Functions in GoLang, Python, and C++

Building Blocks of Reusable Code

What are Functions?

- Functions are blocks of code that perform a specific task.
- They are reusable and can be called from various parts of a program.
- Functions help in modularizing code for better organization and readability.

Function Declaration in Go

- Functions are declared using the `func` keyword.
- `sayHello` is the function name.

```
func sayHello() {
    fmt.Println("Hello, World!")
}
```

Python

- Python functions are defined using the `def` keyword.

```
def say_hello():
    print("Hello, World!")
```

C++

- C++ functions use `void` to indicate no return value.

```
#include <iostream>
using namespace std;

void sayHello() {
    cout << "Hello, World!" << endl;
}
```

Function Invocation in Go

- To execute a function, simply write its name followed by parentheses.

```
sayHello() // Call the sayHello function
```

Function Invocation in Python

- Python function invocation is similar to Go.

```
say_hello() # Call the say_hello function
```

Function Invocation in C++

- C++ function invocation also uses parentheses.

```
sayHello(); // Call the sayHello function
```

Function Parameters in Go

- Functions can accept parameters, like `name` of type `string`.

```
func greet(name string) {
    fmt.Printf("Hello, %s!\n", name)
}
```

Function Parameters in Python

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

- Python functions define parameters similarly.

Function Parameters in C++

```cpp
#include <iostream>
using namespace std;

void greet(string name) {
    cout << "Hello, " << name << "!" << endl;
}
```

- C++ function parameters use type and name.

Function Arguments in Go

```
greet("Alice") // Pass "Alice" as an argument
```

- Values passed to a function are called arguments.

Function Arguments in Python

```
greet("Alice") # Pass "Alice" as an argument
```

- Python function arguments are straightforward.

Function Arguments in C++

```
greet("Alice"); // Pass "Alice" as an argument
```

- C++ function arguments use parentheses.

Return Values in Go

```
func add(a, b int) int {
    return a + b
}
```

- Functions can return values, specified after the parameter list.

Return Values in Python

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

- Python functions use `return` for returning values.

Return Values in C++

```
int add(int a, int b) {
    return a + b;
}
```

- C++ functions specify return type.

Calling Functions with Return Values in Go

```
result := add(3, 5)
fmt.Println("3 + 5 =", result)
```

- Capture the return value of a function when calling it.

Calling Functions with Return Values in Python

```
result = add(3, 5)
print("3 + 5 =", result)
```

- Python captures return values in variables.

Calling Functions with Return Values in C++

```
int result = add(3, 5);
cout << "3 + 5 = " << result << endl;
```

- C++ captures and uses return values similarly.

Multiple Return Values in Go

```
func swap(a, b int) (int, int) {
    return b, a
}
```

- Functions can return multiple values, separated by commas.

Multiple Return Values in Python

```
def swap(a, b):
    return b, a
```

- Python functions can return multiple values as well.

Multiple Return Values in C++

```cpp
#include <iostream>
using namespace std;

void swap(int &a, int &b) {
    int temp = a;
    a = b;
    b = temp;
}
```

- In C++, multiple values can be returned through reference parameters.

Calling Functions with Multiple Return Values in Go

```
x, y := swap(10, 20)
fmt.Printf("Swapped: x=%d, y=%d\n", x, y)
```

- Capture multiple return values using multiple variables.

Calling Functions with Multiple Return Values in Python

```
x, y = swap(10, 20)
print(f"Swapped: x={x}, y={y}")
```

- Python assigns multiple return values to multiple variables.

Named Return Values in Go

```
func divide(dividend, divisor float64) (result float64, err error) {
    if divisor == 0 {
        err = errors.New("division by zero")
        return
    }
    result = dividend / divisor
    return
}
```

- You can name return values in the function signature.

Named Return Values in Python

```
def divide(dividend, divisor):
    if divisor == 0:
        return None, "division by zero"
    return dividend
```