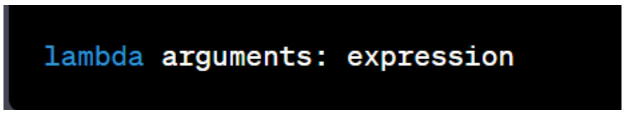
Assignment 9

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

Ans: In Python, a lambda function is a small, anonymous function that is defined using the **‘lambda’** keyword. It allows to create functions without a formal function definition, making them convenient for one-line, simple operations.

Syntax used for lambda function:



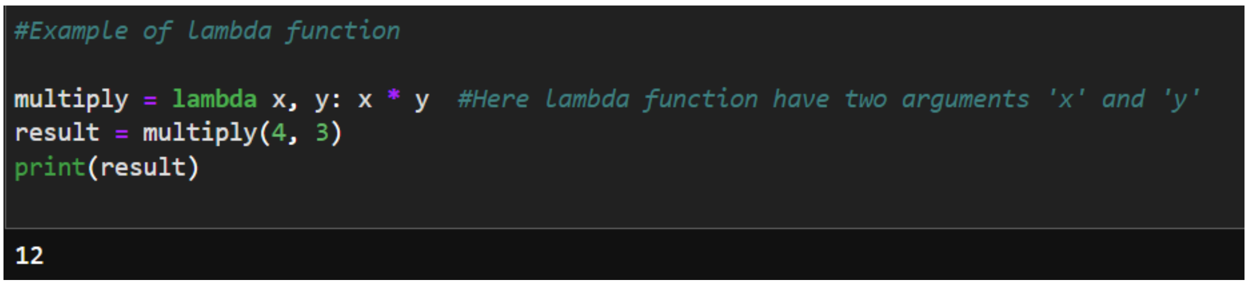
```
lambda arguments: expression
```

Lambda functions can take any arguments, but they can only have one expression. The expression is evaluated and returned as the result of the function

Syntax: Lambda functions are defined using the **‘lambda’** keyword, whereas regular functions are defined using the ’**def**’ keyword.
Function Name: Lambda functions are anonymous, meaning they don't have a name assigned to them. They are generally used when a function is needed for a short duration and does not require a formal function definition.
Function Body: Lambda functions can only have a single expression as their body, whereas regular functions can have multiple statements and a more complex structure.
Return Statement: In a lambda function, the expression itself is automatically returned as the result. Regular functions use the **‘return’** statement to explicitly return a value.

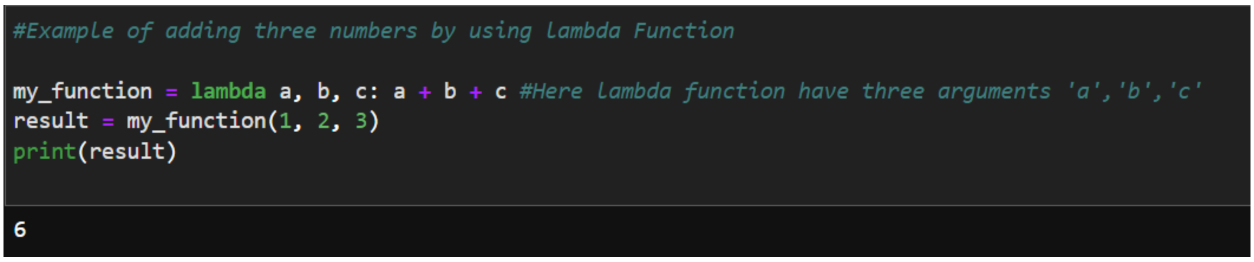
2. **Can a lambda function in Python have multiple arguments? If yes, how can you define and use them?**

Ans: Yes, a lambda function in Python can have multiple arguments. You can define and use multiple arguments in a lambda function by separating them with commas, just like in a regular function.



```
#Example of lambda function

multiply = lambda x, y: x * y  #Here lambda function have two arguments 'x' and 'y'
result = multiply(4, 3)
print(result)
```

```
12
```

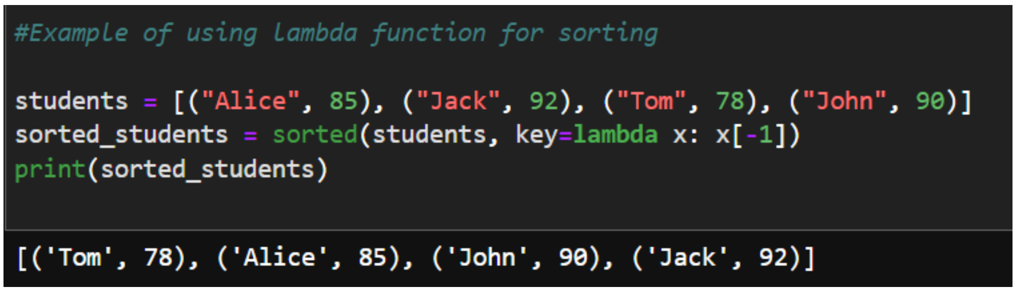


```
#Example of adding three numbers by using lambda Function

my_function = lambda a, b, c: a + b + c #Here lambda function have three arguments 'a','b','c'
result = my_function(1, 2, 3)
print(result)
```

```
6
```

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

Ans: Lambda functions in Python are typically used in situations where a small, one-time function is required, especially when it's used as an argument to another function. They offer a more concise and readable way to define functions on the fly without the need for a full function definition.



```python
#Example of using lambda function for sorting

students = [("Alice", 85), ("Jack", 92), ("Tom", 78), ("John", 90)]
sorted_students = sorted(students, key=lambda x: x[-1])
print(sorted_students)
```

```
[('Tom', 78), ('Alice', 85), ('John', 90), ('Jack', 92)]
```

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

Ans:

Advantages of Lambda Functions:

Conciseness: Lambda functions provide a concise way to define small, one-time functions without the need for a full function definition. They can be defined in a single line, making the code more compact and readable.

Readability: Since lambda functions are often used for simple operations, their inline definition and short length make the code more readable and self-contained. They can be easily understood within the context of their usage.

Function as Arguments: Lambda functions are particularly useful when a function is required as an argument to another function, such as in sorting, filtering, or mapping operations. Instead of defining a separate named function, lambda functions allow you to define the function directly at the point of use, making the code more focused and concise.

Scope: Lambda functions have access to the variables defined in the containing scope. This allows them to capture and use variables from the surrounding code without the need for explicit parameter passing.

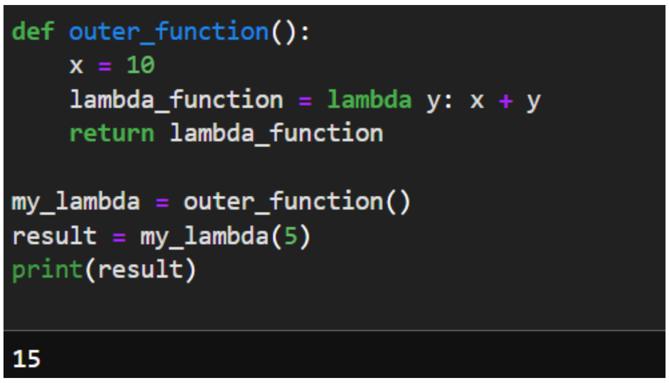
Limitations of Lambda Functions:

Single Expression: Lambda functions can only have a single expression as their body. They are not suitable for complex operations that require multiple statements or control flow structures.

No Documentation: Since lambda functions are anonymous and do not have names, they lack the ability to include docstrings or provide detailed documentation. This can make it harder for other developers to understand the intended purpose and usage of the function.
Limited Functionality: Due to their limited body structure, lambda functions are not capable of performing advanced or extensive operations. They are best suited for simple, one-line functions.
Limited Reusability: Lambda functions are typically used for immediate, one-time use cases. They are not designed for reuse across multiple parts of the codebase. If a function needs to be reused, it is generally recommended to define a regular named function instead.

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 have access to variables defined outside of their own scope. They can capture and use variables from the containing scope in which they are defined. This is known as variable capturing or closure

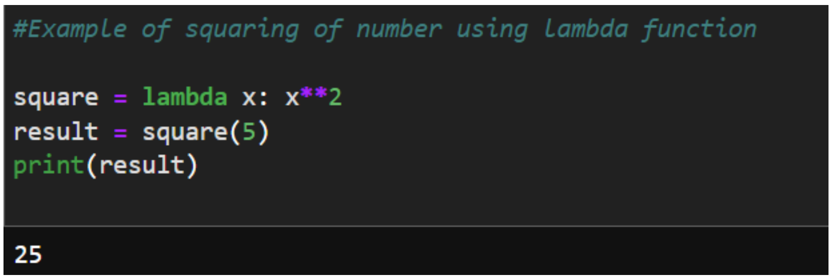


```python
def outer_function():
    x = 10
    lambda_function = lambda y: x + y
    return lambda_function

my_lambda = outer_function()
result = my_lambda(5)
print(result)
```

```
15
```

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

   Ans:

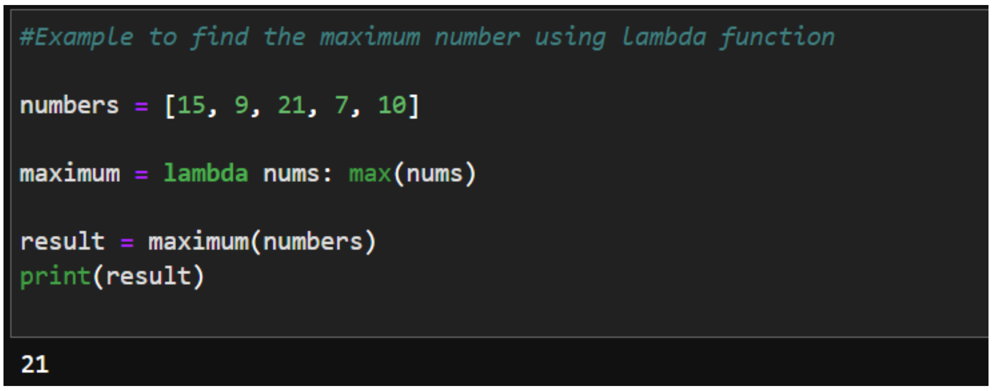


```python
#Example of squaring of number using lambda function

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

```
25
```

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

   Ans:

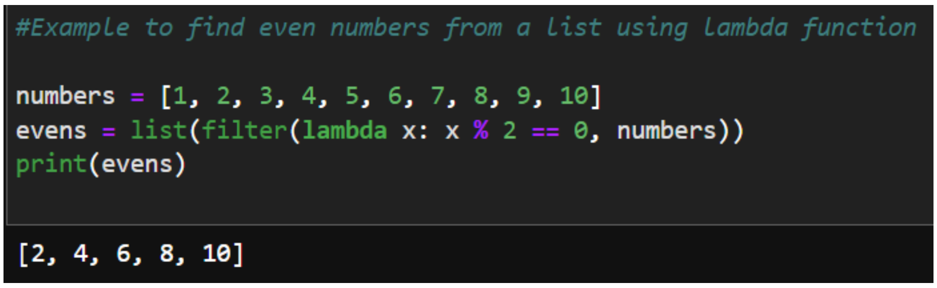


```python
#Example to find the maximum number using lambda function

numbers = [15, 9, 21, 7, 10]

maximum = lambda nums: max(nums)

result = maximum(numbers)
print(result)
```

```
21
```

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

Ans:

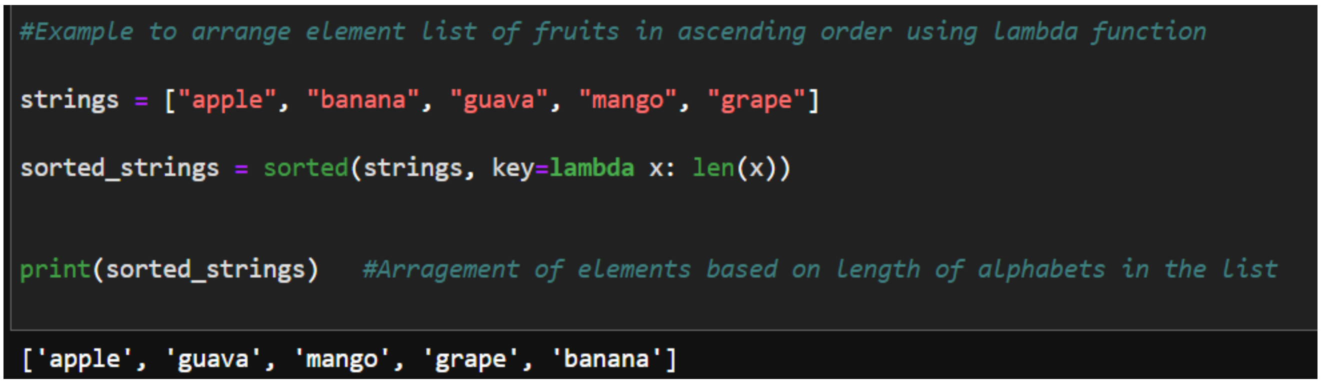


```
#Example to find even numbers from a list using lambda function

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
evens = list(filter(lambda x: x % 2 == 0, numbers))
print(evens)
```

```
[2, 4, 6, 8, 10]
```

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

Ans:

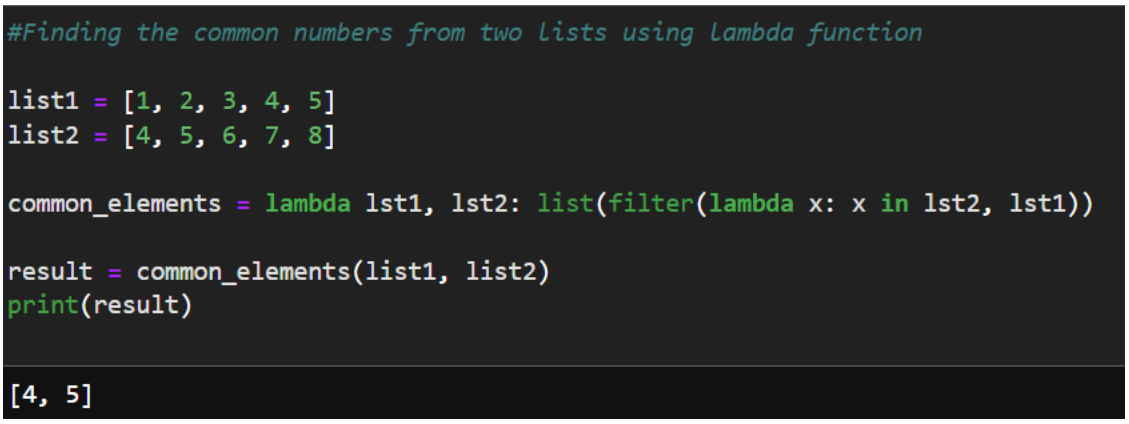


```
#Example to arrange element list of fruits in ascending order using lambda function

strings = ["apple", "banana", "guava", "mango", "grape"]

sorted_strings = sorted(strings, key=lambda x: len(x))


print(sorted_strings)   #Arragement of elements based on length of alphabets in the list
```

```
['apple', 'guava', 'mango', 'grape', 'banana']
```

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:

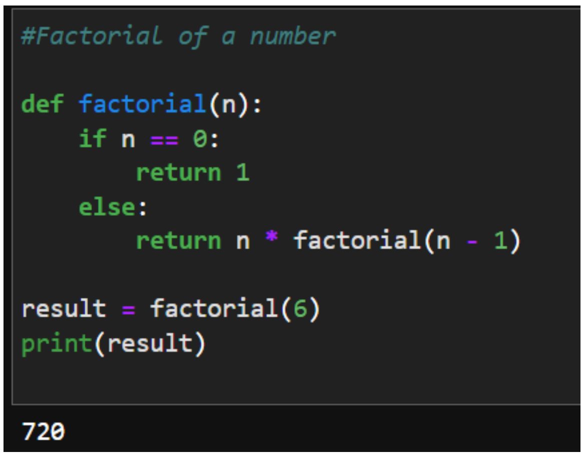


```
#Finding the common numbers from two lists using lambda function

list1 = [1, 2, 3, 4, 5]
list2 = [4, 5, 6, 7, 8]

common_elements = lambda lst1, lst2: list(filter(lambda x: x in lst2, lst1))

result = common_elements(list1, list2)
print(result)
```

```
[4, 5]
```

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

Ans:

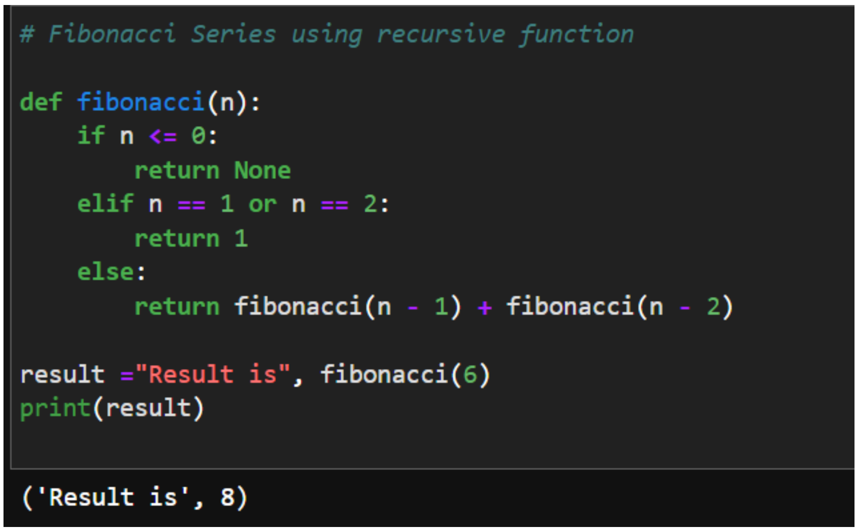


```python
#Factorial of a number

def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)

result = factorial(6)
print(result)
```

```
720
```

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

Ans:

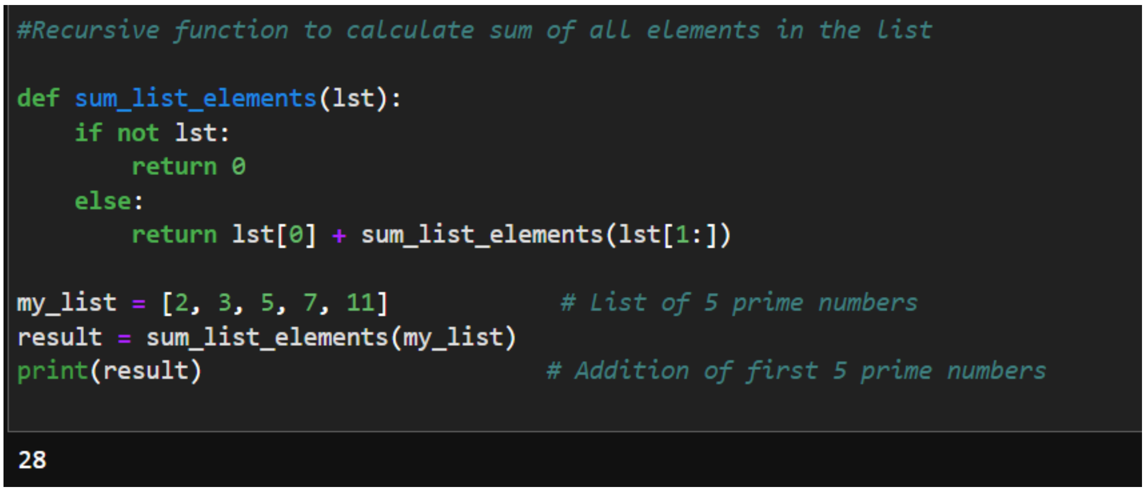


```python
# Fibonacci Series using recursive function

def fibonacci(n):
    if n <= 0:
        return None
    elif n == 1 or n == 2:
        return 1
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)

result ="Result is", fibonacci(6)
print(result)
```

```
('Result is', 8)
```

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

Ans:

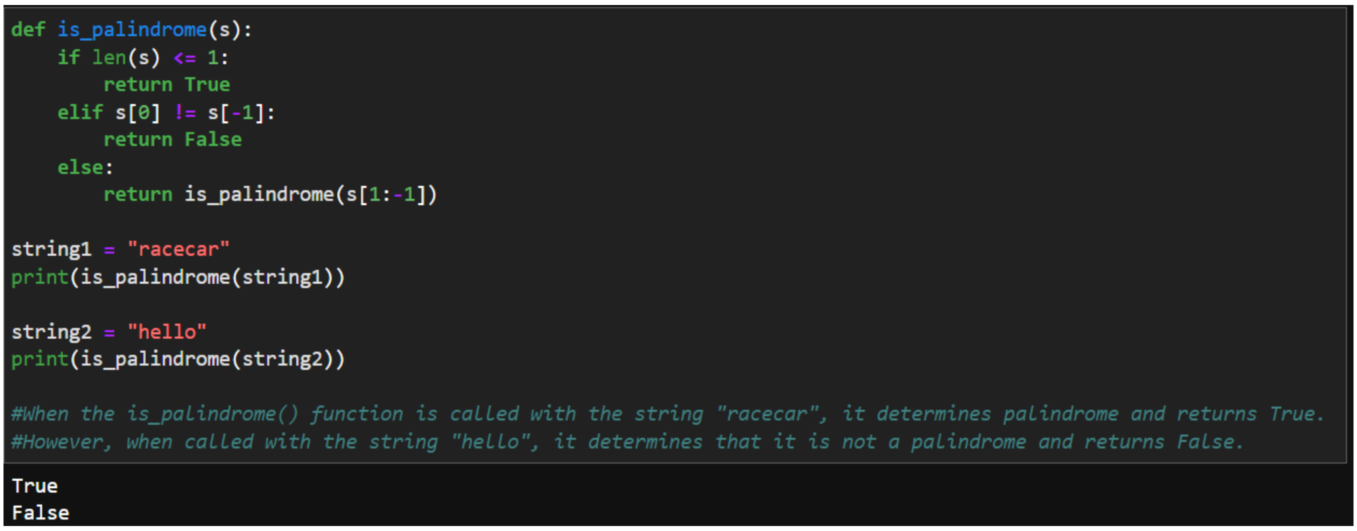


```python
#Recursive function to calculate sum of all elements in the list

def sum_list_elements(lst):
    if not lst:
        return 0
    else:
        return lst[0] + sum_list_elements(lst[1:])

my_list = [2, 3, 5, 7, 11]             # List of 5 prime numbers
result = sum_list_elements(my_list)
print(result)                          # Addition of first 5 prime numbers
```

```
28
```

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

Ans:

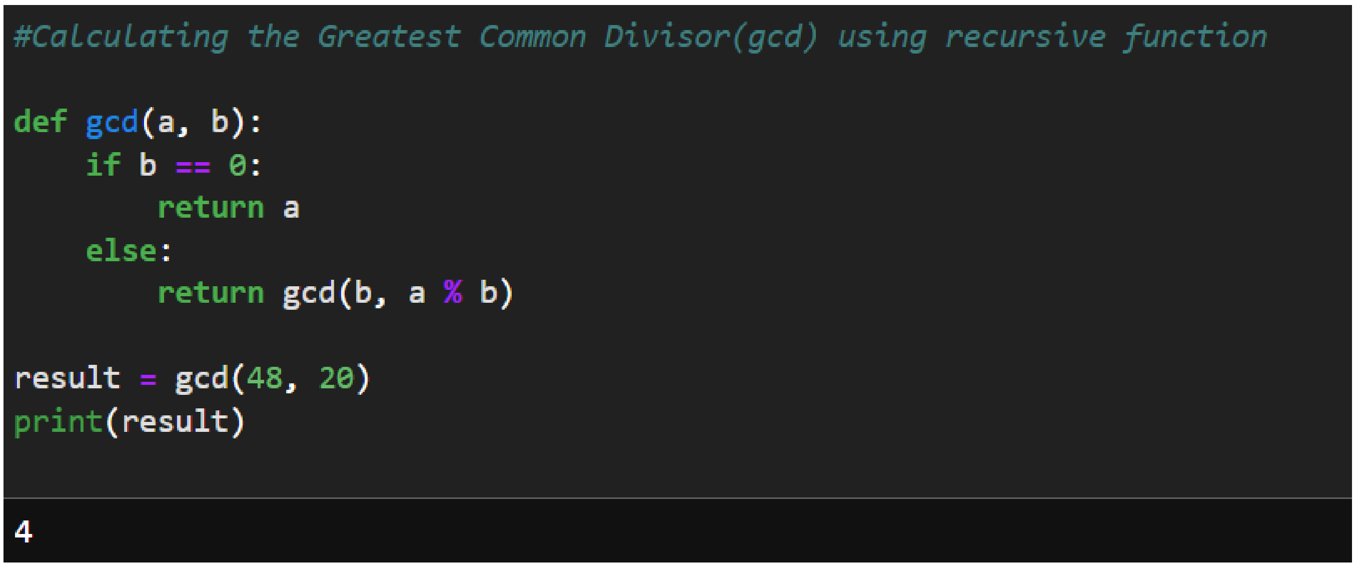


```python
def is_palindrome(s):
    if len(s) <= 1:
        return True
    elif s[0] != s[-1]:
        return False
    else:
        return is_palindrome(s[1:-1])

string1 = "racecar"
print(is_palindrome(string1))

string2 = "hello"
print(is_palindrome(string2))

#When the is_palindrome() function is called with the string "racecar", it determines palindrome and returns True.
#However, when called with the string "hello", it determines that it is not a palindrome and returns False.
```

```
True
False
```

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

Ans:



```
#Calculating the Greatest Common Divisor(gcd) using recursive function

def gcd(a, b):
    if b == 0:
        return a
    else:
        return gcd(b, a % b)

result = gcd(48, 20)
print(result)
```

```
4
```