

Return Types, Arguments, and Capturing Returned Data in C++

- ◆ Function Signature
- ◆ Function Body
- ▣ Return Types
 - Definition:
 - Common Return Types:
 - Example:
- ✿ Function Arguments (Parameters)
 - Definition:
 - Types of Parameter Passing:
- ▣ Capturing Returned Data
 - Definition:
 - Syntax:
 - Example:
 - Why Capture Return Values?
- Best Practices
- Quiz!

In C++, functions are fundamental building blocks that allow you to encapsulate logic and reuse code. Understanding how to define return types, pass arguments, and capture returned values is essential for writing modular and reusable code.

◆ Function Signature

A function in C++ is defined by its **signature**, which includes:

- The return type
- The function name
- The parameter list

```
ReturnType FunctionName(ParameterType1 param1, ParameterType2 param2, ...);
```

```
int add(int a, int b) //this is the signature of the function
{
    return a + b;
}
```

◆ Function Body

The code for a function is in the **function body**. The body consists of the {} and the statements inside the curly braces.

```
int add(int a, int b)
//function body starts here
{
    return a + b;
}
//function body ends here
```

▣ Return Types

Definition:

The **return type** specifies the type of value a function sends back to the caller.

Common Return Types:

- Primitive types: `int`, `double`, `char`, `bool`
- Standard library types: `std::string`, `std::vector`, etc.
- User-defined types: classes or structs
- `void`: indicates no value is returned

Example:

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

✿ Function Arguments (Parameters)

Definition:

- Arguments are values passed to a function when it is called.
- Parameters are the variables that receive those values.

```
// Parameters: a and b
int add(int a, int b) {
    return a + b;
}

//constant arguments (5 and 2) passed to add
int sum = add(5, 2);

//variables passed as arguments
int num1 = 5, num2 = 2;
int result = add(num1, num2);
```

Types of Parameter Passing:

Method	Syntax Example	Behavior
Pass by Value	<code>void f(int x)</code>	Copies the value
Pass by Reference	<code>void f(int& x)</code>	Modifies the original
Pass by Pointer	<code>void f(int* x)</code>	Modifies via address
Const Reference	<code>void f(const std::string&)</code>	Read-only access without copying

▣ Capturing Returned Data

Definition:

When a function returns a value, you can **capture** it by assigning the result to a variable.

Syntax:

```
ReturnType result = FunctionName(arguments);
```

Example:

```
double calculateArea(double width, double height) {
    return width * height;
}

int main() {
    double w = 5.0, h = 3.0;
    double area = calculateArea(w, h); // Capturing the return value
    std::cout << "Area: " << area << std::endl;
    return 0;
}
```

Why Capture Return Values?

- To store results for later use
- To pass results into other functions
- To make decisions based on returned data

● Best Practices

- Always match the **return type** of the function with the type of the variable capturing the result.
- Use `auto` for type inference when the return type is complex:

```
auto result = someFunction();
```

- Avoid ignoring return values unless the function is `void` or the result is truly unnecessary.

● Quiz!

Here's a short quiz on the topic: [quiz](#)

Footer Separator

▷ Markdown Viewer

How to view the markdown files in a browser...

- [Markdown Viewer](#)

● Lecture Practices

Here are the lecture Practices...

- [Day 1](#)
- [Day 2](#)
- [Day 3](#)

✿ Lecture Quizzes

Here are the lecture quizzes...

- [Day 1](#)
- [Day 2](#)
- [Day 3](#)

● Weekly Topics

Here are the topics for the week...

- [Calling Methods](#)
- [Calling Methods 2](#)
- [Creating Methods](#)
- [Iterators](#)
- [Vectors](#)
- [References](#)
- [Const](#)
- [Erasing in a Loop](#)
- [Default Parameters](#)