Defining, Calling & Parameterizing Functions

AiCE Software Engineering Foundations — Python Programming (Intermediate)

In the beginner section, your programs ran from top to bottom, one instruction after another. Functions change that entirely. A function is a named, reusable block of code that you define once and call as many times as you need — the foundation of organized, professional Python programming. This lesson opens that foundation: defining, calling, and customizing functions with parameters.

1. What Is a Function?

Consider how a market trader in Kano organizes their day. Every morning they follow the same sequence — open the stall, arrange the goods, check stock levels, greet the first customer. They don't invent a new process each day; they follow a routine they've defined and refined, repeating it consistently.

A function works the same way. You define a set of instructions once, give those instructions a name, and call that name whenever you want them to run. The code executes on demand — as many times as needed — without you rewriting it.

Without functions, every time you needed to calculate a student's grade, check a payment balance, or validate a phone number, you'd write the same logic again from scratch. Functions eliminate that repetition entirely, and make code easier to read, test, and fix — when something breaks, you go to one function and fix it there, instead of hunting through repeated blocks scattered across the program.

> **Name Your Logic**
> *A function is the programmer's way of saying: this matters enough to have a name. Organize your thinking. Write it once and use it everywhere.*

2. Defining a Function

In Python, you define a function using the def keyword — short for 'define':

```
def function_name():
    # code that runs when the function is called
    # (indented 4 spaces)
```

The def keyword tells Python you're about to define a function. The function name follows the same naming rules as variables — lowercase, underscores for spaces, descriptive. The empty brackets () indicate the function takes no input for now. The colon ends the definition line, and everything indented beneath it is the function body.

```
def greet():
    print("Hello!")
    print("Welcome to AiCE Python Intermediate.")

greet()
```

This produces:

```
Hello!
Welcome to AiCE Python Intermediate.
```

greet is defined with def, its body holds two print statements, and greet() at the bottom calls the function — which is what actually runs those two lines.

Defining Does Not Run — Calling Does

This distinction catches many beginners off guard. Defining a function does not execute it — the code inside sits dormant until the function is called.

```
def greet():
    print("Hello!")
    print("Welcome to AiCE Python Intermediate.")

# No call — nothing happens
```

> **Nothing Prints**
> *Removing the greet() call means the function exists in memory but is never asked to run — so nothing prints. Add greet() back, and the output reappears. A function runs only when called.*

Calling a Function Multiple Times

One of the most important benefits of functions is that a single definition can be called as many times as needed:

```
def greet():
    print("Hello!")
    print("Welcome to AiCE Python Intermediate.")

greet()
greet()
greet()
```

Three calls, three executions, one definition — that's the power of functions: write the logic once, deploy it anywhere.

3. Parameters — Making Functions Flexible

A function that always does the same thing is useful. A function that can be customized each time it's called is far more powerful. Parameters make this possible.

A parameter is a variable declared inside the function's brackets. When you call the function, you provide a value for that parameter, and the function uses it in its body:

```
def greet(name):
    print(f"Hello, {name}!")
    print(f"Welcome to AiCE, {name}. Your journey continues
here.")

greet("Adesina")
greet("Ngozi")
greet("Kwame")
```

The parameter name inside greet(name) acts as a placeholder. Calling greet("Adesina") assigns "Adesina" to name for that call; calling greet("Ngozi") assigns "Ngozi" instead. Each call is independent.

> **Parameters vs. Arguments**
> *These two words are often used interchangeably, but they mean different things. A parameter is the variable name in the function definition — name in def greet(name). An argument is the actual value passed in when calling — "Adesina" in greet("Adesina"). Parameter is the placeholder; argument is the value.*

Multiple Parameters

A function can accept more than one parameter, separated by commas — both in the definition and the call. Values are matched in order: the first argument goes to the first parameter, the second to the second, and so on:

```
def greet(name, course):
    print(f"Hello, {name}!")
    print(f"You are enrolled in {course}.")
    print(f"We are glad to have you, {name}.")

greet("Adesina", "Python Programming")
greet("Chiamaka", "Web Development")
greet("Diallo", "Data Engineering")
```

Each call receives two arguments — one for name, one for course — matched positionally, left to right.

Key Takeaways

A function, defined with def and a name, sits dormant until it's called with parentheses — and once defined, it can be called as many times as needed from anywhere in the program. Parameters are the placeholders declared in the definition; arguments are the actual values supplied at each call, matched positionally. The next lesson introduces the return statement, which lets a function hand a computed value back to the rest of the program, and default parameter values, which give a function a fallback when a caller doesn't supply every argument.