 = [10, 20, [300, 400, [5000, 6000], 500], 30, 40]
add = int(input("Enter number 7000:"))
list1[2][2].append(add)
print(list1)
```

```
Enter number 7000:7000
[10, 20, [300, 400, [5000, 6000, 7000], 500], 30, 40]
```

Result :

The Python program for Adding an Element to a Nested List is compiled & executed successfully.

Teacher's signature

Merging Two Dictionaries in Python

Exp. No. : 08

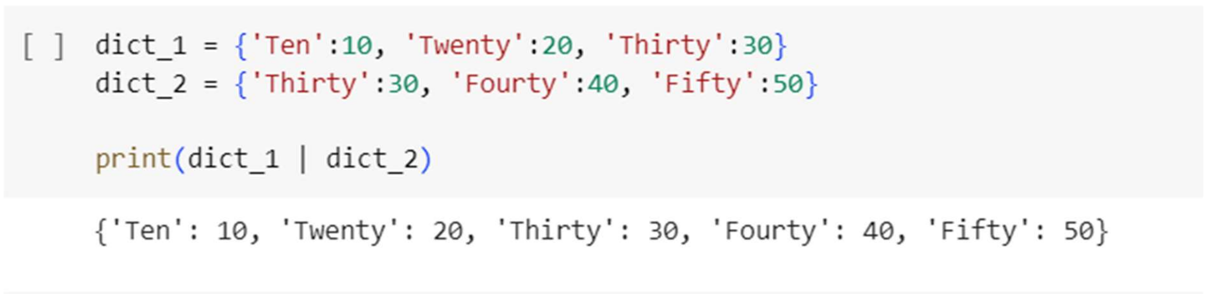
Date :

Aim : To Write a Python Program for Merging Two Dictionaries in Python

Algorithm :

- Start the program.
- Create two dictionaries, 'dict_1' and 'dict_2', with key-value pairs.
- Use the '|' operator to merge 'dict_1' and 'dict_2'. This operator combines the key-value pairs from both dictionaries into a new merged dictionary.
- Print the merged dictionary, which contains all the key-value pairs from 'dict_1' and 'dict_2.
- End the program.

Source Code & Output :



```
[ ] dict_1 = {'Ten':10, 'Twenty':20, 'Thirty':30}
    dict_2 = {'Thirty':30, 'Fourty':40, 'Fifty':50}

    print(dict_1 | dict_2)
```

```
{'Ten': 10, 'Twenty': 20, 'Thirty': 30, 'Fourty': 40, 'Fifty': 50}
```

Result :

The Python program for Creating a New String from User Input is compiled & executed successfully.

Teacher's signature

Extracting Specific Keys from a Dictionary

Exp. No. : 09

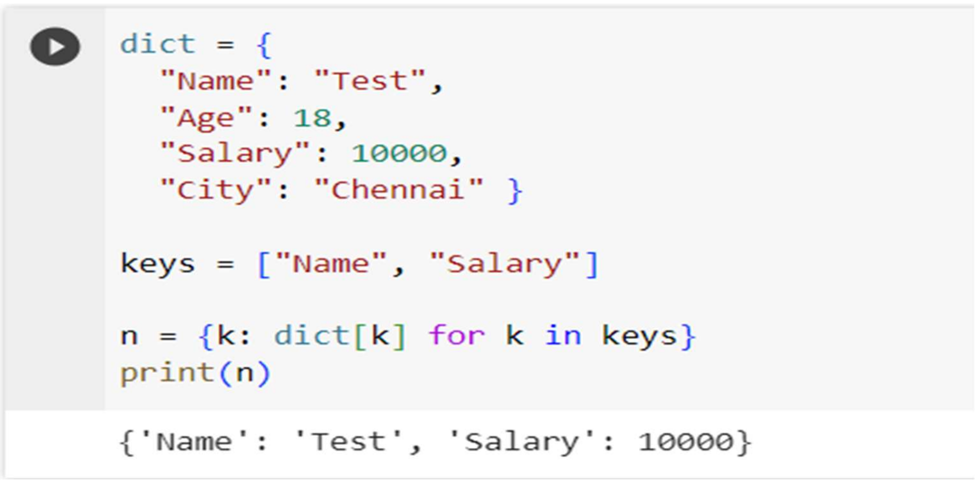
Date :

Aim : To Write a Python Program for Extracting Specific Keys from a Dictionary.

Algorithm :

- Start the program.
- Create a dictionary 'dict' with key-value pairs representing different attributes of an individual, such as name, age, salary, and city.
- Define a list 'keys' containing the keys you want to extract from the dictionary, in this case, "Name" and "Salary."
- Use a dictionary comprehension to create a new dictionary 'n' that includes only the specified keys from the original 'dict' along with their corresponding values.
- Print the 'n' dictionary, which contains a subset of key-value pairs extracted from the original dictionary.
- End the program.

Source Code & Output :



```
dict = {
  "Name": "Test",
  "Age": 18,
  "Salary": 10000,
  "City": "Chennai" }

keys = ["Name", "Salary"]

n = {k: dict[k] for k in keys}
print(n)
```

```
{'Name': 'Test', 'Salary': 10000}
```

Result :

The Python program for Extracting Specific Keys from a Dictionary is compiled & executed successfully.

Teacher’s signature

Set Operations

Exp. No. : 10

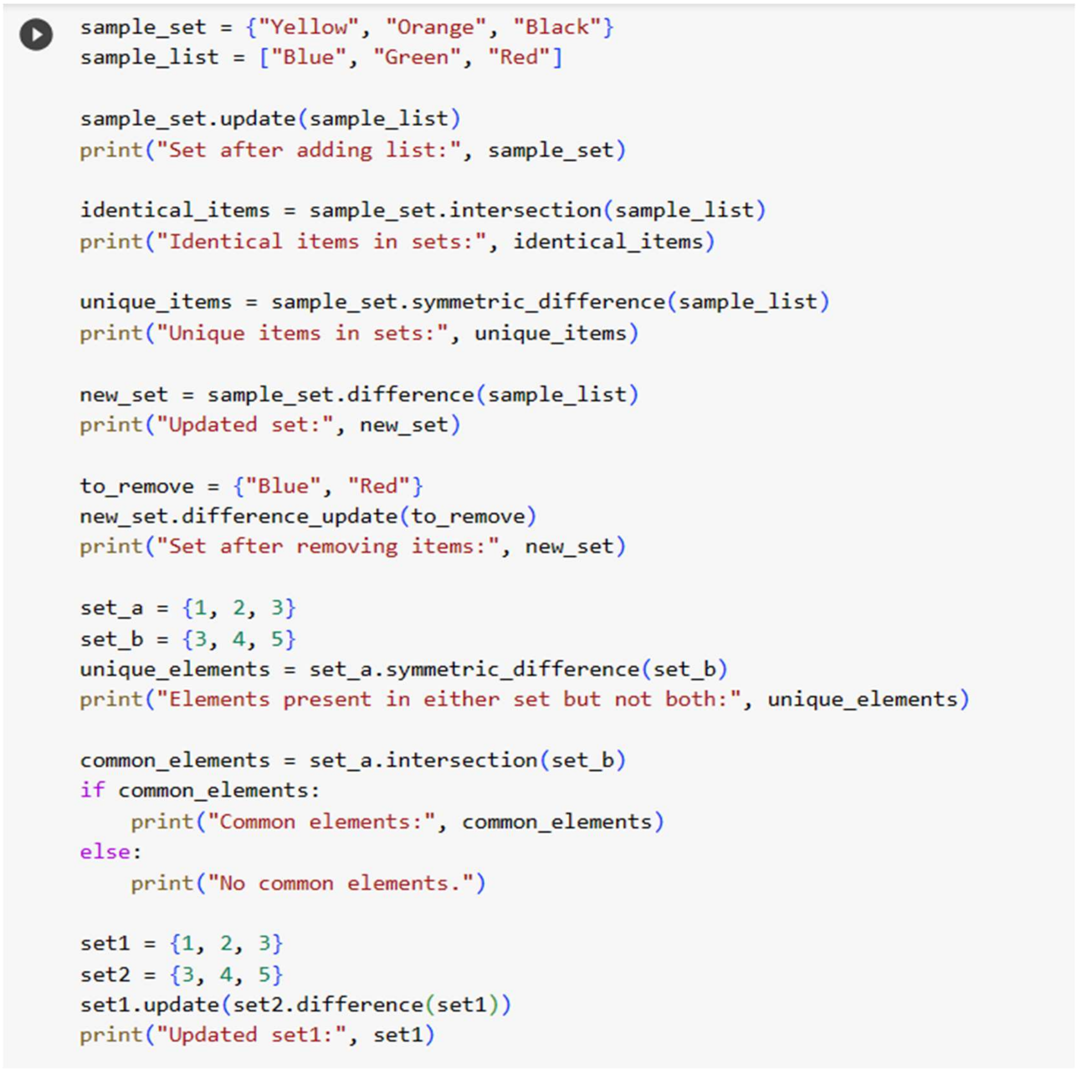
Aim : To write a program to perform the following operations in:

sample_set={“Yellow”, ”Orange”, ”Black”}, sample_list=[“Blue”, ”Green”, ”Red”]

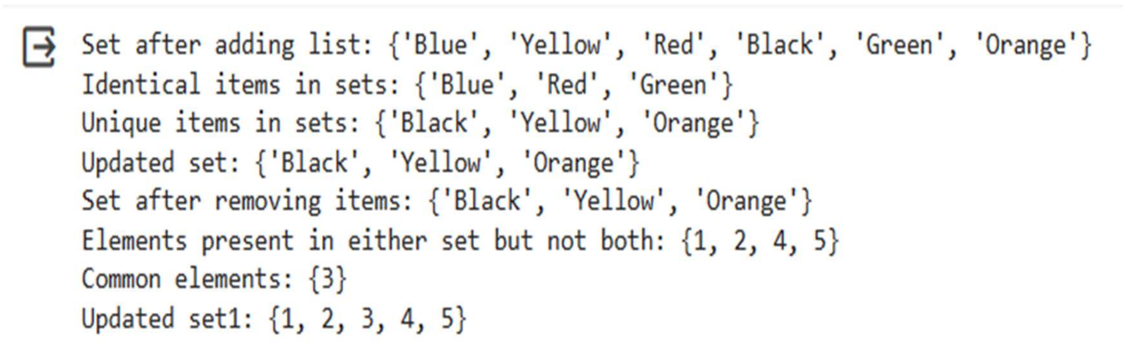
Algorithm :

- Start the Program
- Create the Sample_set and the Sample_list
- Use the following functions to perform the tasks
- Print the dict1
- Stop the program

Source Code & Output:



```python
sample_set = {"Yellow", "Orange", "Black"}
sample_list = ["Blue", "Green", "Red"]

sample_set.update(sample_list)
print("Set after adding list:", sample_set)

identical_items = sample_set.intersection(sample_list)
print("Identical items in sets:", identical_items)

unique_items = sample_set.symmetric_difference(sample_list)
print("Unique items in sets:", unique_items)

new_set = sample_set.difference(sample_list)
print("Updated set:", new_set)

to_remove = {"Blue", "Red"}
new_set.difference_update(to_remove)
print("Set after removing items:", new_set)

set_a = {1, 2, 3}
set_b = {3, 4, 5}
unique_elements = set_a.symmetric_difference(set_b)
print("Elements present in either set but not both:", unique_elements)

common_elements = set_a.intersection(set_b)
if common_elements:
    print("Common elements:", common_elements)
else:
    print("No common elements.")

set1 = {1, 2, 3}
set2 = {3, 4, 5}
set1.update(set2.difference(set1))
print("Updated set1:", set1)
```



```
Set after adding list: {'Blue', 'Yellow', 'Red', 'Black', 'Green', 'Orange'}
Identical items in sets: {'Blue', 'Red', 'Green'}
Unique items in sets: {'Black', 'Yellow', 'Orange'}
Updated set: {'Black', 'Yellow', 'Orange'}
Set after removing items: {'Black', 'Yellow', 'Orange'}
Elements present in either set but not both: {1, 2, 4, 5}
Common elements: {3}
Updated set1: {1, 2, 3, 4, 5}
```

Result :

The Python program to perform the following operations in:

sample_set={“Yellow”, ”Orange”, ”Black”}, sample_list=[“Blue”, ”Green”, ”Red”] is compiled & executed successfully.

Teacher’s signature

Printing 3 random integers between 100 and 999 divisible by 5

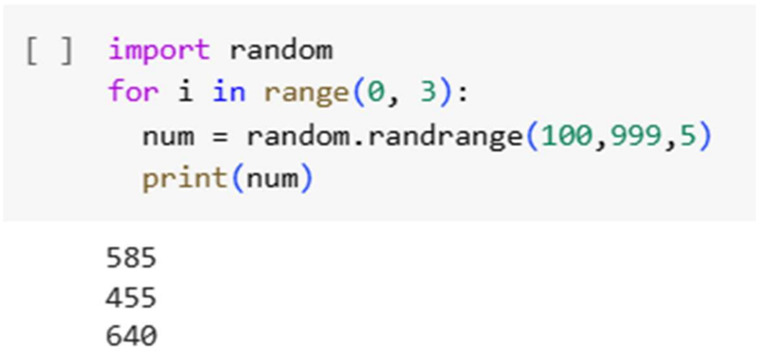
Exp. No. : 11

Aim : To write a python program to generate 3 random integers between 100 and 999 which is divisible by 5.

Algorithm :

- Start the Program
- Import the module random
- Using random.randrange() function, find the required numbers.
- Print the result
- Stop the program

Source Code & Output:



```
[ ]  import random
     for i in range(0, 3):
       num = random.randrange(100,999,5)
       print(num)

585
455
640
```

Result :

The Python program to generate 3 random integers between 100 and 999 which is divisible by 5 is compiled & executed successfully.

Teacher's signature

Summing Numeric Values in a List

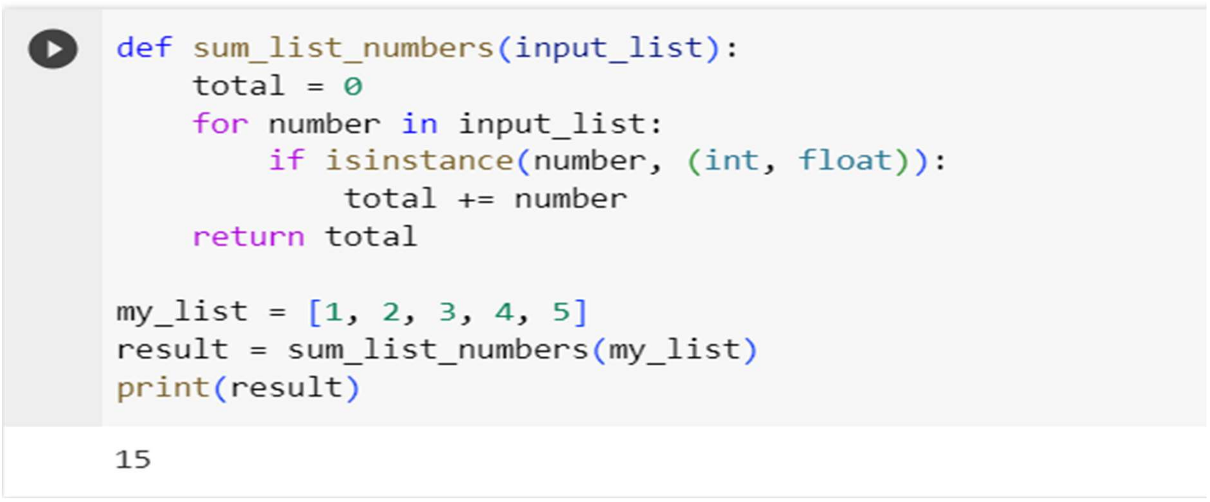
Exp. No. : 12

Aim : To write a python program to Summing Numeric Values in a List.

Algorithm :

- Start the program.
- Define a function 'sum_list_numbers' that takes a list as input.
- Initialize 'total' to 0.
- Loop through the list, and for each element, if it's a number (int or float), add it to 'total'.
- Return 'total'.
- Create a list of numbers.
- Call 'sum_list_numbers' with the list as input and print the result.
- End the program.

Source Code & Output:



```python
def sum_list_numbers(input_list):
    total = 0
    for number in input_list:
        if isinstance(number, (int, float)):
            total += number
    return total

my_list = [1, 2, 3, 4, 5]
result = sum_list_numbers(my_list)
print(result)
```

```
15
```

Result :

The Python program to Summing Numeric Values in a List is compiled & executed successfully.

Teacher's signature

Calculating Factorial Using Recursion

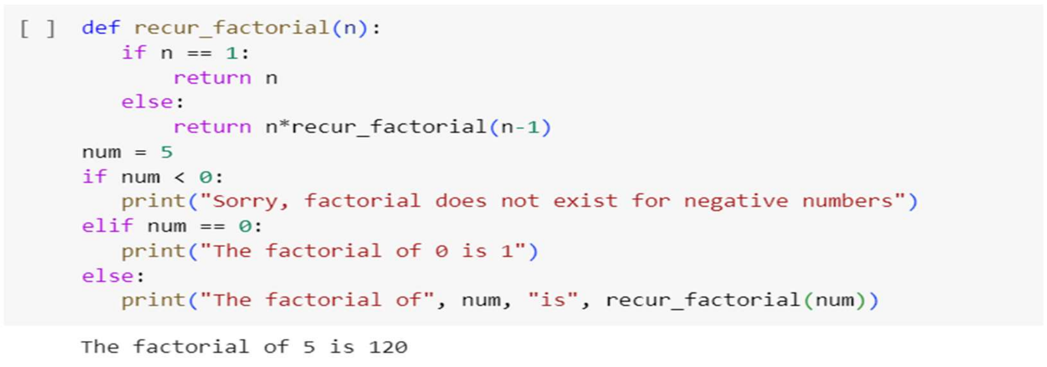
Exp. No. : 13

Aim : To write a python program to Summing Numeric Values in a List.

Algorithm :

- Start the program.
- Define a recursive function 'recur_factorial' that calculates the factorial of a positive integer 'n'.
    - If 'n' is 1, return 1 (base case).
    - Otherwise, return 'n' multiplied by the result of calling 'recur_factorial' with 'n-1' as the argument.
- Set 'num' to 5, representing the number for which you want to calculate the factorial.
- Check if 'num' is less than 0. If so, print "Sorry, factorial does not exist for negative numbers."
- If 'num' is 0, print "The factorial of 0 is 1."
- If 'num' is positive, call the 'recur_factorial' function with 'num' as the argument and print the result, displaying the calculated factorial.
- End the program.

Source Code & Output:



```
[ ]  def recur_factorial(n):
        if n == 1:
            return n
        else:
            return n*recur_factorial(n-1)
     num = 5
     if num < 0:
        print("Sorry, factorial does not exist for negative numbers")
     elif num == 0:
        print("The factorial of 0 is 1")
     else:
        print("The factorial of", num, "is", recur_factorial(num))

     The factorial of 5 is 120
```

Result :

The Python program to Summing Numeric Values in a List is compiled & executed successfully.

Teacher's signature

Simple Calculator with Input Validation

Exp. No. : 14

Date :

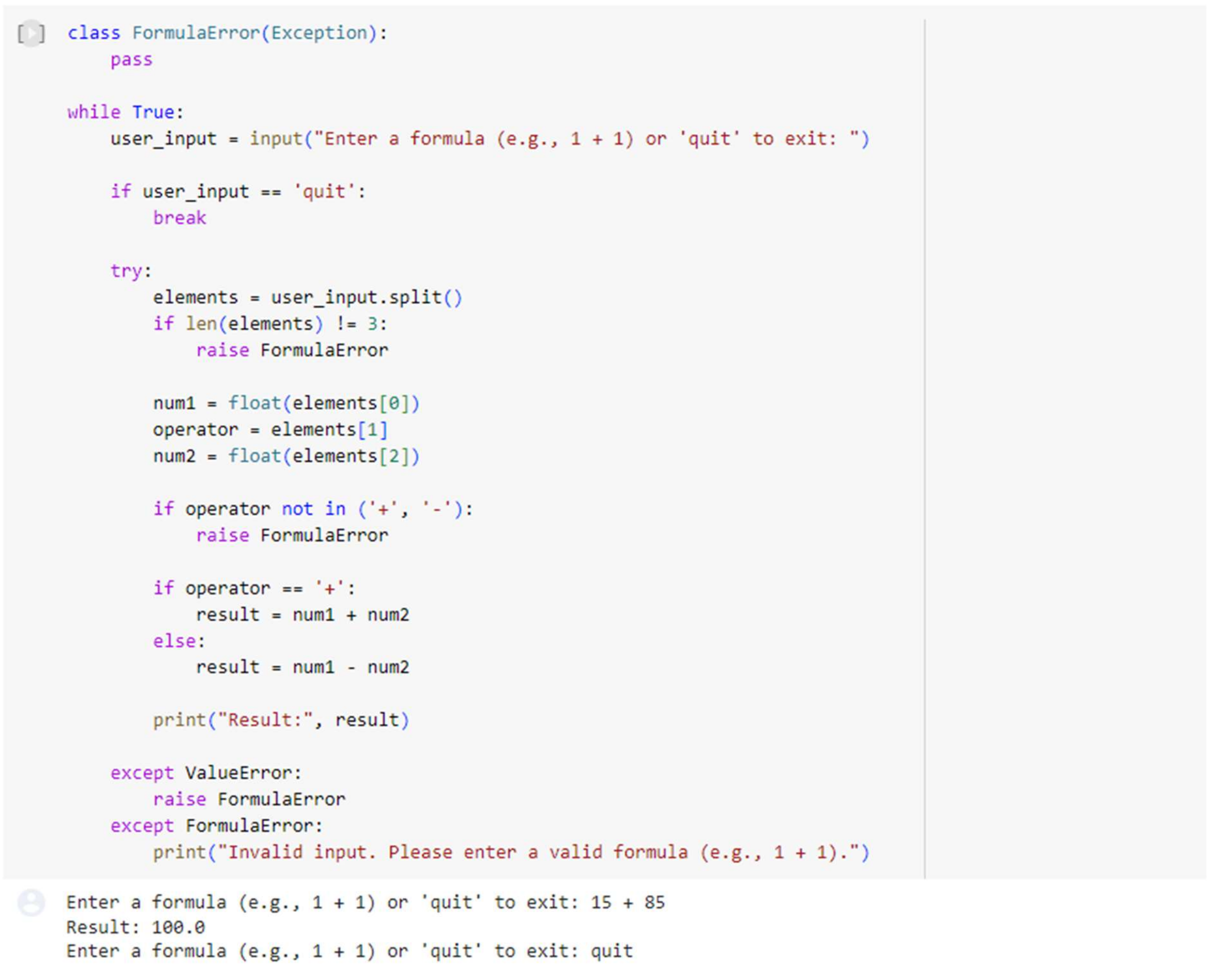
Aim : To Write a Python Program to Simple Calculator with Input Validation.

Algorithm :

- Start the program.
- Define a custom exception class 'FormulaError' that inherits from the built-in 'Exception' class.
- Set up a continuous loop using 'while True' to allow the user to input formulas.
- Prompt the user to enter a formula (e.g., "1 + 1") or type 'quit' to exit.
- If the user enters 'quit', break out of the loop and exit the program.
- Split the user input into elements using the 'split()' function, and store them in the 'elements' list.
- Check if there are exactly three elements in 'elements' (two numbers and one operator). If not, raise a 'FormulaError' to indicate an invalid formula.
- Convert the first and third elements to floating-point numbers, and store them in 'num1' and 'num2'. Extract the operator from the second element.
- Check if the operator is either '+' or '-'. If not, raise a 'FormulaError'.
- Perform the requested calculation based on the operator. If it's '+', add 'num1' and 'num2'; if it's '-', subtract 'num2' from 'num1'. Store the result in the 'result' variable.
- Print the result of the calculation.
- Handle potential exceptions:
  - If a 'ValueError' occurs (e.g., when converting strings to floats), raise a 'FormulaError'.
  - If a 'FormulaError' is raised, print an error message.

- Continue the loop to allow the user to input more formulas.
- When the user types 'quit', exit the program.

Source Code & Output :



```
class FormulaError(Exception):
    pass

while True:
    user_input = input("Enter a formula (e.g., 1 + 1) or 'quit' to exit: ")

    if user_input == 'quit':
        break

    try:
        elements = user_input.split()
        if len(elements) != 3:
            raise FormulaError

        num1 = float(elements[0])
        operator = elements[1]
        num2 = float(elements[2])

        if operator not in ('+', '-'):
            raise FormulaError

        if operator == '+':
            result = num1 + num2
        else:
            result = num1 - num2

        print("Result:", result)

    except ValueError:
        raise FormulaError
    except FormulaError:
        print("Invalid input. Please enter a valid formula (e.g., 1 + 1).")
```

```
Enter a formula (e.g., 1 + 1) or 'quit' to exit: 15 + 85
Result: 100.0
Enter a formula (e.g., 1 + 1) or 'quit' to exit: quit
```

Result :

The Python program for Simple Calculator with Input Validation is compiled & executed successfully.

Teacher's signature

Inheritance in Python: Creating an F14 Jet

Exp. No. : 15

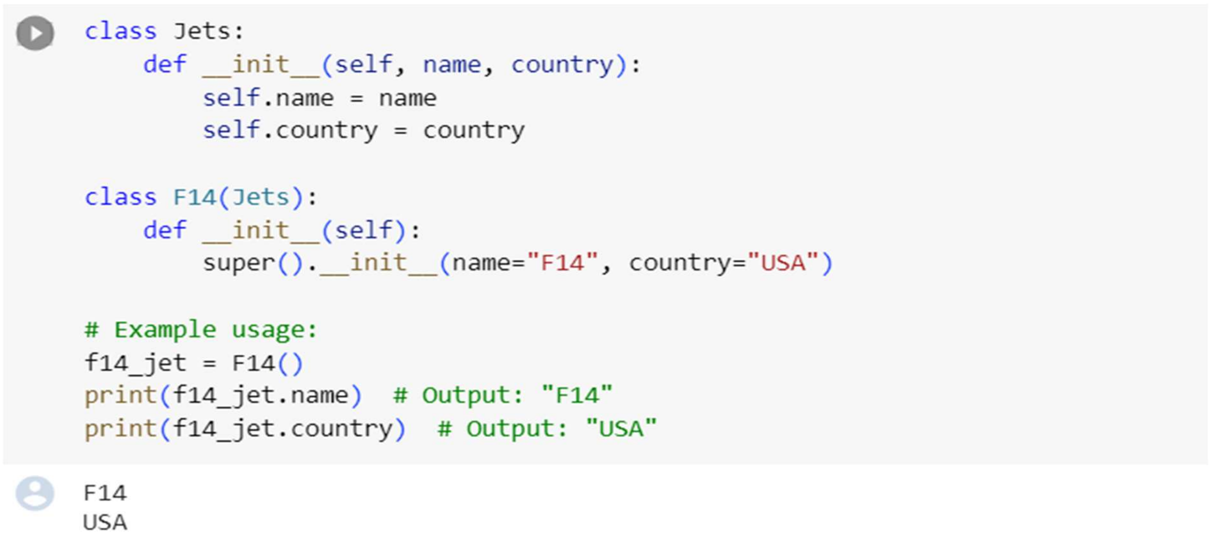
Date :

Aim : To Write a Python Program to Inheritance in Python: Creating an F14 Jet.

Algorithm :

- Start the program.
- Define a base class 'Jets' with an '__init__' constructor that takes 'name' and 'country' as attributes and assigns them to the instance variables.
- Define a derived class 'F14' that inherits from the 'Jets' base class.
- In the 'F14' class, define an '__init__' constructor that doesn't take any arguments, but it uses super().'__init__()' to call the constructor of the base class ('Jets') with specific values for 'name' and 'country ("F14" and "USA" respectively).
- Create an instance of the 'F14' class, named 'f14_jet'.
- Print the 'name' and 'country' attributes of 'f14_jet', which will output "F14" and "USA", respectively.
- End the program.

Source Code & Output :



```python
class Jets:
    def __init__(self, name, country):
        self.name = name
        self.country = country

class F14(Jets):
    def __init__(self):
        super().__init__(name="F14", country="USA")

# Example usage:
f14_jet = F14()
print(f14_jet.name)  # Output: "F14"
print(f14_jet.country)  # Output: "USA"
```

```
F14
USA
```

Result :

The Python program for Inheritance in Python: Creating an F14 Jet is compiled & executed successfully.

Teacher’s signature

Creating a Bank Account Using Python Class

Exp. No. : 16

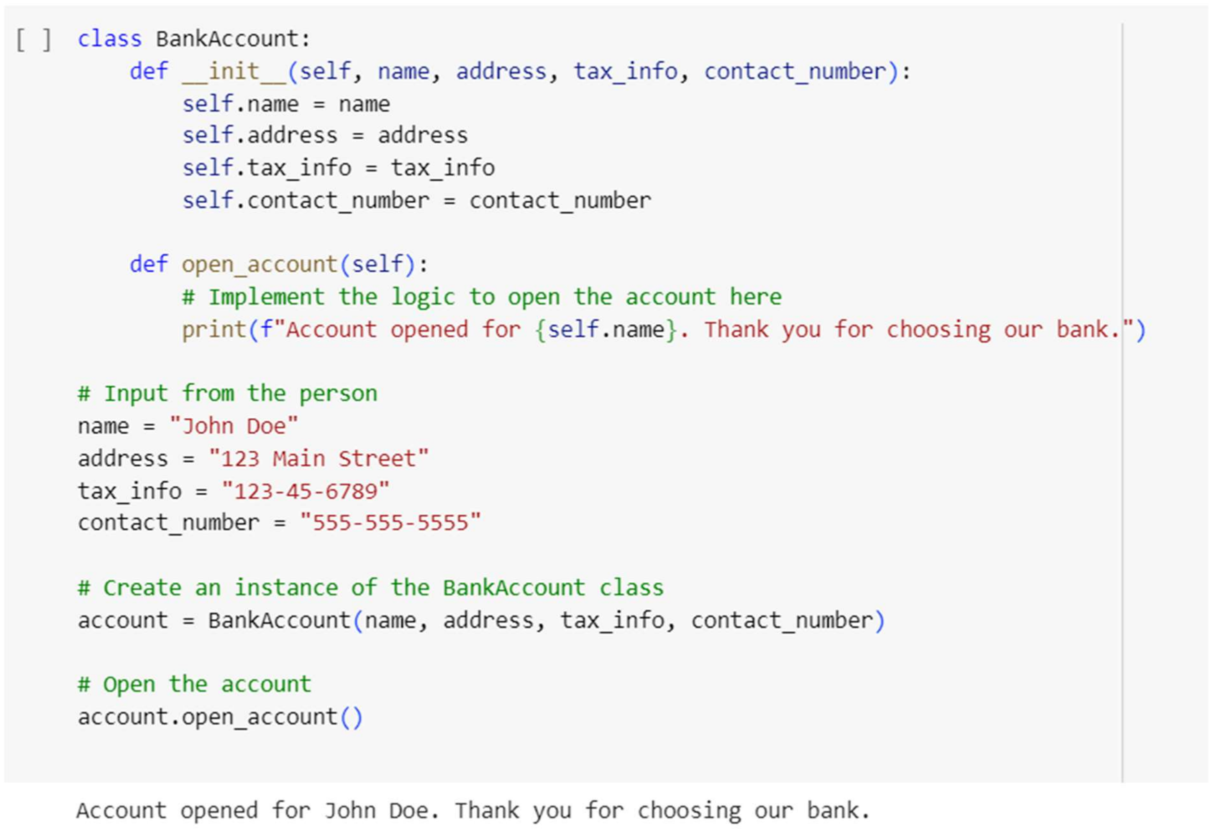
Date :

Aim : To Write a Python Program to Creating a Bank Account using Python Class.

Algorithm :

- Start the program.
- Define a Python class named 'BankAccount' with an '__init__' constructor that initializes attributes for 'name', 'address', 'tax_info', and 'contact_number' based on the provided arguments.
- Inside the 'BankAccount' class, define a method named 'open_account'. This method should include the logic for opening a bank account, such as printing a welcome message with the account holder's name.
- Provide values for the 'name', 'address', 'tax_info', and 'contact_number' based on user input or predefined values.
- Create an instance of the 'BankAccount' class, named 'account', by passing the provided values as arguments to the constructor.
- Call the 'open_account' method on the 'account' object, which will print a message welcoming the account holder and thanking them for choosing the bank.
- End the program.

Source Code & Output :



```python
class BankAccount:
    def __init__(self, name, address, tax_info, contact_number):
        self.name = name
        self.address = address
        self.tax_info = tax_info
        self.contact_number = contact_number

    def open_account(self):
        # Implement the logic to open the account here
        print(f"Account opened for {self.name}. Thank you for choosing our bank.")

# Input from the person
name = "John Doe"
address = "123 Main Street"
tax_info = "123-45-6789"
contact_number = "555-555-5555"

# Create an instance of the BankAccount class
account = BankAccount(name, address, tax_info, contact_number)

# Open the account
account.open_account()
```

```
Account opened for John Doe. Thank you for choosing our bank.
```

Result :

The Python program for Creating a Bank Account using Python Class is compiled & executed successfully.

Teacher’s signature

Comparing Student Scores Using Custom Comparison Methods

Exp. No. : 17

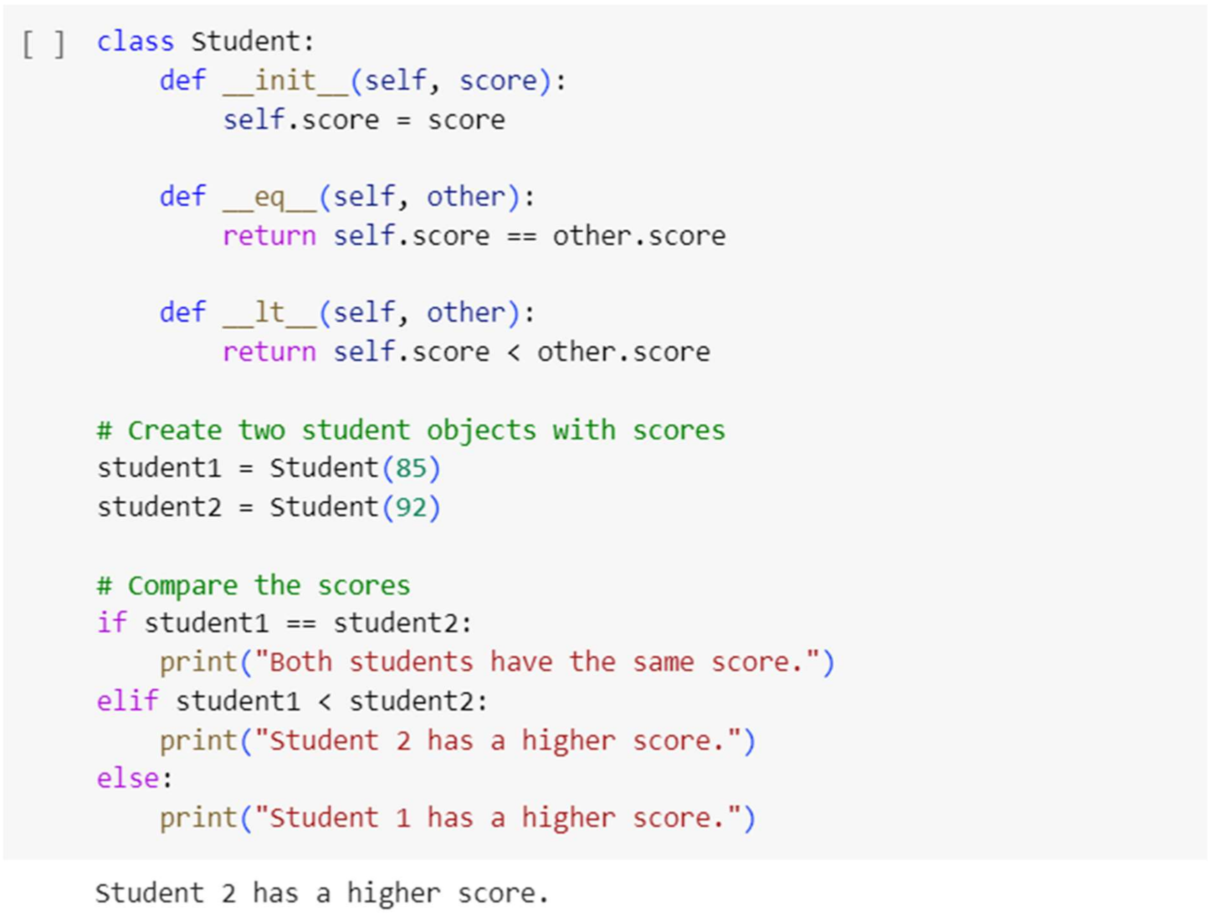
Date :

Aim : To Write a Python Program to Comparing Student Scores using Custom Comparison Methods.

Algorithm :

- Start the program.
- Define a Python class named 'Student' with an __init__ constructor that initializes an attribute 'score' based on the provided argument.
- Inside the 'Student' class, define custom comparison methods:
    - ‘__eq__(self, other)’: This method checks if the score of the current student is equal to the score of another student (defined by 'other').
    - ‘__lt__(self, other)’: This method checks if the score of the current student is less than the score of another student (defined by 'other').
- Create two 'Student' objects, 'student1' and 'student2', with different scores.
- Compare the scores of 'student1' and 'student2':
    - If they have the same score, print "Both students have the same score."
    - If 'student2' has a higher score, print "Student 2 has a higher score."
    - If 'student1' has a higher score, print "Student 1 has a higher score."
- End the program.

Source Code & Output :



```
class Student:
    def __init__(self, score):
        self.score = score

    def __eq__(self, other):
        return self.score == other.score

    def __lt__(self, other):
        return self.score < other.score

# Create two student objects with scores
student1 = Student(85)
student2 = Student(92)

# Compare the scores
if student1 == student2:
    print("Both students have the same score.")
elif student1 < student2:
    print("Student 2 has a higher score.")
else:
    print("Student 1 has a higher score.")
```

```
Student 2 has a higher score.
```

Result :

The Python program for Comparing Student Scores using Custom Comparison Methods is compiled & executed successfully.

Teacher's signature

Creating the Array using Numpy

Exp. No. : 18

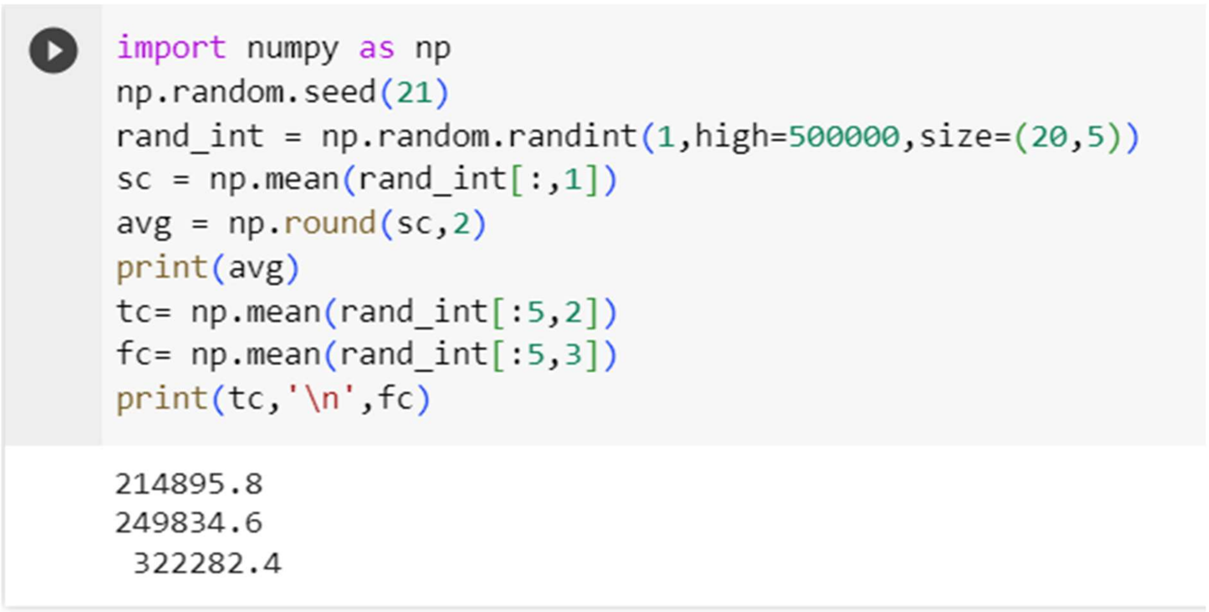
Date :

Aim : To Write a Python Program to Create the Array using Numpy.

Algorithm :

- Start the program.
- Generate a 20x5 array of random integers between 1 and 500,000 using NumPy.
- Calculate the average of the values in the second column of the array and round it to two decimal places.
- Print the rounded average, indicating it's the average of the second column.
- Compute the average of values in the third column for the first 5 rows.
- Calculate the average of values in the fourth column for the first 5 rows.
- Print the averages of the third and fourth columns separately.
- End the program.

Source Code & Output :



```
import numpy as np
np.random.seed(21)
rand_int = np.random.randint(1,high=500000,size=(20,5))
sc = np.mean(rand_int[:,1])
avg = np.round(sc,2)
print(avg)
tc= np.mean(rand_int[:5,2])
fc= np.mean(rand_int[:5,3])
print(tc,'\n',fc)
```

```
214895.8
249834.6
 322282.4
```

Result :

The Python program for creating the array using numpy is compiled & executed successfully.

Teacher’s signature

Create a 4X2 Array using Numpy

Exp. No. : 19

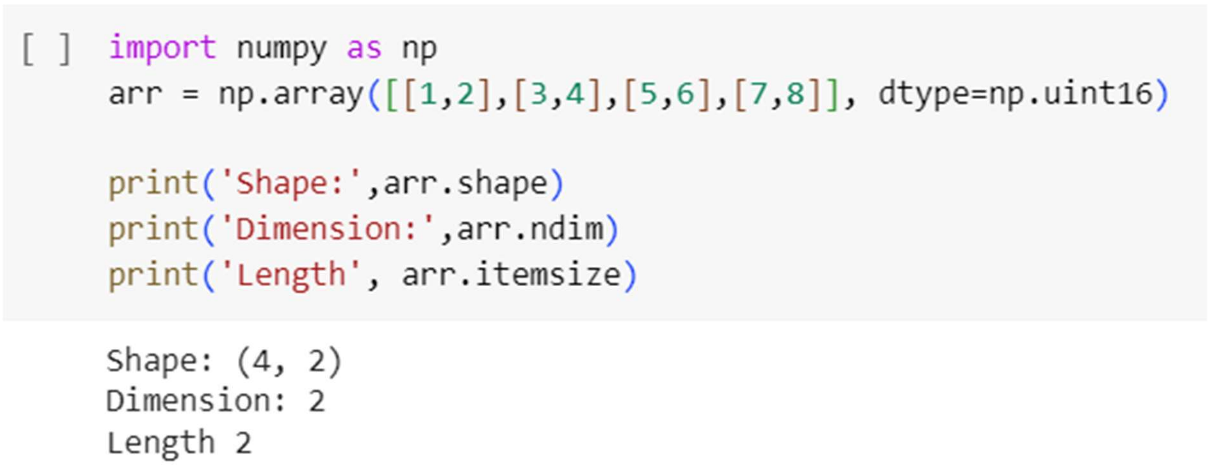
Date :

Aim : To Write a Python Program to Create a 4X2 Integer Array using Numpy and print its attributes like shape, dimension and length.

Algorithm :

- Start the program.
- Import the NumPy library for array operations.
- Create a NumPy array named 'arr' with four sub-arrays, each containing two elements, and specify the data type as unsigned 16-bit integers (uint16).
- Print the shape of the 'arr' array using arr.shape, which indicates the number of rows and columns.
- Print the dimension of the 'arr' array using arr.ndim, showing the number of axes (usually 2 for a 2D array).
- Print the length (size in bytes) of a single array element in 'arr' using arr.itemsize, revealing the memory size of each element.
- End the program.

Source Code & Output :



```
[ ]  import numpy as np
     arr = np.array([[1,2],[3,4],[5,6],[7,8]], dtype=np.uint16)

     print('Shape:',arr.shape)
     print('Dimension:',arr.ndim)
     print('Length', arr.itemsize)
```

```
Shape: (4, 2)
Dimension: 2
Length 2
```

Result :

The Python program for Create a 4X2 Integer Array using Numpy and print its attributes like shape, dimension and length is compiled & executed successfully.

Teacher’s signature

Create a 5X2 Array using Numpy

Exp. No. : 20

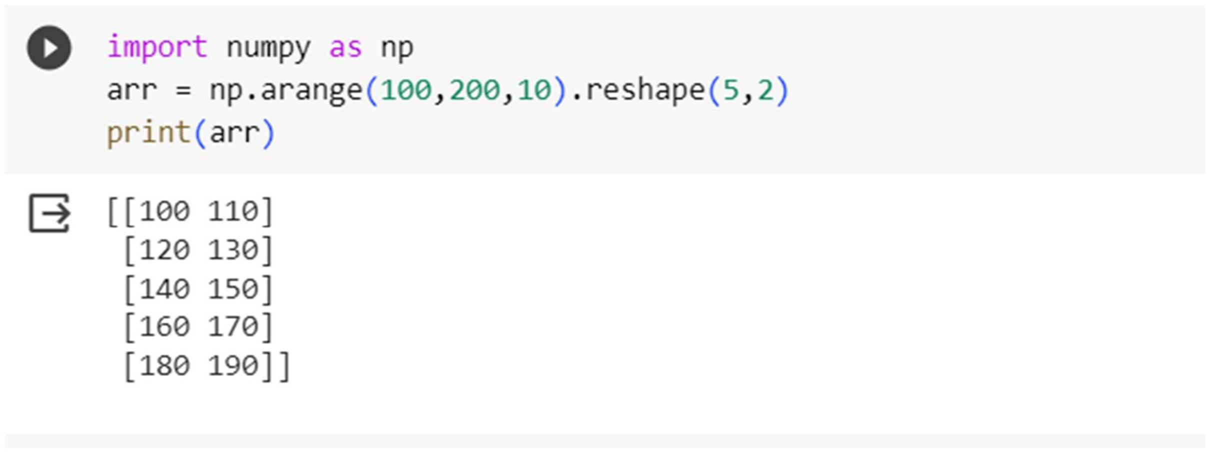
Date :

Aim : To Write a Python Program to Create a 5X2 Array using Numpy from a range between 100 to 200 such that the difference between each element is 10.

Algorithm :

- Start the program.
- Import the NumPy library to perform array operations.
- Generate a NumPy array named 'arr' using np.arange(100, 200, 10), which creates an array with values starting from 100, incrementing by 10, and stopping before 200.
- Reshape the 'arr' array into a 5x2 format using .reshape(5, 2). This rearranges the elements into five rows and two columns.
- Print the 'arr' array, displaying the resulting 5x2 matrix.
- End the program.

Source Code & Output :



```python
import numpy as np
arr = np.arange(100,200,10).reshape(5,2)
print(arr)
```

```
[[100 110]
 [120 130]
 [140 150]
 [160 170]
 [180 190]]
```

Result :

The Python program for Creating a 5X2 Array using Numpy from a range between 100 to 200 such that the difference between each element is 10 is compiled & executed successfully.

Teacher's signature

NumPy Array Slicing: Selecting a Subset of Rows & Columns

Exp. No. : 21

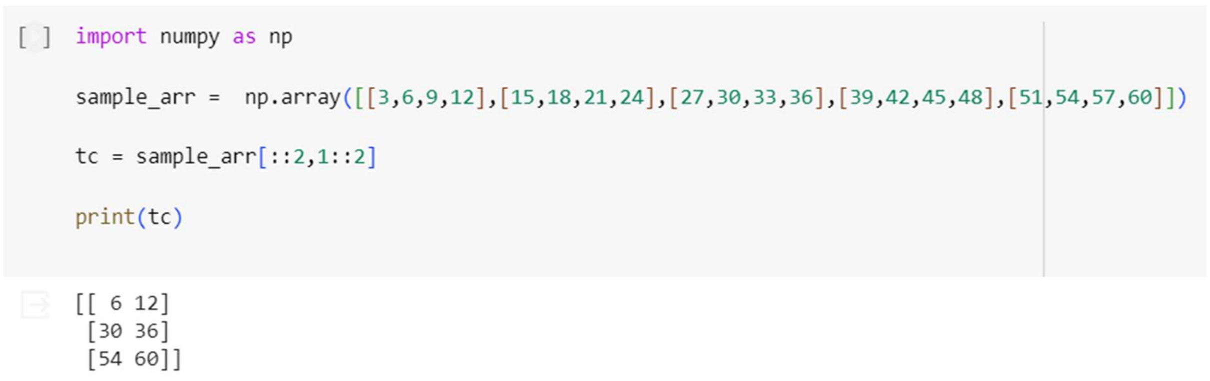
Date :

Aim : To Write a Python Program for NumPy Array Slicing by Selecting a subset of Rows & Columns

Algorithm :

- Start the program.
- Import the NumPy library for array operations.
- Create a NumPy array named 'sample_arr' with a 5x4 matrix containing numbers, where each row consists of consecutive multiples of 3.
- Extract a subarray 'tc' from 'sample_arr' using slicing:
    - Rows are selected at intervals of 2 using sample_arr[::2], resulting in every other row.
    - Columns are selected at intervals of 2 starting from the second column using 1::2, which includes every other column.
- Print the 'tc' subarray, which contains a subset of rows and columns from 'sample_arr'.
- End the program.

Source Code & Output :



```
import numpy as np

sample_arr =  np.array([[3,6,9,12],[15,18,21,24],[27,30,33,36],[39,42,45,48],[51,54,57,60]])

tc = sample_arr[::2,1::2]

print(tc)
```

```
[[ 6 12]
 [30 36]
 [54 60]]
```

Result :

The Python program for NumPy Array Slicing by Selecting a subset of Rows & Columns is compiled & executed successfully.

Teacher’s signature

NumPy Array Column Extraction: Isolating a Specific Column

Exp. No. : 22

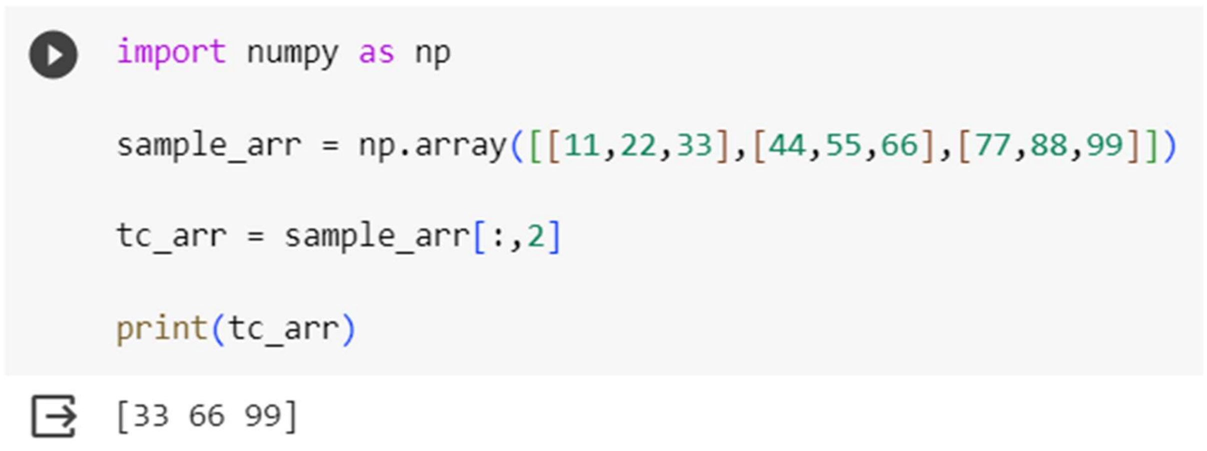
Date :

Aim : To Write a Python Program for NumPy Array Column Extraction By Isolating a Specific Column

Algorithm :

- Start the program.
- Import the NumPy library for array operations.
- Create a NumPy array named 'sample_arr' with a 3x3 matrix containing various numbers.
- Extract a single column 'tc_arr' from 'sample_arr' using slicing with [:, 2], which selects all rows and the third column.
- Print the 'tc_arr' column, which represents the third column of 'sample_arr'.
- End the program.

Source Code & Output :



```python
import numpy as np

sample_arr = np.array([[11,22,33],[44,55,66],[77,88,99]])

tc_arr = sample_arr[:,2]

print(tc_arr)
```

```
[33 66 99]
```

Result :

The Python program for NumPy Array Column Extraction By Isolating a Specific Column is compiled & executed successfully.

Teacher's signature

NumPy Array Operations: Element-Wise Addition and Squaring

Exp. No. : 23

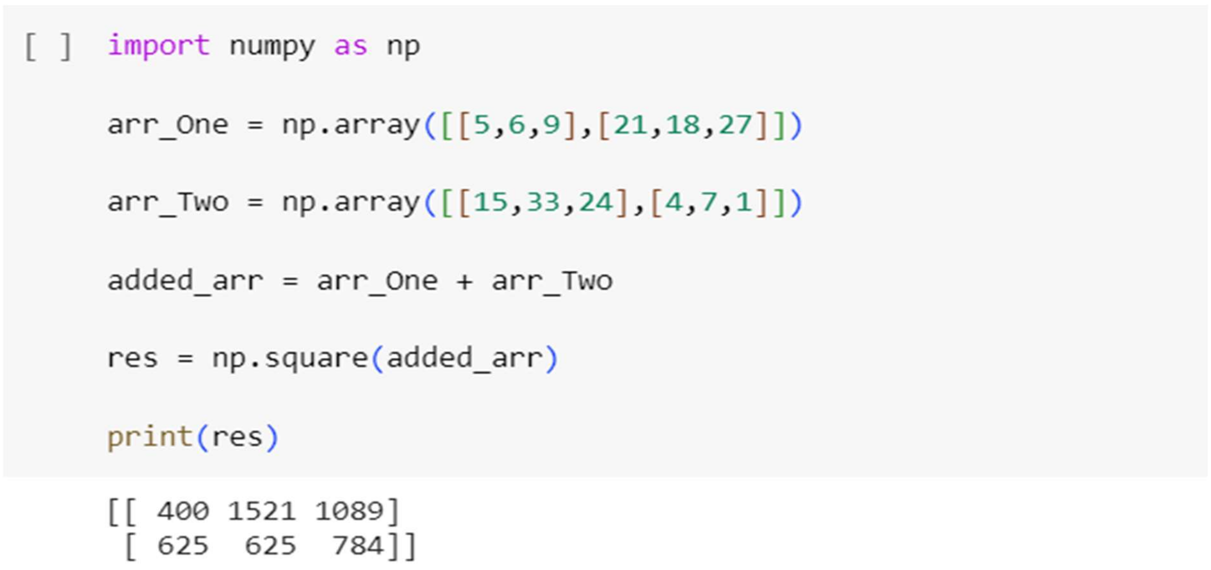
Date :

Aim : To Write a Python Program to Numpy Array Operations by element-wise Addition & Squaring.

Algorithm :

- Start the program.
- Import the NumPy library for array operations.
- Create two NumPy arrays, 'arr_One' and 'arr_Two,' each with a 2x3 matrix containing numerical values.
- Add the corresponding elements of 'arr_One' and 'arr_Two' to create a new array 'added_arr' using the '+' operator. This results in element-wise addition of the two arrays.
- Calculate the square of each element in 'added_arr' and store the result in the 'res' array using 'np.square()'.
- Print the 'res' array, which represents the element-wise squares of the sum of 'arr_One' and 'arr_Two'.
- End the program.

Source Code & Output :



```
[ ]  import numpy as np

     arr_One = np.array([[5,6,9],[21,18,27]])

     arr_Two = np.array([[15,33,24],[4,7,1]])

     added_arr = arr_One + arr_Two

     res = np.square(added_arr)

     print(res)
```

```
[[ 400 1521 1089]
 [ 625  625  784]]
```

Result :

The Python program for Numpy Array Operations by element-wise Addition & Squaring is compiled & executed successfully.

Teacher's signature

Creating a Binary Border in a NumPy Array

Exp. No. : 24

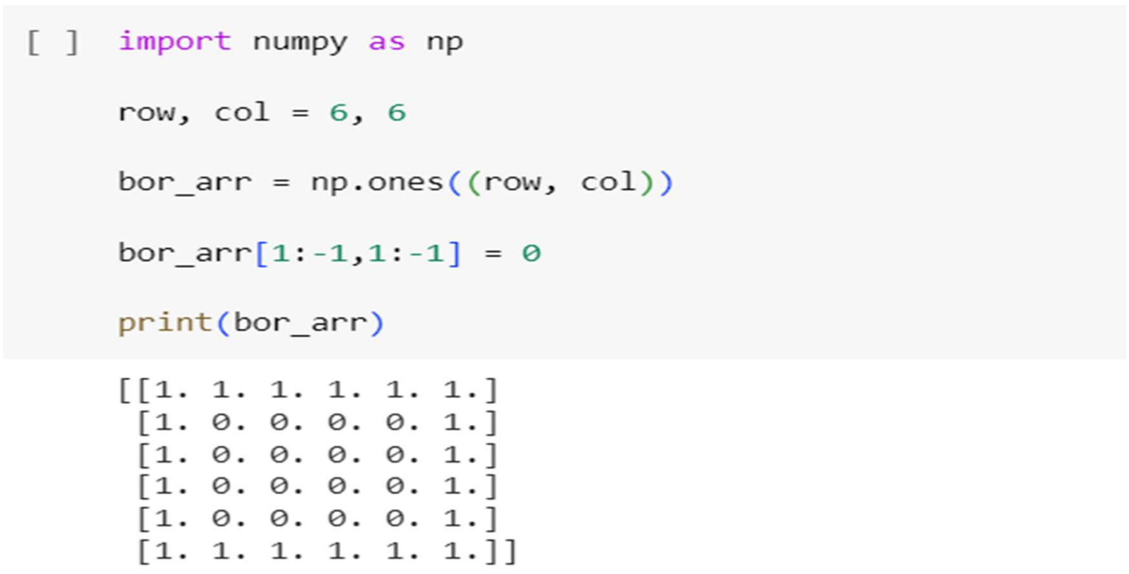
Date :

Aim : To Write a Python Program to Create a Binary Border in a NumPy Array.

Algorithm :

- Start the program.
- Import the NumPy library for array operations.
- Define the number of rows and columns as 'row' and 'col', both set to 6.
- Create a NumPy array 'bor_arr' filled with ones, forming a 6x6 matrix, using 'np.ones((row, col))'.
- Modify the 'bor_arr' array to create a binary border by setting the interior elements to 0. This is done by slicing the array from the second row to the second-to-last row and from the second column to the second-to-last column: 'bor_arr[1:-1, 1:-1] = 0'.
- Print the 'bor_arr' array, which now contains a binary border with 1s on the edges and 0s in the interior.
- End the program.

Source Code & Output :



```
[ ]  import numpy as np

     row, col = 6, 6

     bor_arr = np.ones((row, col))

     bor_arr[1:-1,1:-1] = 0

     print(bor_arr)
```

```
[[1. 1. 1. 1. 1. 1.]
 [1. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 1.]
 [1. 1. 1. 1. 1. 1.]]
```

Teacher’s signature

Result :

The Python program for Creating a Binary Border in a NumPy Array is compiled & executed successfully.

Teacher’s signature

Creating and Printing a NumPy Array Using a Loop

Exp. No. : 25

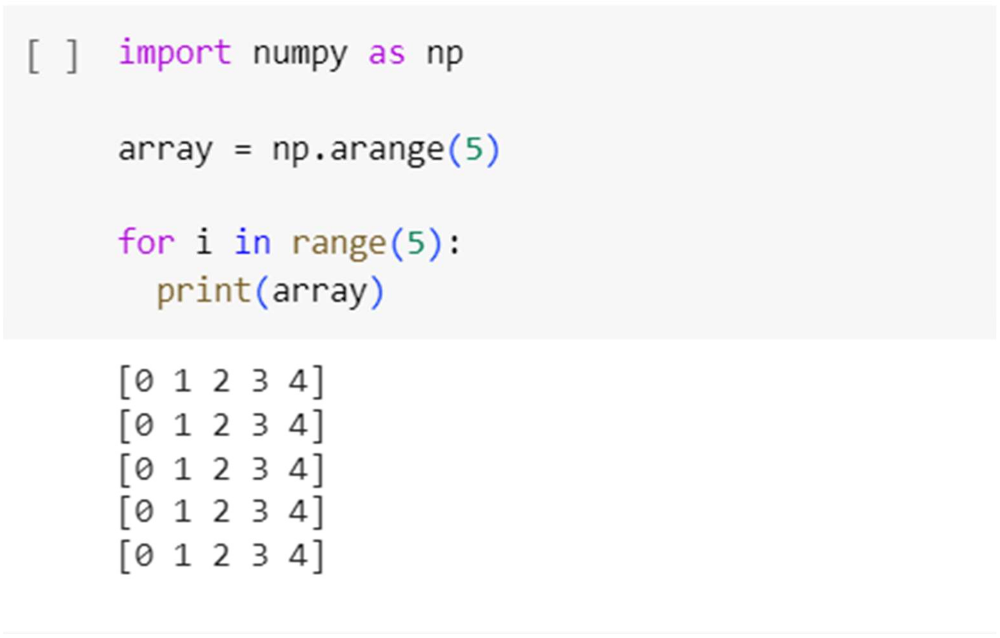
Date :

Aim : To Write a Python Program to Create and Printing a NumPy Array using a Loop.

Algorithm :

- Start the program.
- Import the NumPy library for array operations.
- Create a NumPy array named 'array' 'using np.arange(5)', which generates an array containing values from 0 to 4.
- Use a 'for' loop that iterates from 0 to 4 (5 iterations in total) with the variable 'i'.
- Within the loop, print the 'array' on each iteration. The 'array' remains the same in each iteration of the loop.
- End the program.

Source Code & Output :



```
[ ]  import numpy as np

     array = np.arange(5)

     for i in range(5):
       print(array)
```

```
[0 1 2 3 4]
[0 1 2 3 4]
[0 1 2 3 4]
[0 1 2 3 4]
[0 1 2 3 4]
```

Result :

The Python program for Creating and Printing a NumPy Array using a Loop is compiled & executed successfully.

Teacher's signature

NumPy Array Slicing and Indexing

Exp. No. : 26

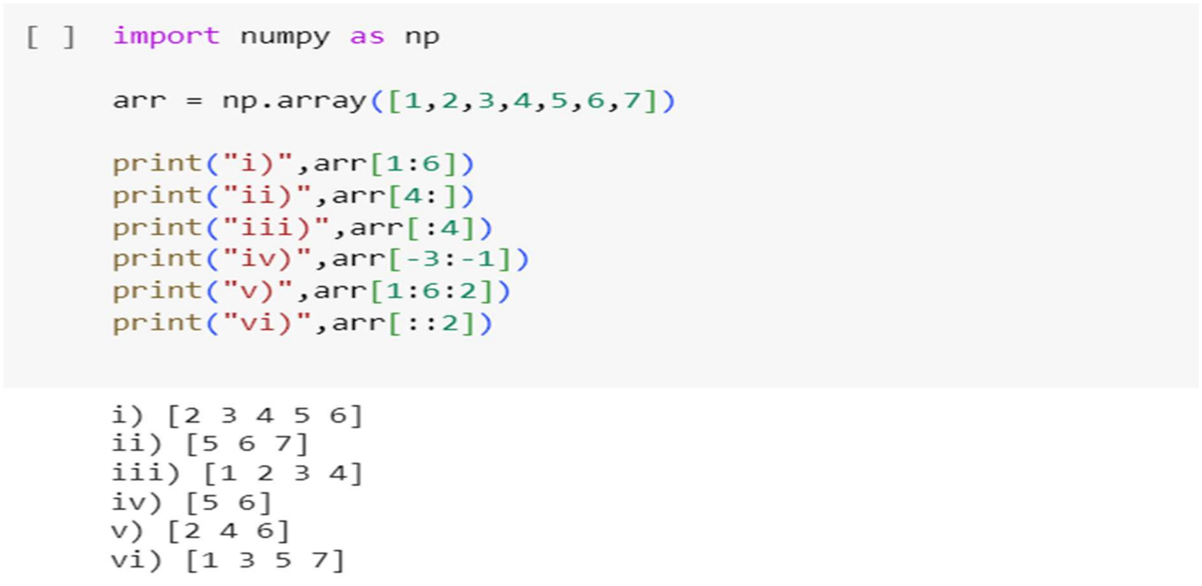
Date :

Aim : To Write a Python Program to NumPy Array Slicing and Indexing.

Algorithm :

- Start the program.
- Import the NumPy library for array operations.
- Create a NumPy array named 'arr' containing values from 1 to 7.
- Print the following slices and indices of 'arr':
    - "i)" Slice from index 1 to 6 (exclusive).
    - "ii)" Slice from index 4 to the end.
    - "iii)" Slice from the beginning to index 4 (exclusive).
    - "iv)" Slice from the third-to-last element to the second-to-last element.
    - "v)" Slice from index 1 to 6 (exclusive) with a step size of 2.
    - "vi)" Slice with a step size of 2, including all elements.
- End the program.

Source Code & Output :



```
[ ]  import numpy as np

     arr = np.array([1,2,3,4,5,6,7])

     print("i)",arr[1:6])
     print("ii)",arr[4:])
     print("iii)",arr[:4])
     print("iv)",arr[-3:-1])
     print("v)",arr[1:6:2])
     print("vi)",arr[::2])
```

```
i) [2 3 4 5 6]
ii) [5 6 7]
iii) [1 2 3 4]
iv) [5 6]
v) [2 4 6]
vi) [1 3 5 7]
```

Teacher's signature

Result :

The Python program for NumPy Array Slicing and Indexing is compiled & executed successfully.

Teacher's signature

Reshaping a NumPy Array into a 3x4 Matrix

Exp. No. : 27

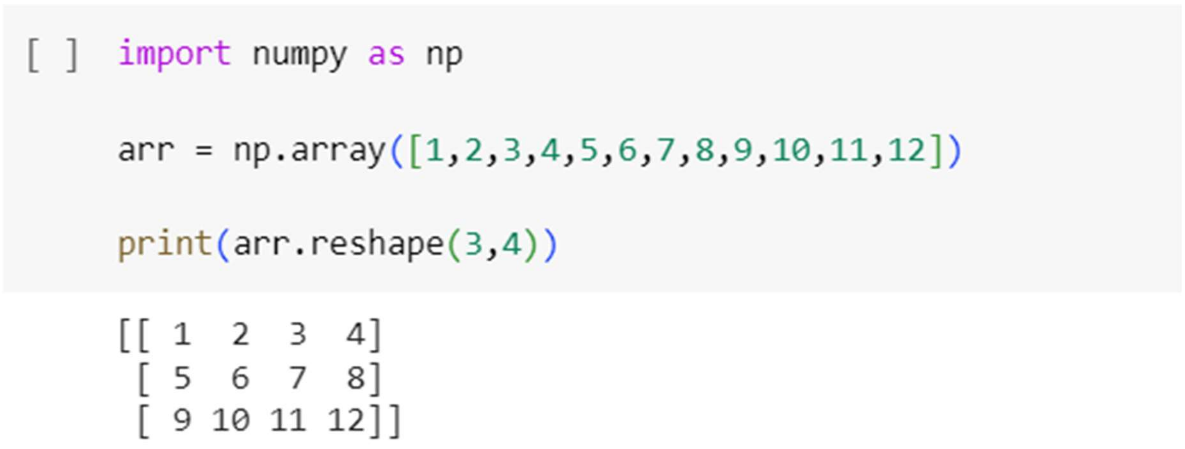
Date :

Aim : To Write a Python Program to Reshaping a NumPy Array into a 3x4 Matrix.

Algorithm :

- Start the program.
- Import the NumPy library for array operations.
- Create a NumPy array named 'arr' containing values from 1 to 12.
- Use the '.reshape(3, 4)' function to reshape the 'arr' array into a 3x4 matrix.
- Print the reshaped 'arr', which now appears as a 3x4 matrix with the values distributed in rows and columns.
- End the program.

Source Code & Output :



```
[ ]  import numpy as np

     arr = np.array([1,2,3,4,5,6,7,8,9,10,11,12])

     print(arr.reshape(3,4))
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

Result :

The Python program for Reshaping a NumPy Array into a 3x4 Matrix is compiled & executed successfully.

Teacher's signature

Reshaping a NumPy Array into a 3D Matrix

Exp. No. : 28

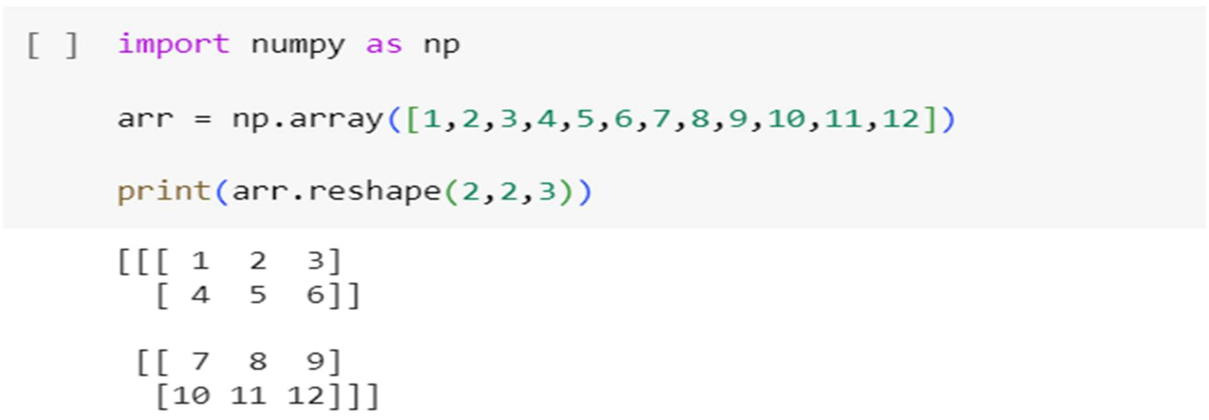
Date :

Aim : To Write a Python Program to Reshaping a NumPy Array into a 3D Matrix.

Algorithm :

- Start the program.
- Import the NumPy library for array operations.
- Create a NumPy array named 'arr' containing values from 1 to 12.
- Use the '.reshape(2, 2, 3)' function to reshape the 'arr' array into a 3D matrix with dimensions 2x2x3.
- Print the reshaped 'arr', which now appears as a 3D matrix with the values distributed in three dimensions.
- End the program.

Source Code & Output :



```
[ ]  import numpy as np

     arr = np.array([1,2,3,4,5,6,7,8,9,10,11,12])

     print(arr.reshape(2,2,3))
```

```
[[[ 1  2  3]
  [ 4  5  6]]

 [[ 7  8  9]
  [10 11 12]]]
```

Result :

The Python program for Reshaping a NumPy Array into a 3D Matrix is compiled & executed successfully.

Teacher's signature

Reshaping a NumPy Array into a 1D Array

Exp. No. : 29

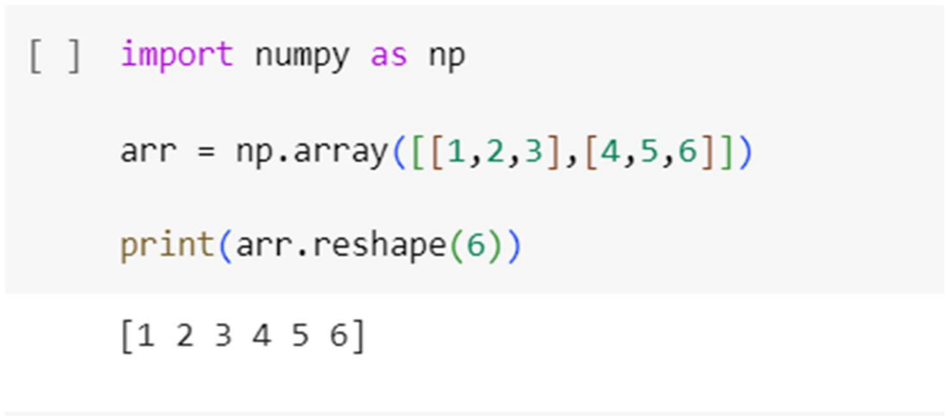
Date :

Aim : To Write a Python Program to Reshaping a NumPy Array into a 1D Array.

Algorithm :

- Start the program.
- Import the NumPy library for array operations.
- Create a NumPy array named 'arr' as a 2x3 matrix with values.
- Use the '.reshape(6)' function to reshape the 'arr' array into a 1D array containing 6 elements.
- Print the reshaped 'arr', which now appears as a 1D array.
- End the program.

Source Code & Output :



```
[ ]  import numpy as np

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

     print(arr.reshape(6))

     [1 2 3 4 5 6]
```

Result :

The Python program for Reshaping a NumPy Array into a 1D Array is compiled & executed successfully.

Teacher's signature

Slicing and Indexing a 2D NumPy Array

Exp. No. : 30

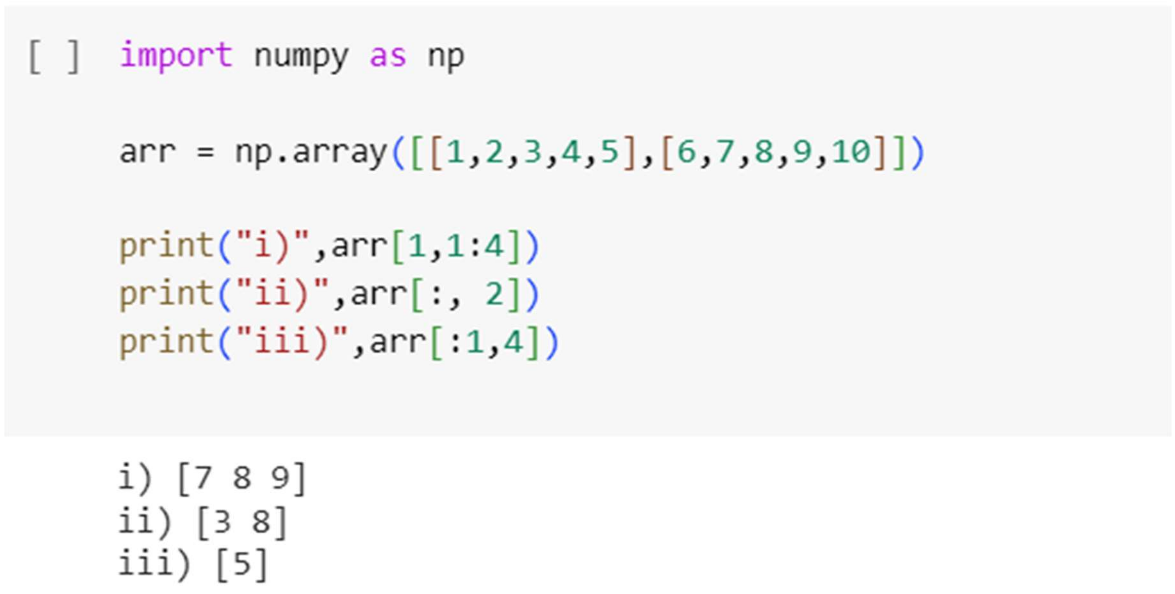
Date :

Aim : To Write a Python Program to Slicing and Indexing a 2D NumPy Array.

Algorithm :

- Start the program.
- Import the NumPy library for array operations.
- Create a NumPy array named 'arr' with a 2x5 matrix containing numerical values.
- Print the following slices and indices of 'arr':
    - "i)" Elements from the second row (index 1) and columns 1 to 3 (indices 1 to 3, inclusive).
    - "ii)" All elements from the third column (index 2) of 'arr'.
    - "iii)" Elements from the first row (index 0) and the fifth column (index 4) of 'arr'.
- End the program.

Source Code & Output :



```
[ ]  import numpy as np

     arr = np.array([[1,2,3,4,5],[6,7,8,9,10]])

     print("i)",arr[1,1:4])
     print("ii)",arr[:, 2])
     print("iii)",arr[:1,4])
```

```
i) [7 8 9]
ii) [3 8]
iii) [5]
```

Teacher's signature

Result :

The Python program for Slicing and Indexing a 2D NumPy Array is compiled & executed successfully.

Teacher's signature

Creating and Printing a One-Dimensional Pandas Series

Exp. No. : 31

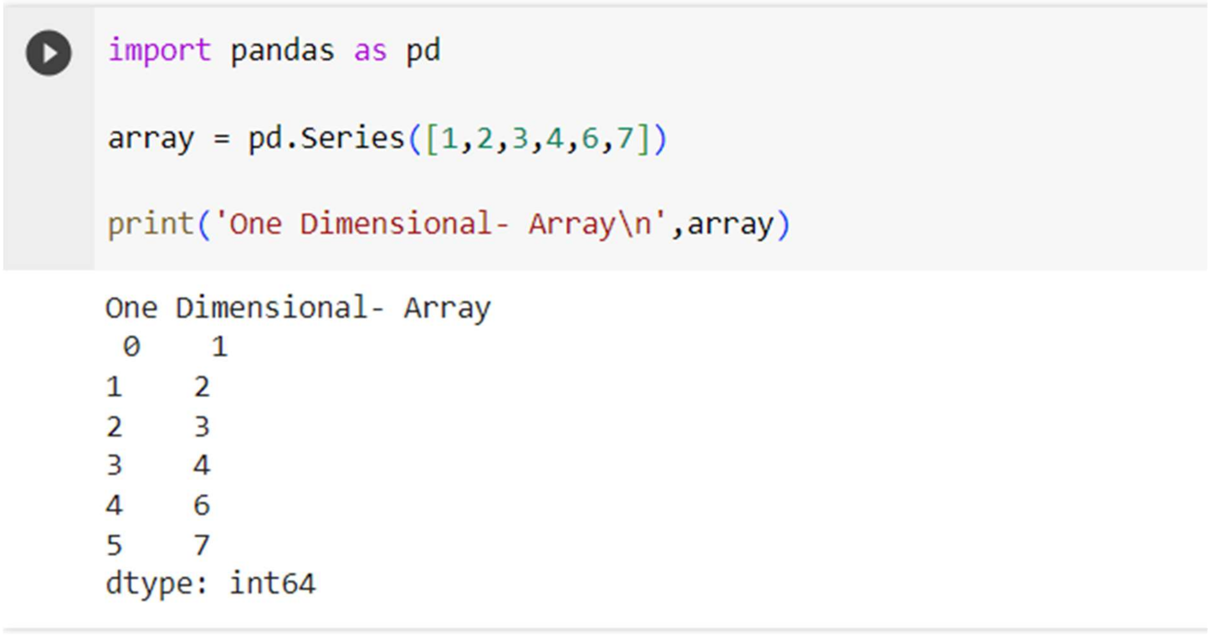
Date :

Aim : To Write a Python Program to Creating & Printing a One-Dimensional Pandas Series.

Algorithm :

- Start the program.
- Import the Pandas library as 'pd' to work with data structures.
- Create a one-dimensional Pandas Series named 'array' containing a sequence of numbers.
- Print the 'array' Pandas Series, displaying its one-dimensional structure.
- End the program.

Source Code & Output :



```
import pandas as pd

array = pd.Series([1,2,3,4,6,7])

print('One Dimensional- Array\n',array)
```

```
One Dimensional- Array
 0    1
1    2
2    3
3    4
4    6
5    7
dtype: int64
```

Result :

The Python program for Creating & Printing a One-Dimensional Pandas Series is compiled & executed successfully.

Teacher’s signature

Performing Basic Operations on Pandas Series

Exp. No. : 32

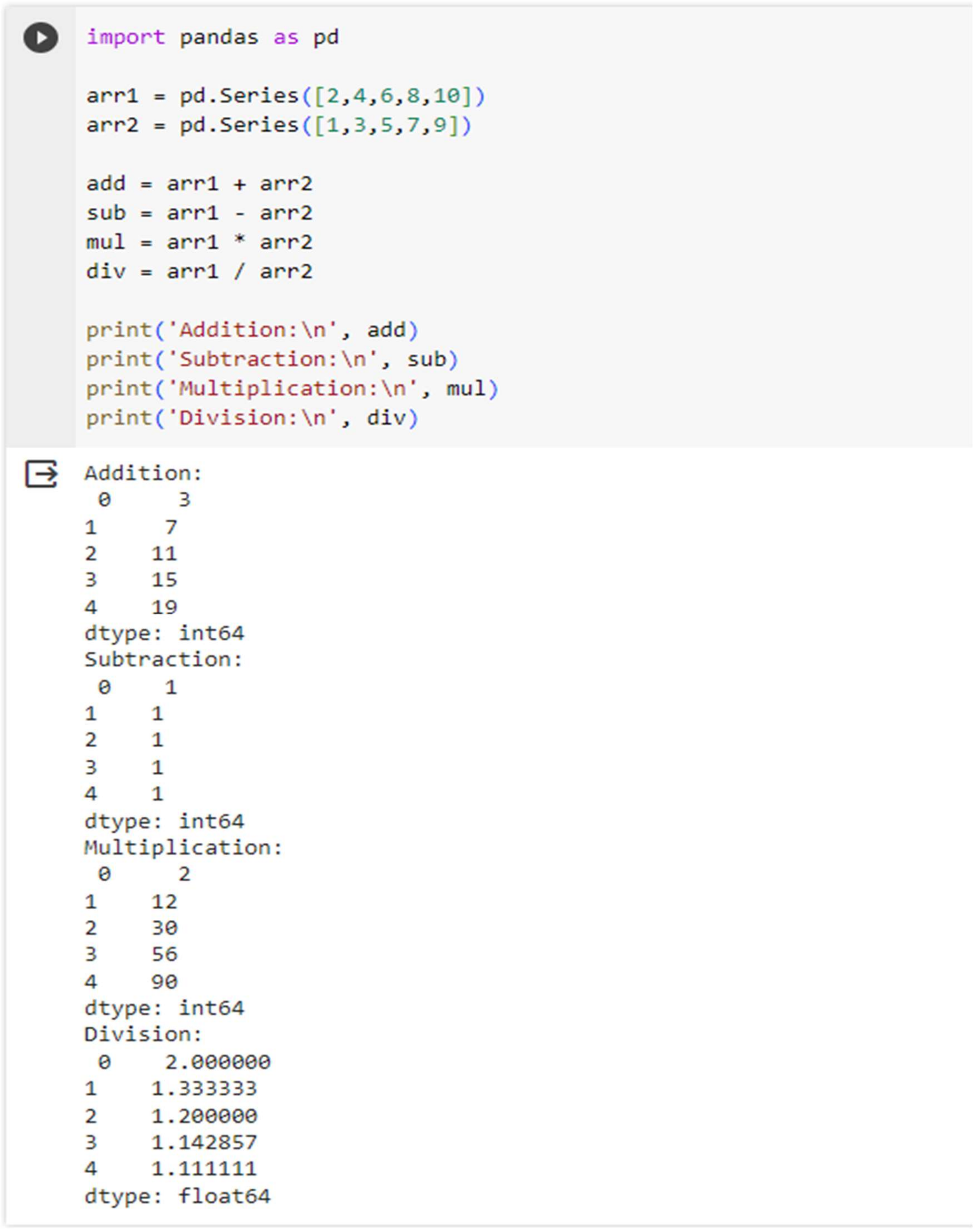
Date :

Aim : To Write a Python Program to Performing Basic Operations on Pandas Series.

Algorithm :

- Start the program.
- Import the Pandas library as 'pd' to work with data structures.
- Create two Pandas Series, 'arr1' and 'arr2', each containing a sequence of numbers.
- Perform the following operations element-wise on the Pandas Series:
    - Addition: Create a new Series 'add' by adding 'arr1' and 'arr2' together.
    - Subtraction: Create a new Series 'sub' by subtracting 'arr2' from 'arr1'.
    - Multiplication: Create a new Series 'mul' by multiplying 'arr1' and 'arr2' element-wise.
    - Division: Create a new Series 'div' by dividing 'arr1' by 'arr2' element-wise.
- Print the results of each operation to display the addition, subtraction, multiplication, and division of the Pandas Series.
- End the program.

Source Code & Output :



```python
import pandas as pd

arr1 = pd.Series([2,4,6,8,10])
arr2 = pd.Series([1,3,5,7,9])

add = arr1 + arr2
sub = arr1 - arr2
mul = arr1 * arr2
div = arr1 / arr2

print('Addition:\n', add)
print('Subtraction:\n', sub)
print('Multiplication:\n', mul)
print('Division:\n', div)
```

```
Addition:
 0     3
1     7
2    11
3    15
4    19
dtype: int64
Subtraction:
 0    1
1    1
2    1
3    1
4    1
dtype: int64
Multiplication:
 0     2
1    12
2    30
3    56
4    90
dtype: int64
Division:
 0    2.000000
1    1.333333
2    1.200000
3    1.142857
4    1.111111
dtype: float64
```

Result :

The Python program for Performing Basic Operations on Pandas Series is compiled & executed successfully.

Teacher's signature

Sorting a Pandas Series

Exp. No. : 33

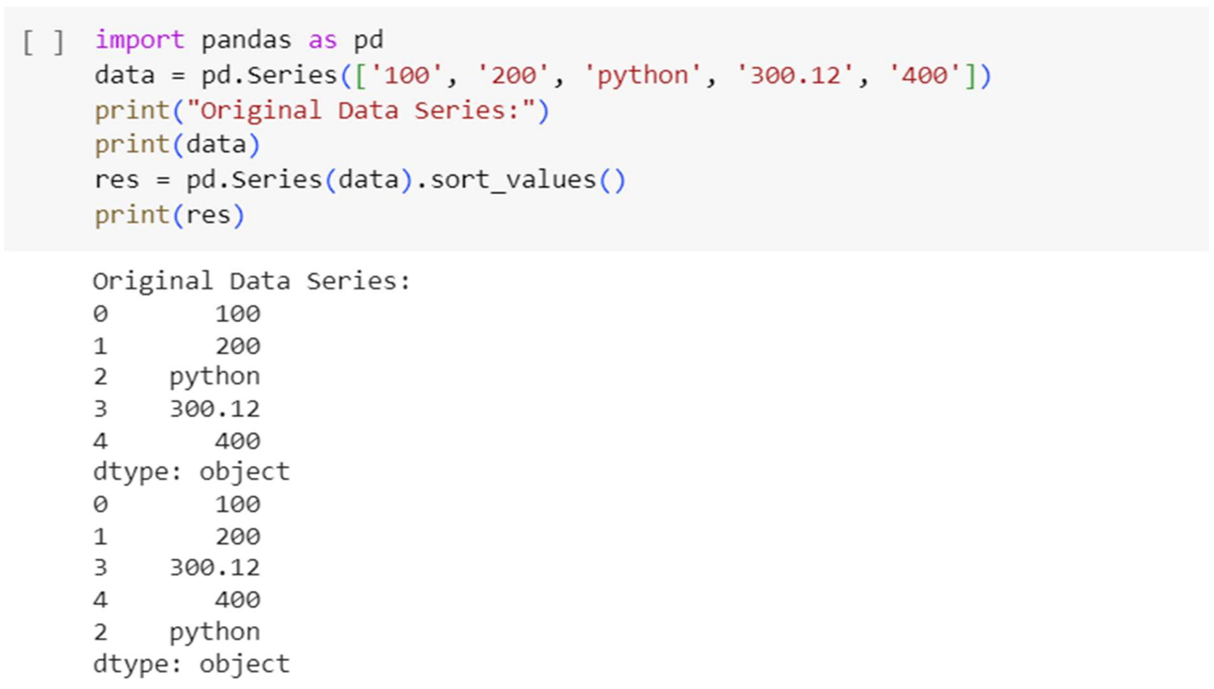
Date :

Aim : To Write a Python Program to Find the Addition of Two Numbers.

Algorithm :

- Start the program.
- Import the Pandas library as 'pd' to work with data structures.
- Create a Pandas Series named 'data' containing a sequence of values, including integers, strings, and floating-point numbers.
- Print the original 'data' Pandas Series to display its contents.
- Create a new Pandas Series 'res' by sorting the values of the original 'data' Series using ‘.sort_values()’. This sorts the values in ascending order, and the sorted Series is assigned to 'res'.
- Print the sorted 'res' Series, which displays the values from 'data' in ascending order.
- End the program.

Source Code & Output :



```
[ ]  import pandas as pd
     data = pd.Series(['100', '200', 'python', '300.12', '400'])
     print("Original Data Series:")
     print(data)
     res = pd.Series(data).sort_values()
     print(res)
```

```
Original Data Series:
0       100
1       200
2    python
3    300.12
4       400
dtype: object
0       100
1       200
3    300.12
4       400
2    python
dtype: object
```

Result :

The Python program for Adding Two Numbers is compiled & executed successfully.

Teacher’s signature

Creating a Pandas DataFrame from a Dictionary

Exp. No. : 34

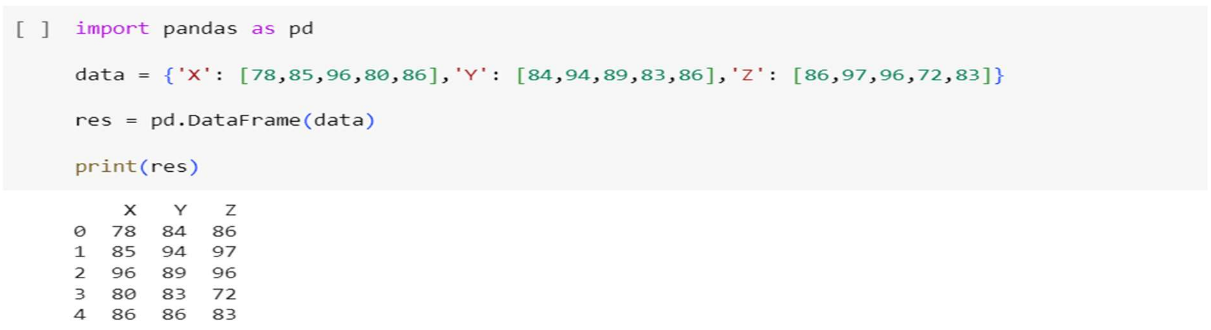
Date :

Aim : To Write a Python Program to Creating a Pandas DataFrame from a Dictionary.

Algorithm :

- Start the program.
- Import the Pandas library as 'pd' to work with data structures.
- Create a Python dictionary 'data' containing key-value pairs where each key represents a column name ('X', 'Y', 'Z') and the corresponding values are lists of numerical data.
- Use the 'data' dictionary to create a Pandas DataFrame 'res' by calling ‘pd.DataFrame(data)’.
- Print the 'res' DataFrame to display the data in tabular form, with columns 'X', 'Y', and 'Z'.
- End the program.

Source Code & Output :



```
[ ]  import pandas as pd

     data = {'X': [78,85,96,80,86],'Y': [84,94,89,83,86],'Z': [86,97,96,72,83]}

     res = pd.DataFrame(data)

     print(res)
```

```
    X   Y   Z
0  78  84  86
1  85  94  97
2  96  89  96
3  80  83  72
4  86  86  83
```

Result :

The Python program for Creating a Pandas DataFrame from a Dictionary is compiled & executed successfully.

Teacher’s signature

Concatenating Pandas DataFrames Vertically

Exp. No. : 35

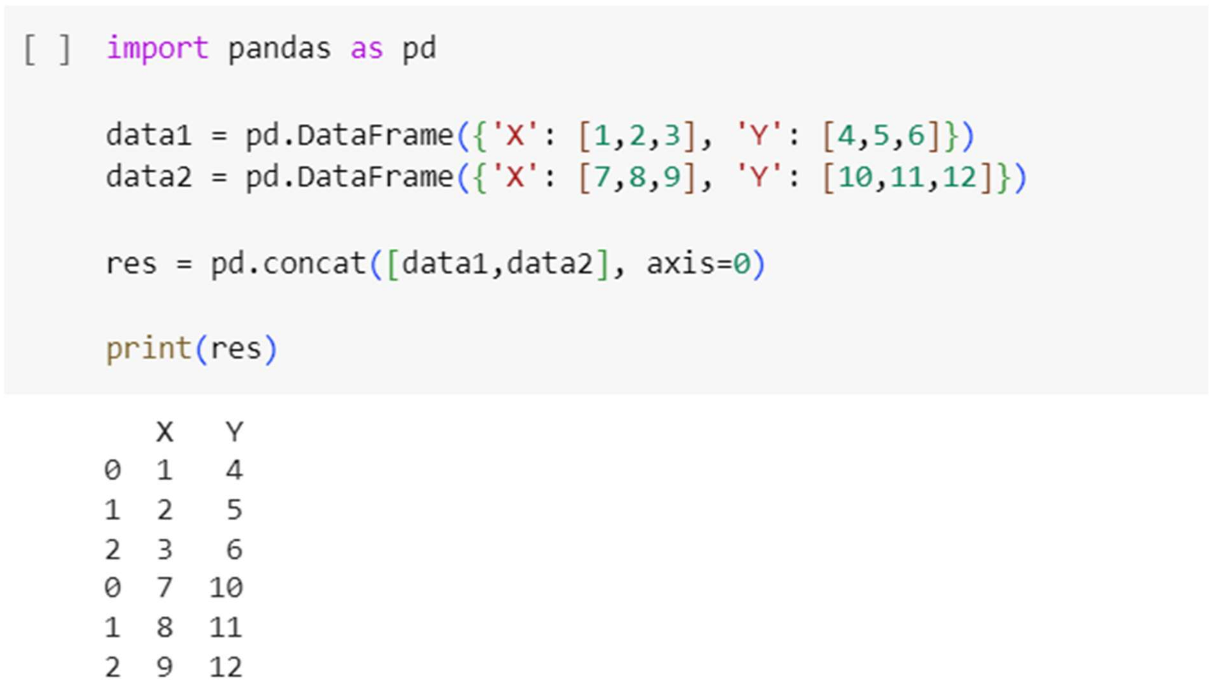
Date :

Aim : To Write a Python Program to Concatenating Pandas DataFrames Vertically.

Algorithm :

- Start the program.
- Import the Pandas library as 'pd' to work with data structures.
- Create two Pandas DataFrames, 'data1' and 'data2,' where each DataFrame has columns 'X' and 'Y' with different sets of data.
- Concatenate the 'data1' and 'data2' DataFrames vertically (along the rows) using ‘pd.concat([data1, data2], axis=0)’. This combines the two DataFrames into a single DataFrame 'res' with the same column names 'X' and 'Y.
- Print the 'res' DataFrame to display the concatenated data with all rows from 'data1' followed by all rows from 'data2'.
- End the program.

Source Code & Output :



```
[ ]  import pandas as pd

     data1 = pd.DataFrame({'X': [1,2,3], 'Y': [4,5,6]})
     data2 = pd.DataFrame({'X': [7,8,9], 'Y': [10,11,12]})

     res = pd.concat([data1,data2], axis=0)

     print(res)
```

```
   X   Y
0  1   4
1  2   5
2  3   6
0  7  10
1  8  11
2  9  12
```

Teacher's signature

Result :

The Python program for Adding Two Numbers is compiled & executed successfully.

Teacher's signature

Appending DataFrames in Pandas

Exp. No. : 36

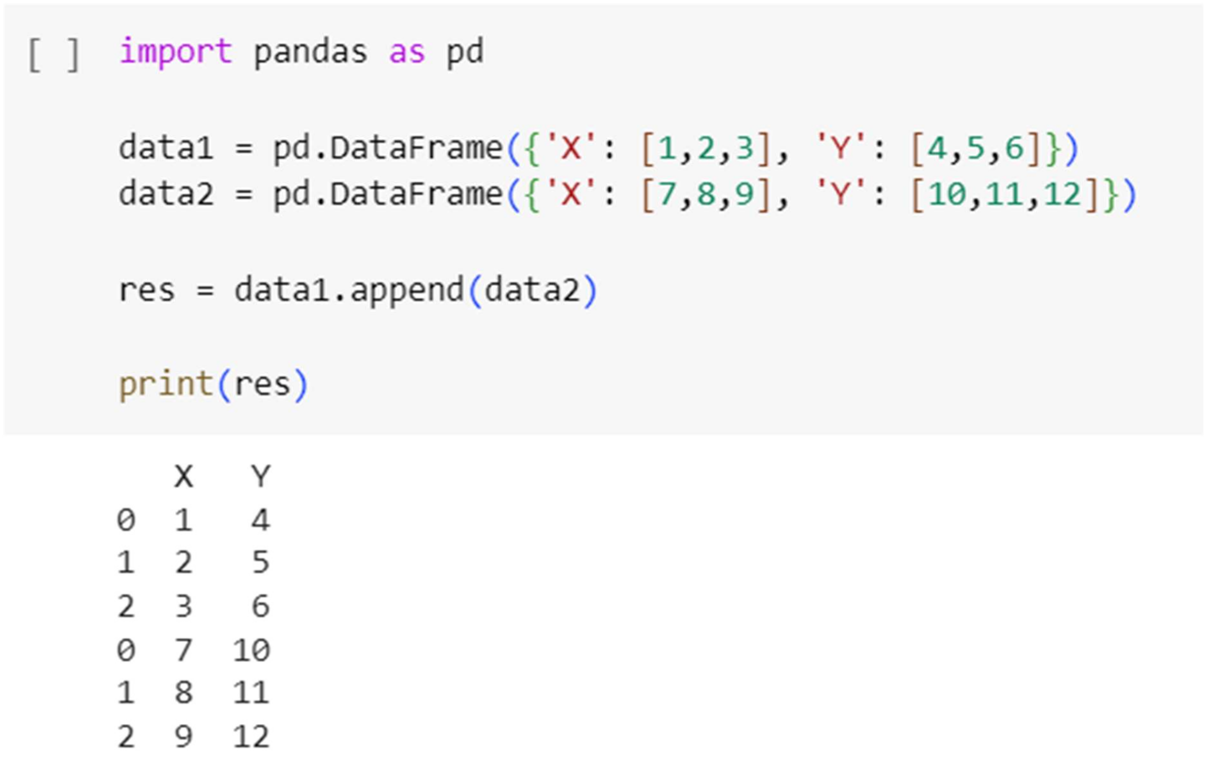
Date :

Aim : To Write a Python Program to Appending DataFrames in Pandas.

Algorithm :

- Start the program.
- Import the Pandas library as 'pd' to work with data structures.
- Create two Pandas DataFrames, 'data1' and 'data2,' where each DataFrame has columns 'X' and 'Y' with different sets of data.
- Append 'data2' to 'data1' using the ‘.append()’ method, which creates a new DataFrame 'res' containing all rows from both 'data1' and 'data2'.
- Print the 'res' DataFrame to display the appended data with all rows from 'data1' followed by all rows from 'data2'.
- End the program.

Source Code & Output :



```
[ ]  import pandas as pd

     data1 = pd.DataFrame({'X': [1,2,3], 'Y': [4,5,6]})
     data2 = pd.DataFrame({'X': [7,8,9], 'Y': [10,11,12]})

     res = data1.append(data2)

     print(res)
```

```
   X   Y
0  1   4
1  2   5
2  3   6
0  7  10
1  8  11
2  9  12
```

Teacher’s signature

Result :

The Python program for Appending DataFrames in Pandas is compiled & executed successfully.

Teacher’s signature

Reading and Exploring a CSV File Using Pandas

Exp. No. : 37

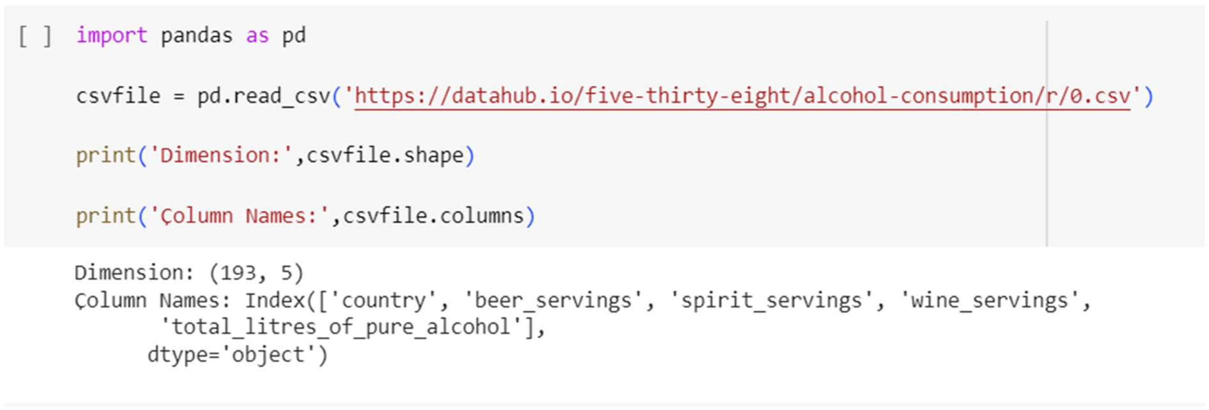
Date :

Aim : To Write a Python Program to Reading & Exploring a CSV File using Pandas.

Algorithm :

- Start the program.
- Import the Pandas library as 'pd' to work with data structures.
- Use Pandas to read a CSV file from a specified URL ('https://datahub.io/five-thirty-eight/alcohol-consumption/r/0.csv') and store it in a DataFrame named 'csvfile'.
- Print the dimensions of the DataFrame using ‘csvfile.shape’. This provides the number of rows and columns in the CSV data.
- Print the column names of the DataFrame using ‘csvfile.columns’. This displays the names of all columns in the CSV data.
- End the program.

Source Code & Output :



```
[ ]  import pandas as pd

     csvfile = pd.read_csv('https://datahub.io/five-thirty-eight/alcohol-consumption/r/0.csv')

     print('Dimension:',csvfile.shape)

     print('Column Names:',csvfile.columns)
```

```
Dimension: (193, 5)
Column Names: Index(['country', 'beer_servings', 'spirit_servings', 'wine_servings',
       'total_litres_of_pure_alcohol'],
      dtype='object')
```

Result :

The Python program for Reading & Exploring a CSV File using Pandas is compiled & executed successfully.

Teacher’s signature

Reading and Cleaning a CSV File Using Pandas

Exp. No. : 38

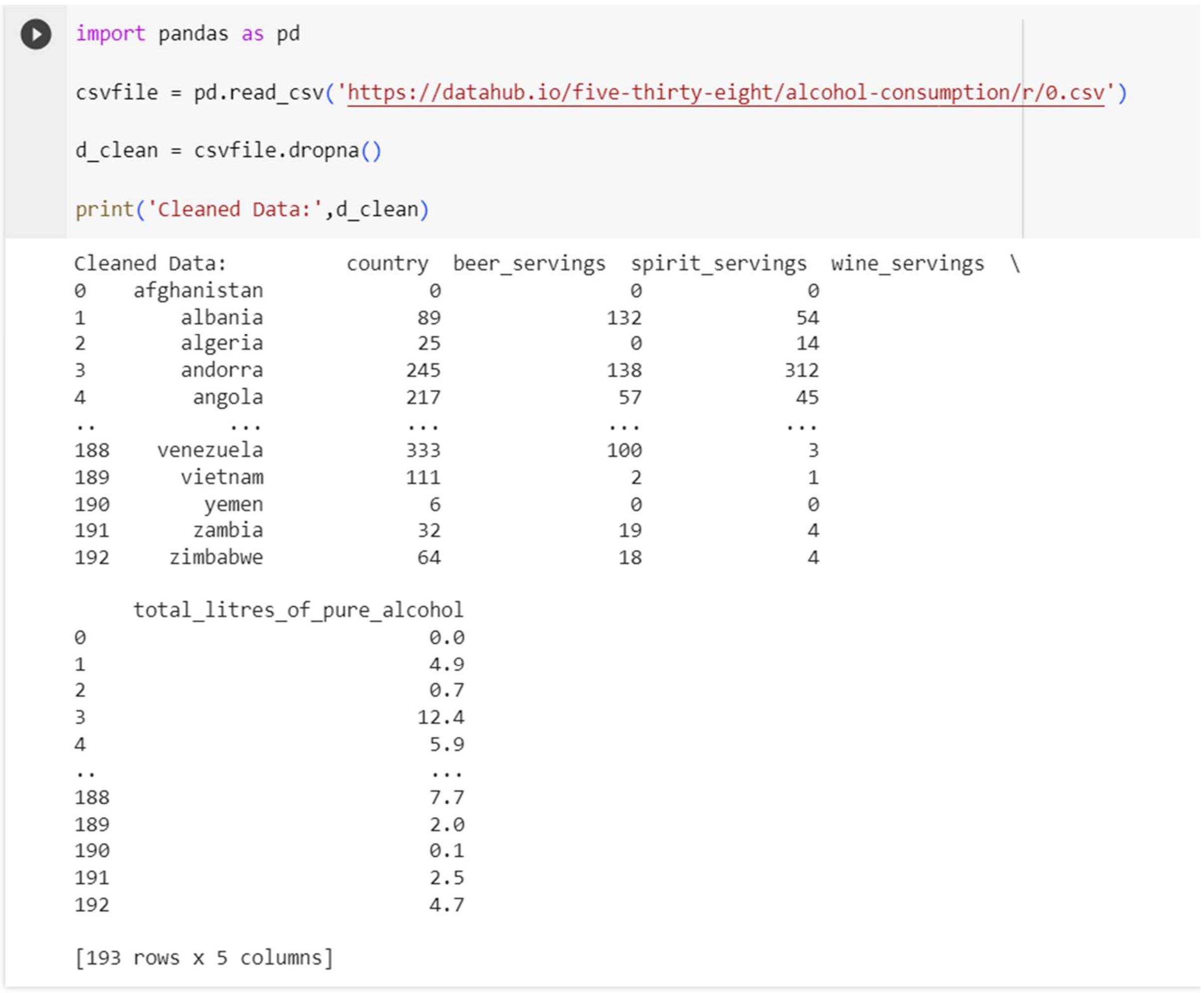
Date :

Aim : To Write a Python Program to Reading & Cleaning a CSV File Using Pandas.

Algorithm :

- Start the program.
- Import the Pandas library as 'pd' to work with data structures.
- Use Pandas to read a CSV file from a specified URL ('https://datahub.io/five-thirty-eight/alcohol-consumption/r/0.csv') and store it in a DataFrame named 'csvfile'.
- Use the ‘.dropna()’ method to remove rows with missing values (NaN) from the DataFrame. Store the cleaned DataFrame in 'd_clean'.
- Print the 'd_clean' DataFrame, which contains the data after removing rows with missing values.
- End the program.

Source Code & Output :



```python
import pandas as pd

csvfile = pd.read_csv('https://datahub.io/five-thirty-eight/alcohol-consumption/r/0.csv')

d_clean = csvfile.dropna()

print('Cleaned Data:',d_clean)
```

```
Cleaned Data:          country  beer_servings  spirit_servings  wine_servings  \
0    afghanistan              0                0              0   
1        albania             89              132             54   
2        algeria             25                0             14   
3        andorra            245              138            312   
4         angola            217               57             45   
..           ...            ...              ...            ...   
188    venezuela            333              100              3   
189      vietnam            111                2              1   
190        yemen              6                0              0   
191       zambia             32               19              4   
192     zimbabwe             64               18              4   

     total_litres_of_pure_alcohol  
0                             0.0  
1                             4.9  
2                             0.7  
3                            12.4  
4                             5.9  
..                            ...  
188                           7.7  
189                           2.0  
190                           0.1  
191                           2.5  
192                           4.7  

[193 rows x 5 columns]
```

Result :

The Python program for Reading & Cleaning a CSV File Using Pandas is compiled & executed successfully.

Teacher's signature

Generating Quarterly Dates Using Pandas

Exp. No. : 39

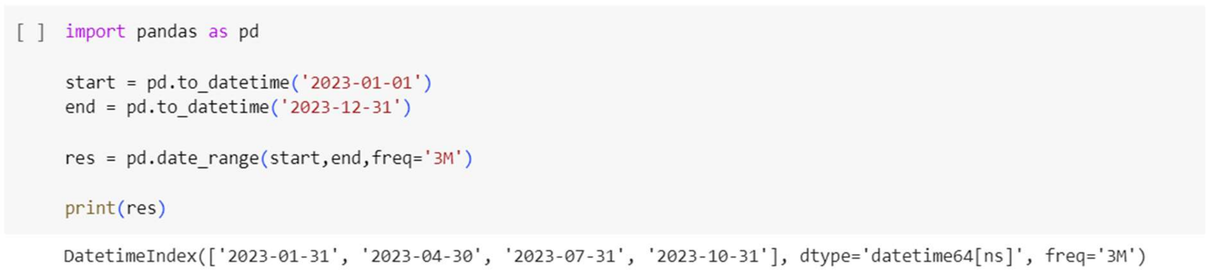
Date :

Aim : To Write a Python Program to Generating Quarterly Dates using Pandas.

Algorithm :

- Start the program.
- Import the Pandas library as 'pd' to work with data structures.
- Define the start date 'start' as '2023-01-01' and the end date 'end' as '2023-12-31' using ‘pd.to_datetime()’.
- Generate a date range 'res' using ‘pd.date_range(start, end, freq='3M')’. This creates a range of dates at 3-month intervals between 'start' and 'end'.
- Print the 'res' date range, which contains quarterly dates within the specified range.
- End the program.

Source Code & Output :



```
[ ]  import pandas as pd

     start = pd.to_datetime('2023-01-01')
     end = pd.to_datetime('2023-12-31')

     res = pd.date_range(start,end,freq='3M')

     print(res)
```

```
DatetimeIndex(['2023-01-31', '2023-04-30', '2023-07-31', '2023-10-31'], dtype='datetime64[ns]', freq='3M')
```

Result :

The Python program for Generating Quarterly Dates using Pandas is compiled & executed successfully.

Teacher’s signature