Functions in Python

Functions are reusable blocks of code that perform a specific task. They help organize code, make it modular, and improve readability and maintainability.

Defining a Function

- Use the def keyword followed by the function name and parentheses ().
- Parameters can be listed inside the parentheses.
- The function body starts with a colon : and is indented.
- Optional: Use a docstring (triple quotes) to describe the function.
- Use return to send back a value from the function.

Syntax:

python

```python
def function_name(parameters):
  """Docstring explaining function purpose"""
  # function body
  return value
```

Example:

python

```python
def greet(name):
  """Display a personalized greeting"""
  print(f"Hello, {name}!")

greet("Alice")
```

Output:text

```text
Hello, Alice!
```

Calling a Function

Invoke the function by its name followed by parentheses, passing required arguments if any:

python

```python
greet("Bob")
```

Function Parameters and Arguments

- Parameters: Variables listed in the function definition.
- Arguments: Values passed to the function during the call.

Example:

python

```python
def add(a, b):
  return a + b

result = add(3, 5)  # Arguments 3 and 5 passed
print(result)     # Output: 8
```

Types of Arguments

- Positional Arguments: Passed in order.
- Keyword Arguments: Passed with parameter names.
- Default Arguments: Parameters with default values.

- Arbitrary Arguments: Using *args (non-keyword) and **kwargs (keyword arguments) to accept variable numbers of arguments.

Example with keyword and default arguments:

python

```python
def describe_pet(pet_name, animal_type='dog'):
  print(f"I have a {animal_type} named {pet_name}.")

describe_pet('Buddy')  # animal_type defaults to dog
describe_pet('Whiskers', animal_type='cat')
```

Return Statement

- Used to return a value from a function.
- Functions without a return statement return None.

Lambda Functions

- Small anonymous functions defined with lambda keyword.
- Typically for simple operations.

python

```python
square = lambda x: x * x
print(square(5))  # Output: 25
```

Python Modules

A Python module is a file containing Python code—such as functions, classes, variables, and runnable code—saved with a .py extension. Modules help organize code into reusable and logically grouped components, making programs easier to understand, maintain, and scale.

Purpose of Modules

- Group related functions, classes, or variables.
- Promote code reuse by importing modules into other programs.
- Help maintain clean, organized codebases.
- Enable modular programming, breaking complex tasks into simpler subtasks.

Creating a Module

To create a module, write Python code in a file and save it with .py extension.

Example: Create mymodule.py

python

```python
def greet(name):
  print(f"Hello, {name}!")

pi = 3.14159
```

Using a Module

Import a module using the import statement.

Example:

python

```python
import mymodule

mymodule.greet("Alice")
print(mymodule.pi)
```

Output:text

```
Hello, Alice!
3.14159
```

You can also import specific functions or variables:

python

```python
from mymodule import greet, pi

greet("Bob")
print(pi)
```

Standard Library Modules

Python comes with many built-in standard modules for various tasks, such as:

- math: Mathematical functions
- os: Operating system interfaces
- sys: System-specific parameters and functions
- random: Random number generation
- datetime: Date and time manipulation