|  |  |  |  |
| --- | --- | --- | --- |
| **Name of Student: Pushkar Sane** | | | |
| **Roll Number: 45** | | **Lab Assignment Number: 4** | |
| **Title of Lab Assignment: To implement programs on Python Functions and Modules.** | | | |
| **DOP:** | | **DOS:** | |
| **CO Mapped:**  **CO1** | **PO Mapped:**  **PO3, PO5, PSO1, PSO2** | **Signature:** | **Marks:** |

**Practical No. 4**

**Aim:** To implement programs on Python Functions and Modules.

**Description:**

* **Function In Python:**

1. A function in Python is a named block of code that performs a specific task or set of tasks.
2. Functions are essential for organizing and structuring your code, promoting code reuse, and making your code more readable and maintainable. They encapsulate a series of instructions, allowing you to call and reuse them throughout your program.

* **Function Syntax:**

Here’s the general syntax of defining a function in Python:

def function\_name(parameter1, parameter2, ...):

# Function body - code to perform the task

# You can use parameters in the function body

result = parameter1 + parameter2

return result # Optional, specifies the value to be returned

1. **def:** This keyword is used to define a function in Python.
2. **function\_name:** Choose a descriptive name for your function that reflects its purpose. Function names should follow naming conventions (e.g., use lowercase letters and underscores).
3. **parameter1, parameter2, ... :** These are optional input parameters or arguments that the function accepts. Parameters allow you to pass data into the function for processing.
4. **‘:’** : A colon is used to denote the beginning of the function's body.
5. **Function body:** The function body consists of the actual code that performs the desired task. It can include calculations, loops, conditionals, and other statements.
6. **return:** This keyword, followed by an expression, is used to specify the value that the function should return as its result. Not all functions need to return a value; it's optional.

**Example: Adding Two Numbers:**

def add\_numbers(num1, num2):

result = num1 + num2

return result

# Call the function with arguments 5 and 3

sum\_result = add\_numbers(5, 3)

print(sum\_result) # Output: 8

# Call the function with different arguments

result2 = add\_numbers(10, 20)

print(result2) # Output: 30

* **Recursion:**

Recursion is a programming technique in which a function calls itself to solve a problem. In Python, you can implement recursive functions to break down complex problems into simpler, self-similar sub problems. To successfully use recursion, you need to define a base case (the terminating condition) and one or more recursive cases (the cases where the function calls itself).

Here's a detailed explanation of recursion with an example:

**Factorial Calculation Using Recursion:**

def factorial(n):

# Base case: When n is 0 or 1, the factorial is 1.

if n == 0 or n == 1:

return 1

# Recursive case: Calculate factorial by calling the function recursively.

else:

return n \* factorial(n - 1)

**In this recursive function:**

1. The base case is when `n` is either `0` or `1`. In these cases, we return `1` because `0!` and `1!` are both equal to `1`.
2. The recursive case calculates the factorial of `n` by calling the `factorial` function recursively with the argument `n - 1`. This reduces the problem to a smaller subproblem.

* **Lambda Function**

In Python, a lambda function is a small, anonymous, and inline function defined using the lambda keyword. Lambda functions are also known as anonymous functions because they don't have a name. They are typically used for short, simple operations where defining a full function using the def keyword would be overly verbose.

The basic syntax of a lambda function is as follows:

lambda arguments: expression

1. **lambda:** The lambda keyword is used to indicate the creation of a lambda function.
2. **arguments:** These are the input parameters to the lambda function, separated by commas. Lambda functions can take any number of arguments but are often used with one or two.
3. **expression:** This is a single expression that the lambda function evaluates and returns as its result.

Here's an example to illustrate how lambda functions work:

# Regular function to add two numbers

def add(x, y):

return x + y

# Equivalent lambda function

add\_lambda = lambda x, y: x + y

# Using both functions to add numbers

result1 = add(5, 3)

result2 = add\_lambda(5, 3)

print("Using the regular function:", result1) # Output: Using the regular function: 8

print("Using the lambda function:", result2)

* **Module:**

In Python, a module is a file containing Python code, including variables, functions, and classes, that can be imported and used in other Python scripts or programs. Modules allow you to organize your code into reusable and manageable units, making your code more modular and maintainable. A file in Python can serve as a module if it contains Python code. To create a module, you typically save your Python code in a `.py` file with the same name as the module you want to create. For example, if you want to create a module named `my\_module`, you could create a file named `my\_module.py`.

Here's a simple example of a Python module:

**my\_module.py**

# This is a Python module named 'my\_module'

def greet(name):

return f"Hello, {name}!"

def add(a, b):

return a + b

pi = 3.14159265359

Now, you can use this module in another Python script by importing it using the `import` statement:

**main.py**

# Importing the 'my\_module' module

import my\_module

# Using functions and variables from the module

print(my\_module.greet("Alice")) # Output: Hello, Alice!

print(my\_module.add(5, 3)) # Output: 8

print(my\_module.pi) # Output: 3.14159265359

1. **To check whether a string is palindrome or not using function recursion.**

**Code:**

# Define a function

def isPalindrome(string):

if len(string) < 1:

return True

else:

if string[0] == string[-1]:

return isPalindrome(string[1:-1])

else:

return False

#Enter input string

str1 = input("Enter string:")

if(isPalindrome(str1)==True):

print("The string is palindrome.")

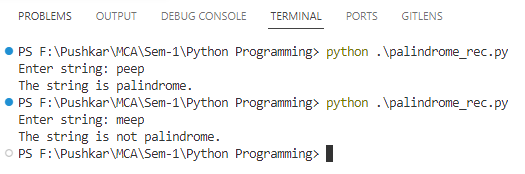
else:

print("The string is not palindrome.")

**Conclusion:**

Here, we reverse a string using the recursion method which is a process where the function calls itself. Then we check if the reversed string matches with the original string and demonstrate the program for checking if a string is palindrome or not using recursion.

**Output:**



1. **To find Fibonacci series using recursion.**

**Code:**

def recur\_fibo(n):

if n <= 1:

return n

else:

return(recur\_fibo(n-1) + recur\_fibo(n-2))

# take input from the user

nterms = int(input("How many terms? "))

# check if the number of terms is valid

if nterms <= 0:

print("Please enter a positive integer")

else:

print("Fibonacci sequence:")

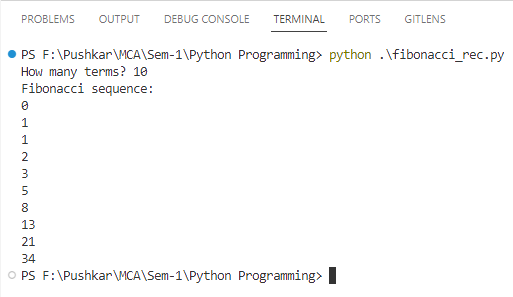
for i in range(nterms):

print(recur\_fibo(i))

**Conclusion:**

In this program, we store the number of terms to be displayed in nterms. A recursive function recur\_fibo() is used to calculate the nth term of the sequence. We use a for loop to iterate and calculate each term recursively.

**Output:**



1. **To find the binary equivalent of a number using recursion.**

**Code:**

# Decimal to binary conversion using recursion

def find( decimal\_number ):

if decimal\_number == 0:

return 0

else:

return (decimal\_number % 2 + 10 \*

find(int(decimal\_number // 2)))

# Driver Code

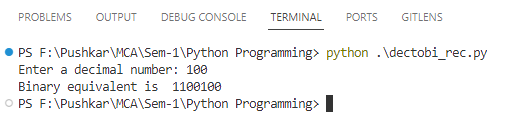
decimal\_number = int(input(“Enter a decimal number: ”))

print(find(decimal\_number))

**Conclusion:**

Demonstrated the program for finding the binary equivalent or converting a decimal number to binary number using recursion.

**Output:**



1. **To use lambda function on list to generate filtered list, mapped list and reduced list.**

**Code:**

from functools import reduce

# Sample list

my\_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Filter using a lambda function

filtered\_list = list(filter(lambda x: x % 2 == 0, my\_list))

# Map using a lambda function

mapped\_list = list(map(lambda x: x \* 2, my\_list))

# Reduce using a lambda function

reduced\_result = reduce(lambda x, y: x + y, my\_list)

# Print the results

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

print("Filtered List (even numbers):", filtered\_list)

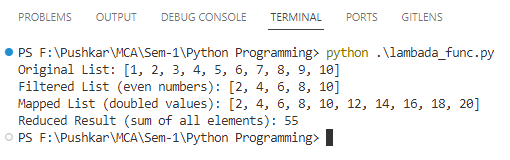
print("Mapped List (doubled values):", mapped\_list)

print("Reduced Result (sum of all elements):", reduced\_result)

**Conclusion**

In this Python program, we demonstrated how to use lambda functions in conjunction with the filter(), map(), and reduce() functions to perform different operations on a given list.

**Output:**

****

1. **Convert the temperature in Celsius to Fahrenheit in a list using an anonymous function.**

**Code:**

# Convert Celsius to Fahrenheit using a lambda function

celsius\_to\_fahrenheit = lambda c: (c \* 9/5) + 32

# Temperature in Celsius

celsius\_temperature = 25

# Use the lambda function to convert

fahrenheit\_temperature = celsius\_to\_fahrenheit(celsius\_temperature)

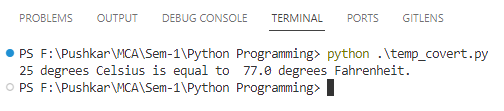
# Print the result

print(celsius\_temperature, "degrees Celsius is equal to ", fahrenheit\_temperature , "degrees Fahrenheit.")

**Conclusion:**

This program demonstrates how to perform a straightforward temperature conversion on a list using an anonymous lambda function and then display the results. It's a concise way to achieve the desired conversion without the need for additional functions or complex string formatting.

**Output:**

****

1. **To create modules in python and access functions of the module by importing it to another file/module. (Calculator program)**

**Code:**

Calculator.py

# calculator.py

def add(x, y):

return x + y

def subtract(x, y):

return x - y

def multiply(x, y):

return x \* y

def divide(x, y):

if y == 0:

return "Error: Division by zero"

return x / y

Main.py

# main.py

import calculator

def main():

print("Simple Calculator")

print("1. Add")

print("2. Subtract")

print("3. Multiply")

print("4. Divide")

choice = input("Enter operation (1/2/3/4): ")

num1 = float(input("Enter first number: "))

num2 = float(input("Enter second number: "))

if choice == '1':

result = calculator.add(num1, num2)

print(f"Result: {result}")

elif choice == '2':

result = calculator.subtract(num1, num2)

print(f"Result: {result}")

elif choice == '3':

result = calculator.multiply(num1, num2)

print(f"Result: {result}")

elif choice == '4':

result = calculator.divide(num1, num2)

print(f"Result: {result}")

else:

print("Invalid choice")

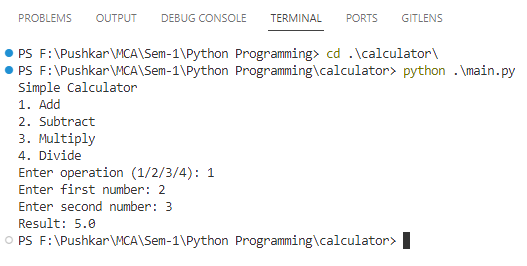
if \_\_name\_\_ == "\_\_main\_\_":

main()

**Conclusion:**

Here, we've demonstrated the creation of a modular calculator program using two separate files. The calculator.py module encapsulates core arithmetic functions, while the main.py script serves as a user interface, importing and utilizing these functions. This modular approach enhances code organization, reusability, and readability, making it easier to maintain and expand the calculator program.

**Output:**

****