1. Introduction to Functions

1.1 What are Functions?

Functions are self-contained blocks of code that perform specific tasks. They are fundamental building blocks that help organize code, promote reusability, and make programs more modular and maintainable.

Benefits of Functions:

- **Code Reusability:** Write once, use multiple times
- **Modularity:** Break complex problems into smaller, manageable pieces
- **Maintainability:** Easier to debug, test, and modify
- **Readability:** Makes code more organized and understandable
- **Abstraction:** Hide implementation details from the caller

1.2 Why Use Functions? (Comparison with and without)

To understand the practical benefits of functions, let's consider a simple task: calculating the area of three different rectangles.

Without Functions:

```
#include <stdio.h>

int main() {
    // Rectangle 1
    int length1 = 10;
    int width1 = 5;
    int area1 = length1 * width1;
    printf("Area of Rectangle 1: %d\n", area1);

    // Rectangle 2
    int length2 = 12;
    int width2 = 8;
    int area2 = length2 * width2;
    printf("Area of Rectangle 2: %d\n", area2);
```

```
    // Rectangle 3
    int length3 = 7;
    int width3 = 3;
    int area3 = length3 * width3;
    printf("Area of Rectangle 3: %d\n", area3);

    return 0;
}
```

In this example, the logic for calculating the area is repeated three times. If we needed to change how the area is calculated (e.g., add a margin), we would have to modify each instance, which is error-prone and inefficient.

With Functions:

```
#include <stdio.h>

// Function to calculate rectangle area
int calculate_rectangle_area(int length, int width) {
    return length * width;
}

int main() {
    // Rectangle 1
    int area1 = calculate_rectangle_area(10, 5);
    printf("Area of Rectangle 1: %d\n", area1);

    // Rectangle 2
    int area2 = calculate_rectangle_area(12, 8);
    printf("Area of Rectangle 2: %d\n", area2);

    // Rectangle 3
    int area3 = calculate_rectangle_area(7, 3);
    printf("Area of Rectangle 3: %d\n", area3);

    return 0;
}
```

By using a function `calculate_rectangle_area`, we write the area calculation logic only once. This demonstrates:

- **Reusability:** The `calculate_rectangle_area` function can be used multiple times with different inputs.
- **Readability:** The `main` function becomes cleaner and easier to understand, as the details of area calculation are encapsulated within the function.
- **Maintainability:** If the area calculation logic needs to change, we only need to modify it in one place (inside the function definition), and all calls to that function will automatically use the updated logic.

1.3 Python vs C Functions Comparison

Aspect	Python	C
Declaration	Not required	Function prototype usually required (unless defined before use)
Definition	`def function_name():`	`return_type function_name() { }`
Return Type	Dynamic (any type)	Must be explicitly declared
Parameters	Dynamic typing	Static typing required
Call Before Definition	Allowed	Requires prototype
Multiple Return Values	`return a, b`	Use pointers or structures

Python Example:

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

result = add_numbers(5, 3)
print(result)  # Output: 8
```

C Equivalent:

```
#include <stdio.h>

// Function declaration (prototype)
int add_numbers(int a, int b);

int main() {
    int result = add_numbers(5, 3);
    printf("%d\n", result);  // Output: 8
    return 0;
}

// Function definition
```

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

Revision #2
Created 2025-09-06 10:13:20 UTC by DS
Updated 2025-09-06 10:38:50 UTC by DS