Roll No. 41 Exam Seat No._____

VIVEKANANDEDUCATION SOCIETY'S INSTITUTE OF TECHNOLOGY

HashuAdvani Memorial Complex, Collector's Colony, R. C. Marg,
Chembur, Mumbai – 400074. Contact No. 02261532532

Since 1962

CERTIFICATE

Certified that Mr./Miss ABHAY OMPRAKASH PRAJAPATI

of FYMCA/A has

satisfactorily completed a course of the necessary experiments in

Python Programming Lab under my supervision

in the Institute of Technology in the academic year 2023 - 2024.

Principal Head of Department

Lab In-charge Subject Teacher

V.E.S. Institute of Technology, Collector Colony,Chembur,
Mumbai, Maharashtra 400047
Department of M.C.A

INDEX

S. No.	Contents	Marks	Faculty Sign
1.	**To write, test, and debug Basic Python programs.** 1. Add Three Numbers 2. To Swap two No using third variable and without using third variable. 3. Calculate area of triangle 4. To Solve Quadratic equation 5. To use Bitwise operators 6. To compute compound interest given all the required values. 7. To generate a random number between 0 and 100 8. To display calendar for the January 2019 9. To add two binary numbers		
2.	**To implement Python programs with conditionals and loops** 1. To find all the prime numbers in the interval 0 to 100 2. To check if the given number is Armstrong no or not 3. To check if the given char is vowel or consonant 4. Write a Program to Take in the Marks of 3 Subjects and Display the Grade 5. To add two matrices 6. To convert month name to a number of days. 7. To check the validity of password input by users Validation: At least 1 letter between [a-z] and 1 letter between [A-Z].		

	At least 1 number between [0-9]. At least 1 character from [$#@]. Minimum length 6 characters. Maximum length 16 characters. 8. To check if a number is palindrome or not		
3.	**To implement Python programs using List, String, Set and Dictionary** 1. To merge two list and find second largest element in the list using bubble sort 2. To calculate the no of uppercase ,lowercase letters and digits in a string 3. To count the occurrences of each word in a given string sentence 4. To add key value pair to the dictionary and search and then delete the given key from the dictionary 5. Create one dictionary of 5 students with their name, address, age, class and marks of 5 subjects. Perform all the operations on the created dictionary 6. To concatenate two dictionaries and find sum of all values in dictionary 7. To add and remove elements from set and perform all the set operations like Union, Intersection, Difference and Symmetric Difference 8. Perform different operations on Tuple. 9. Write a Python program to count the elements in a list until an element is a tuple		
4.	**To implement programs on Python Functions and Modules** 1. To check whether string is palindrome or not using function recursion 2. To find Fibonacci series using recursion		

	3. To find binary equivalent of number using recursion 4. To use lambda function on list to generate filtered list, mapped list and reduced list 5. Convert the temperature in Celsius to Fahrenheit in list using anonymous function 6. To create modules in python and access functions of the module by importing it to another file/module. (Calculator program)		
5.	**To implement programs on OOP Concepts in python** 1. Python Program to Create a Class and Compute the Area and the Perimeter of the Circle 2. To Implement Multiple Inheritance in python 3. To Implement a program with same method name and multiple arguments 4. To Implement Operator Overloading in python. 5. Write a program which handles various exceptions in python		
6.	**To implement programs on Data Structures using Python** 1. To Create, Traverse, Insert and remove data using Linked List 2. Implementation of stacks 3. Implementation of Queue 4. Implementation of Dequeue		
7.	**To implement GUI programming and Database Connectivity** 1. To Design Login Page 2. To Design Student Information Form/Library management Form/Hospital Management Form 3. Implement Database connectivity For Login Page i.e. Connect		

	Login GUI with Sqlite3		
8.	**To implement Threads in Python** 1. To do design the program for starting the thread in python 2. Write a program to illustrate the concept of Synchronization 3. Write a program for creating multithreaded priority queue		
9.	**To implement NumPy library in Python** 1. Creating ndarray objects using array() in NumPy 2. Creating 2D arrays to implement Matrix Multiplication. 3. Program for Indexing and slicing in NumPy arrays. 4. To implement NumPy - Data Types		
10	**To implement Pandas library in Python** 1. Write a Pandas program to create and display a one-dimensional array-like object containing an array of data using Pandas module. 2. Write a Pandas program to convert a dictionary to a Pandas series. 3. Write a Pandas program to create a dataframe from a dictionary and display it. Sample data: {'X':[78,85,96,80,86], 'Y':[84,94,89,83,86],'Z':[86,97,96,72,83]} 4. Write a Pandas program to aggregate the two given dataframes along rows and assign all data. 5. Write a Pandas program to merge two given dataframes with different columns.		

Final Grade	Instructor Signature

Name of student: Abhay Omprakash Prajapati		
Roll no: 41	**Tutorial No: 1**	
Title of LAB Assignment: To write, test, and debug Basic Python programs.		
DOP: 25-09-2023	**DOS:02-10-2023**	
CO Mapped: Co1,Co2	**PO Mapped:** PO3 ,PO6	**Signature:**

1. Add Three Numbers:

```
num1 = 5
num2 = 8
num3 = 10
sum_of_numbers = num1 + num2 + num3
print("The sum of", num1, "+", num2, "+", num3, "is:", sum_of_numbers)
```

Output:

2. Swap two Numbers with and without a Third Variable:

```
num1 = 5
num2 = 8
num3 = 10
sum_of_numbers = num1 + num2 + num3
print("The sum of", num1, "+", num2, "+", num3, "is:", sum_of_numbers)
```

Output:

3. Calculate the Area of a Triangle:

```
base = 6
height = 8
area = 0.5 * base * height
print("The area of the triangle with base", base, "and height", height, "is:",
area)
```

Output:

4. Solve Quadratic Equation:

```
import math
```

```
a = 1
b = 5
c = 6

discriminant = b**2 - 4*a*c

if discriminant > 0:
    root1 = (-b + math.sqrt(discriminant)) / (2*a)
    root2 = (-b - math.sqrt(discriminant)) / (2*a)
    print("Two real roots: Root 1 =", root1, "Root 2 =", root2)
elif discriminant == 0:
    root = -b / (2*a)
    print("One real root:", root)
else:
    real_part = -b / (2*a)
    imaginary_part = math.sqrt(-discriminant) / (2*a)
    print("Complex roots: Root 1 =", real_part, "+", imaginary_part, "i and
Root 2 =", real_part, "-", imaginary_part, "i")
```

Output:

5. Use Bitwise Operators:

```
x = 5
y = 3
result_and = x & y
print("Bitwise AND:", result_and)

result_or = x | y
print("Bitwise OR:", result_or)

result_xor = x ^ y
print("Bitwise XOR:", result_xor)
```

Output:

6. Compute Compound Interest:

```python
# Task 6: Compute Compound Interest
principal = 1000
rate = 5
time = 3
n = 12  # Compounded annually

amount = principal * (1 + (rate / (100 * n))) ** (n * time)
interest = amount - principal
print("Principal Amount:", principal)
print("Rate of Interest:", rate)
print("Time (in years):", time)
print("Number of times interest is compounded per year:", n)
print("Amount after compound interest:", amount)
print("Interest earned:", interest)
```

7. Generate a Random Number between 0 and 100:

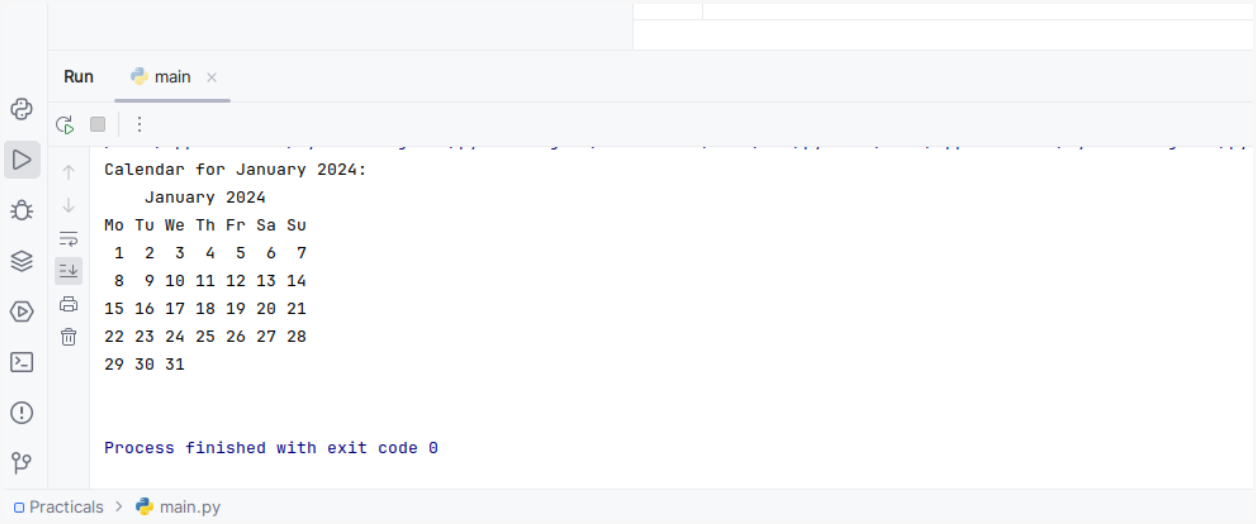
```python
import random

random_number = random.randint(0, 100)
print("Random Number between 0 and 100:", random_number)
```

Output:

8. Display Calendar for January 2024:

```
import calendar

year = 2024
month = 1
print("Calendar for January 2024:")
print(calendar.month(year, month))
```

Output:

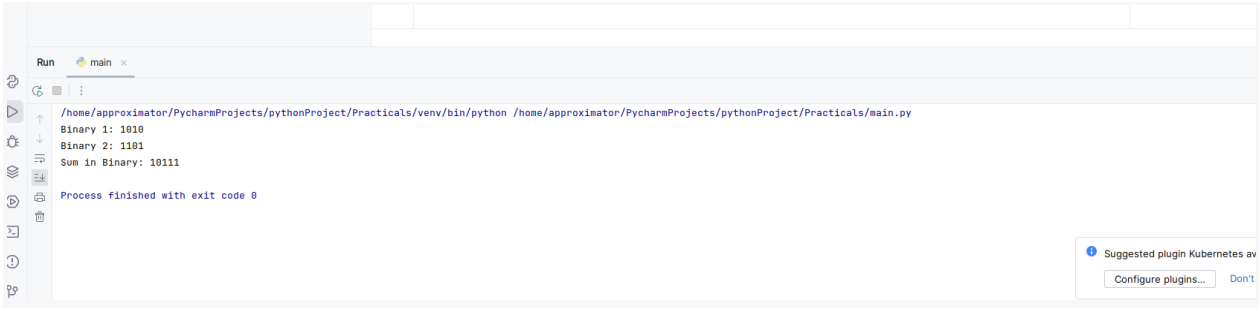


9. Add Two Binary Numbers:

```
binary1 = "1010"
binary2 = "1101"

decimal1 = int(binary1, 2)
decimal2 = int(binary2, 2)
result_decimal = decimal1 + decimal2
result_binary = bin(result_decimal).replace("0b", "")

print("Binary 1:", binary1)
```

```
print("Binary 2:", binary2)
print("Sum in Binary:", result_binary)
```

Output:



Name of student: Abhay Omprakash Prajapati		
Roll no: 41	**Tutorial No: 2**	
Title of LAB Assignment: To implement Python programs with conditionals and loops		
DOP: 25-09-2023	**DOS:02-10-2023**	
CO Mapped: Co1,Co2	**PO Mapped:** PO3 ,PO6	**Signature:**

1. To find all the prime numbers in the interval 0 to 100

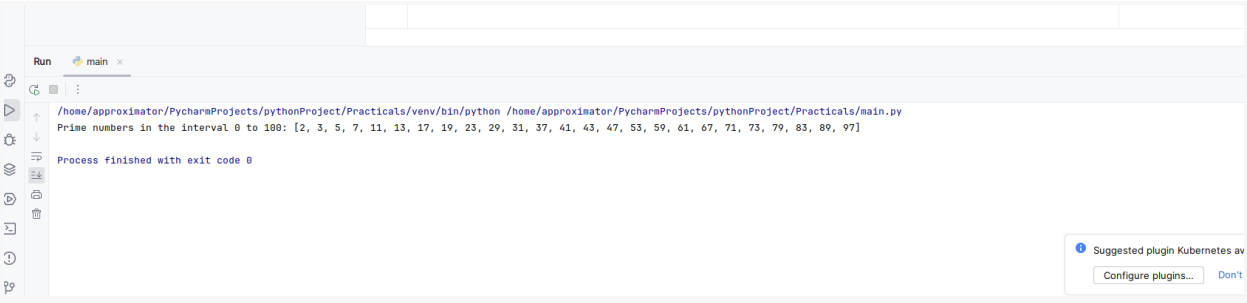
Aim: To find and display all prime numbers in the range from 0 to 100.
Theory: Prime numbers are natural numbers greater than 1 that have no positive divisors other than 1 and themselves. In this task, we'll iterate through numbers from 2 to 100 and check if each number is prime or not. We'll display all the prime numbers found in the given interval.
Code:

```
def is_prime(num):
    if num <= 1:
        return False
    for i in range(2, int(num ** 0.5) + 1):
```

```
        if num % i == 0:
            return False
    return True

primes = [num for num in range(2, 101) if is_prime(num)]
print("Prime numbers in the interval 0 to 100:", primes)
```

Conclusion: We checked if the given number is an Armstrong number and provided the result.

Output:

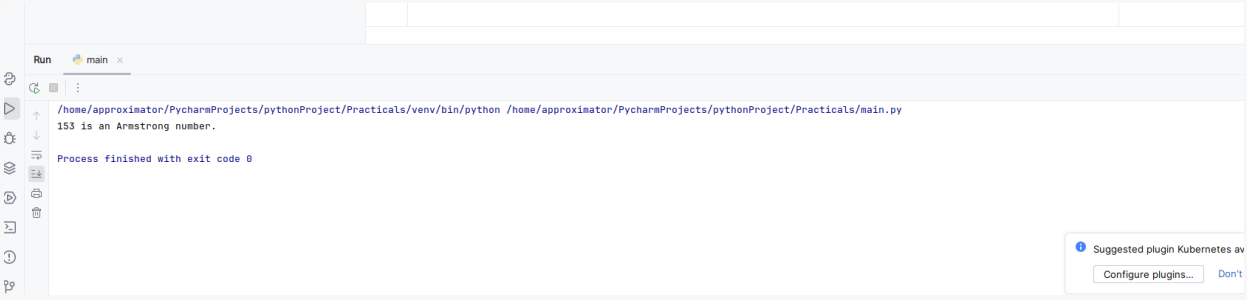


2. To check if the given number is Armstrong number or not

Aim: To check if a given number is an Armstrong number.
Theory: An Armstrong number (or narcissistic number) is a number that is equal to the sum of its own digits raised to the power of the number of digits. In this task, we'll check if a given number is an Armstrong number or not.
Code:

```
def is_armstrong(num):
    num_str = str(num)
    num_digits = len(num_str)
    total = sum(int(digit) ** num_digits for digit in num_str)
    return num == total

number = 153
if is_armstrong(number):
    print(number, "is an Armstrong number.")
else:
    print(number, "is not an Armstrong number.")
```

Conclusion: We checked if the given number is an Armstrong number and provided the result.

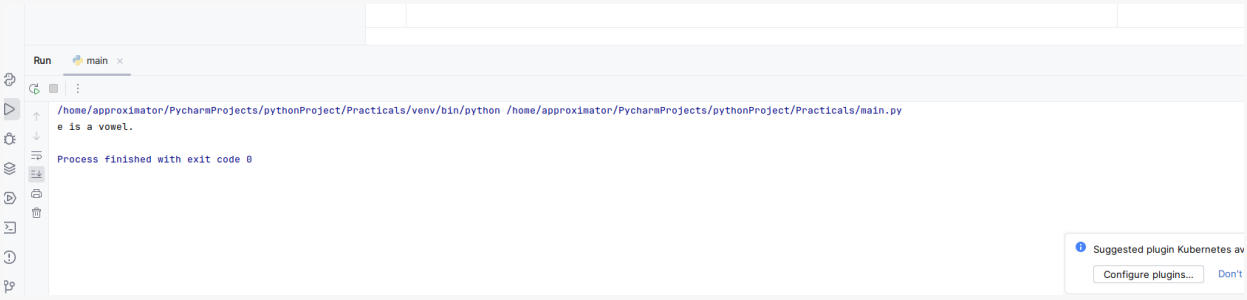


3. To check if the given character is a vowel or consonant

Aim: To check if a given character is a vowel or consonant.
Theory: Vowels are the letters 'a', 'e', 'i', 'o', and 'u'. In this task, we'll check if a given character is a vowel or a consonant.
Code:

```
char = 'e'  # You can change this to any character you want to check
if char.lower() in ('a', 'e', 'i', 'o', 'u'):
    print(char, "is a vowel.")
else:
    print(char, "is a consonant.")
```

Conclusion: We checked if the given character is a vowel or a consonant and provided the result.



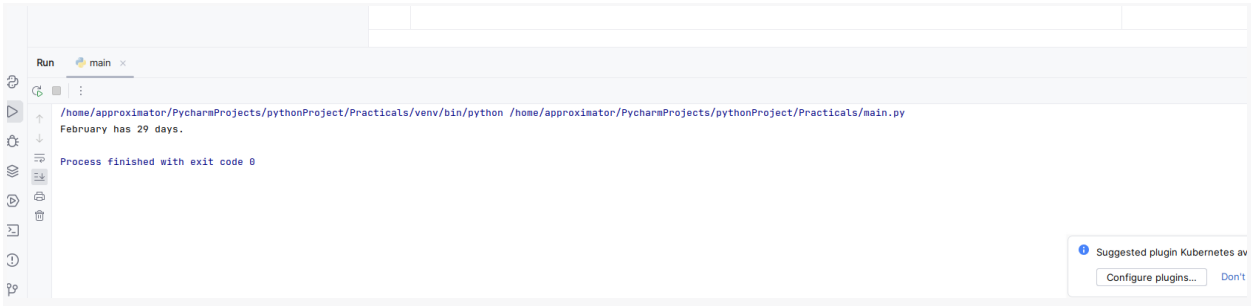
4. To convert a month to a number of days

Aim: To convert a month to the number of days it contains.
Theory: Different months have different numbers of days. In this task, we'll convert a given month to the number of days it contains, considering both common years and leap years.
Code:

```
def month_to_days(month, is_leap_year=False):
    month = month.lower()
    if month in ("january", "march", "may", "july", "august", "october",
"december"):
        return 31
    elif month in ("april", "june", "september", "november"):
        return 30
    elif month == "february":
        return 29 if is_leap_year else 28
    else:
        return None  # Invalid month

given_month = "February"  # You can change this to any month you want to check
is_leap = True  # You can change this to False for a non-leap year
days = month_to_days(given_month, is_leap)
if days is not None:
    print(given_month, "has", days, "days.")
else:
    print("Invalid month.")
```

Conclusion: We converted a given month to the number of days it contains and provided the result.

Output:



5. To check if a number is a palindrome or not

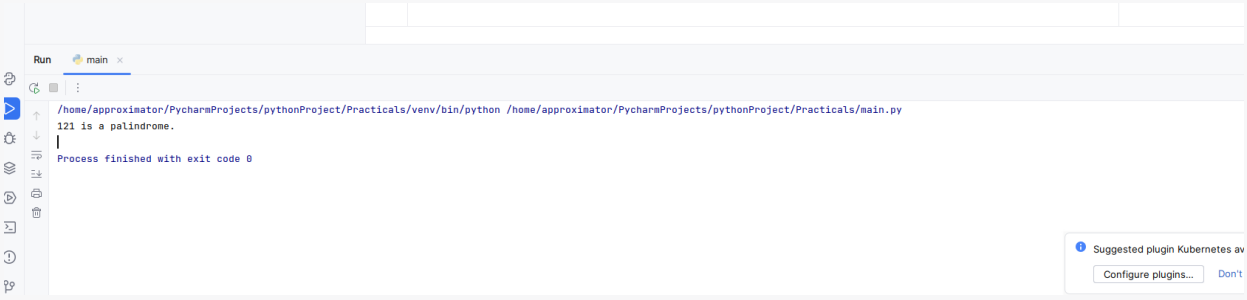
Aim: To check if a given number is a palindrome.
Theory: A palindrome is a number that remains the same when its digits are reversed. In this task, we'll check if a given number is a palindrome or not.
Code:

```
def is_palindrome(number):
    num_str = str(number)
```

```
    return num_str == num_str[::-1]

given_number = 121  # You can change this to any number you want to check
if is_palindrome(given_number):
    print(given_number, "is a palindrome.")
else:
    print(given_number, "is not a palindrome.")
```

Conclusion: We checked if the given number is a palindrome and provided the result.

Output:



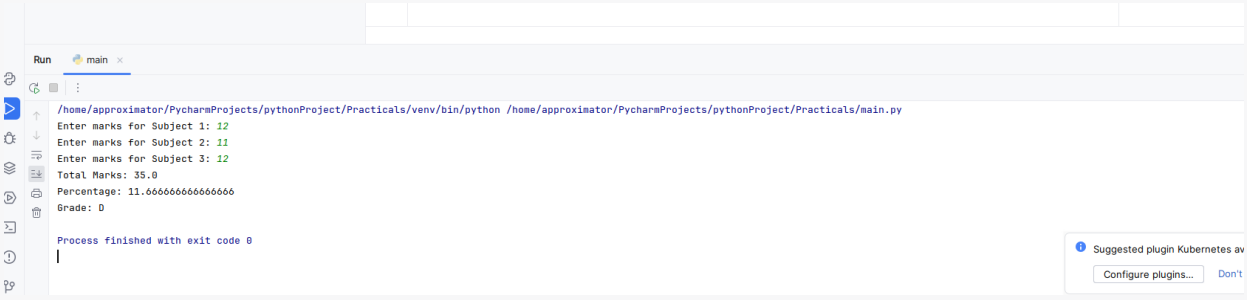
6. Program to Take in the Marks of 3 Subjects and Display the Grade

Aim: To calculate the grade based on the marks of 3 subjects.
Theory: In this task, we'll take marks for 3 subjects as input and then calculate the average percentage. Based on the percentage, we'll assign a grade.
Code:

```
subject1 = float(input("Enter marks for Subject 1: "))
subject2 = float(input("Enter marks for Subject 2: "))
subject3 = float(input("Enter marks for Subject 3: "))

total_marks = subject1 + subject2 + subject3
percentage = (total_marks / 300) * 100

if percentage >= 90:
    grade = "A+"
elif percentage >= 80:
    grade = "A"
elif percentage >= 70:
    grade = "B"
elif percentage >= 60:
    grade = "C"
else:
    grade = "D"
```

```
print("Total Marks:", total_marks)
print("Percentage:", percentage)
print("Grade:", grade)
```

Conclusion: We calculated the grade based on the marks of 3 subjects and displayed the result.
Output:



7. To add two matrices

Aim: To add two matrices.
Theory: In this task, we'll add two matrices of the same dimensions. Matrix addition involves adding corresponding elements of two matrices to form a new matrix.
Code:

```
matrix1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
matrix2 = [[9, 8, 7], [6, 5, 4], [3, 2, 1]]

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

for i in range(len(matrix1)):
    for j in range(len(matrix1[0])):
        result[i][j] = matrix1[i][j] + matrix2[i][j]


print("Matrix 1:")
for row in matrix1:
    print(row)

print("Matrix 2:")
for row in matrix2:
    print(row)

print("Result Matrix:")
for row in result:
```

```
    print(row)
```

Output:

8. To check the validity of a password input by users

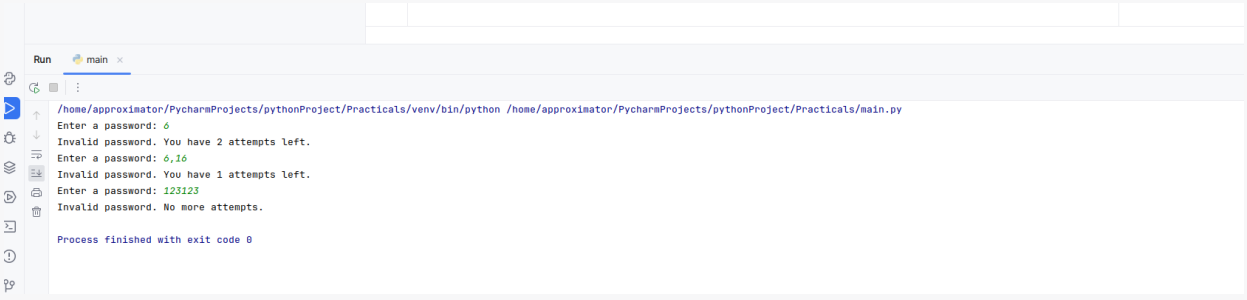
Aim: To check the validity of a user-entered password.
Theory: In this task, we'll validate a password based on the specified criteria. The password must contain at least 1 lowercase letter, 1 uppercase letter, 1 digit, and 1 special character from the set [$#@]. It should also have a minimum length of 6 characters and a maximum length of 16 characters. The user has 3 chances to enter a valid password.
Code:

Code:

```
import re

attempts = 3

while attempts > 0:
    password = input("Enter a password: ")
    if re.match(r"^(?=.*[a-z])(?=.*[A-Z])(?=.*[0-9])(?=.*[$#@]).{6,16}$",
password):
        print("Password is valid.")
        break
    else:
        attempts -= 1
        if attempts > 0:
            print("Invalid password. You have", attempts, "attempts left.")
        else:
            print("Invalid password. No more attempts.")
```

Conclusion: We checked the validity of a user-entered password and provided the result within 3 chances.

Output:



Name of student: Abhay Omprakash Prajapati		
Roll no: 41	**Tutorial No: 3**	
Title of LAB Assignment: To implement Python programs using List, String, Set and Dictionary		
DOP: 25-09-2023	**DOS:02-10-2023**	
CO Mapped: Co1,Co2	**PO Mapped:** PO3 ,PO6	**Signature:**

1. Merge two lists and find the second largest element using bubble sort

Aim: To merge two lists and find the second largest element using bubble sort.
Theory: In this task, we'll merge two lists into one and then use the bubble sort algorithm to find the second largest element in the merged list.
Code:

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
```

```
list1 = [3, 9, 7]
list2 = [6, 5, 4]
merged_list = list1 + list2
bubble_sort(merged_list)
second_largest = merged_list[-2]
print("Merged List:", merged_list)
print("Second Largest Element:", second_largest)
```

Conclusion: We successfully merged two lists and found the second largest element using bubble sort.

Output:

2. Calculate the number of uppercase, lowercase letters, and digits in a string

Aim: To calculate the number of uppercase, lowercase letters, and digits in a given string.

Theory: In this task, we'll iterate through the characters in a string and count the number of uppercase letters, lowercase letters, and digits.

Code:

```
input_string = "Hello World 123"

upper_count = sum(1 for char in input_string if char.isupper())
lower_count = sum(1 for char in input_string if char.islower())
digit_count = sum(1 for char in input_string if char.isdigit())

print("Uppercase Letters:", upper_count)
print("Lowercase Letters:", lower_count)
print("Digits:", digit_count)
```

Conclusion: We successfully counted the number of uppercase, lowercase letters, and digits in the given string.

Output:

3. Count the occurrences of each word in a given string sentence

Aim: To count the occurrences of each word in a given string sentence.
Theory: In this task, we'll tokenize the string into words, count the occurrences of each word, and store the results in a dictionary.
Code:

```
input_sentence = "This is a simple sentence. This is another sentence."

words = input_sentence.split()
word_count = {}

for word in words:
    word = word.lower()
    word_count[word] = word_count.get(word, 0) + 1

print("Word Count:")
for word, count in word_count.items():
    print(f"{word}: {count}")
```

Conclusion: We successfully counted the occurrences of each word in the given string sentence.

Output:

4. Add a key-value pair to a dictionary, search for a given key, and then delete the key

Aim: To add a key-value pair to a dictionary, search for a given key, and then delete the key.

Theory: In this task, we'll demonstrate how to add a key-value pair to a dictionary, search for a specific key, and delete that key if found.

Code:

```
sample_dict = {"name": "John", "age": 30, "city": "New York"}

sample_dict["gender"] = "Male"
print("After Adding:", sample_dict)

search_key = "age"
if search_key in sample_dict:
    print(f"{search_key} found. Value: {sample_dict[search_key]}")
else:
    print(f"{search_key} not found.")

delete_key = "city"
if delete_key in sample_dict:
    del sample_dict[delete_key]
    print(f"{delete_key} deleted. New Dictionary: {sample_dict}")
else:
    print(f"{delete_key} not found, no deletion.")
```

Conclusion: We successfully added a key-value pair, searched for a key, and deleted a key in the dictionary.

Output:

5. Concatenate two dictionaries and find the sum of all values in the resulting dictionary

Aim: To concatenate two dictionaries and find the sum of all values in the resulting dictionary.

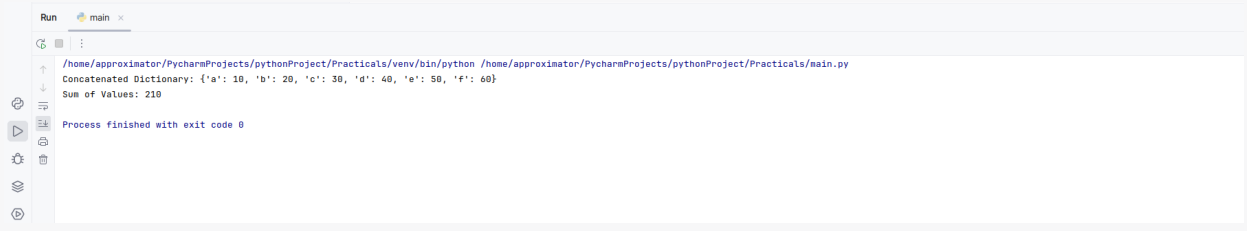
Theory: In this task, we'll merge two dictionaries into one, and then calculate the sum of all values in the merged dictionary.

Code:

```
dict1 = {'a': 10, 'b': 20, 'c': 30}
dict2 = {'d': 40, 'e': 50, 'f': 60}

concatenated_dict = {**dict1, **dict2}
total_sum = sum(concatenated_dict.values())

print("Concatenated Dictionary:", concatenated_dict)
print("Sum of Values:", total_sum)
```

Conclusion: We successfully concatenated two dictionaries and found the sum of all values in the merged dictionary.

Output:



6. Add and remove elements from a set and perform all set operations, such as union, intersection, difference, and symmetric difference

Aim: To perform various operations on sets, including adding and removing elements, and set operations.

Theory: In this task, we'll create two sets, add and remove elements, and perform set operations such as union, intersection, difference, and symmetric difference.

Code:

```
set1 = {1, 2, 3}
set2 = {3, 4, 5}

set1.add(6)

set2.remove(4)

union_set = set1 | set2
intersection_set = set1 & set2
difference_set = set1 - set2
symmetric_difference_set = set1 ^ set2

print("Set 1:", set1)
```

```
print("Set 2:", set2)
print("Union:", union_set)
print("Intersection:", intersection_set)
print("Difference:", difference_set)
print("Symmetric Difference:", symmetric_difference_set)
```

Conclusion: We successfully performed various operations on sets and set operations.
Output:

7. Perform different operations on tuples

Aim: To perform different operations on tuples.
Theory: In this task, we'll demonstrate various operations on tuples, including creating a tuple, accessing elements, finding the length, and checking for the presence of an element.
Code:

```
my_tuple = (1, 2, 3, 4, 5)

first_element = my_tuple[0]
last_element = my_tuple[-1]

tuple_length = len(my_tuple)

is_present = 3 in my_tuple

print("Tuple:", my_tuple)
print("First Element:", first_element)
print("Last Element:", last_element)
print("Tuple Length:", tuple_length)
print("Is 3 in Tuple:", is_present)
```

Conclusion: We performed various operations on tuples, including creation, element access, length calculation, and element presence check.
Output:

8. Count the elements in a list until an element is a tuple

Aim: To count the elements in a list until an element is a tuple.
Theory: In this task, we'll iterate through a list and count the elements until we encounter a tuple. We'll then stop counting.
Code:

```
my_list = [1, 2, 3, 'a', 'b', (4, 5), 6, 7]

count_until_tuple = 0
for element in my_list:
    if isinstance(element, tuple):
        break
    count_until_tuple += 1

print("List:", my_list)
print("Count Until Tuple:", count_until_tuple)
```

Conclusion: We counted the elements in the list until an element of the tuple type was encountered.
Output:

Name of student: Abhay Omprakash Prajapati		
Roll no: 41	**Tutorial No: 4**	
Title of LAB Assignment: To write, test, and debug Basic Python programs.		
DOP: 25-09-2023	**DOS:02-10-2023**	
CO Mapped: Co1,Co2	**PO Mapped:** PO3 ,PO6	**Signature:**

1. Check if a string is a palindrome using a recursive function

Aim: To check if a string is a palindrome using a recursive function.
Theory: In this task, we'll create a recursive function to check if a given string is a palindrome by comparing characters from the beginning and end of the string.
Code:

```
def is_palindrome(s):
    s = s.lower().replace(" ", "")
    if len(s) <= 1:
        return True
    if s[0] != s[-1]:
        return False
    return is_palindrome(s[1:-1])
```

```
input_string = "A man a plan a canal Panama"
if is_palindrome(input_string):
    print(f'"{input_string}" is a palindrome.')
else:
    print(f'"{input_string}" is not a palindrome.')
```

Conclusion: We successfully checked if the given string is a palindrome using a recursive function.
Output:

2. Find the Fibonacci sequence using recursion

Aim: To find the Fibonacci sequence using recursion.
Theory: In this task, we'll create a recursive function to find the Fibonacci sequence, where each number is the sum of the two preceding ones.
Code:

```
def fibonacci(n):
    if n <= 1:
        return n
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)

num_terms = 10
fib_sequence = [fibonacci(i) for i in range(num_terms)]
print("Fibonacci Sequence (first", num_terms, "terms):", fib_sequence)
```

Conclusion: We successfully found the Fibonacci sequence using recursion.
Output:

3. Find the binary equivalent of a number using recursion

Aim: To find the binary equivalent of a number using recursion.
Theory: In this task, we'll create a recursive function to convert a decimal number into its binary equivalent.
Code:

```
def decimal_to_binary(n):
    if n == 0:
        return '0'
    elif n == 1:
        return '1'
    else:
        return decimal_to_binary(n // 2) + str(n % 2)

decimal_number = 10
binary_equivalent = decimal_to_binary(decimal_number)
print(f"Binary equivalent of {decimal_number} is {binary_equivalent}.")
```

Conclusion: We successfully found the binary equivalent of a decimal number using recursion.
Output:

4. Use lambda functions to generate filtered, mapped, and reduced lists

Aim: To use lambda functions for filtering, mapping, and reducing lists.
Theory: In this task, we'll use lambda functions with the filter(), map(), and reduce() functions to perform operations on a list.
Code:

```
from functools import reduce

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

filtered_numbers = list(filter(lambda x: x % 2 == 0, numbers))

squared_numbers = list(map(lambda x: x ** 2, numbers))

sum_of_numbers = reduce(lambda x, y: x + y, numbers)
```

```
print("Original Numbers:", numbers)
print("Filtered Even Numbers:", filtered_numbers)
print("Mapped to Squares:", squared_numbers)
print("Reduced to Sum:", sum_of_numbers)
```

Conclusion: We successfully used lambda functions for filtering, mapping, and reducing operations on a list.
Output:

5. Convert temperatures from Celsius to Fahrenheit in a list using an anonymous function

Aim: To convert temperatures from Celsius to Fahrenheit using an anonymous function and a list of temperatures.
Theory: In this task, we'll create an anonymous (lambda) function to convert Celsius temperatures to Fahrenheit and apply it to a list of temperatures.
Code:

```
celsius_temperatures = [0, 25, 100, -10]

convert_to_fahrenheit = lambda c: (c * 9/5) + 32

fahrenheit_temperatures = list(map(convert_to_fahrenheit,
celsius_temperatures))

print("Celsius Temperatures:", celsius_temperatures)
print("Fahrenheit Temperatures:", fahrenheit_temperatures)
```

Conclusion: We successfully converted Celsius temperatures to Fahrenheit using an anonymous (lambda) function and a list.
Output:

6. Create Python modules and access their functions by importing them to other files/modules (calculator program)

Aim: To create a Python module and access its functions by importing them into another file.
Theory: In this task, we'll create a simple Python module containing a function, and then import and use that function in another file.
Code:

```
def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

def multiply(a, b):
    return a * b

def divide(a, b):
    if b == 0:
        return "Cannot divide by zero"
    return a / b
```

Main.py

```
import calculator

num1 = 10
num2 = 5

result_add = calculator.add(num1, num2)
result_subtract = calculator.subtract(num1, num2)
result_multiply = calculator.multiply(num1, num2)
result_divide = calculator.divide(num1, num2)

print(f"Addition: {num1} + {num2} = {result_add}")
print(f"Subtraction: {num1} - {num2} = {result_subtract}")
print(f"Multiplication: {num1} * {num2} = {result_multiply}")
print(f"Division: {num1} / {num2} = {result_divide}")
```

Conclusion: We created a Python module with basic calculator functions and accessed them by importing the module in another file.

Output:

.

Name of student: Abhay Omprakash Prajapati		
Roll no: 41	**Tutorial No: 5**	
Title of LAB Assignment: **To implement programs on OOP Concepts in python**		
DOP: 25-10-2023	**DOS:**	
CO Mapped:	**PO Mapped:**	**Signature:**

1. Aim:

To demonstrate various Python programming concepts, including class creation, multiple inheritance, method overloading, operator overloading, and exception handling.

2. Theory:

Creating a class to compute the area and perimeter of a circle.
Implementing multiple inheritance in Python.
Defining a program with the same method name but multiple arguments.
Overloading operators in Python.
Handling various exceptions in Python.

3. Code:

A. Creating a Class to Compute the Area and Perimeter of a Circle:

```
import math

class Circle:
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return math.pi * self.radius**2

    def perimeter(self):
        return 2 * math.pi * self.radius

circle = Circle(5)
print("Circle Area:", circle.area())
print("Circle Perimeter:", circle.perimeter())
```

B. Implementing Multiple Inheritance in Python:

```
class Parent1:
    def show(self):
        print("This is from Parent1")

class Parent2:
    def show(self):
        print("This is from Parent2")

class Child(Parent1, Parent2):
    pass

child = Child()
child.show()
```

C. Defining a Program with the Same Method Name but Multiple Arguments:

```
class MyClass:
    def argsMethods(self, arg1, arg2=None):
        if arg2 is not None:
            print(f"Arguments: {arg1}, {arg2}")
        else:
            print(f"Argument: {arg1}")

obj = MyClass()
obj.example_method(10)
obj.example_method(20, 30)
```

D. Overloading Operators in Python:

```
class ComplexNumber:
    def __init__(self, real, imag):
        self.real = real
        self.imag = imag

    def __add__(self, other):
        return ComplexNumber(self.real + other.real, self.imag + other.imag)

    def __str__(self):
        return f"{self.real} + {self.imag}i"

c1 = ComplexNumber(1, 2)
c2 = ComplexNumber(3, 4)
c3 = c1 + c2
print("Sum of Complex Numbers:", c3)
```

E. Handling Various Exceptions in Python:

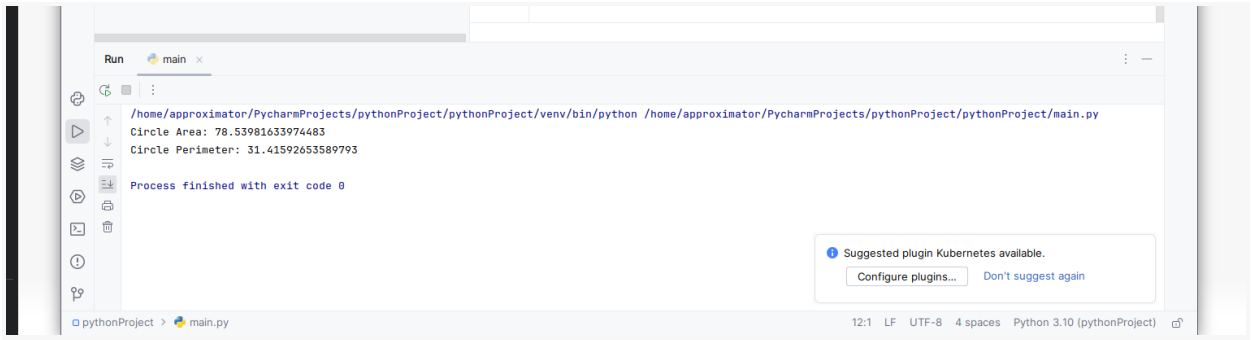
```
try:
    num1 = int(input("Enter a number: "))
    num2 = int(input("Enter another number: "))
    result = num1 / num2
    print("Result:", result)
except ZeroDivisionError:
    print("Division by zero is not allowed.")
except ValueError:
    print("Invalid input. Please enter a valid number.")
except Exception as e:
    print(f"An error occurred: {e}")
```

4. Conclusion:

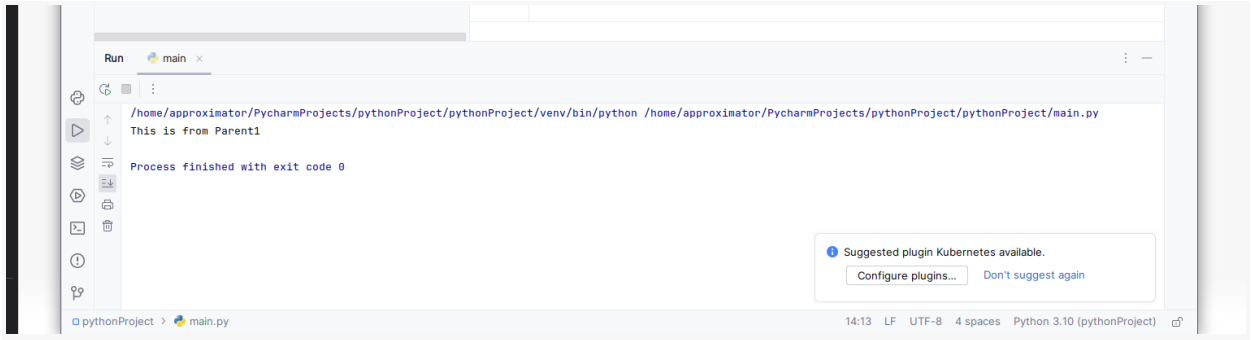
In this practical, we demonstrated various Python programming concepts, including class creation, multiple inheritance, method overloading, operator overloading, and exception handling. These concepts are fundamental for building robust and versatile Python programs.

5. Output:

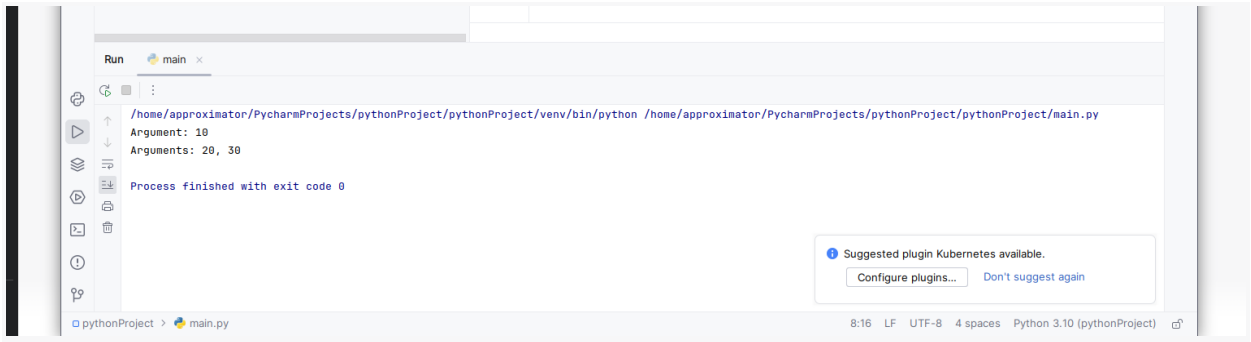
A.


Run
main
/home/approximator/PycharmProjects/pythonProject/pythonProject/venv/bin/python /home/approximator/PycharmProjects/pythonProject/pythonProject/main.py
Circle Area: 78.53981633974483
Circle Perimeter: 31.41592653589793
Process finished with exit code 0
Suggested plugin Kubernetes available.
Configure plugins...
Don't suggest again
pythonProject
main.py
12:1
LF
UTF-8
4 spaces
Python 3.10 (pythonProject)

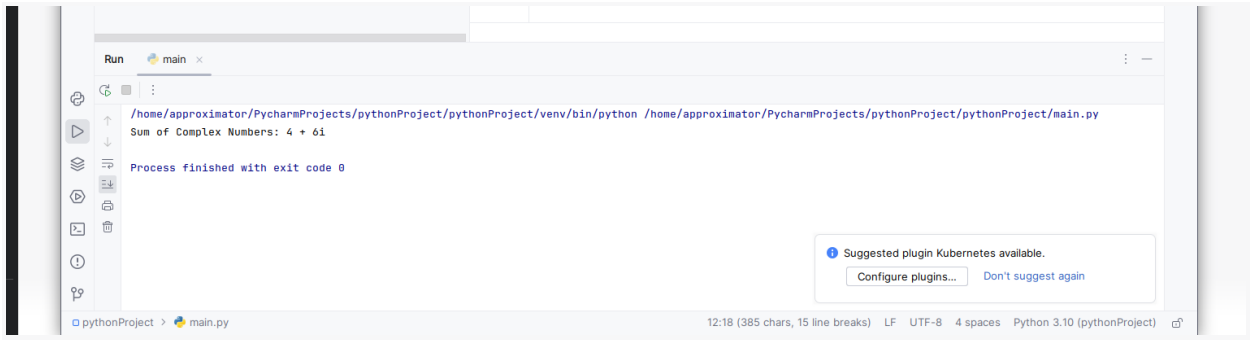
B.


Run
main
/home/approximator/PycharmProjects/pythonProject/pythonProject/venv/bin/python /home/approximator/PycharmProjects/pythonProject/pythonProject/main.py
This is from Parent1
Process finished with exit code 0
Suggested plugin Kubernetes available.
Configure plugins...
Don't suggest again
pythonProject
main.py
14:13
LF
UTF-8
4 spaces
Python 3.10 (pythonProject)

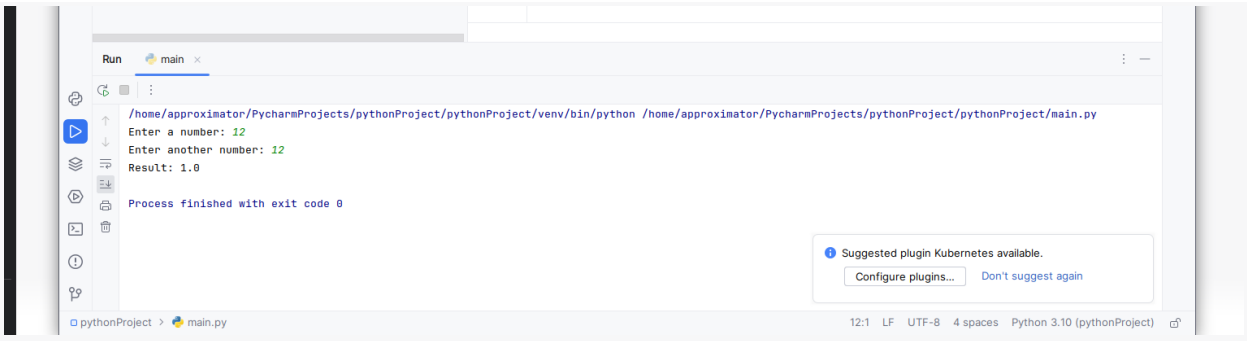
C.


Run
main
/home/approximator/PycharmProjects/pythonProject/pythonProject/venv/bin/python /home/approximator/PycharmProjects/pythonProject/pythonProject/main.py
Argument: 10
Arguments: 20, 30
Process finished with exit code 0
Suggested plugin Kubernetes available.
Configure plugins...
Don't suggest again
pythonProject
main.py
8:16
LF
UTF-8
4 spaces
Python 3.10 (pythonProject)

D.


Run
main
/home/approximator/PycharmProjects/pythonProject/pythonProject/venv/bin/python /home/approximator/PycharmProjects/pythonProject/pythonProject/main.py
Sum of Complex Numbers: 4 + 6i
Process finished with exit code 0
Suggested plugin Kubernetes available.
Configure plugins...
Don't suggest again
pythonProject
main.py
12:18 (385 chars, 15 line breaks)
LF
UTF-8
4 spaces
Python 3.10 (pythonProject)

E.


Run
main
/home/approximator/PycharmProjects/pythonProject/pythonProject/venv/bin/python /home/approximator/PycharmProjects/pythonProject/pythonProject/main.py
Enter a number: 12
Enter another number: 12
Result: 1.0
Process finished with exit code 0
Suggested plugin Kubernetes available.
Configure plugins...
Don't suggest again
pythonProject
main.py
12:1
LF
UTF-8
4 spaces
Python 3.10 (pythonProject)

Name of student: Abhay Omprakash Prajapati		
Roll no: 41		**Tutorial No: 6**
Title of LAB Assignment:To implement programs on Data Structures using Python		
DOP: 25-09-2023		**DOS:02-10-2023**
CO Mapped: Co1,Co2	**PO Mapped:** PO3 ,PO6	**Signature:**

1. Create, Traverse, Insert, and Remove Data Using Linked List

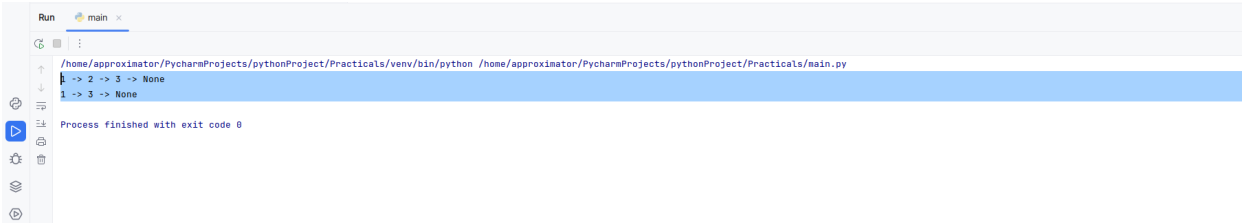
Aim: To create, traverse, insert, and remove data using a linked list.
Theory: In this task, we'll create a basic linked list structure and implement operations like insertion, traversal, and removal of data.
Code:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None
```

```
    def append(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
        else:
            current = self.head
            while current.next:
                current = current.next
            current.next = new_node

    def remove(self, data):
        if self.head and self.head.data == data:
            self.head = self.head.next
            return
        current = self.head
        while current and current.next:
            if current.next.data == data:
                current.next = current.next.next
            current = current.next

    def display(self):
        current = self.head
        while current:
            print(current.data, end=" -> ")
            current = current.next
        print("None")

linked_list = LinkedList()
linked_list.append(1)
linked_list.append(2)
linked_list.append(3)
linked_list.display()
linked_list.remove(2)
linked_list.display()
```

Conclusion: We successfully created a linked list, inserted data, and removed data from it.

Output:



2. Implementation of Stacks

Aim: To implement a stack data structure.
Theory: In this task, we'll create a basic stack structure and implement operations like push and pop.
Code:

```
class Stack:
    def __init__(self):
        self.items = []

    def is_empty(self):
        return len(self.items) == 0

    def push(self, item):
        self.items.append(item)

    def pop(self):
        if not self.is_empty():
            return self.items.pop()
        else:
            return "Stack is empty"

stack = Stack()
stack.push(1)
stack.push(2)
stack.push(3)
print("Popped:", stack.pop())
print("Popped:", stack.pop())
```

Conclusion: We successfully implemented a stack and demonstrated push and pop operations.
Output:

3. Implementation of Queue

Aim: To implement a queue data structure.
Theory: In this task, we'll create a basic queue structure and implement operations like enqueue and dequeue.
Code:

```
class Queue:
    def __init__(self):
        self.items = []

    def is_empty(self):
        return len(self.items) == 0

    def enqueue(self, item):
        self.items.insert(0, item)

    def dequeue(self):
        if not self.is_empty():
            return self.items.pop()
        else:
            return "Queue is empty"

queue = Queue()
queue.enqueue(1)
queue.enqueue(2)
queue.enqueue(3)
print("Dequeued:", queue.dequeue())
print("Dequeued:", queue.dequeue())
```

Conclusion: We successfully implemented a queue and demonstrated enqueue and dequeue operations.
Output: The output will display the items dequeued from the queue.

4. Implementation of Dequeue (Double-Ended Queue)

Aim: To implement a double-ended queue (deque) data structure.
Theory: In this task, we'll create a basic deque structure and implement operations for both ends, such as inserting and removing elements.
Code:

```
from collections import deque

dq = deque()
dq.append(1)
dq.append(2)
dq.appendleft(3)
dq.appendleft(4)
```

```
print("Deque:", dq)

popped_from_right = dq.pop()
popped_from_left = dq.popleft()
print("Popped from right:", popped_from_right)
print("Popped from left:", popped_from_left)
```

Conclusion: We successfully implemented a deque and demonstrated append, appendleft, pop, and popleft operations.
Output:

Name of student: Abhay Omprakash Prajapati		
Roll no: 41		**Tutorial No: 7**
Title of LAB Assignment: To write, test, and debug Basic Python programs.		
DOP: 25-09-2023		**DOS:02-10-2023**
CO Mapped:	**PO Mapped:**	**Signature:**

1. Aim:
The aim of this project is to create a login system with a sign-up feature using Python and the tkinter library. This system will allow users to register and log in securely.

2. Theory:
In this project, we will create a graphical user interface (GUI) application that includes two main features:

Sign-up: Users can register by providing a username and password. The entered data will be stored in an SQLite database for later use.

Login: Users can enter their username and password to log in. The application will check if the credentials match those stored in the database and provide access if they are correct.
We'll use the tkinter library for the graphical interface and SQLite for database operations.

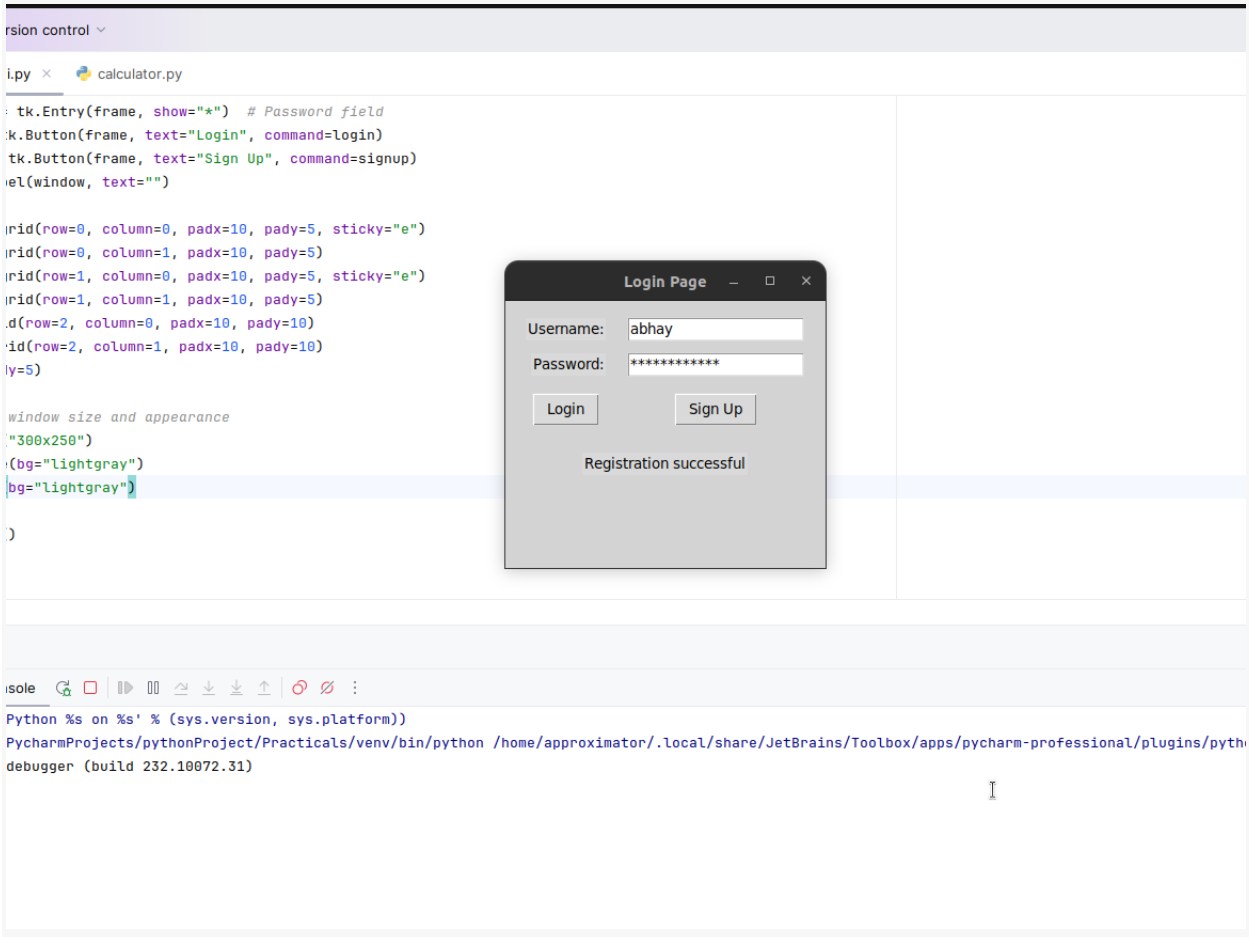
Code:

```
import tkinter as tk
import sqlite3

# Connect to the SQLite database (this will create the database if it doesn't
exist)
conn = sqlite3.connect('user_database.db')
cursor = conn.cursor()

# Create a 'users' table if it doesn't exist
cursor.execute('''
    CREATE TABLE IF NOT EXISTS users (
        id INTEGER PRIMARY KEY,
        username TEXT NOT NULL,
        password TEXT NOT NULL
    )
''')

# Commit the changes and close the connection
conn.commit()
conn.close()

def signup():
    username = entry_username.get()
    password = entry_password.get()

    # Connect to the SQLite database
    conn = sqlite3.connect('user_database.db')
    cursor = conn.cursor()

    # Insert user into the 'users' table
    cursor.execute("INSERT INTO users (username, password) VALUES (?, ?)",
(username, password))
    conn.commit()
    conn.close()

    message.config(text="Registration successful")

def login():
    username = entry_username.get()
    password = entry_password.get()
```

```
    # Connect to the SQLite database
    conn = sqlite3.connect('user_database.db')
    cursor = conn.cursor()

    # Check if the provided credentials are in the database
    cursor.execute('SELECT * FROM users WHERE username=? AND password=?',
(username, password))
    user = cursor.fetchone()

    if user:
        message.config(text="Login successful")
    else:
        message.config(text="Login failed")

    conn.close()

# Create the main window
window = tk.Tk()
window.title("Login Page")

# Create a frame for better organization
frame = tk.Frame(window)
frame.pack(pady=10)

# Create and place widgets
label_username = tk.Label(frame, text="Username:")
entry_username = tk.Entry(frame)
label_password = tk.Label(frame, text="Password:")
entry_password = tk.Entry(frame, show="*")  # Password field
button_login = tk.Button(frame, text="Login", command=login)
button_signup = tk.Button(frame, text="Sign Up", command=signup)
message = tk.Label(window, text="")

label_username.grid(row=0, column=0, padx=10, pady=5, sticky="e")
entry_username.grid(row=0, column=1, padx=10, pady=5)
label_password.grid(row=1, column=0, padx=10, pady=5, sticky="e")
entry_password.grid(row=1, column=1, padx=10, pady=5)
button_login.grid(row=2, column=0, padx=10, pady=10)
button_signup.grid(row=2, column=1, padx=10, pady=10)
message.pack(pady=5)

# Customize the window size and appearance
window.geometry("300x250")
window.configure(bg="lightgray")
frame.configure(bg="lightgray")

window.mainloop()
```

Output:

Login/SignUP:



Name of student: Abhay Omprakash Prajapati		
Roll no: 41	**Tutorial No: 8**	
Title of LAB Assignment: To implement Threads in Python		
DOP: 25-09-2023	**DOS:02-10-2023**	
CO Mapped:	**PO Mapped:**	**Signature:**

1. Aim:
To understand and implement Python threading with a focus on synchronization and multithreaded priority queue.

2. Theory:
In this practical, we will cover the following topics:

Threading in Python: Python's threading module allows you to create and manage threads.
Synchronization: We'll explore synchronization techniques using locks to ensure that multiple threads work together harmoniously without interfering with each other.

Multithreaded Priority Queue: We will implement a multithreaded priority queue using Python's queue module.

3. Code:

Here's the Python code for each part of your practical:

Starting a Thread:

```
import threading

def print_numbers():
    for i in range(1, 6):
        print(f"Number {i}")

# Create a thread
thread = threading.Thread(target=print_numbers)

# Start the thread
thread.start()
```

2. Synchronization:

```
import threading

counter = 0
lock = threading.Lock()

def increment():
    global counter
    for _ in range(100000):
        with lock:
            counter += 1

thread1 = threading.Thread(target=increment)
thread2 = threading.Thread(target=increment)

thread1.start()
thread2.start()

thread1.join()
thread2.join()
```

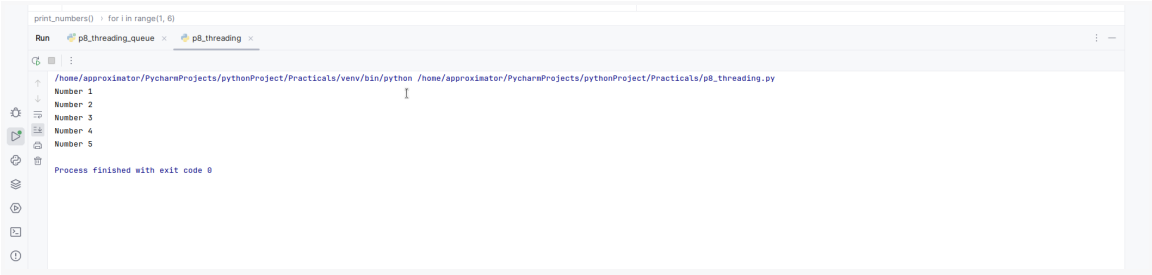
```
print("Counter:", counter)
```

3. Multithreaded Priority Queue:

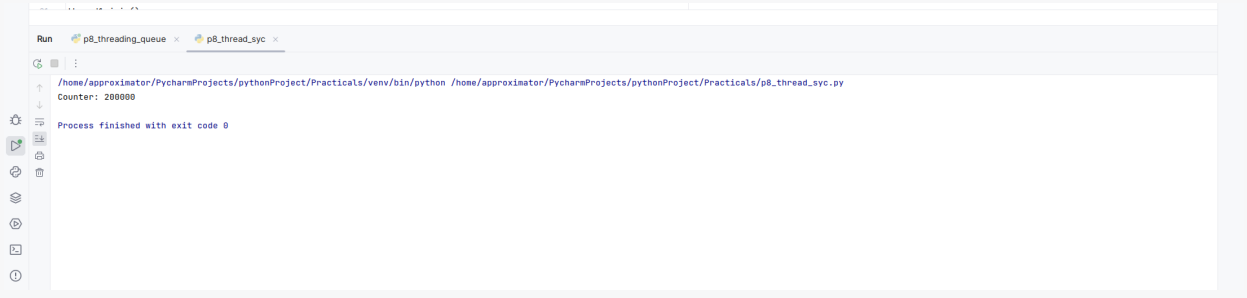
```
import threading
import queue

priority_queue = queue.PriorityQueue()

def producer():
    for i in range(5):
        priority_queue.put(i)

def consumer():
    while True:
        item = priority_queue.get()
        print(f"Consumed: {item}")
        priority_queue.task_done()

producer_thread = threading.Thread(target=producer)
consumer_thread = threading.Thread(target=consumer)

producer_thread.start()
consumer_thread.start()

producer_thread.join()
```

Output:

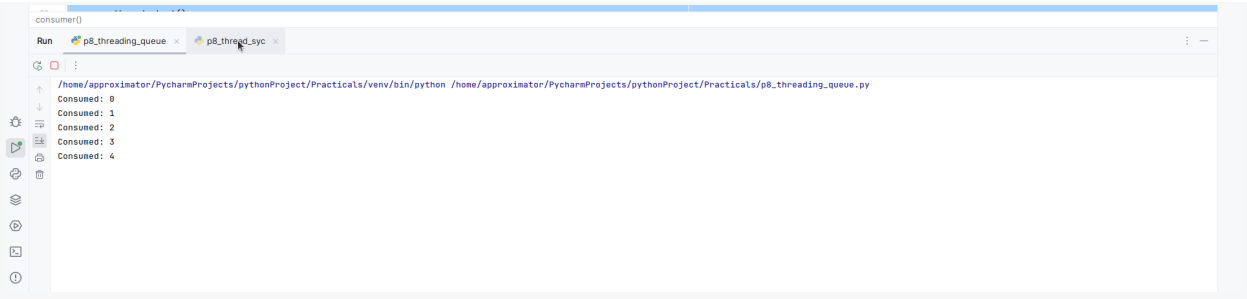
1.



2.


Run
p8_threading_queue
p8_thread_syc
/home/approximator/PycharmProjects/pythonProject/Practicals/venv/bin/python /home/approximator/PycharmProjects/pythonProject/Practicals/p8_thread_syc.py
Counter: 200000
Process finished with exit code 0

3.


consumer()
Run
p8_threading_queue
p8_thread_syc
/home/approximator/PycharmProjects/pythonProject/Practicals/venv/bin/python /home/approximator/PycharmProjects/pythonProject/Practicals/p8_threading_queue.py
Consumed: 0
Consumed: 1
Consumed: 2
Consumed: 3
Consumed: 4

Name of student: Abhay Omprakash Prajapati		
Roll no: 41	**Tutorial No: 9**	
Title of LAB Assignment: **To implement NumPy library in Python**		
DOP: 30-10-2023	**DOS: 04-11-2023**	
CO Mapped:	**PO Mapped:**	**Signature:**

1. Aim:
To understand and implement basic operations in NumPy, including creating ndarray objects, performing matrix multiplication, indexing and slicing NumPy arrays, and working with data types.

2. Theory:
In this practical, we will cover the following topics:

Creating ndarray objects using array() in NumPy.
Implementing 2D arrays to perform matrix multiplication using the dot() function.
Indexing and slicing in NumPy arrays to access and manipulate elements.

Exploring NumPy data types for efficient memory usage and performance.
3. Code:

Here's the Python code for each part of your practical:

1. **Creating ndarray objects using array() in NumPy:**

```
import numpy as np

# Create a 1D array
arr1 = np.array([1, 2, 3, 4, 5])

# Create a 2D array
arr2 = np.array([[1, 2, 3], [4, 5, 6]])

print("1D Array:")
print(arr1)
print("2D Array:")
print(arr2)
```

2. **Program for Indexing and Slicing in NumPy arrays:**

```
import numpy as np

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

element = arr[1, 2]
row_slice = arr[0:2, 1:]
print("Element at (1, 2):", element)
print("Sliced Array:")
print(row_slice)
```

3. **Implementing NumPy - Data Types:**

```
import numpy as np

# Create arrays with specific data types
int_array = np.array([1, 2, 3], dtype=np.int32)
float_array = np.array([1.2, 2.3, 3.4], dtype=np.float64)
complex_array = np.array([1 + 2j, 2 + 3j], dtype=np.complex128)

print("Int Array:", int_array)
print("Float Array:", float_array)
print("Complex Array:", complex_array)
```

Conclusion:

In this practical, we learned how to create ndarray objects in NumPy using the array() function, perform matrix multiplication with 2D arrays, and use indexing and slicing to access specific elements in arrays. Additionally, we explored the use of NumPy data types for efficient memory and performance management.

5. Output:

1.

2.

3.

Name of student: Abhay Omprakash Prajapati		
Roll no: 41	**Tutorial No: 10**	
Title of LAB Assignment: **Implementing Pandas in Python**		
DOP: 30-10-2023	**DOS: 04-11-2023**	
CO Mapped:	**PO Mapped:**	**Signature:**

1. Aim:

To understand and implement basic operations with the Pandas library in Python, including creating Series and DataFrames, converting dictionaries, aggregating data frames along rows, and merging dataframes.

2. Theory:

In this practical, we will cover the following topics:

Creating a one-dimensional array-like object containing data using Pandas Series.
Converting a dictionary to a Pandas Series.

Creating a DataFrame from a dictionary and displaying it.
Aggregating two data frames along rows.
Merging two dataframes with different columns.

3. Code:

Here's the Python code for each part of your practical:

1. **Creating a one-dimensional -like object with Pandas Series: array**

```
import pandas as pd

data = [78, 85, 96, 80, 86]
series = pd.Series(data)

print("Pandas Series:")
print(series)
```

2. **Converting a dictionary to a Pandas Series:**

```
import pandas as pd

data = {'X': [78, 85, 96, 80, 86]}

series = pd.Series(data['X'])

print("Pandas Series from Dictionary:")
print(series)
```

3. **Creating a DataFrame from a dictionary and displaying it:**

```
import pandas as pd

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

df = pd.DataFrame(data)

print("Pandas DataFrame:")
print(df)
```

4. **Aggregating two given data frames along rows:**

```
import pandas as pd
```

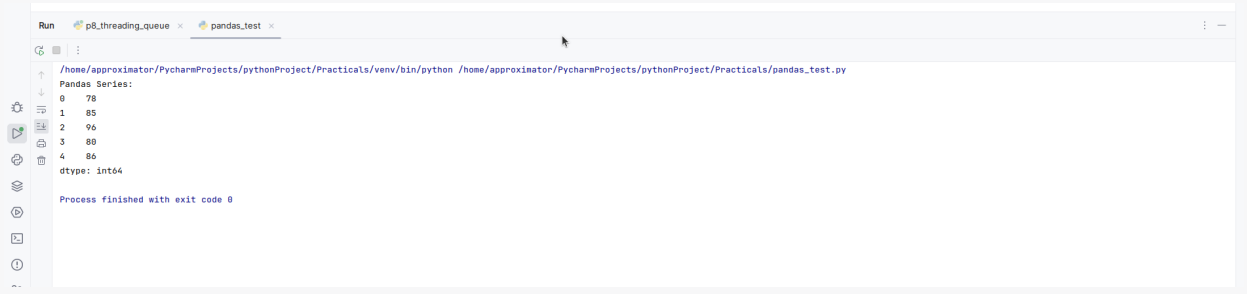
```
df1 = pd.DataFrame({'A': [1, 2], 'B': [3, 4]})
df2 = pd.DataFrame({'A': [5, 6], 'B': [7, 8]})

result = pd.concat([df1, df2])

print("Aggregated Dataframe:")
print(result)
```

5. **Merging two given dataframes with different columns:**

```
import pandas as pd

df1 = pd.DataFrame({'ID': [1, 2, 3], 'Name': ['Alice', 'Bob', 'Charlie']})
df2 = pd.DataFrame({'ID': [2, 3, 4], 'Age': [25, 30, 22]})

merged_df = pd.merge(df1, df2, on='ID', how='outer')

print("Merged Dataframe:")
print(merged_df)
```

Output:

1.



2.

3.

4.

5.

Conclusion:

In this practice, we learned how to use the Pandas library to create Series and DataFrames, convert dictionaries into Series, aggregate data frames along rows, and merge dataframes with different columns. Pandas is a powerful library for data manipulation and analysis.