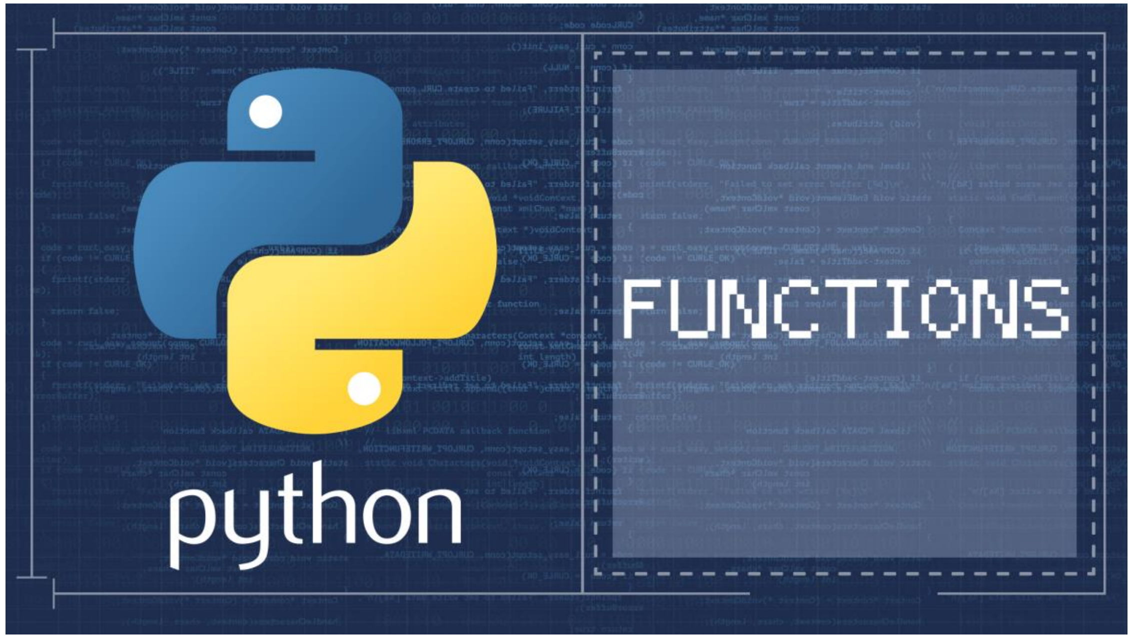
Python

Full stack Skills Bootcamp

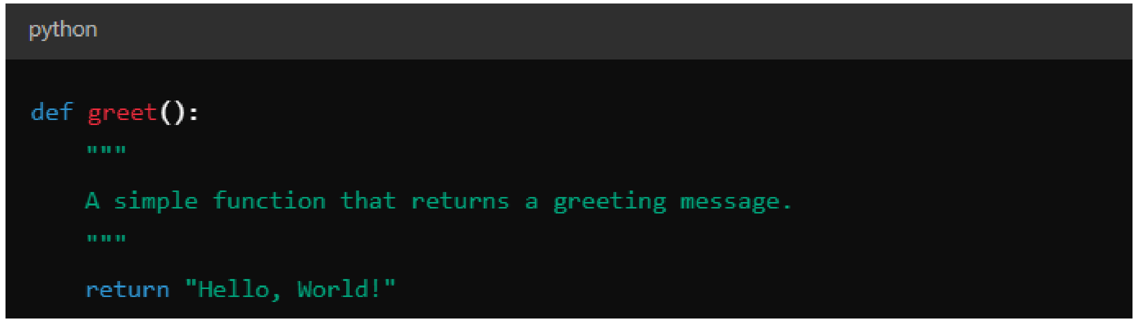
Introducing Python Functions

- What are Functions?
  - A block of reusable code that performs a specific task.
  - Helps in organizing and structuring your code.
- Key Benefits:
  - Code Reusability.
  - Modularity.
  - Improved Readability.



Defining a Basic Function

Creating a Function:



python

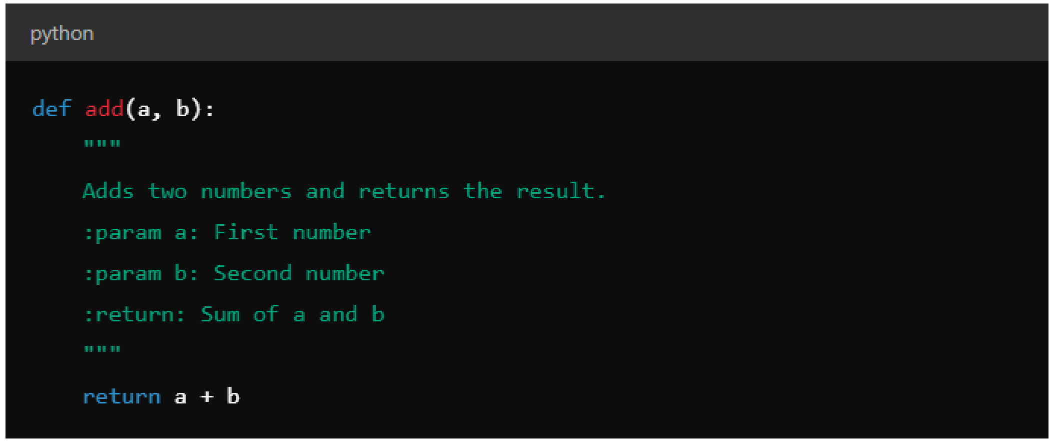
```python
def greet():
    """
    A simple function that returns a greeting message.
    """
    return "Hello, World!"
```

- "def greet()", this defines a function called 'greet' with no parameters.
- return "Hello, World!" , this function returns a greeting message when called.

KEY POINTS:

- No Parameters: The function takes no input arguments.
- Return Statements: The function returns a value, in this case, a string.
- Function Call: The function is executed when called using greet().

Functions with Parameters


python

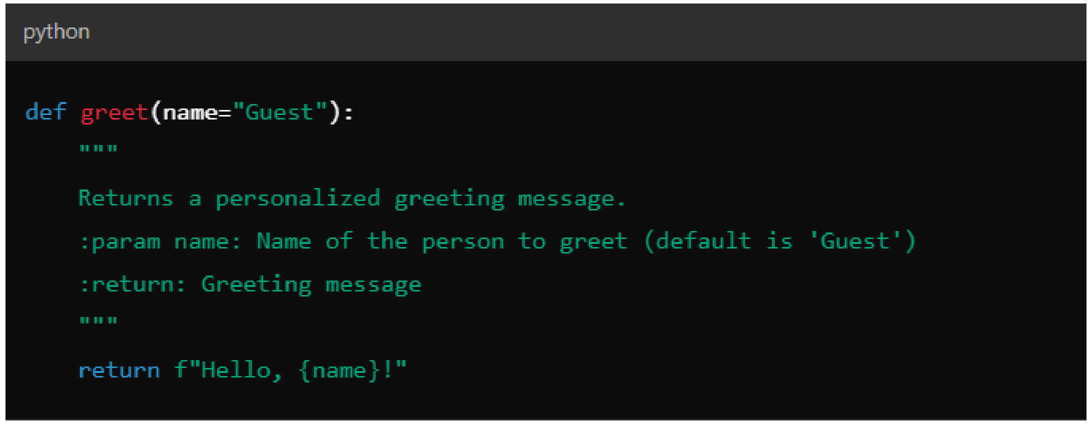
```python
def add(a, b):
    """
    Adds two numbers and returns the result.
    :param a: First number
    :param b: Second number
    :return: Sum of a and b
    """
    return a + b
```

- “def add(a, b)”, the function add accepts two parameters, a and b.
- Parameters: These are inputs to the function. In this case, both, a and b are numbers that will be added together.
- Return: The sum of the two parameters is returned.

KEY POINTS:

- Flexible Input: By passing different values as parameters, the function can compute the sum of any two numbers.
- Reusability: This function can now be used to add any two numbers without rewriting the logic.

Functions with Default Arguments


python

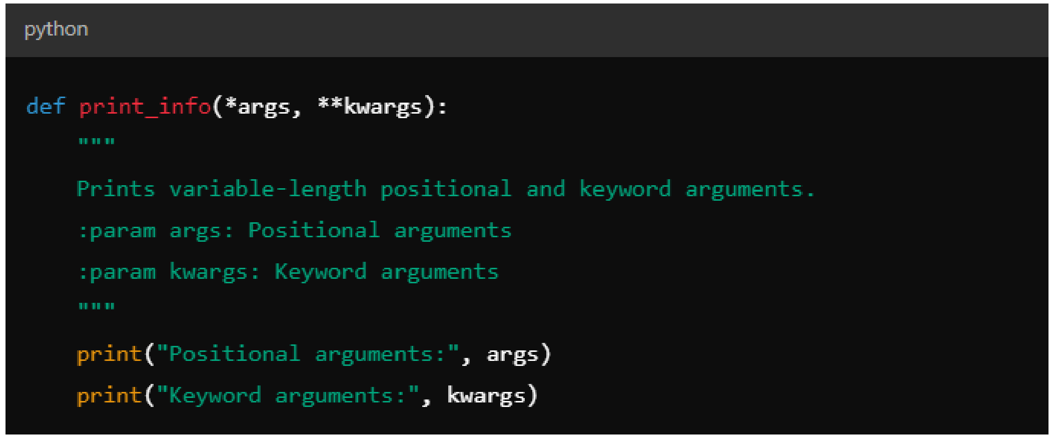
```python
def greet(name="Guest"):
    """
    Returns a personalized greeting message.
    :param name: Name of the person to greet (default is 'Guest')
    :return: Greeting message
    """
    return f"Hello, {name}!"
```

- “def greet(name="Guest")”, The function takes an optional parameter name with a default value of “Guest”.
- Default Argument: If no value is provided for name, the function will use the default value.

KEY POINTS:

- Flexibility: The function can be called with or without the name parameter. When no argument is passed, it defaults to Guest.
- Optional Parameters: Default arguments make it easy to handle cases where input might be optional.

Variable-Length Arguments (*args and **kwargs)



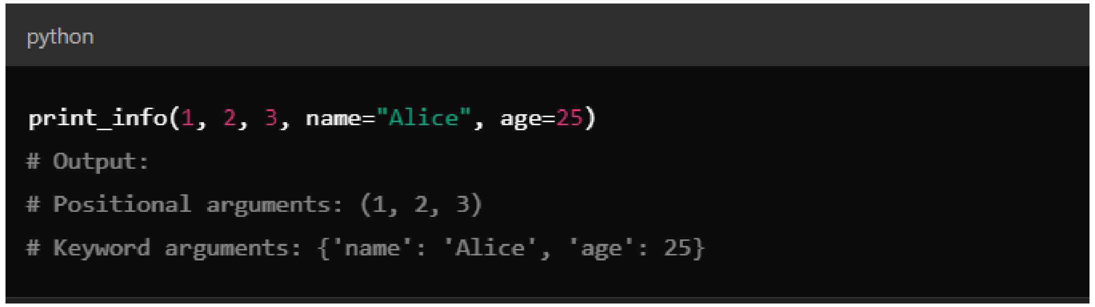
python

```python
def print_info(*args, **kwargs):
    """
    Prints variable-length positional and keyword arguments.
    :param args: Positional arguments
    :param kwargs: Keyword arguments
    """
    print("Positional arguments:", args)
    print("Keyword arguments:", kwargs)
```

- *args, allows you to pass a variable number of positional arguments to a function.
- **kwargs, allows you to pass a variable number of keyword arguments.

KEY POINTS:

- Positional Arguments (*args): Collects all unnamed arguments into a tuple.
- Keyword Arguments (**kwargs): Collects all named arguments into a dictionary.



python

```python
print_info(1, 2, 3, name="Alice", age=25)
# Output:
# Positional arguments: (1, 2, 3)
# Keyword arguments: {'name': 'Alice', 'age': 25}
```

Conclusion

- Functions in Modular Programming:

  - Why Functions Matter: Reusability, Modularity, Readability.
  - Modular Design: Functions help in dividing the entire program into logical modules. Each function performs a specific task, making it easier to understand and debug.
  - Practical Application: When working on large projects, dividing the tasks into multiple functions makes the code more organized and scalable.