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:**

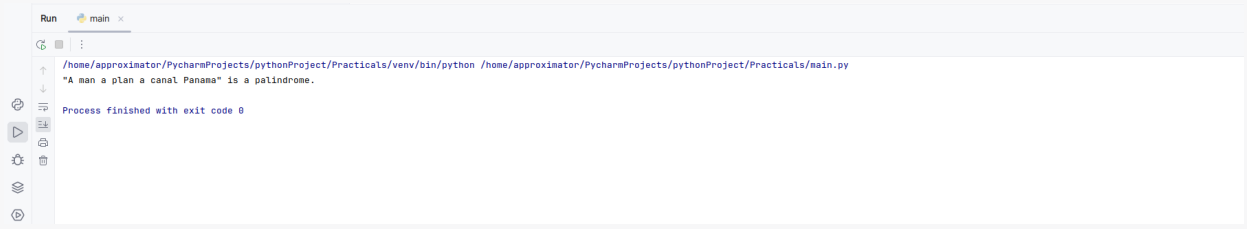
1. Check if a string is a palindrome using a recursive function

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:

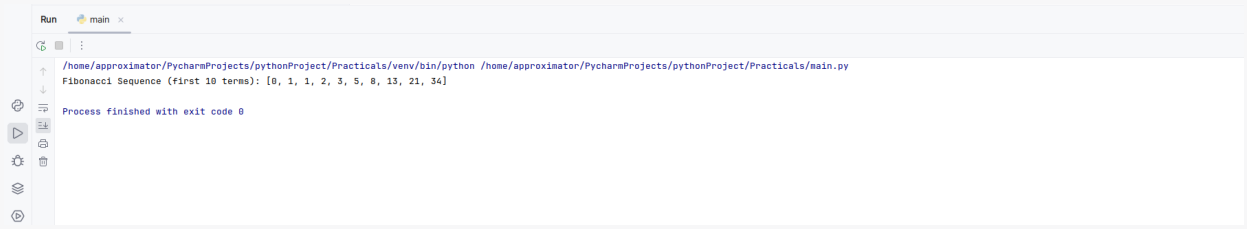


2. Find the Fibonacci sequence using recursion

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:

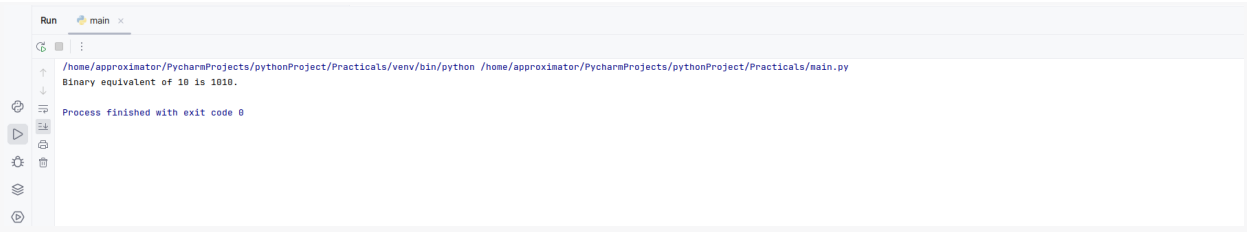


3. Find the binary equivalent of a number using recursion

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:



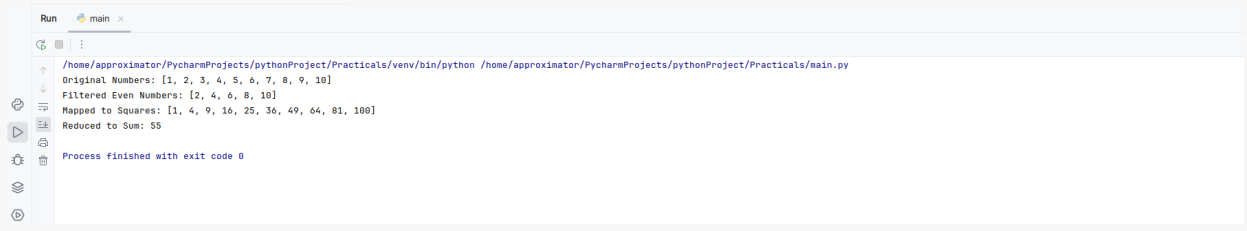
4. Use lambda functions to generate filtered, mapped, and reduced lists

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:

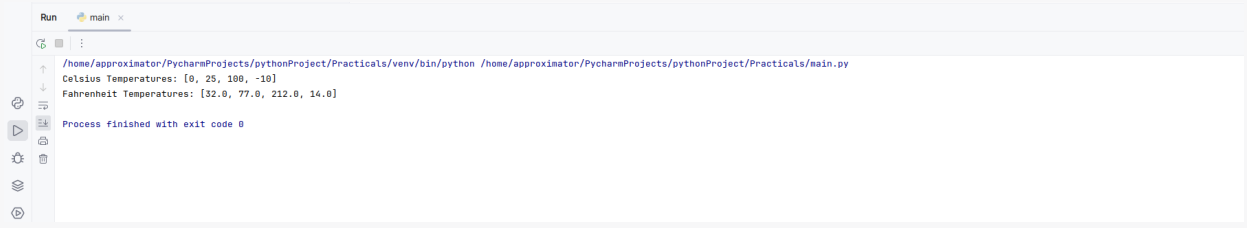


5. Convert temperatures from Celsius to Fahrenheit in a list using an anonymous function

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:

.