11th June Assignment

1. **What is a lambda function in Python, and how does it differ from a regular function?**

Ans- A lambda function is a small anonymous function in Python that can have any number of arguments but can only have one expression. It is defined using the lambda keyword instead of the def keyword used for defining regular functions. The lambda function can be used wherever function objects are required. The main difference between a lambda function and a regular function is that the lambda function can be written in one line and does not require a return statement. It returns the value of the expression automatically. Regular functions can have multiple expressions and statements and can be written in multiple lines

2. **Can a lambda function in Python have multiple arguments? If yes, how can you define and use them?**

Ans- Yes, a lambda function in Python can have multiple arguments. You can define and use them by separating the arguments with commas.
Example:

```
sum = lambda arg1, arg2: arg1 + arg2
print("Value of total : ", sum( 10, 20 ))
```

Output:

```
Value of total:  30
```

We have defined a lambda function that takes two arguments and returns their sum. We then call this function with the arguments 10 and 20 and print the result.

3. **How are lambda functions typically used in Python? Provide an example use case.**

Ans- Examples of using lambda functions in Python include :

- A squaring function
- A subtraction function with multiple arguments
- Using lambda with map () to make the map function more concise
- Using lambda with filter () to simplify the filter () function
- Using lambda with higher-order functions like reduce (), sort (), sorted (), min (), and max ()

4. **What are the advantages and limitations of lambda functions compared to regular functions in Python?**

Ans- Lambda functions have some advantages over regular functions, but they also have limitations. The main advantage is their concise and readable syntax, which makes them ideal for

simple operations. However, lambda functions can only contain a single expression and cannot include statements, making them less suitable for complex operations .

Lambda functions are anonymous functions that are defined using the lambda keyword. They are used to create small, one-time-use functions that can be passed as arguments to other functions.

Lambda functions are used in conjunction with built-in Python functions like filter (), map (), and reduce(). These functions take other functions as arguments and apply them to a sequence of values .

Lambda functions are also useful when you need to create a function that is only used once in your code. Instead of defining a named function that you will only use once, you can define a lambda function inline .

5. Are lambda functions in Python able to access variables defined outside of their own scope? Explain with an example.

Ans- Yes, lambda functions in Python can access variables defined outside of their own scope. This is known as “closing over” a variable12. When a lambda function is defined, it captures the current value of any variables that are used within its body. For example, the lambda in button.clicked.connect (lambda: self.squareButtonHandler (buttonNumber)) is an inner function that contains a reference to buttonNumber in the outer scope .

6. Write a lambda function to calculate the square of a given number.

Ans- Example

```
square = lambda x: x ** 2
print(square(5))
```

Output

```
25
```

7. Create a lambda function to find the maximum value in a list of integers.

Ans- A lambda function is an anonymous function that can take any number of arguments, but can only have one expression. To find the maximum value in a list of integers, we can use the built-in max function as the expression.

Example:

```
# Define a list of integers
nums = [3, 5, 7, 9, 2, 4, 6, 8]

# Create a lambda function to find the maximum value
max_value = lambda x: max(x)
```

```
# Call the lambda function with the list of integers
print(max_value(nums))
```

Output:

```
9
```

8. Implement a lambda function to filter out all the even numbers from a list of integers.

Ans- To filter out all the even numbers from a list of integers, we can use the built-in filter function, which takes a function and an iterable as arguments and returns an iterator of the elements that satisfy the function.

Example:

```
# Define a list of integers
nums = [3, 5, 7, 9, 2, 4, 6, 8]

# Create a lambda function to check if a number is odd
is_odd = lambda x: x % 2 != 0

# Filter out all the even numbers using the lambda function and the filter function
odd_nums = list(filter(is_odd, nums))

# Print the filtered list
print(odd_nums)
```

Output

```
[3, 5, 7, 9]
```

9. Write a lambda function to sort a list of strings in ascending order based on the length of each string.

Ans- To sort a list of strings in ascending order based on the length of each string, you can use the **sorted** function with a lambda function as the key argument.

Example:

```
# Define a list of strings
my_list = ["apple", "banana", "cherry", "dragonfruit", "elderberry"]

# Sort the list using a lambda function
sorted_list = sorted(my_list, key=lambda x: len(x))

print(sorted_list)
```

Output:

```
['apple', 'banana', 'cherry', 'elderberry', 'dragonfruit']
```

10. Create a lambda function that takes two lists as input and returns a new list containing the common elements between the two lists.

Ans- To create a lambda function that takes two lists as input and returns a new list containing the common elements between the two lists, we can use the filter function with a lambda function as the argument.

Example:

```
# Define two lists
list1 = [1, 2, 3, 4, 5]
list2 = [3, 4, 5, 6, 7]

# Create a lambda function that checks if an element is in both lists
common = lambda x: x in list1 and x in list2

# Filter the elements of list2 using the lambda function
new_list = list(filter(common, list2))

# Print the new list
print(new_list)
```

Output:

```
[3, 4, 5]
```

11. Write a recursive function to calculate the factorial of a given positive integer.

Ans- A recursive function is a function that calls itself until a base case is reached. A factorial of a positive integer n is the product of all positive integers less than or equal to n.

Example

```
def factorial(n):
    # base case: n is zero or one
    if n <= 1:
        return 1
    # recursive case: n is greater than one
```

```
    else:
        return n * factorial(n - 1) # multiply n by the factorial of n – 1


print(factorial(0))
print(factorial(1))
print(factorial(5)
print(factorial(10))
```

Output:

```
1
1
120
3628800
```

12. Implement a recursive function to compute the nth Fibonacci number.

Ans- A recursive function is a function that calls itself until a base case is reached. The Fibonacci sequence is a series of numbers where each number is the sum of the two preceding ones, starting from 0 and 1. To implement a recursive function to compute the nth Fibonacci number.

Example:

```
# Define a recursive function to compute the nth Fibonacci number
def fibonacci(n):
  # Base case: if n is 0 or 1, return n
  if n == 0 or n == 1:
    return n
  # Recursive case: if n is greater than 1, return the sum of the previous two Fibonacci numbers
  else:
    return fibonacci(n-1) + fibonacci(n-2)

# Test the function with some examples
print(fibonacci(0)) # 0
print(fibonacci(1)) # 1
```

```
print(fibonacci(2)) # 1
print(fibonacci(3)) # 2
print(fibonacci(4)) # 3
print(fibonacci(5)) # 5
print(fibonacci(6)) # 8
```

Output:

```
0
1
1
2
3
5
8
```

13. Create a recursive function to find the sum of all the elements in a given list.

Ans-

```
# Create function
def sum_list(lst):
  # base case: empty list
  if len(lst) == 0:
    return 0
  # recursive case: sum the first element and the rest of the list
  else:
    return lst[0] + sum_list(lst[1:])

# list
lst = [1, 2, 3, 4, 5]
# print the sum
print(sum_list(lst))
```

Output:

15

14. Write a recursive function to determine whether a given string is a palindrome.

Ans-

```
def is_palindrome(s):
  # base case: empty or single-character string
  if len(s) <= 1:
    return True
  # recursive case: check if the first and last characters are equal and the rest of the string is a palindrome
  else:
    return s[0] == s[-1] and is_palindrome(s[1:-1])

# example string
s = "racecar"
# print the result
print(is_palindrome(s))
```

Output:

True

15. Implement a recursive function to find the greatest common divisor (GCD) of two positive integers

Ans-

```
def gcd(a, b):
  # base case: b is zero
  if b == 0:
    return a
  # recursive case: apply the Euclidean algorithm
  else:
    return gcd(b, a % b)

# Example numbers
```

```
a = 24

b = 36

# print the result

print(gcd(a, b))
```

Output:

```
12
```

==========*==========*==========*==========*==========