Functions and Modules

1. Defining Functions in Python

Functions help in reusability and modularity in Python.

Syntax:

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

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

Key Points:

- Defined using `def` keyword.
- Function name should be meaningful.
- Use `return` to send a value back.

---

2. Function Arguments & Return Values

Functions can take parameters and return values.

Types of Arguments:

1. **Positional Arguments**

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

```
print(add(5, 3))  # Output: 8
```

2. **Default Arguments**

```
def greet(name="Guest"):
    return f"Hello, {name}!"

print(greet())  # Output: Hello, Guest!
```

3. **Keyword Arguments**

```
def student(name, age):
    print(f"Name: {name}, Age: {age}")

student(age=20, name="Bob")
```

---

3. Lambda Functions in Python

Lambda functions are anonymous, inline functions.

Syntax:

```
square = lambda x: x * x
print(square(4))  # Output: 16
```

Example:

```
numbers = [1, 2, 3, 4]
squared = list(map(lambda x: x**2, numbers))
print(squared)  # Output: [1, 4, 9, 16]
```

---

4. Recursion in Python

A function calling itself to solve a problem.

Example: Factorial using Recursion

```
def factorial(n):
    if n == 1:
        return 1
    return n * factorial(n-1)

print(factorial(5))  # Output: 120
```

Important Notes:

- Must have a base case to avoid infinite recursion.
- Used in algorithms like Fibonacci, Tree Traversals.

---

5. Modules and Pip - Using External Libraries

Importing Modules

Python provides built-in and third-party modules.

Example: Using the `math` module

```
import math

print(math.sqrt(16))  # Output: 4.0
```

Creating Your Own Module

Save this as `mymodule.py` :

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

Import in another file:

```
import mymodule
print(mymodule.greet("Alice"))  # Output: Hello, Alice!
```

Installing External Libraries with `pip`

```
pip install requests
```

Example usage:

```
import requests

response = requests.get("https://api.github.com")
print(response.status_code)
```

6. Function Scope and Lifetime

In Python, variables have **scope** (where they can be accessed) and **lifetime** (how long they exist). Variables are created when a function is called and destroyed when it returns. Understanding scope helps avoid unintended errors and improves code organization.

Types of Scope in Python

1. **Local Scope** (inside a function) – Variables declared inside a function are accessible only within that function.
2. **Global Scope** (accessible everywhere) – Variables declared outside any function can be used throughout the program.

Example:

```
x = 10  # Global variable

def my_func():
    x = 5  # Local variable
    print(x)  # Output: 5

my_func()
print(x)  # Output: 10 (global x remains unchanged)
```

Using the `global` Keyword

To modify a global variable inside a function, use the `global` keyword:

```
x = 10  # Global variable

def modify_global():
    global x
    x = 5  # Modifies the global x

modify_global()
print(x)  # Output: 5
```

This allows functions to change global variables, but excessive use of `global` is discouraged as it can make debugging harder.

7. Docstrings - Writing Function Documentation

Docstrings are used to document functions, classes, and modules. In Python, they are written in triple quotes. They are accessible using the `__doc__` attribute. Here's an example:

```
def add(a, b):
    """Returns the sum of two numbers."""
    return a + b
```

```
print(add.__doc__)  # Output: Returns the sum of two numbers.
```

Here is even proper way to write docstrings:

```
def add(a, b):
    """
    Returns the sum of two numbers.

    Parameters:
    a (int): The first number.
    b (int): The second number.

    Returns:
    int: The sum of the two numbers.
    """
    return a + b
```

Summary

- Functions help in reusability and modularity.
- Functions can take arguments and return values.
- Lambda functions are short, inline functions.
- Recursion is a technique where a function calls itself.
- Modules help in organizing code and using external libraries.
- Scope and lifetime of variables decide their accessibility.
- Docstrings are used to document functions, classes, and modules.