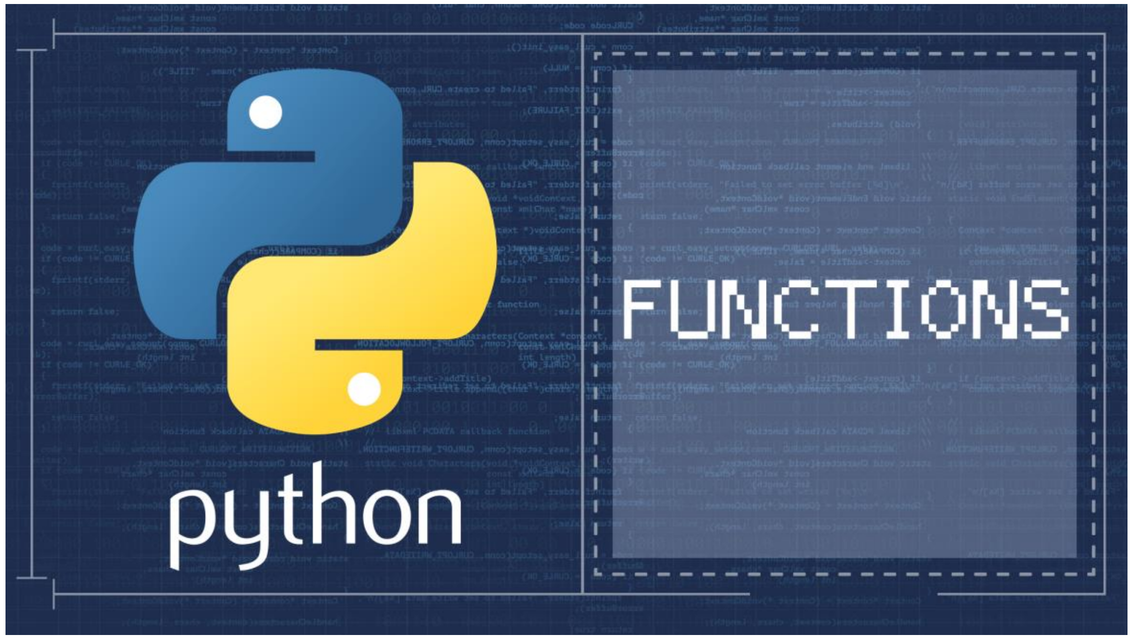
Python

Full stack Skills Bootcamp

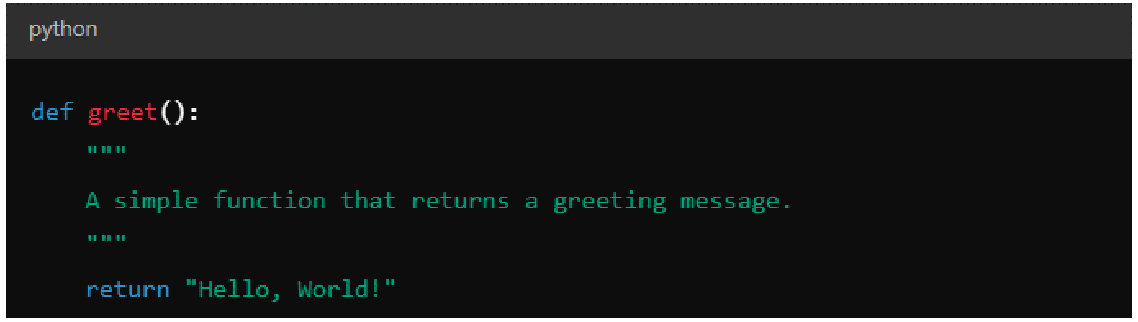
Introducing Python Functions

- 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

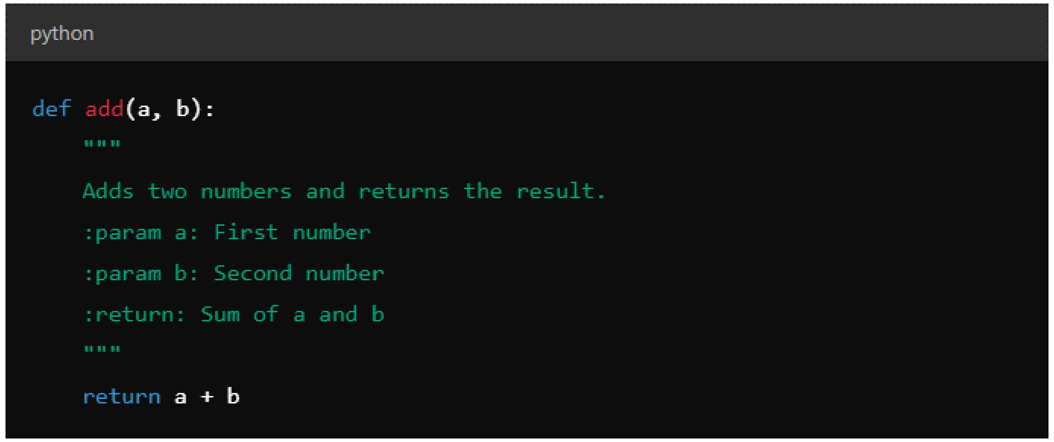
```python
def greet():
    """
    A simple function that returns a greeting message.
    """
    return "Hello, World!"
```

- “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

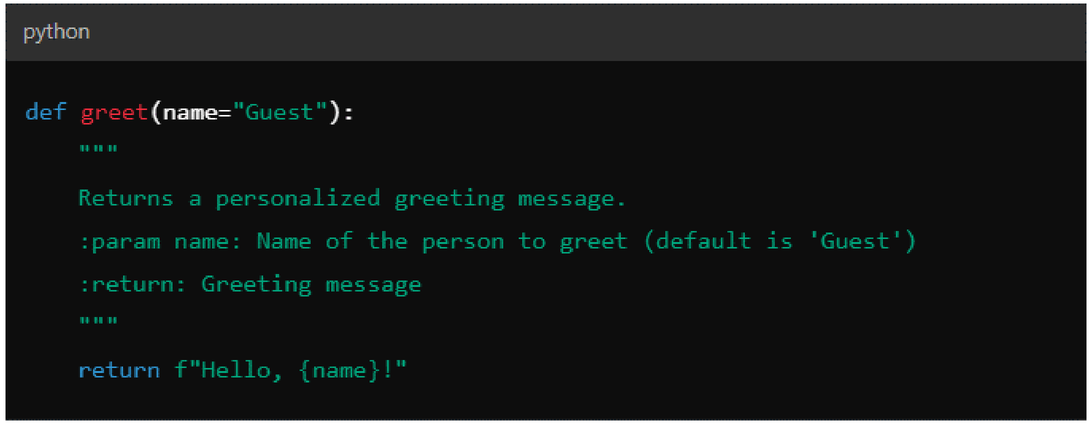
```python
def add(a, b):
    """
    Adds two numbers and returns the result.
    :param a: First number
    :param b: Second number
    :return: Sum of a and b
    """
    return a + b
```

- “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

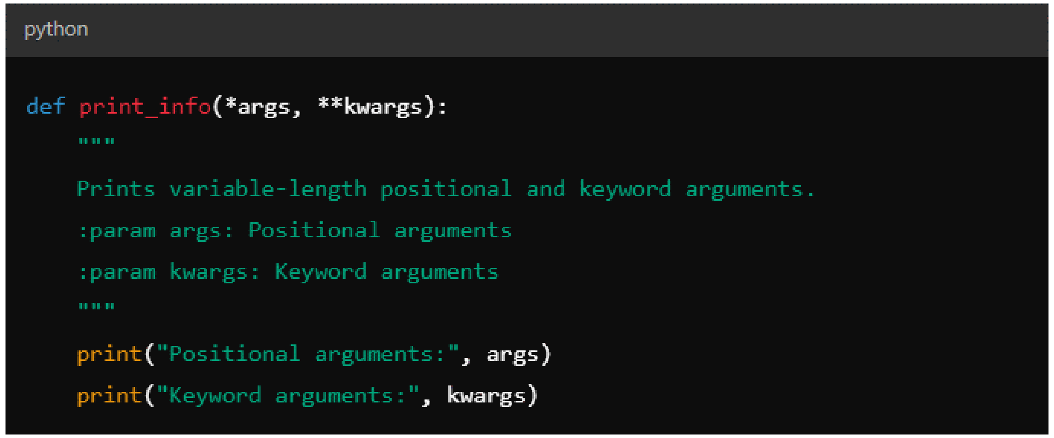
```python
def greet(name="Guest"):
    """
    Returns a personalized greeting message.
    :param name: Name of the person to greet (default is 'Guest')
    :return: Greeting message
    """
    return f"Hello, {name}!"
```

- “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)

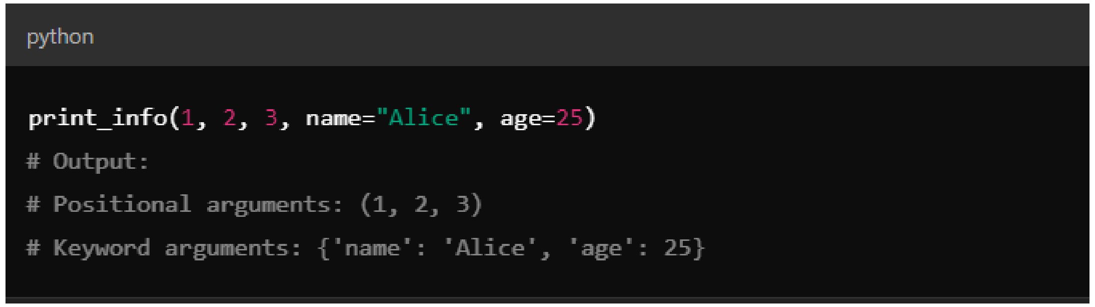


```
python
def print_info(*args, **kwargs):
    """
    Prints variable-length positional and keyword arguments.
    :param args: Positional arguments
    :param kwargs: Keyword arguments
    """
    print("Positional arguments:", args)
    print("Keyword arguments:", kwargs)
```

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

Concluding Functions

- Functions in Modular Programming:

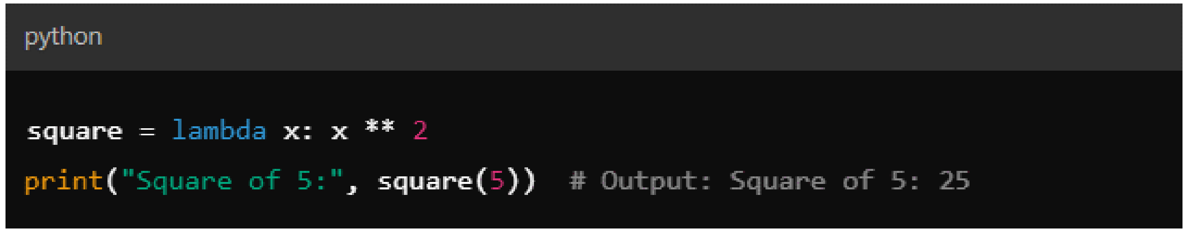
  - Why Functions Matter: Reusability, Modularity, Readability.
  - Modular Design: Functions help in dividing the entire program into logical modules. Each function performs a specific task, making it easier to understand and debug.
  - Practical Application: When working on large projects, dividing the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

Introducing Python Lambdas

- What are Functions?
  - Lambda functions are small, anonymous functions defined using the lambda keyword. They are designed for situations where a simple function is needed for a short duration.
- Characteristics:
  - Number of Arguments: They can take any number of arguments (including none).
  - Single Expression: They can only contain a single expression, which makes them concise and easy to use for simple operations.

Defining a Basic Lambda

Creating a Lambda Function:



python

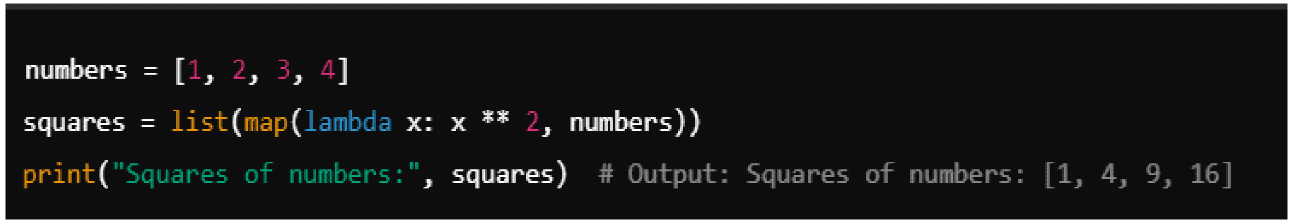
```python
square = lambda x: x ** 2
print("Square of 5:", square(5))  # Output: Square of 5: 25
```

```python
def a(x, y):
    return x + y

b = lambda x, y: x + y
```

- In this example, we define a lambda function that calculates the square of a number x.
- The function is assigned to the variable “square”, which can then be called like a regular function.
- Output: When we call square(5), it computes 5^2 and returns 25.

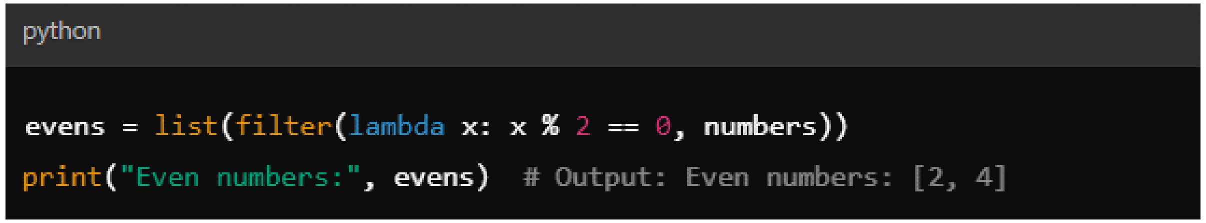
Lambda with map( )



```python
numbers = [1, 2, 3, 4]
squares = list(map(lambda x: x ** 2, numbers))
print("Squares of numbers:", squares)  # Output: Squares of numbers: [1, 4, 9, 16]
```

- The map() function applies the provided lambda function to each element in the list numbers.
- Here, the lambda function takes each number x and returns its square.
- The result of map() is an iterable, which is converted to a list using the list() function.
- Output: The list of squares, [1, 4, 9, 16], is produced by mapping the square function over the original list.

Lambda with filter( )


python

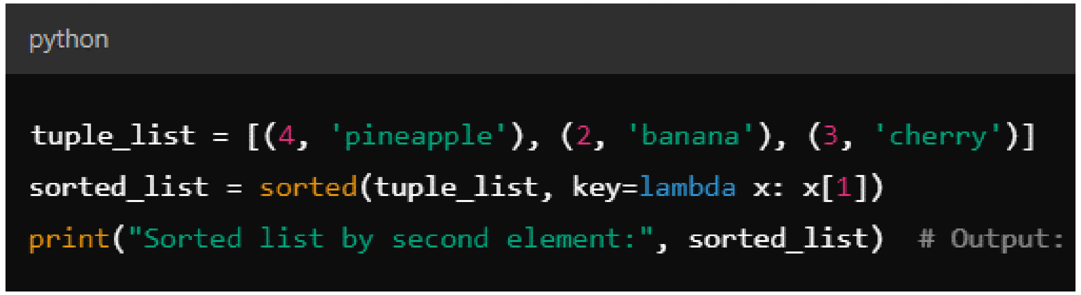
```python
evens = list(filter(lambda x: x % 2 == 0, numbers))
print("Even numbers:", evens)  # Output: Even numbers: [2, 4]
```

- The filter() constructs an iterator from elements of the iterable numbers for which the lambda function returns true.

- In this case, the lambda checks if each number x is even (i.e.., x % 2 == 0).
- The filtered result is converted to a list.
- Output: The even numbers extracted from the list are [2, 4], demonstrating how filtering works with lambda functions.

Step8Up Academy

Lambda with sorted( )



python

```
tuple_list = [(4, 'pineapple'), (2, 'banana'), (3, 'cherry')]
sorted_list = sorted(tuple_list, key=lambda x: x[1])
print("Sorted list by second element:", sorted_list)  # Output:
```

- The sorted() function sorts the list of tuples based on the second element of each tuple.
- The lambda function is used as the sorting key, taking each tuple x and returning x[1], which is a fruit name.
- Output: The sorted list, [(2, 'banana'), (3, 'cherry'), (4, 'pineapple')], reflects the alphabetical order of the second elements.

Concluding Lambdas

- Key Points:

  - Conciseness: Lambda functions allow you to write shorter code for simple functions, reducing boilerplate.

  - Higher-Order Functions: They are commonly used with functions like map(), filter(), and sorted(), enabling functional programming paradigms.

  - Use with Care: While lambda functions are powerful, they can reduce code readability if overused or if the expression is too complex. For maintainability, consider using named functions for more complicated logic.