Return Values & Default Parameters

AiCE Software Engineering Foundations — Python Programming (Intermediate)

Functions that print directly are useful for displaying information, but many functions need to compute something and hand the result back to the rest of the program. This lesson covers the return statement that makes that possible, and default parameter values, which let a function work sensibly even when a caller doesn't provide every argument.

1. The return Statement

return does two things: it sends a value back from the function to wherever it was called from, and it immediately stops the function — any code written after return inside the function body never runs.

```
def add(num1, num2):
    result = num1 + num2
    return result

answer = add(10, 5)
print(answer)

# You can also use the returned value directly
print(add(100, 250))
```

This produces:

```
15
350
```

When add(10, 5) is called, the function runs, calculates result as 15, and returns it — stored in answer and then printed. The second call passes the returned value directly to print() without storing it first; both approaches are valid.

return Stops the Function Immediately

The moment Python hits a return statement, the function ends — regardless of what code follows it:

```
def check_payment(amount):
    if amount <= 0:
        return "Invalid amount. Payment cancelled."
    # This line only runs if amount > 0
    return f"Payment of NGN {amount:,} processed
successfully."

print(check_payment(-500))
print(check_payment(0))
print(check_payment(15000))
```

This produces:

```
Invalid amount. Payment cancelled.
Invalid amount. Payment cancelled.
Payment of NGN 15,000 processed successfully.
```

For the first two calls, the condition is met and the first return fires immediately — the function ends there, and the second return is never reached. For the third call, the condition is False, so Python skips the first return and executes the second.

> **The Guard Clause Pattern**
> *This early-return pattern — checking for invalid or edge cases at the top of a function and returning immediately — is a professional technique called a guard clause. It keeps the main logic of a function clean and unindented, rather than nesting everything inside an else block.*

A Practical return Example — Grade Calculator

A real function that computes a student's grade and returns it, rather than printing it directly:

```
def get_grade(score):
    if score >= 70:
        return "A"
    elif score >= 60:
        return "B"
    elif score >= 50:
        return "C"
    elif score >= 40:
        return "D"
    else:
        return "F"

# The returned value can be stored and used
grade = get_grade(85)
print(f"Score 85 -> Grade: {grade}")

# Or passed directly to another function
print(f"Score 45 -> Grade: {get_grade(45)}")
```

get_grade doesn't print anything itself — it only computes and returns. This separation of concerns, keeping computation and display as separate responsibilities, is excellent function design.

2. Default Parameter Values

When a function expects a parameter and you call it without providing that argument, Python raises a TypeError. Default parameter values solve this by providing a fallback used when no argument is supplied.

To demonstrate the problem first:

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

greet("Adesina", "Good morning")  # Works fine
greet("Ngozi")                    # TypeError — message
missing
```

Running this gives:

```
Good morning, Adesina!
TypeError: greet() missing 1 required positional argument:
'message'
```

Now add a default value for message:

```
def greet(name, message="Good morning"):
    print(f"{message}, {name}!")

greet("Adesina")                        # Uses default: Good
morning
greet("Ngozi", "Good evening")           # Overrides default
greet("Kwame", "Welcome back")           # Overrides default
```

This produces:

```
Good morning, Adesina!
Good evening, Ngozi!
Welcome back, Kwame!
```

When no message argument is provided, Python uses "Good morning" automatically; when one is provided, it replaces the default. Default parameters let callers supply only what's necessary and rely on sensible defaults for everything else.

> **Default Parameters Must Come Last**
> *In a function definition, parameters with default values must always come after parameters without defaults. def greet(name, message="Good morning") is valid; def greet(message="Good morning", name) is a SyntaxError. Python enforces this so it can match positional arguments unambiguously.*

Key Takeaways

return sends a value back to the caller and ends the function immediately — the guard clause pattern uses this to handle edge cases at the top of a function, keeping the main

logic clean. Default parameter values, written as param=value in the function definition, give a function a sensible fallback when a caller omits an argument, and must always come after any parameters without defaults. The next lesson returns to local and global variables with more depth, shows functions calling other functions, and applies functions directly to lists and dictionaries.