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

**Ans:** *Lambda functions, also known as anonymous functions or lambda expressions, are a concise way to create small, unnamed functions in Python. They are defined using the `lambda` keyword, followed by a list of arguments, a colon `:`, and an expression. Lambda functions are typically used for simple operations that can be expressed in a single line of code. The general syntax of a lambda function is:*



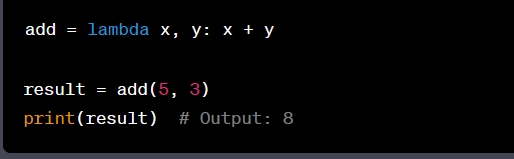
Key differences between lambda functions and regular (or named) functions:

* Anonymous Nature: Lambda functions are anonymous, meaning they don't have a name like regular functions. They are often used for short, one-time tasks and are not reusable across the codebase.

* Simplicity: Lambda functions are concise and designed for simple operations, making them ideal for functions that don't require extensive code.

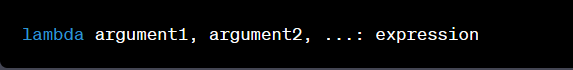
* No Statements: Lambda functions can only contain a single expression, not multiple statements. Regular functions can have multiple statements and more complex logic.

Here's an example of a lambda function that adds two numbers:

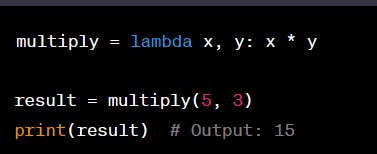


1. **Can a lambda function in Python have multiple arguments? If yes, how can you define and use them?**

**Ans:** *Yes, a lambda function in Python can have multiple arguments. You can define and use them in a lambda function in the same way as you define arguments in a regular function, by separating them with commas. Here's the general syntax:*

**

*Here's an example of a lambda function with multiple arguments:*

**

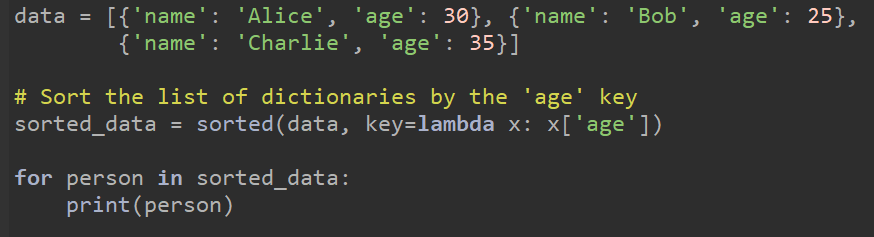
*In this example, the lambda function `multiply` takes two arguments, `x` and `y`, and returns their product. When you call the lambda function with two values, it performs the multiplication and returns the result.*

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

**Ans***; Lambda functions are typically used in Python for concise, one-time, small, and simple operations. They are commonly used in situations where you need to pass a simple function as an argument to another function, like in the case of higher-order functions. Here's a typical use case for lambda functions in Python:*

*Sorting with `sorted()` and `lambda`:*

*The `sorted () ` function in Python allows you to sort lists or other iterable data structures. You can use a `lambda` function to specify a custom sorting key or criteria.*

*Example:* ****

*In this example, the sorted() function is used to sort a list of dictionaries by the 'age' key. The key parameter takes a function (in this case, a lambda function) that extracts the 'age' value from each dictionary. This allows you to sort the list based on the 'age' attribute. The resulting sorted\_data list is sorted by age in ascending order.*

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

**Ans:** *Lambda functions and regular functions in Python each have their own set of advantages and limitations. Let's discuss these in more detail:*

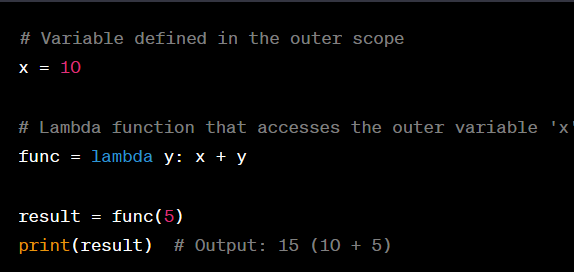
|  |  |
| --- | --- |
| **Advantages of Lambda Functions:**  *1. Conciseness: Lambda functions are concise and allow you to define simple functions in a single line of code. This can lead to more readable code when used appropriately.*  *2. Use as a Short Function: Lambda functions are useful for small, one-time operations where defining a regular function might be overkill.*  *3. Functional Programming: Lambda functions are often used in functional programming constructs like `map`, `filter`, and `sorted` to perform operations on sequences of data.*  **Limitations of Lambda Functions:**  *1. Limited Complexity: Lambda functions are restricted to a single expression and cannot contain multiple statements or complex logic.*  *2. Limited Reusability: Lambda functions are typically not reusable across different parts of your code. They are usually designed for specific, short-term use cases.*  *3. Readability: Complex lambda functions can reduce code readability, as they may not be self-explanatory.* | **Advantages of Regular Functions:**  *1. Complex Logic: Regular functions allow you to define complex logic with multiple statements, which can improve code organization and readability for more involved tasks.*  *2. Reusability: Regular functions are highly reusable and can be called from various parts of your code, promoting modularity.*  *3. Documentation: Regular functions encourage the use of docstrings, making code more self-documenting and understandable.*  **Limitations of Regular Functions:**  *1. Verbosity: Regular functions require more boilerplate code with the `def` keyword, indentation, and sometimes explicit `return` statements, which can make code longer and less concise.*  *2. Scope: Regular functions can introduce variables into various scopes (local, global, and nonlocal), which may lead to unintentional side effects or make debugging more challenging.*  *3. Customizability: While regular functions provide greater flexibility, they may require more effort for customization and are often more complex than necessary for simple tasks.* |

**5. Are lambda functions in Python able to access variables defined outside of their own scope?**

**Explain with an example.**

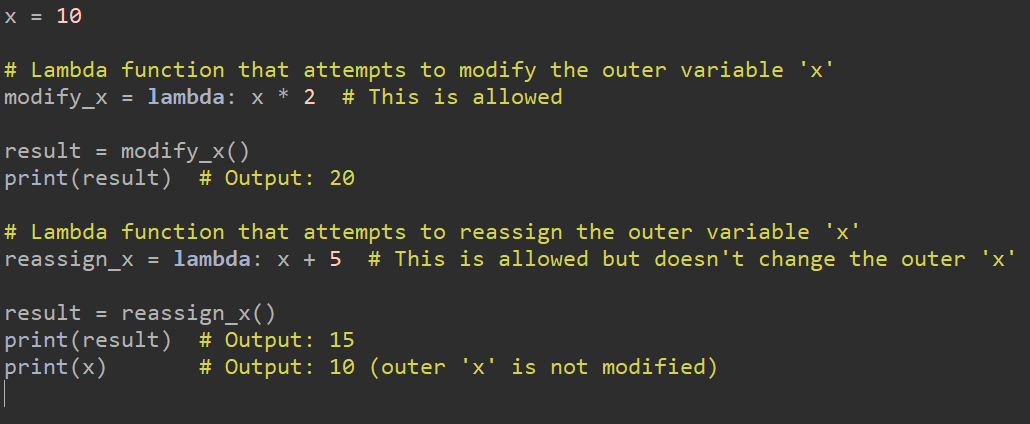
**Ans:** *Yes, lambda functions in Python can access variables defined outside of their own scope. They can access and use variables from the enclosing scope in a read-only manner. This is known as "lexical scoping" or "closure."*

*Here's an example to illustrate how lambda functions can access variables from the outer scope:*

**

*In this example, the lambda function `func` takes an argument `y` and adds it to the variable `x`, which is defined in the outer scope. The lambda function is able to access and use the value of `x` from the enclosing scope.*

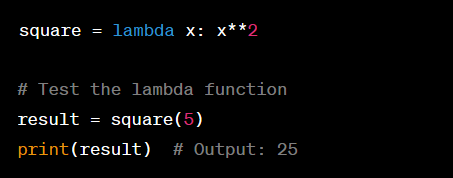
*However, it's important to note that lambda functions can only read variables from the enclosing scope. They cannot modify or reassign those variables. If you attempt to modify a variable from the enclosing scope within a lambda function, Python will treat it as a new local variable in the lambda function's scope.*

**

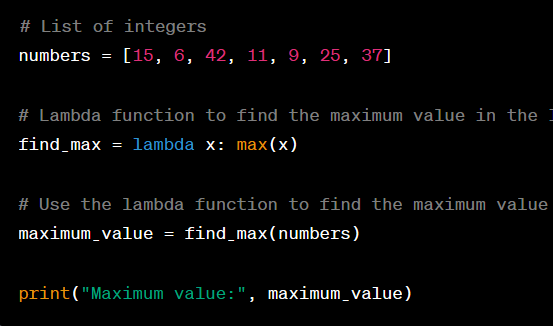
*In the above example, the lambda functions `modify\_x` and `reassign\_x` can access the variable `x` from the enclosing scope but cannot modify or reassign it.*

*The ability of lambda functions to access variables from the enclosing scope makes them useful in various contexts, such as when defining custom sorting or filtering criteria in functions like `sorted()` or `filter()`.*

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

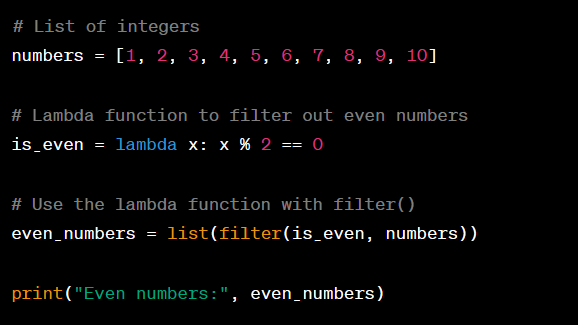
**Ans:** ****

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

**Ans:** ****

Output: ****

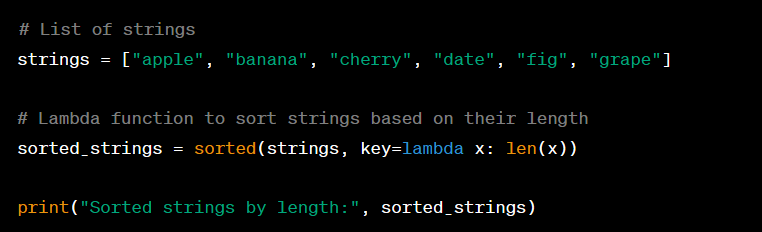
1. **Implement a lambda function to filter out all the even numbers from a list of integers.**

**Ans:** ****

*In this example, the lambda function is\_even checks if a number is even by using the modulo operator (%). The filter() function is used to filter the even numbers from the list based on the lambda function's condition, and the result is converted to a list. The even\_numbers list will contain the filtered even integers*

*Output:* ****

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

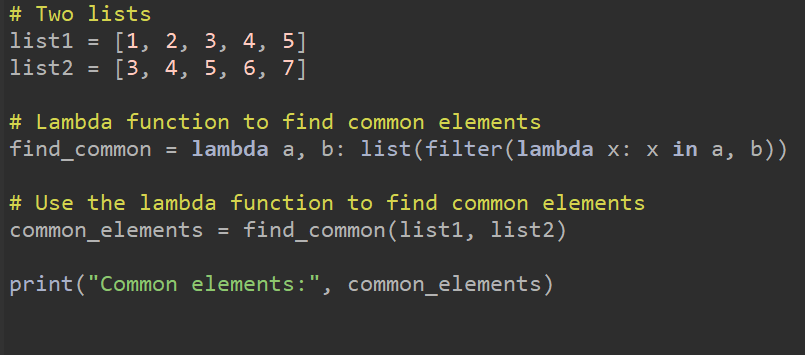
**Ans:** ****

*In this example, the lambda function is used as the key argument for the sorted() function. The lambda function lambda x: len(x) calculates the length of each string x, and the sorted() function sorts the strings based on their lengths in ascending order. The result, stored in the sorted\_strings list, will contain the sorted strings.*

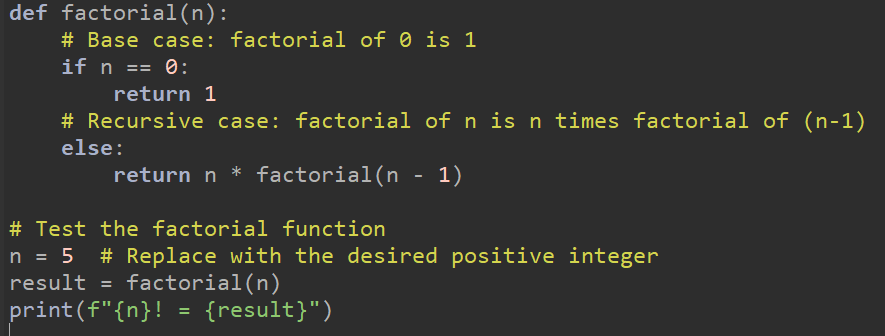
*Output:*

**

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

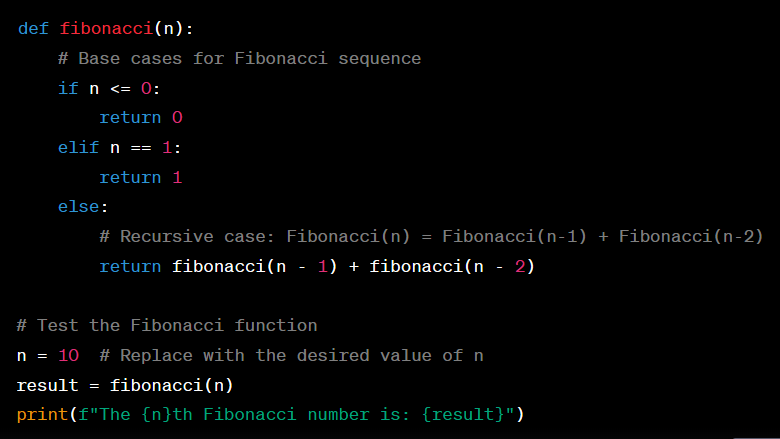
**Ans:** ****OUTPUT: 

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

**Ans:** ****

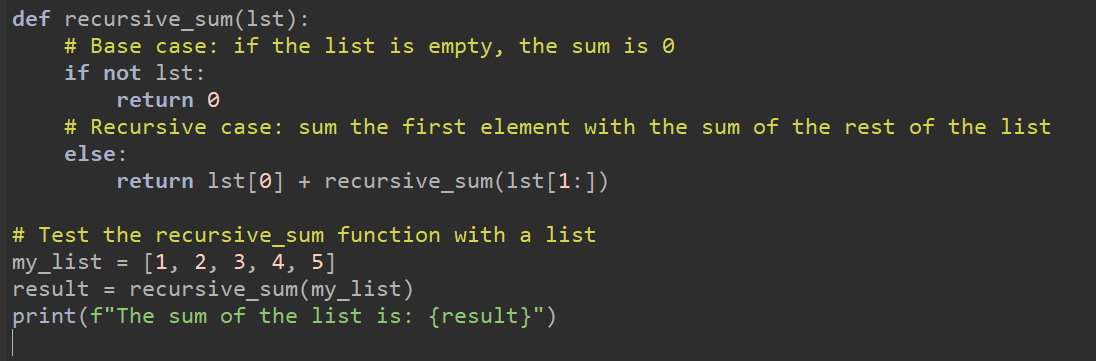
**OUTPUT :** ****

1. **Implement a recursive function to compute the nth Fibonacci number.**

**Ans:** ****

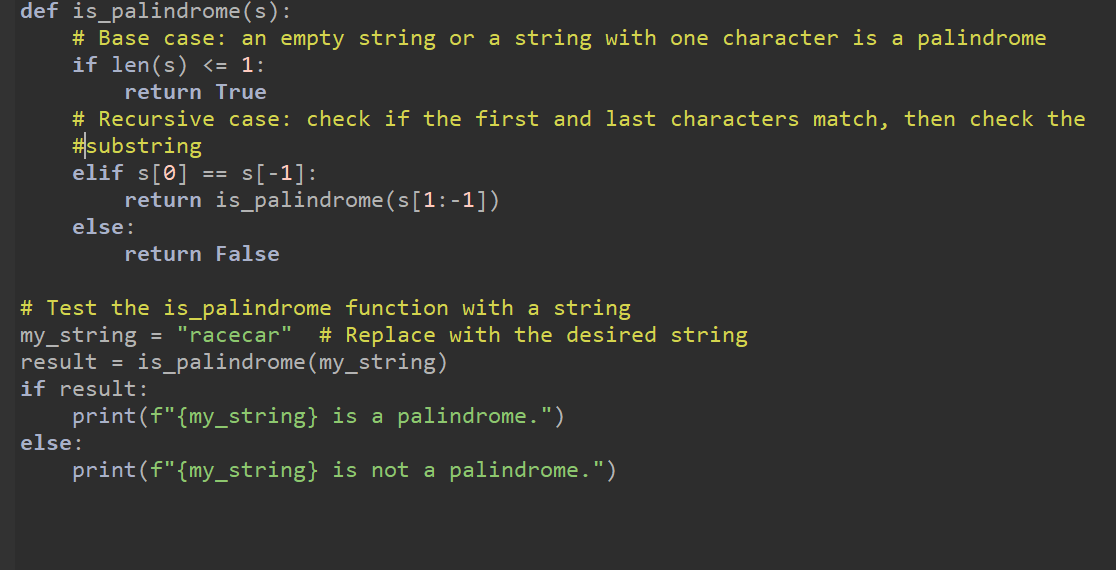
**OUTPUT:** ****

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

**Ans:** ****

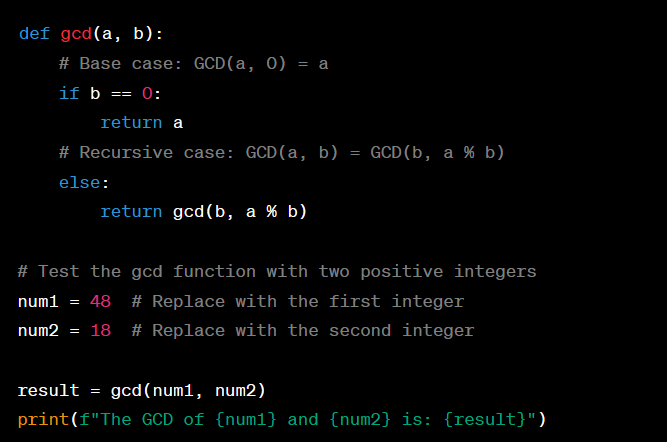
OUTPUT: 

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

**ANS:** ****

OUTPUT: 

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

**Ans:** ****

OUTPUT: 