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

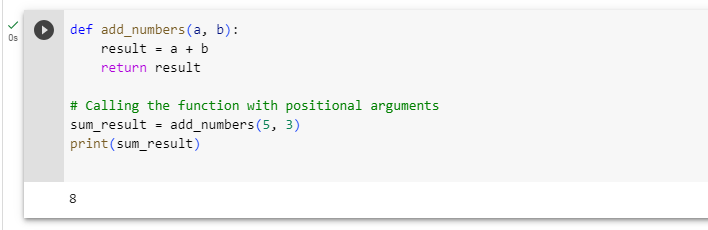
Ans.)

|  |  |
| --- | --- |
| **Built-in Functions** | **User-defined Functions** |
| Built-in functions are functions that are provided as part of the Python language. | User-defined functions are functions created by the user to perform a specific task or set of operations. |
| These functions are readily available and can be used directly without requiring any additional steps or code. | These functions are defined by the user using the 'def' keyword and can be called and reused throughout the program. |
| Ex.) print("Hello, World!") | Ex.) # Defining a user-defined function def greet(name):  print("Hello, " + name + "!")  # Calling the user-defined function greet("Alice") |

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

Ans.) In Python, you can pass arguments to a function in multiple ways. The two common ways of passing arguments to a function are:

1. **Positional Arguments:** Positional arguments are passed to a function based on their position or order in the function call. The order and number of arguments passed should match the order and number of parameters defined in the function's parameter list.

**Ex.**) 

**2.Keyword Arguments**: Keyword arguments are passed to a function with explicit names (keywords) along with the values. In this case, the order of the arguments doesn't matter, as the function matches the values to the parameter names based on the keywords.

Ex.)

A screenshot of a computer program

Description automatically generated

**3. What is the purpose of the return statement in a function? Can a function have multiple return statements? Explain with an example.**

Ans.) The ‘**return’** statement in a function is used to specify the value that the function should return when it is called. It serves the purpose of providing the result or output of the function's computations back to the caller.

A function can indeed have multiple ‘**return**’ statements. Once a ‘**return’** statement is executed, the function immediately exits, and the specified value is returned to the caller. Therefore, only the code up to the first executed ‘**return’** statement is executed, and subsequent ‘**return’** statements are not reached.

Ex.)

A screenshot of a computer code

Description automatically generated

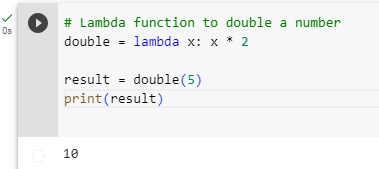
1. **What are lambda functions in Python? How are they different from regular functions? Provide an example where a lambda function can be useful.**

Ans.)

Lambda functions in Python, also known as anonymous functions, they are small, single-expression functions without a name. They are defined using the lambda keyword and can take any number of arguments but can only have one expression. Lambda functions are typically used when you need a simple function that you don't want to define using the def statement.

**Syntax: lambda arguments: expression**

**Ex.)**

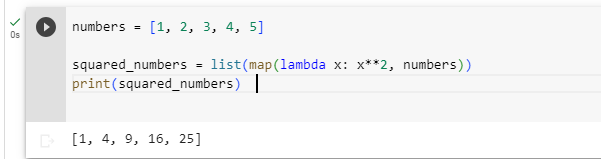


Lambda functions are different from regular functions in a few ways:

1. **No Function Name**: Lambda functions are anonymous functions, meaning they do not have a specific name.
2. **Single Expression:** Lambda functions can only contain a single expression. Unlike regular functions, they cannot contain multiple statements or have complex logic.
3. **Concise Syntax:** The syntax for lambda functions is more compact compared to regular functions. They provide a concise way to define functions without the need for a formal function definition.

Lambda functions can be useful in scenarios **where** you need a small function for immediate use without the need to define and name a regular function. They are commonly used in combination with other functions such as ‘**map()’, ‘filter()**’, and ‘**reduce()**’ to perform simple operations on sequences of data.

Ex.)

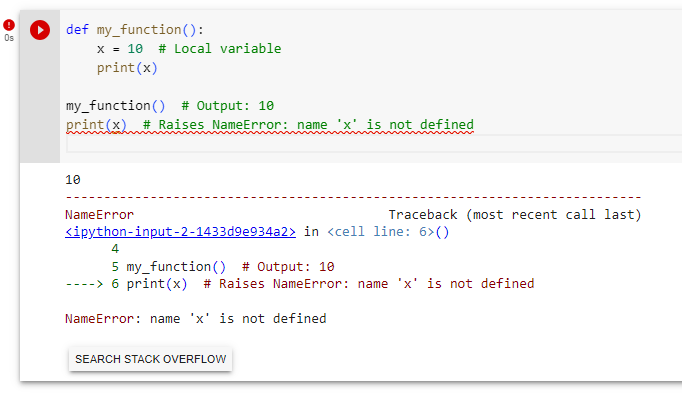


**5. How does the concept of "scope" apply to functions in Python? Explain the difference between local scope and global scope.**

Ans.) In Python, the concept of "scope" refers to the visibility and accessibility of variables within different parts of a program. It determines the portion of the code where a variable can be referenced or modified. Understanding scope is crucial for understanding how variables are accessed and how they interact with each other within functions.

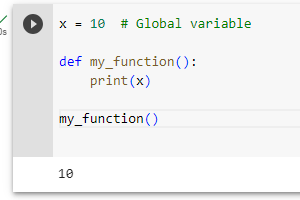
1. **Local Scope:** Local scope refers to the scope within a specific function or block of code. Variables defined inside a function are considered local to that function and can only be accessed from within that function. Local variables are created when the function is called and are destroyed when the function completes execution.

**Ex.**)



1. **Global Scope**: Global scope refers to the scope outside of any function or block of code. Variables defined outside of any function or within a module are considered global variables and can be accessed from anywhere within the program, including inside functions.

**Ex**.)

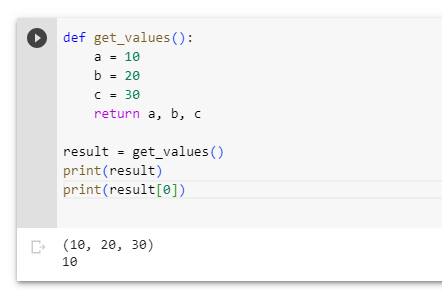


**6. How can you use the "return" statement in a Python function to return multiple values?**

Ans.) In Python, you can use the return statement in a function to return multiple values by returning them as a tuple, list, or any other iterable data structure. Below mentioned are a few ways to achieve this:

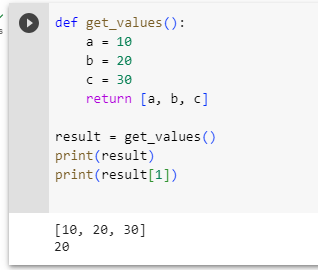
**Returning as a Tuple:** You can return multiple values as a tuple by separating them with commas in the return statement. When the function is called, the returned values can be assigned to multiple variables or accessed using indexing.

Ex.)



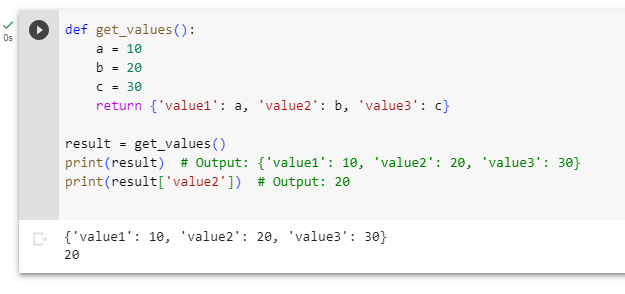
**Returning as a List:** Similar to returning as a tuple, you can also return multiple values as a list. Instead of using commas, you enclose the values in square brackets ‘**[]**’ to create a list.

Ex.)

****

**Returning as a Dictionary or Custom Object:** If you need to return multiple values with specific labels or in a custom format, you can return them as a dictionary or create a custom object and return an instance of that object.

Ex.)

****

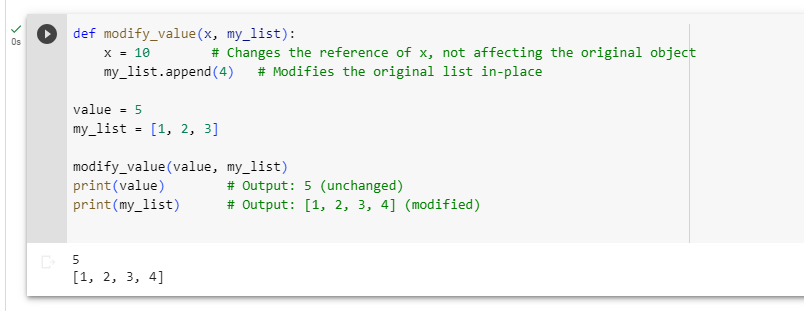
**Q.7 What is the difference between the "pass by value" and "pass by reference" concepts when it comes to function arguments in Python?**

Ans.) In Python, the concepts of "pass by value" and "pass by reference" are not directly applicable. Instead, Python follows a different approach known as "pass by object reference" or "call by object reference".

In Python, variables are references to objects rather than the objects themselves. When a function is called, the arguments passed to the function are actually object references. This means that the function receives a copy of the reference to the object, not the actual object itself.

consider the following:

1. **Immutable Objects:** When an immutable object, such as a number or string, is passed as an argument to a function, a copy of the object's reference is created and passed to the function. Any changes made to the reference within the function do not affect the original object.
2. **Mutable Objects:** When a mutable object, such as a list or dictionary, is passed as an argument to a function, a copy of the object's reference is created and passed to the function. However, the function can modify the contents of the object in-place since the object itself is mutable. These modifications will be reflected outside the function as well.

****

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

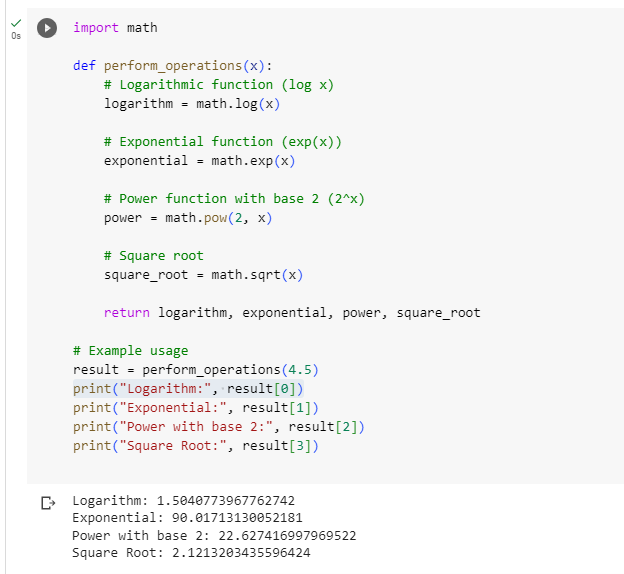
**a. Logarithmic function (log x)**

**b. Exponential function (exp(x))**

**c. Power function with base 2 (2 x )**

**d. Square root**

Ans.)



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

Ans.)

A screenshot of a computer program

Description automatically generated