In [ ]:

```
Q.1    What is a lambda function in Python, and how does it differ from a regular
       Lambda functions are similar to user-defined functions but without a name.
       Lambda functions are efficient whenever you want to create a function that
       that is, expressions that are usually a single line of a statement.
       Lambda functions are often used in scenarios where a small, temporary funct
       filter(), or sorted().

        lambda functions provide a concise and convenient way to create small, ano
        and better suited for larger, reusable code blocks.
```

In [ ]:

```
Q.2 Can a lambda function in Python have multiple arguments? If yes, how can you d
    them?
Ans: Yes, a lambda function in Python can have multiple arguments. The syntax for
     similar to that of a single argument.

        lambda argument1, argument2, ..., argumentN: expression

Example:
add = lambda x, y: x + y
result = add(3, 5)
print(result)
```

In [ ]:

```
Q.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 we need a s
     anonymous function for a short duration. They are often employed in scenarios
     want to pass a simple function as an argument to higher-order functions, like
     filter(), or sorted().

     # List of numbers
numbers = [1, 2, 3, 4, 5]

# Use map() with a lambda function to square each number
squared_numbers = map(lambda x: x**2, numbers)

# Convert the result to a list and print it
result_list = list(squared_numbers)
print(result_list)
```

In [ ]:

```
Q.4 What are the advantages and limitations of lambda functions compared to regula
Python?
Ans:Advantages of Lambda Functions:
    1 : Conciseness: Lambda functions are concise and can be defined in a single l
        simple operations
    2 : Functional Programming: Lambda functions are often used in functional prog
        and sorted() to pass small, inline functions as arguments.
    Limitations of Lambda Functions:
    1:  Limited Expressiveness: Lambda functions are limited to a single expressio
        If your function needs more than one expression or includes control flow s

    2:Readability: While lambda functions can be concise, they may reduce code rea
        descriptive names are often more readable.
```

In [ ]:

```
Q.5 Are lambda functions in Python able to access variables defined outside of the
    Explain with an example.

Ans: Yes, lambda functions in Python can access variables defined outside of their
     When a lambda function is created inside another function, it can remember an
    of variables from that outer function. This is possible because of a feature c


    def outer_function(x):
    # Lambda function inside the outer function
    inner_lambda = lambda y: x + y
    return inner_lambda

# Create an instance of the outer function with x = 10
closure_instance = outer_function(10)

# Use the closure_instance as a lambda function with y = 5
result = closure_instance(5)

# Display the result
print(result)
```

In [ ]:

```
Q.6 Write a lambda function to calculate the square of a given number
Ans:
square = lambda x: x**2

# Use the lambda function to calculate the square of a given number
number = 4
result = square(number)

# Display the result
print(f"The square of {number} is: {result}")
```

In [ ]:

```
Q.7 Create a lambda function to find the maximum value in a list of integers.
Ans: # List of integers
numbers = [10, 5, 8, 20, 15]

# Lambda function to find the maximum value in the list
max_value = lambda lst: max(lst)

# Use the lambda function to find the maximum value
result = max_value(numbers)

# Display the result
print(f"The maximum value in the list is: {result}")
```

In [ ]:

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

Ans: # List of numbers
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Lambda function to filter out even numbers
filter_even = lambda x: x % 2 != 0

# Use the lambda function with filter() to get a new list of odd numbers
odd_numbers = list(filter(filter_even, numbers))

# Display the result
print("List of odd numbers:", odd_numbers)
```

In [ ]:

```
Q.9  Write a lambda function to sort a list of strings in ascending order based on
string.

Ans:
# List of strings
strings = ["apple", "banana", "kiwi", "orange", "grape"]

# Lambda function to sort strings based on length
sorted_strings = sorted(strings, key=lambda x: len(x))

# Display the result
print("Sorted strings by length:", sorted_strings)
```

In [ ]:

In [ ]:

```
Q.10  Create a lambda function that takes two lists as input and returns a new lis
      common elements between the two lists.

Ans:  # Lambda function to find common elements between two lists
common_elements = lambda list1, list2: list(filter(lambda x: x in list1, list2))

# Example usage:
list1 = [1, 2, 3, 4, 5]
list2 = [3, 4, 5, 6, 7]

result = common_elements(list1, list2)

print("Common elements:", result)
```

In [ ]:

```
Q.11   Write a recursive function to calculate the factorial of a given positive i
Ans:
def factorial_recursive(n):
    # Base case: factorial of 0 is 1
    if n == 0:
        return 1
    # Recursive case: factorial(n) = n * factorial(n-1)
    else:
        return n * factorial_recursive(n - 1)

# Example usage:
number = 5
result = factorial_recursive(number)

print(f"The factorial of {number} is: {result}")
```

In [ ]:

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

Ans:
def fibonacci_recursive(n):
    # Base cases: F(0) = 0, F(1) = 1
    if n == 0:
        return 0
    elif n == 1:
        return 1
    # Recursive case: F(n) = F(n-1) + F(n-2)
    else:
        return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)

# Example usage:
n = 6
result = fibonacci_recursive(n)

print(f"The {n}-th Fibonacci number is: {result}")
```

In [ ]:

```
Q.13  Create a recursive function to find the sum of all the elements in a given l
Ans:  def recursive_sum(lst):
    # Base case: if the list is empty, the sum is 0
    if not lst:
        return 0
    # Recursive case: sum the first element and the sum of the rest of the list
    return lst[0] + recursive_sum(lst[1:])

# Example usage:
numbers = [1, 2, 3, 4, 5]

result = recursive_sum(numbers)

print(f"The sum of the elements in the list is: {result}")
```

In [ ]:

```
Q.14  Write a recursive function to determine whether a given string is a palindro
Ans:
def is_palindrome_recursive(s):
    # Base case: if the string has 0 or 1 characters, it's a palindrome
    if len(s) <= 1:
        return True
    # Recursive case: compare the first and last characters, and check the inner s
    return s[0] == s[-1] and is_palindrome_recursive(s[1:-1])

# Example usage:
string1 = "radar"
string2 = "hello"

result1 = is_palindrome_recursive(string1)
result2 = is_palindrome_recursive(string2)

print(f"Is '{string1}' a palindrome? {result1}")
print(f"Is '{string2}' a palindrome? {result2}")
```

In [ ]:

```
Q.15  Implement a recursive function to find the greatest common divisor (GCD) of
Ans:  def gcd_recursive(a, b):
    # Base case: GCD(a, 0) = a
    if b == 0:
        return a
    # Recursive case: GCD(a, b) = GCD(b, a % b)
    return gcd_recursive(b, a % b)

# Example usage:
num1 = 48
num2 = 18

result = gcd_recursive(num1, num2)

print(f"The GCD of {num1} and {num2} is: {result}")
```