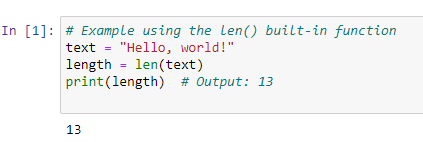
Python (Functions Assignment) – 1 Solution

Submitted by: Sweta Dhara

1. In Python, what is the difference between a built-in function and a user-defined function? Provide an example of each.

A: Built-in Function: Built-in functions are pre-defined functions that are provided by Python's standard library. These functions are readily available for use without requiring any additional code or import statements. Python comes with a rich set of built-in functions that serve various purposes, such as manipulating data, performing mathematical operations, handling strings, and more.

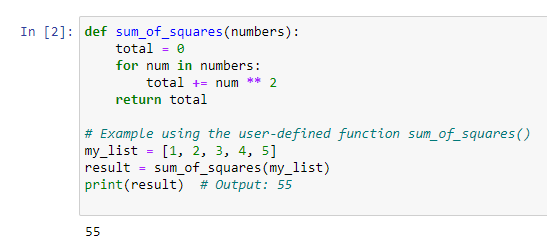
Example of a built-in function: len() The len() function is used to determine the length of a sequence, such as a string, list, tuple, or dictionary.



User-defined Function: User-defined functions, as the name suggests, are functions created by the programmer to fulfil specific requirements in their code. These functions allow us to encapsulate a block of code and execute it whenever needed by calling the function. They enhance code reusability and organization.

To create a user-defined function, you use the def keyword, followed by the function name, a list of parameters (if any), and a colon. The function body is indented under the function definition.

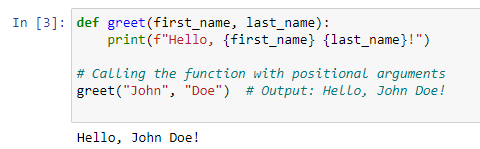
Example of a user-defined function: Sum of Squares



2. How can you pass arguments to a function in Python? Explain the difference between positional Arguments and keyword arguments.

A: Positional Arguments: Positional arguments are passed to a function based on their position or order. When we call a function, the values you provide are assigned to the function's parameters in the order they appear in the function definition. The number of arguments and their order must match the function's parameter list.

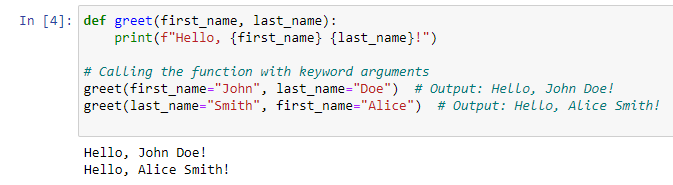
Example of positional arguments:



In this example, the function greet takes two positional arguments: first\_name and last\_name. When calling the function, we pass the values "John" and "Doe" in that order.

Keyword Arguments: Keyword arguments are passed to a function using the names of the parameters as keys. Instead of relying on the position, we explicitly mention the parameter names and their corresponding values when calling the function. This way, the order of arguments does not matter.

Example of keyword arguments:



In the above example, we are passing keyword arguments to the greet function. The function signature remains the same, but the arguments are provided with their corresponding parameter names, making the code more readable and reducing the risk of passing values in the wrong order.

The main difference between positional and keyword arguments is how the values are passed to the function. Positional arguments rely on the order of appearance, while keyword arguments use parameter names for assignment.

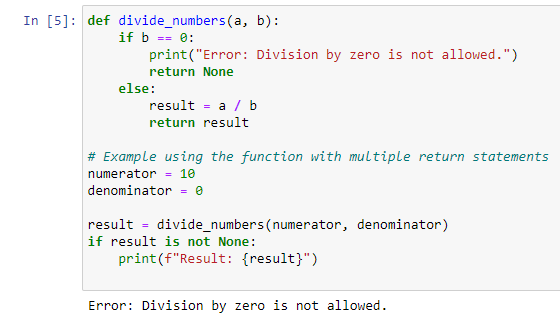
3. What is the purpose of the return statement in a function? Can a function have multiple return

statements? Explain with an example.

A: The return statement in a function serves the purpose of providing a result or value back to the caller of the function. When a function is called and the execution encounters a return statement, the function immediately exits, and the value specified after the return keyword is returned as the output of the function. If there is no return statement or the return statement lacks an expression, the function implicitly returns None.

Yes, a function can have multiple return statements. However, only one return statement will be executed during the function call. Once a return statement is encountered, the function execution stops, and the value associated with that return statement is returned to the caller.

Example of a function with multiple return statements:



In this example, the function divide\_numbers takes two arguments a and b, representing the numerator and denominator of a division operation. The function first checks if the denominator b is zero. If it is zero, the function prints an error message and returns None. Otherwise, it performs the division and returns the result.

When calling the function with numerator = 10 and denominator = 0, the return None statement is executed since the denominator is zero. As a result, the error message is printed, and the variable result is assigned the value None. If the denominator were non-zero, the division would be performed, and the result would be returned instead.

4. What are lambda functions in Python? How are they different from regular functions? Provide an example where a lambda function can be useful.

A: A lambda function is an anonymous function (i.e., defined without a name) that can take any number of arguments but, unlike normal functions, evaluates and returns only one expression. The [lambda functions](https://www.geeksforgeeks.org/python-lambda-anonymous-functions-filter-map-reduce/) can be used without any declaration in the namespace. The lambda functions defined above are like single-line functions. These functions do not have parenthesis like the def defined functions but instead, take parameters after the lambda keyword as shown above. There is no return keyword defined explicitly because the lambda function does return an object by default.

Lambda functions and regular functions in Python differ in several aspects:

Syntax:

Lambda functions are defined using the lambda keyword followed by a list of parameters and an expression. They are typically used for simple, one-liner functions.

Regular functions are defined using the def keyword followed by a function name, a list of parameters, and a block of code. They can be used for more complex functions that require multiple lines of code.

Function Name:

Lambda functions are anonymous functions, meaning they don't have a proper name. They are often used for short-lived and single-use scenarios.

Regular functions have a proper name, allowing them to be called, reused, and referenced from various parts of the code.

Number of Expressions:

Lambda functions can only consist of a single expression, which is evaluated and returned as the function's result.

Regular functions can contain multiple statements and expressions. They allow you to have more complex logic within the function body.

Return Statement:

Lambda functions automatically return the result of the expression without requiring an explicit return statement.

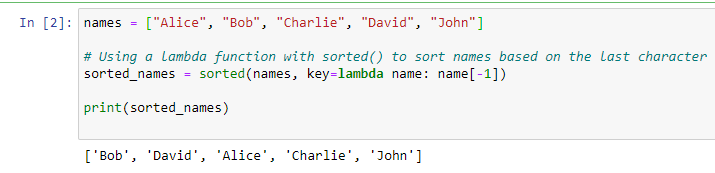
Regular functions use the return statement to specify the value to be returned explicitly.

Usage:

Lambda functions are typically used for short, simple operations where a small function is required, such as in functional programming with map(), filter(), and sorted().

Regular functions are more versatile and can be used for any kind of operation, regardless of complexity. They are suitable for code reusability and creating well-organized programs.

Example:

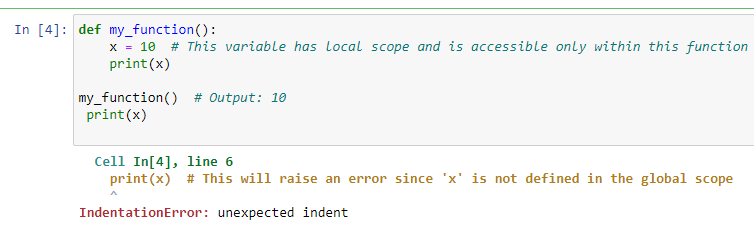


5. How does the concept of "scope" apply to functions in Python? Explain the difference between local scope and global scope.

A: In Python, the concept of "scope" refers to the region or context in which a particular name (variable, function, class, etc.) is valid and can be accessed. The scope determines the visibility and lifetime of a name within the code.

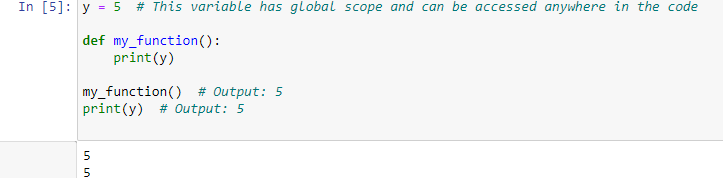
Local Scope: Local scope, also known as function scope, is the innermost scope and is specific to a particular function. Any variable defined inside a function is considered to have a local scope. These variables are only accessible within that function and are not visible outside of it. Once the function finishes executing, the local variables are destroyed, and their values are no longer accessible.

Example of local scope:



Global Scope: Global scope refers to the outermost scope in a Python program and is accessible throughout the entire program. Variables defined outside of any function or class have global scope and can be accessed from any part of the code, including within functions.

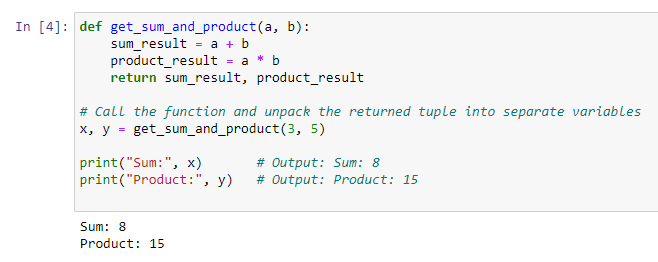
Example of global scope:



How can you use the "return" statement in a Python function to return multiple values?

A: In Python, you can use the return statement in a function to return multiple values by returning them as a tuple, a list, or any other suitable data structure. When you use the return statement with multiple expressions separated by commas, Python automatically packs them into a single object, allowing you to return multiple values as a single entity.

Here's an example of a function that returns multiple values as a tuple:



7. What is the difference between the "pass by value" and "pass by reference" concepts when it

comes to function arguments in Python?

A: When you pass function arguments by reference, those arguments are only references to existing values. In contrast, when you pass arguments by value, those arguments become independent copies of the original values.

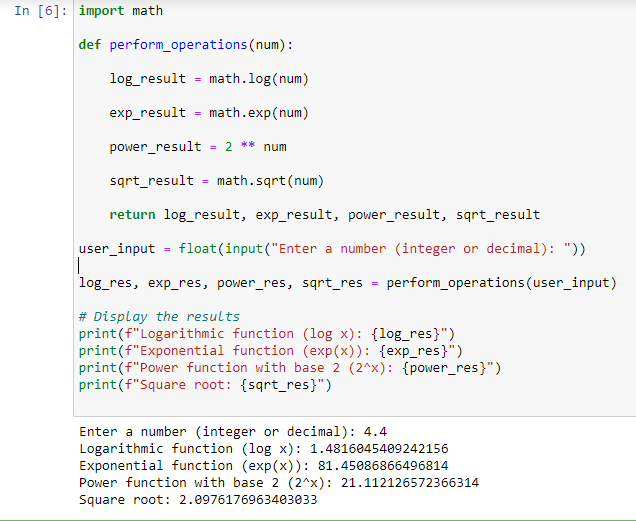
8. Create a function that can intake integer or decimal value and do following operations:

a. Logarithmic function (log x)

b. Exponential function (exp(x))

c. Power function with base 2 (2x)

d. Square root

A: 

9. Create a function that takes a full name as an argument and returns first name and last name.

A: 