info@codeanddebug.in —— Code and Debug —— codeanddebug.in

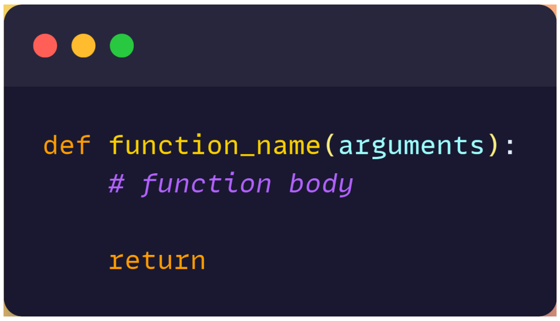
FUNCTIONS IN PYTHON NOTES

A function in Python is a piece of code that operates on its own when you ask it to, giving you a result. Think of it like a machine: when you give it something to work with, it gives you back an answer every time you use it.

There are two types of functions in Python:

- **User-defined Functions:** These are functions created by the user based on their needs. They're tailored to perform particular tasks as required.
- **Built-In Functions:** These are functions already defined within Python. They're handy for performing general tasks within programs without needing to create them from scratch each time.

Python Function Declaration

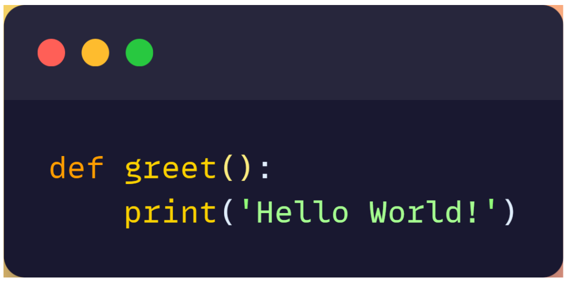


```python
def function_name(arguments):
    # function body

    return
```

Here's what each part does:

- **def:** This keyword is used to define a function.
- **function_name:** You give your function any name you want.
- **arguments:** These are values that can be passed into the function (if needed).
- **return (optional):** If used, it sends a value back from the function.

For example:

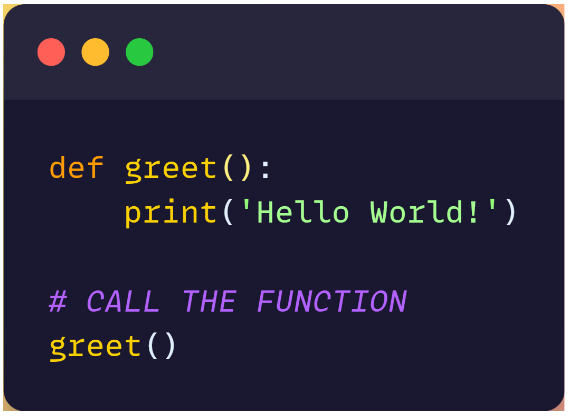


```python
def greet():
    print('Hello World!')
```

In this example, the function greet() is created. It simply prints the text "Hello World!". This specific function doesn't take any arguments and doesn't return any values. Later on, we'll delve into how to use arguments and return statements in functions.

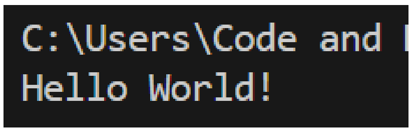
Calling a Function in Python

To use the **greet()** function in Python, you simply need to call it by typing **greet()** in your code.



```python
def greet():
    print('Hello World!')

# CALL THE FUNCTION
greet()
```

Output:



```
C:\Users\Code and
Hello World!
```

When a function is called:

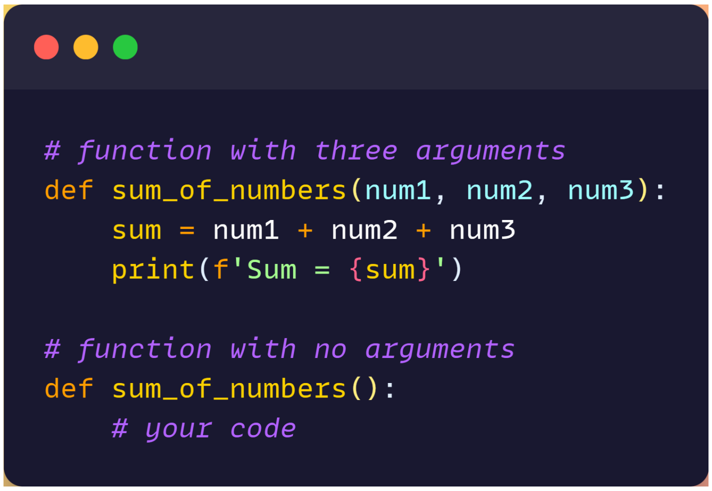
- **Control Flow:** The program's control moves to the function's definition.
- **Execution:** All the code inside the function is executed according to its definition.
- **Return to Caller:** After executing the function, the program's control moves back to the statement immediately after the function call.

Python Function Arguments

Arguments: Information or data that we pass into a function for performing some operations are called arguments.

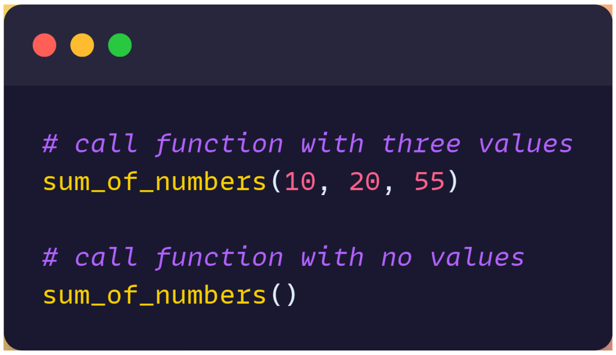
Arguments are placed within the parentheses following the function's name, and they're accompanied by their data type, indicating the kind of data to be provided to the function. Multiple arguments are handled by separating each with commas. The values passed must align with the order of arguments specified in the function. In cases where a function doesn't require any arguments, the space for arguments remains empty within the function's definition.

As discussed earlier, a function can include arguments, which represent accepted values within the function's operation.



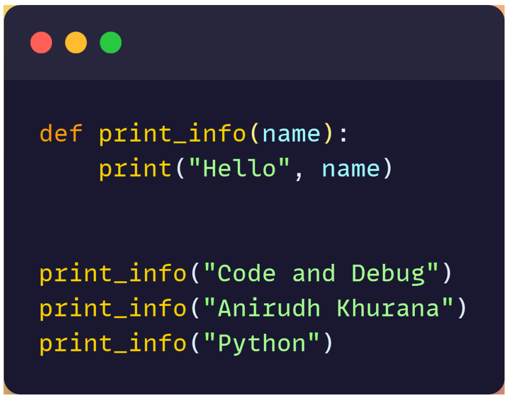
```python
# function with three arguments
def sum_of_numbers(num1, num2, num3):
    sum = num1 + num2 + num3
    print(f'Sum = {sum}')

# function with no arguments
def sum_of_numbers():
    # your code
```

If we create a function with arguments, we need to pass the corresponding values while calling them. For example,



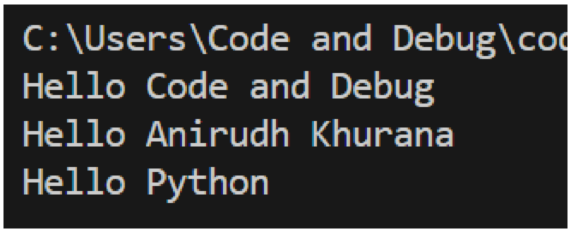
```python
# call function with three values
sum_of_numbers(10, 20, 55)

# call function with no values
sum_of_numbers()
```

Example 1



```python
def print_info(name):
    print("Hello", name)


print_info("Code and Debug")
print_info("Anirudh Khurana")
print_info("Python")
```

Output:



```
C:\Users\Code and Debug\co
Hello Code and Debug
Hello Anirudh Khurana
Hello Python
```

Explanation:

- **Function Definition:**
  - Declares a function named print_info that takes a single argument name.
- **Function Execution:**
  - Calls print_info with different names passed as arguments:
    - "Code and Debug"
    - "Anirudh Khurana"
    - "Python"
- **Output:**
  - Prints "Hello" followed by each provided name when the function is called. For example:
    - "Hello Code and Debug"
    - "Hello Anirudh Khurana"
    - "Hello Python"

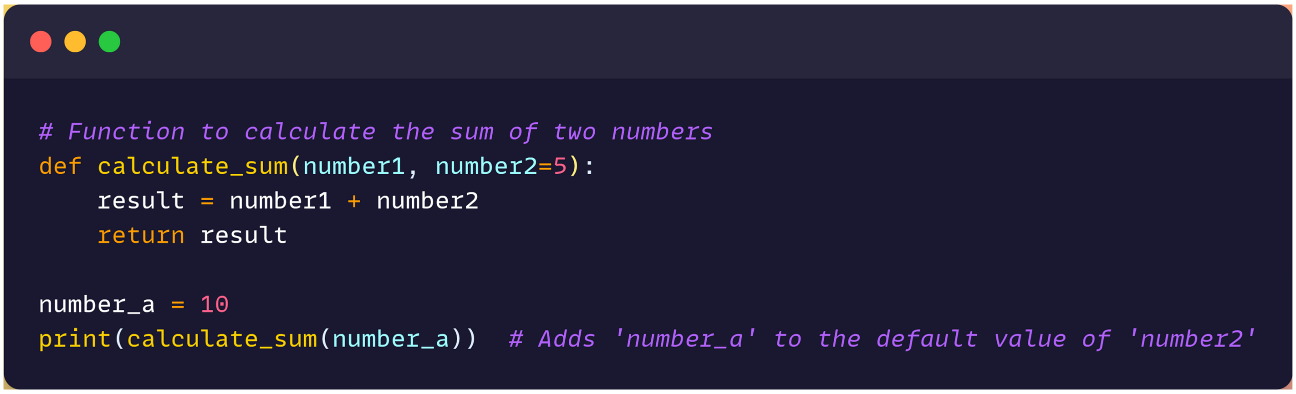
Types of Arguments

Default arguments

Default arguments in a function allow it to use a preset value if no argument is provided when the function is called. Key points about default arguments:

- **Default Argument Behavior:** If a function has a default argument and no value is passed for it, the function uses the default value.

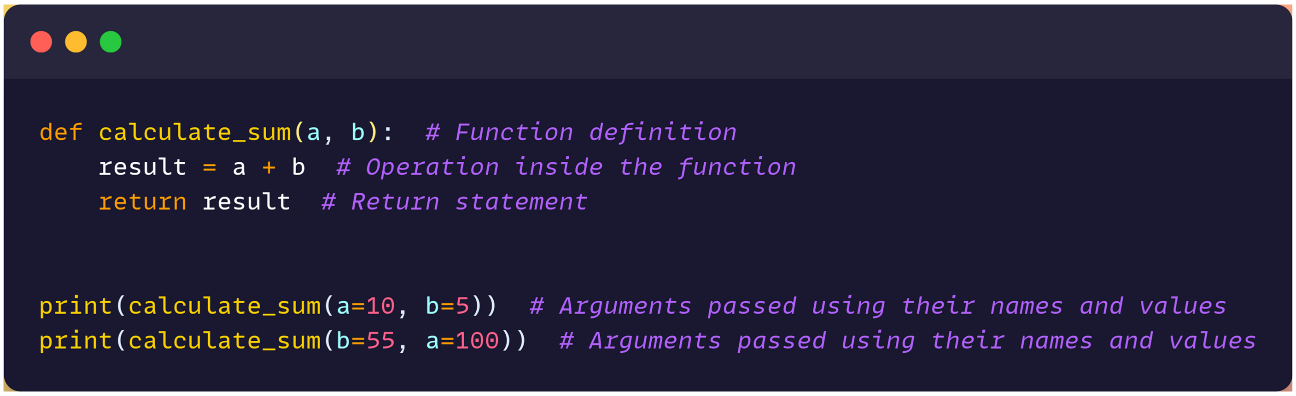
- **Handling Multiple Arguments:**
    - If a function has two arguments, one being default and the other not, providing two values while calling the function assigns each value to its respective argument.
    - However, when only one value is passed, it goes to the non-default argument, and the default argument keeps its default value.
- **Multiple Default Arguments:**
    - When multiple default arguments are used, any subsequent arguments to the right of the default argument(s) must also be default.



```python
# Function to calculate the sum of two numbers
def calculate_sum(number1, number2=5):
    result = number1 + number2
    return result

number_a = 10
print(calculate_sum(number_a))  # Adds 'number_a' to the default value of 'number2'
```

Keyword arguments

Keyword arguments are utilized in functions where values are passed by specifying the argument name followed by its value during the function call. This approach eliminates the need to recall the sequence of arguments, enhancing clarity and ease of use.



```python
def calculate_sum(a, b):  # Function definition
    result = a + b  # Operation inside the function
    return result  # Return statement


print(calculate_sum(a=10, b=5))  # Arguments passed using their names and values
print(calculate_sum(b=55, a=100))  # Arguments passed using their names and values
```

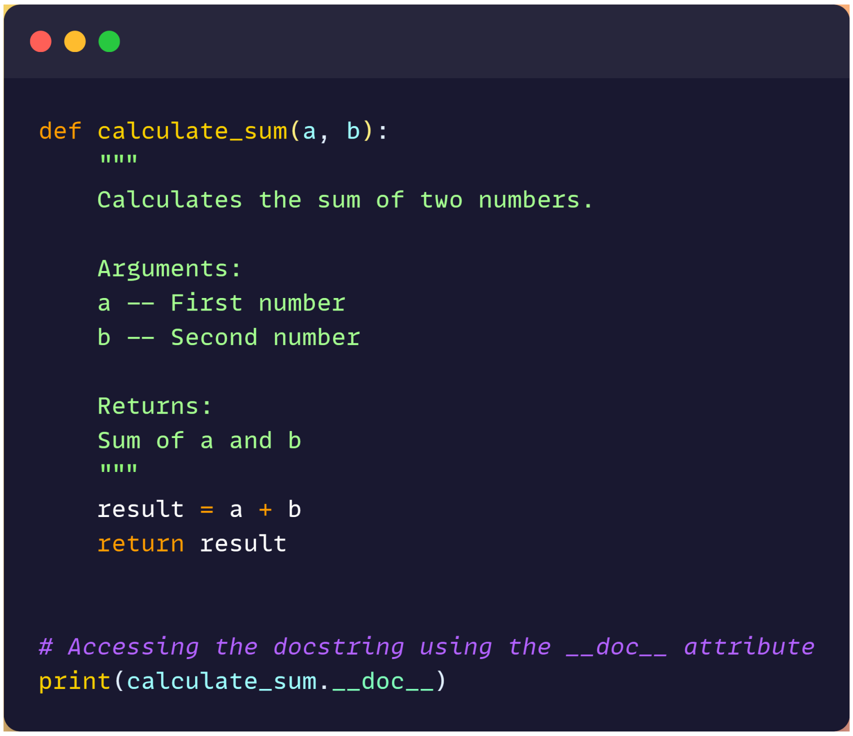
- The function **calculate_sum** takes two arguments, a and b, and computes their sum. It returns the result of this addition.
- The first function call **print(calculate_sum(a=10, b=5))** passes the values 10 and 5 explicitly as **a** and **b** respectively. It calculates the sum of 10 and 5, which is 15, and prints that value.

- The second function call **print(calculate_sum(b=55, a=100))** switches the order of arguments compared to the first call. Here, **a** is assigned the value 100 and **b** is assigned the value 55. It computes the sum of 100 and 55, which is 155, and prints that value.

Both function calls showcase how specifying the argument names (**a=** and **b=**) allows you to pass values in any order while ensuring the correct values are used for the respective arguments within the function.

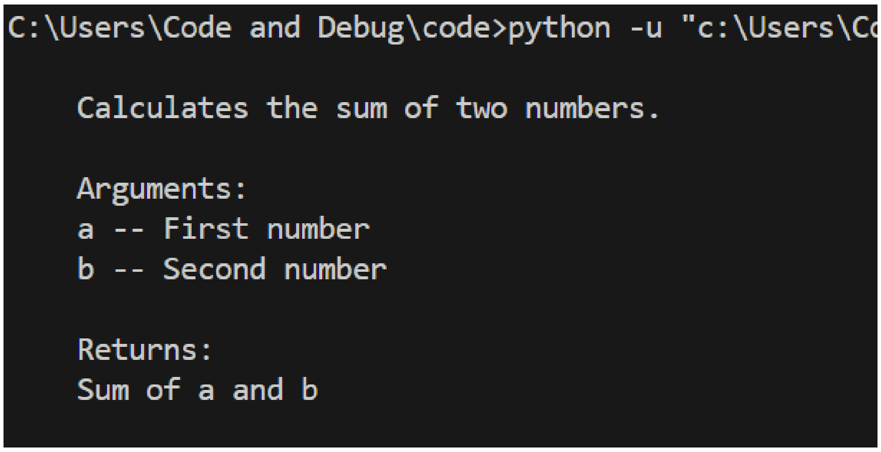
Docstrings

A docstring, the first string right after a function's name, serves as documentation. It explains the function's purpose and functionality to users. While using a docstring is optional, it's considered good practice as it helps others (and yourself) understand the function's intent and usage.



```python
def calculate_sum(a, b):
    """
    Calculates the sum of two numbers.

    Arguments:
    a -- First number
    b -- Second number

    Returns:
    Sum of a and b
    """
    result = a + b
    return result


# Accessing the docstring using the __doc__ attribute
print(calculate_sum.__doc__)
```

Output:



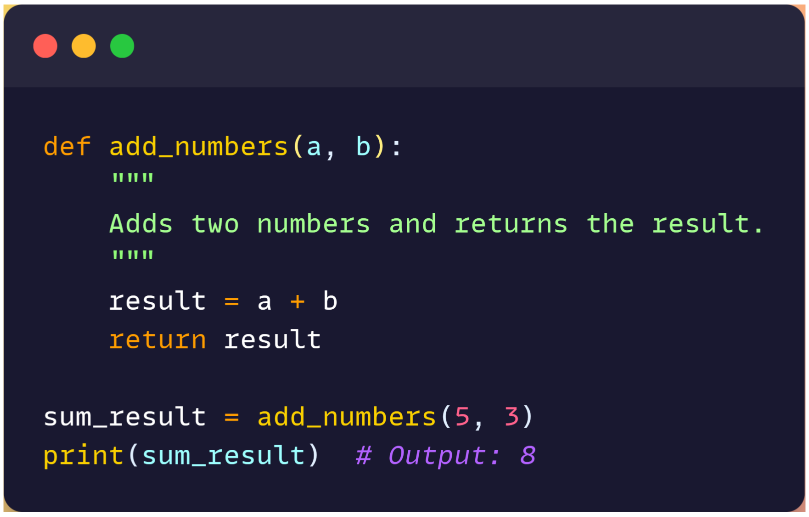
```
C:\Users\Code and Debug\code>python -u "c:\Users\Co

    Calculates the sum of two numbers.

    Arguments:
    a -- First number
    b -- Second number

    Returns:
    Sum of a and b
```

Functions with return statement

In Python, a function has the option to return a value or not. To send a value back to the function call, we employ the return statement.

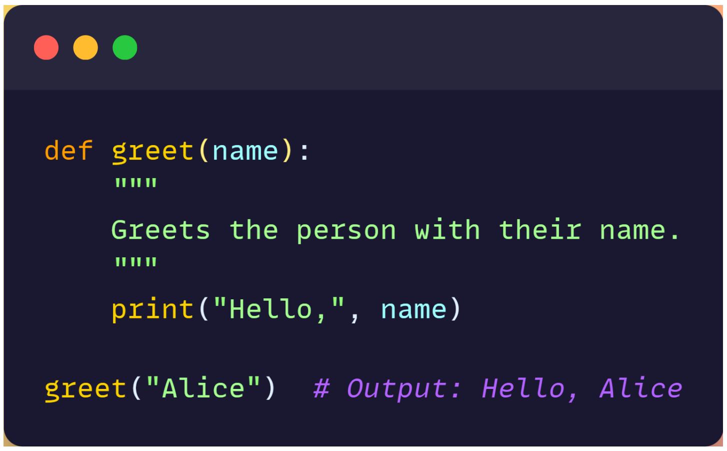
Reminder: The **return** statement marks the conclusion of a function. No code beyond the **return** statement is executed within that function.

Example 1: Function returning a value



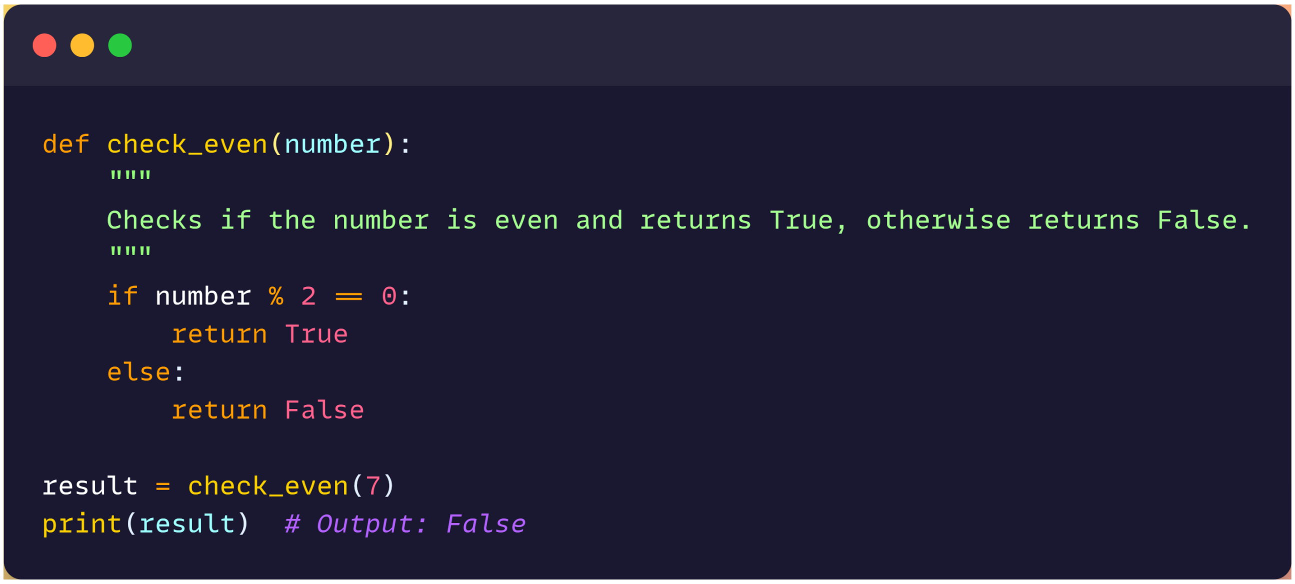
```python
def add_numbers(a, b):
    """
    Adds two numbers and returns the result.
    """
    result = a + b
    return result

sum_result = add_numbers(5, 3)
print(sum_result)  # Output: 8
```

Example 2: Function without a return statement



```python
def greet(name):
    """
    Greets the person with their name.
    """
    print("Hello,", name)

greet("Alice")  # Output: Hello, Alice
```

Example 3: Function with an early return



```python
def check_even(number):
    """
    Checks if the number is even and returns True, otherwise returns False.
    """
    if number % 2 == 0:
        return True
    else:
        return False

result = check_even(7)
print(result)  # Output: False
```

In these examples, the return statement is used to send back values from the functions to the function calls. The first example returns the sum of two numbers, the second example doesn't return anything (just prints), and the third example returns a boolean based on whether the number is even or odd.