In [ ]:
```
1. What is a lambda function in Python, and how does it differ from a regular function?
```

In [ ]:
```
A lambda function is a small, anonymous function defined using the lambda keyword. It differs
from a regular function in that it:
Has no name (anonymous).
Contains a single expression and returns its result implicitly.
Is typically used for short, simple operations.
```

In [41]:
```
# Lambda function
square = lambda x: x ** 2
print(square(5))  # Output: 25

# Regular function
def square(x):
    return x ** 2
print(square(5))  # Output: 25
```
```
25
25
```

In [ ]:
```
2. Can a lambda function in Python have multiple arguments?
```

In [47]:
```
 # Yes, lambda functions can have multiple arguments

 multiply = lambda a, b: a * b
print(multiply(3, 4))  # Output: 12
```
```
12
```

In [ ]:
```
3. How are lambda functions typically used in Python?
```

In [49]:
```
# Lambda functions are often used with functions like map(), filter(), and sorted()

numbers = [1, 2, 3, 4, 5]
squared = map(lambda x: x ** 2, numbers)
print(list(squared))  # Output: [1, 4, 9, 16, 25]
```
```
[1, 4, 9, 16, 25]
```

In [ ]:
```
4. Advantages and limitations of lambda functions
```

In [ ]:
```
Advantages:

Concise syntax for short functions.
Useful for functional programming tools.
Limitations:

Limited to a single expression.
Less readable for complex logic.
No name for debugging or reuse.
```

In [ ]:
```
5. Can lambda functions access variables outside their scope?
```

In [59]:
```
# Yes, lambda functions can access variables in the enclosing scope.

x = 10
double = lambda y: y * x
print(double(5))  # Output: 50
```
```
50
```

In [ ]:
```
6. Lambda function to calculate the square of a given number
```

In [53]:
```
square = lambda x: x ** 2
print(square(7))  # Output: 49
```
```
49
```

In [ ]:
```
7. Lambda function to find the maximum value in a list
```

In [55]:
```
find_max = lambda lst: max(lst)
print(find_max([10, 20, 30, 40]))  # Output: 40
```
```
40
```

In [ ]:
```
8. Lambda function to filter out even numbers
```

In [57]:
```
numbers = [1, 2, 3, 4, 5, 6]
evens = list(filter(lambda x: x % 2 == 0, numbers))
print(evens)  # Output: [2, 4, 6]
```
```
[2, 4, 6]
```

In [ ]:
```
9. Lambda function to sort strings by length
```

In [61]:
```
strings = ["apple", "kiwi", "banana", "cherry"]
sorted_strings = sorted(strings, key=lambda s: len(s))
print(sorted_strings)  # Output: ['kiwi', 'apple', 'cherry', 'banana']
```
```
['kiwi', 'apple', 'banana', 'cherry']
```

In [ ]:
```
10. Lambda function to find common elements between two lists
```

In [63]:
```
common_elements = lambda list1, list2: list(filter(lambda x: x in list2, list1))
print(common_elements([1, 2, 3], [2, 3, 4]))  # Output: [2, 3]
```
```
[2, 3]
```

In [ ]:
```
11. Recursive function for factorial
```

In [65]:
```
def factorial(n):
    if n == 0 or n == 1:
        return 1
    return n * factorial(n - 1)

print(factorial(5))  # Output: 120
```
```
120
```

In [ ]:
```
12. Recursive function for Fibonacci number
```

In [67]:
```
def fibonacci(n):
    if n <= 1:
        return n
    return fibonacci(n - 1) + fibonacci(n - 2)

print(fibonacci(7))  # Output: 13
```
```
13
```

In [ ]:
```
13. Recursive function for sum of list elements
```

In [69]:
```
def sum_list(lst):
    if not lst:
        return 0
    return lst[0] + sum_list(lst[1:])

print(sum_list([1, 2, 3, 4]))  # Output: 10
```
```
10
```

In [ ]:
```
14. Recursive function to check palindrome
```

In [82]:
```
 def is_palindrome(s):
     if len(s) <= 1:
         return True
     if s[0] != s[-1]:
         return False
     return is_palindrome(s[1:-1])

print(is_palindrome("rader"))  # output True
print(is_palindrome("hello"))  # output False
```
```
False
False
```

In [ ]:
```
15. Recursive function for GCD
```

In [84]:
```
def gcd(a, b):
    if b == 0:
        return a
    return gcd(b, a % b)

print(gcd(48, 18))  # Output: 6
```

```
6
```