Sumit Rastogi Assignment 10 June 2023

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

Ans1) In Python, the main difference between a built-in function and a user-defined function lies in their origin and implementation. Let me explain each type and provide an example for better understanding:

a) Built-in function: Built-in functions are pre-defined functions that are available in Python without the need for any additional code or module import. These functions are part of the Python programming language and come with the Python interpreter. Examples of built-in functions include print (), len (), type (), and range ().

b.) User-defined function: User-defined functions are created by the programmer to perform specific tasks or operations. These functions are defined by the user using the def keyword, and they can be reused throughout the program. User-defined functions provide a way to modularize code and improve code readability.

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

Ans2.) In Python, you can pass arguments to a function in multiple ways. The two main methods are using positional arguments and keyword arguments.

a) Positional Arguments: Positional arguments are passed to a function based on their position or order. The function receives the arguments in the same or
der they are provided when calling the function.

b) Keyword Arguments: Keyword arguments are passed to a function using the name of the parameter they are intended for. This allows you to pass arguments to a function in any order, as long as you specify which argument is assigned to which parameter
Keyword arguments are especially useful when a function has a large number of parameters as they make the function call more readable and self-explanatory. They also provide flexibility in case you want to omit some arguments by using default parameter values.

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

Ans3.) The purpose of the return statement in a function is to specify the value that the function should "return" or provide as output when it is called. In other words, it allows the function to pass data back to the code that called it.
A function can have multiple return statements, but only one of them will be executed during the function's execution. When a return statement is encountered, the function immediately exits, and the specified value is returned. This means that any code after the return statement will not be executed.
In this example, the calculate_ discounted _price function takes an original _price and a discount as input. It first checks if the discount is valid (between 0 and 100). If it's invalid, it returns an error message. Otherwise, it calculates the discounted price and checks if it's negative. If the discounted price is negative, it returns 0. Otherwise, it returns the calculated discounted price.

4.What are lambda functions in Python? How are they different from regular functions? Provide an

example where a lambda function can be useful.
Ans4.) Lambda functions, also known as anonymous functions, are a feature in Python that allows you to create small, single-expression functions without giving them a formal name. Unlike regular functions defined using the def keyword, lambda functions are created using the lambda keyword and can be used in situations where a small, throwaway function is needed.
Here, arguments represent the input arguments of the function, and expression is the single expression that the function evaluates and returns.

Lambda functions are often used in situations where a function is required as an argument to another function, such as in functional programming or when working with higher-order functions like map (), filter (), and reduce ().
In Example, a lambda function is used as the key argument to the sorted () function. It specifies that the sorting should be based on the second element of each tuple. The lambda function lambda x: x [1] takes an input tuple x and returns its second element.

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

Ans5) In Python, the concept of "scope" refers to the visibility and accessibility of variables and objects within different parts of a program. It determines where variables can be referenced and manipulated. Python has two main types of scopes: local scope and global scope.
Local Scope: Local scope refers to the part of a program where variables are defined within a function. These variables are known as local variables.
Local variables are created when a function is called and exist only within the function's body.
They are accessible only from within the function and cannot be accessed outside of it.
Once the function finishes executing, the local variables are destroyed, and their values are no longer available. Local variables can have the same name as variables in other scopes, but they are independent and separate entities.
Global Scope: Global scope refers to the part of a program where variables are defined outside of any function, at the top level of the program, or using the global keyword within a function.
Global variables are accessible from any part of the program, including within functions.
Global variables are created when they are first assigned a value or defined, and they exist until the program terminates.
They can be accessed and modified from any part of the program, including within functions.
If a variable is referenced within a function without being explicitly declared as global, Python assumes it to be a local variable within the function's scope, even if a variable with the same name exists in the global scope.

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

Ans6.) In Python, you can use the "return" statement in a function to return multiple values by separating them with commas. There are a few ways to accomplish this:

Using a Tuple: You can return multiple values as a tuple. Here's an example: In this example, the function return _multiple _values () return three values, which are automatically packed into a tuple. The tuple is then assigned to the variable result, which can be used to access individual values.

Using a List: You can also return multiple values as a list. Here's an example: In this case, the function return _multiple _values () return a list containing the values. The list is then assigned to the variable result, which can be used to access individual values.

Using Unpacking: You can assign the returned values to separate variables using unpacking. Here's an example: In this example, the returned values from the function are assigned to separate variables using unpacking. Each variable will hold the corresponding value from the return statement.

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

Ans7.) In Python, the concepts of "pass by value" and "pass by reference" are often misunderstood because the behavior is not strictly one or the other. Python uses a combination of both depending on the type of the object being passed as an argument. Let's explore the differences:

Pass by Value: When a variable is passed by value to a function, a copy of the variable's value is created and passed to the function. Any modifications made to the parameter inside the function do not affect the original variable.

Pass by Reference: When a variable is passed by reference to a function, the function receives a reference to the original variable. Any modifications made to the parameter inside the function directly affect the original variable.

In Python, immutable objects (e.g., numbers, strings, tuples) are passed by value, while mutable objects (e.g., lists, dictionaries) are passed by reference. However, it's important to note that even though mutable objects are passed by reference, the behavior can often resemble pass by value due to the way objects are assigned in Python.

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

Ans8) Certainly! Here's a Python function that can perform the requested operations:

```
import math

def perform_operations(value):
    result = {}
    result['logarithmic'] = math.log(value)
    result['exponential'] = math.exp(value)
    result['power'] = math.pow(2, value)
    result['square_root'] = math.sqrt(value)
    return result
```

You can use this function to perform the operations on an integer or decimal value. It will return a dictionary containing the results of each operation.

Here's an example of how you can use the function:

```
input_value = 3.5
results = perform_operations(input_value)
print(results)
Output:{
  'logarithmic': 1.252762968495368,
  'exponential': 33.11545195869231,
  'power': 11.313708498984761,
  'square_root': 1.8708286933869707
}
```

In this example, the input value is 3.5, and the function calculates the logarithmic value, exponential value, power with base 2, and square root of 3.5. The results are then printed.

A

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

Ans9) Certainly! Here's a Python function that takes a full name as an argument and returns the first name and last name:

```
def get_first_last_name(full_name):
    # Split the full name into individual parts
    name_parts = full_name.split()

    # Extract the first name (first element)
    first_name = name_parts[0]

    # Extract the last name (last element)
    last_name = name_parts[-1]

    # Return the first name and last name as a tuple
    return first_name, last_name
```

You can use this function by calling it and passing a full name as an argument. It will return a tuple containing the first name and last name. Here's an example usage:

```
full_name = "John Doe"
first_name, last_name = get_first_last_name(full_name)
print("First name:", first_name)
print("Last name:", last_name)
```

Output:
```
First name: John
Last name: Doe
```

Note that this function assumes that the full name consists of the first name followed by the last name, with no middle names or additional spaces.