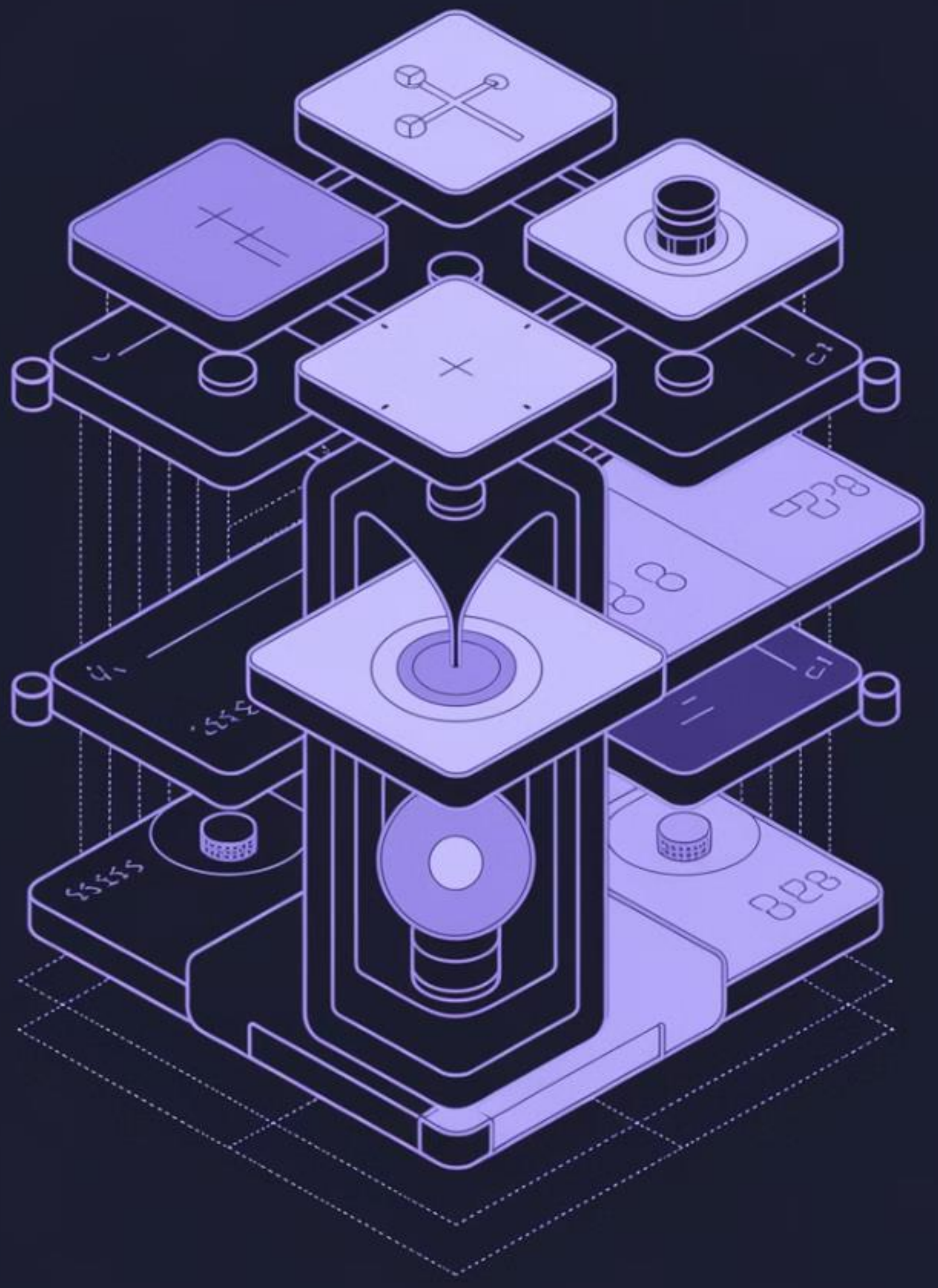
Using Functions in Python



Introduction

Definition

A function is a reusable block of code that performs a specific task.

Benefits

- Modularity: Breaks a large problem into smaller, manageable parts. parts.
- Reusability: Write once, reuse anywhere.
- Maintainability: Easier to debug, test, and update.
- Readability: Improves the clarity of your code.

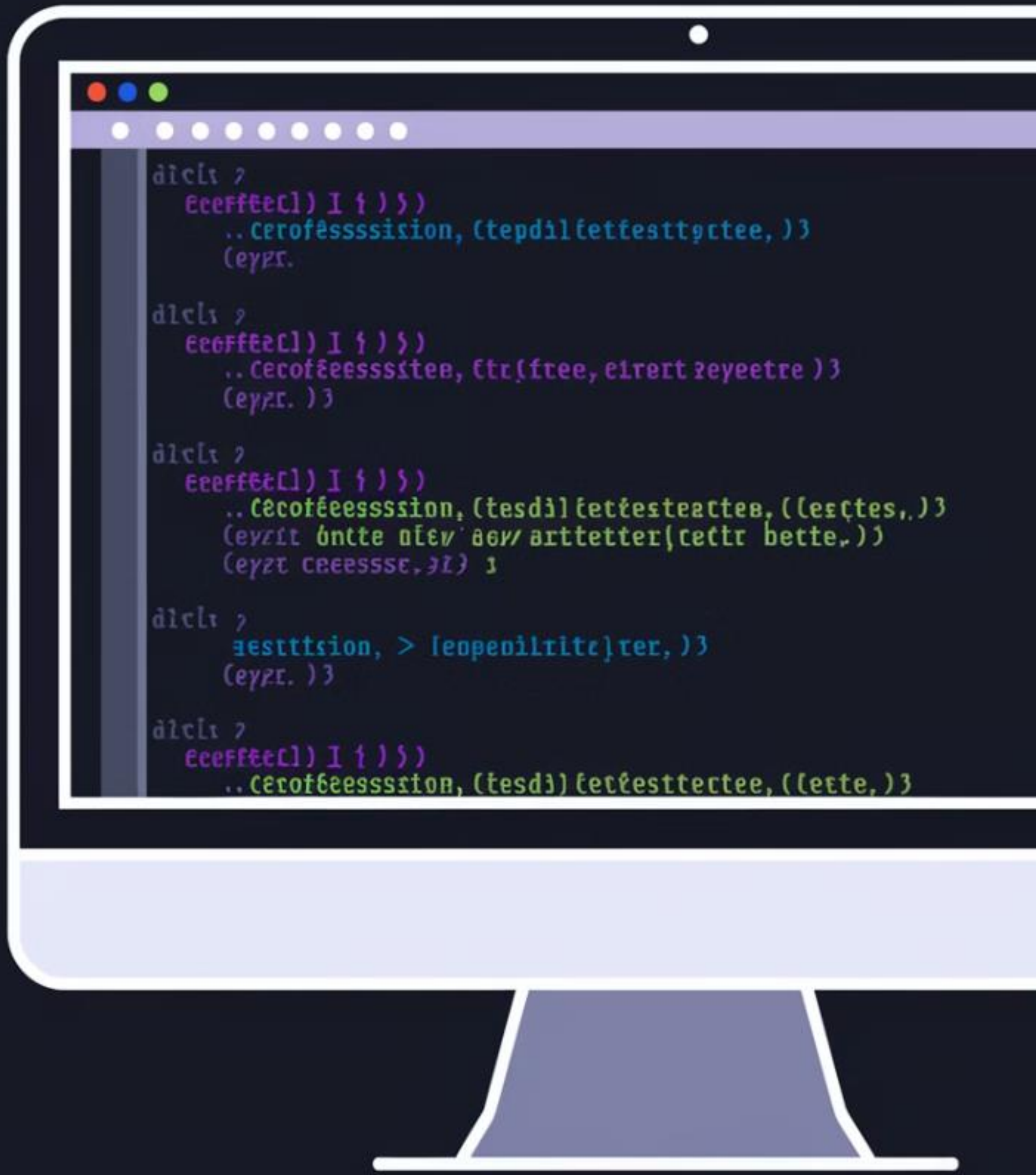
Defining a Function in Python

Syntax

```python
def function_name(parameters):
    """Optional docstring explaining the function."""
    # Function body
    return result  # Optional return statement

```

Example

```python
def greet(name):
    """Return a greeting message for the given name."""
    return f"Hello, {name}!"

print(greet("Alice"))  # Output: Hello, Alice!

```



Parameters and Arguments

Types of Parameters

- Positional Parameters: Must be provided in the correct order.
- Keyword Parameters: Specified by parameter name.
- Default Parameters: Provide a default value if none is given.

Variable-Length Arguments: *args

For a variable number of positional arguments.

```python
def sum_all(*numbers):
    """Return the sum of all numbers provided."""
    return sum(numbers)

print(sum_all(1, 2, 3))  # Output: 6

```

Variable-Length Arguments: **kwargs

For a variable number of keyword arguments.

```python
def display_info(**info):
    """Display key-value pairs from keyword arguments."""
    for key, value in info.items():
        print(f"{key}: {value}")

display_info(name="Bob", age=25)
# Output:
# name: Bob
# age: 25

```

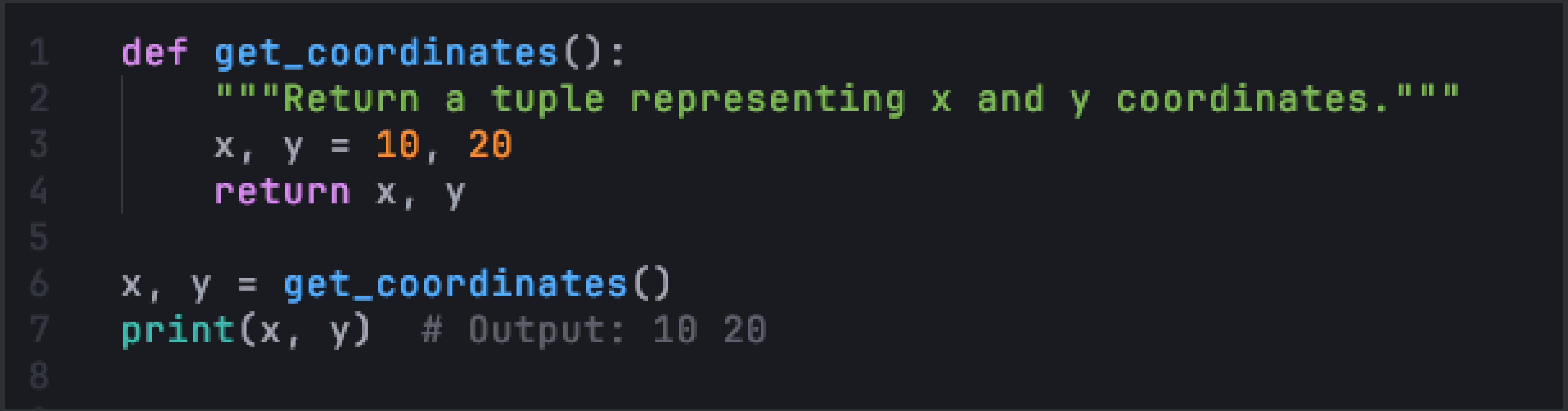
Return Values

Purpose

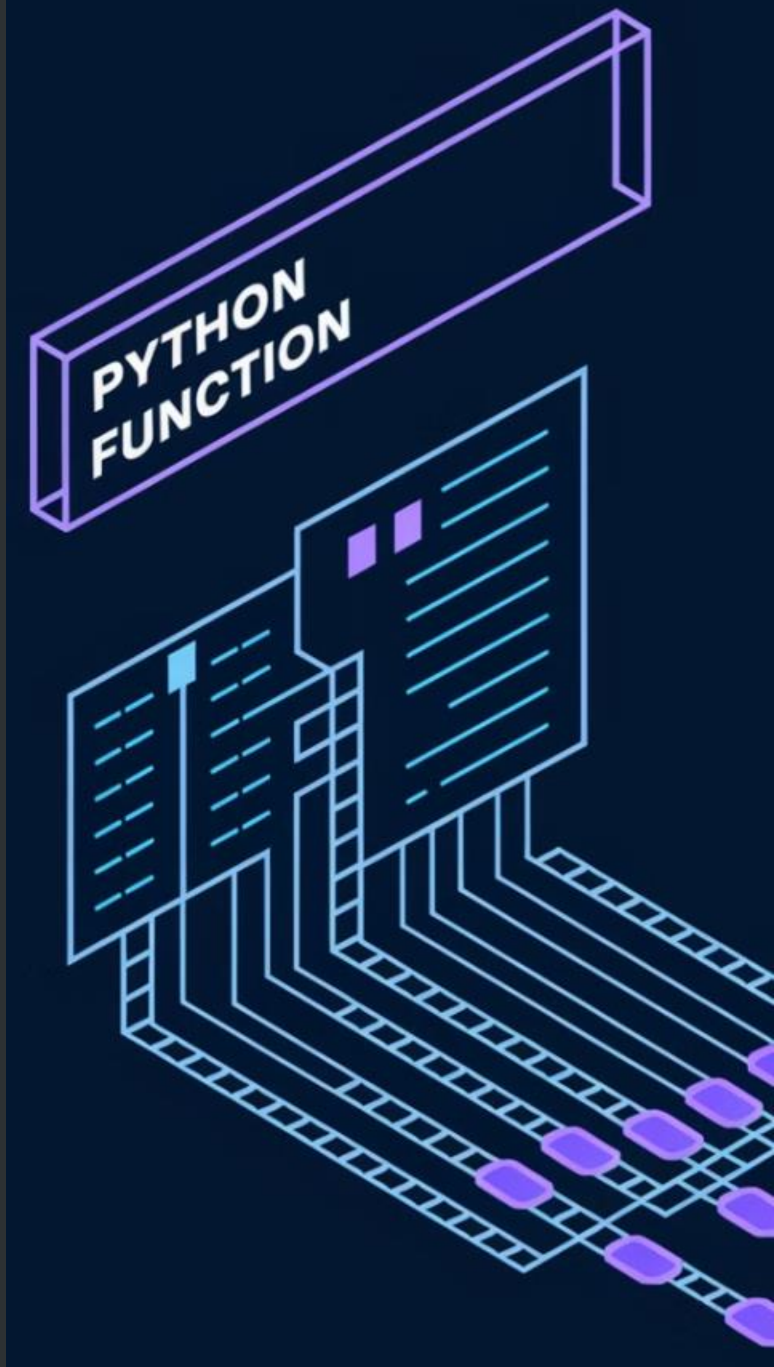
The return statement sends a result back to the caller.

Multiple Return Values

Can return a tuple.



```python
def get_coordinates():
    """Return a tuple representing x and y coordinates."""
    x, y = 10, 20
    return x, y

x, y = get_coordinates()
print(x, y)  # Output: 10 20
```



Scope and Lifetime of Variables

Local Variables

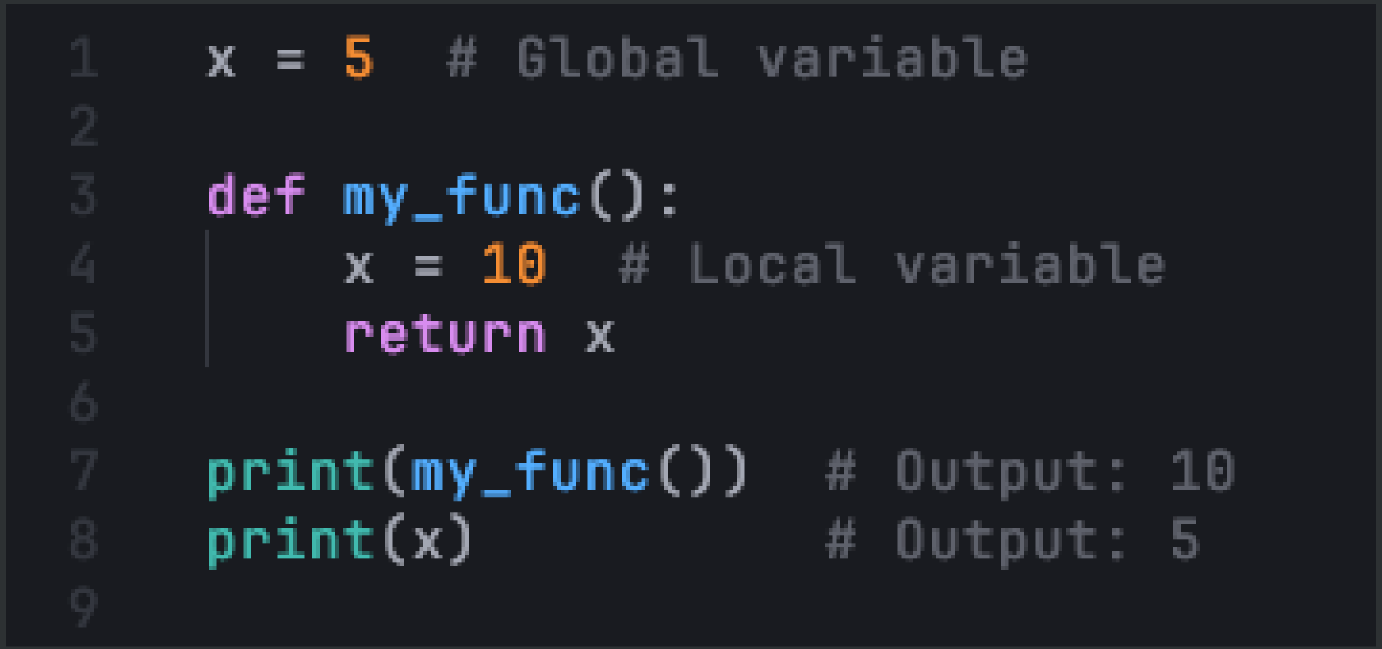
Defined within a function and accessible only there.

Global Variables

Defined outside functions and accessible throughout the module.

module.

Example:



```python
x = 5  # Global variable

def my_func():
    x = 10  # Local variable
    return x

print(my_func())  # Output: 10
print(x)          # Output: 5
```

Lambda Functions (Anonymous Functions)

Definition

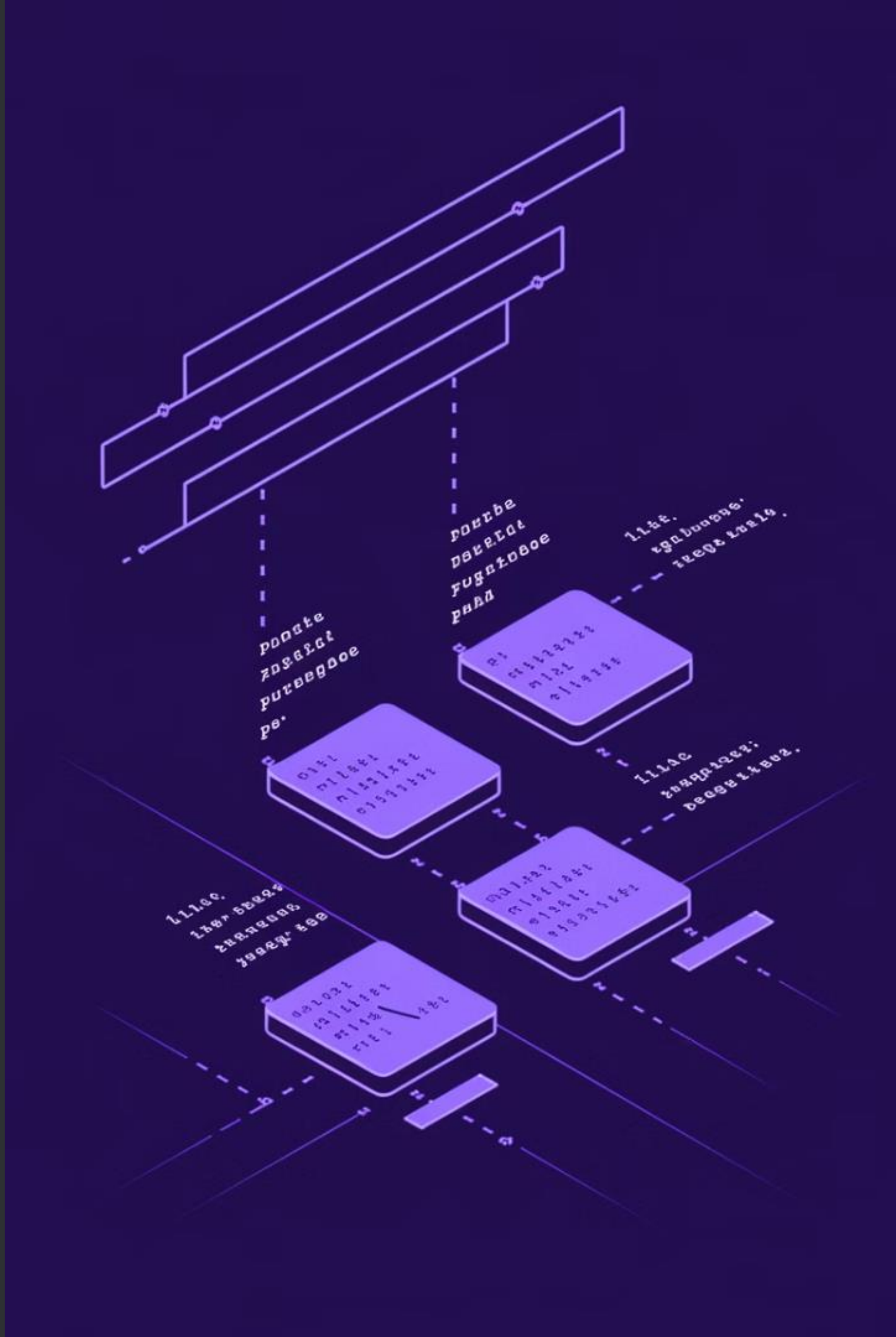
Small, unnamed functions defined with the lambda keyword.

Example

```python
square = lambda x: x * x

print(square(4))  # Output: 16
```

Usage

Often used for short operations or as arguments to higher-order order functions.



Higher-Order Functions and Decorators

First-Class Functions

In Python, functions are objects that can can be passed as arguments, returned returned from other functions, and assigned to variables.

Decorators

Functions that modify the behavior of other functions.

Function Enhancement

Add functionality without modifying the modifying the original function.

Example of a Decorator and returning a function:

```python
def my_decorator(func):
    def wrapper(*args, **kwargs):
        print("Before function call")
        result = func(*args, **kwargs)
        print("After function call")
        return result
    return wrapper

@my_decorator
def say_hello():
    print("Hello!")

say_hello()
# Output:
# Before function call
# Hello!
# After function call
```

```python
def multiplier(factor):
    def multiply(n):
        return n * factor
    return multiply

double = multiplier(2)
print(double(5))
# Output: 10
```

Recursion in Functions

Definition: A function that calls itself to solve a problem.

Key Concept: Always include a base case to prevent infinite recursion.

Base Case

The condition to stop recursion

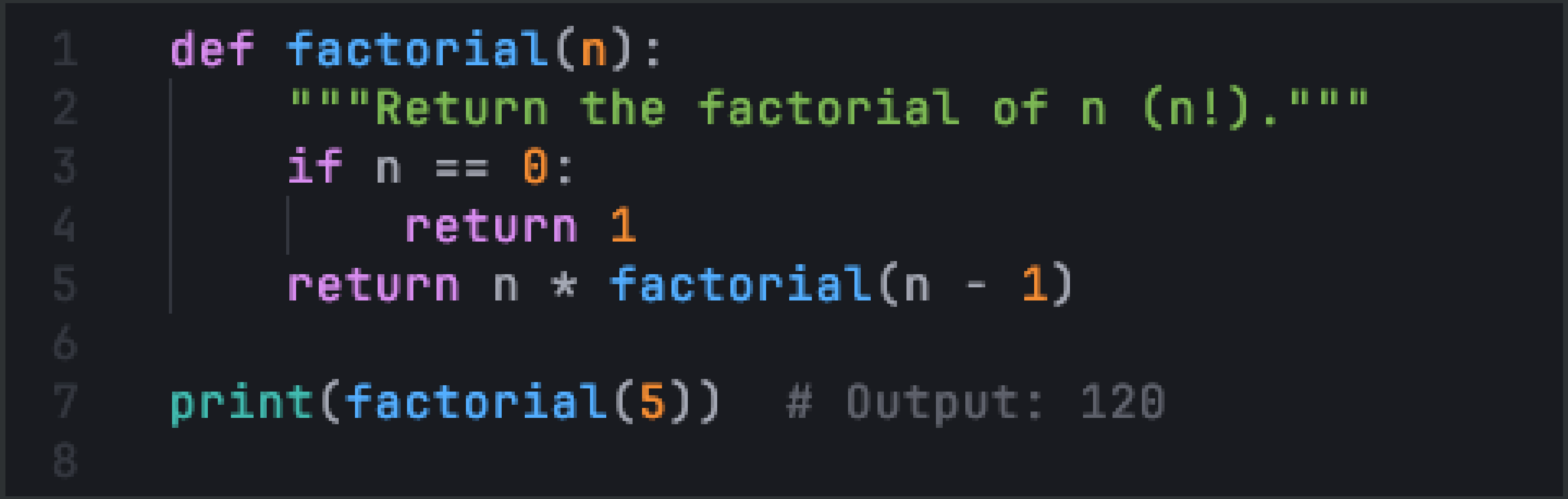
Recursive Case

The function calls itself

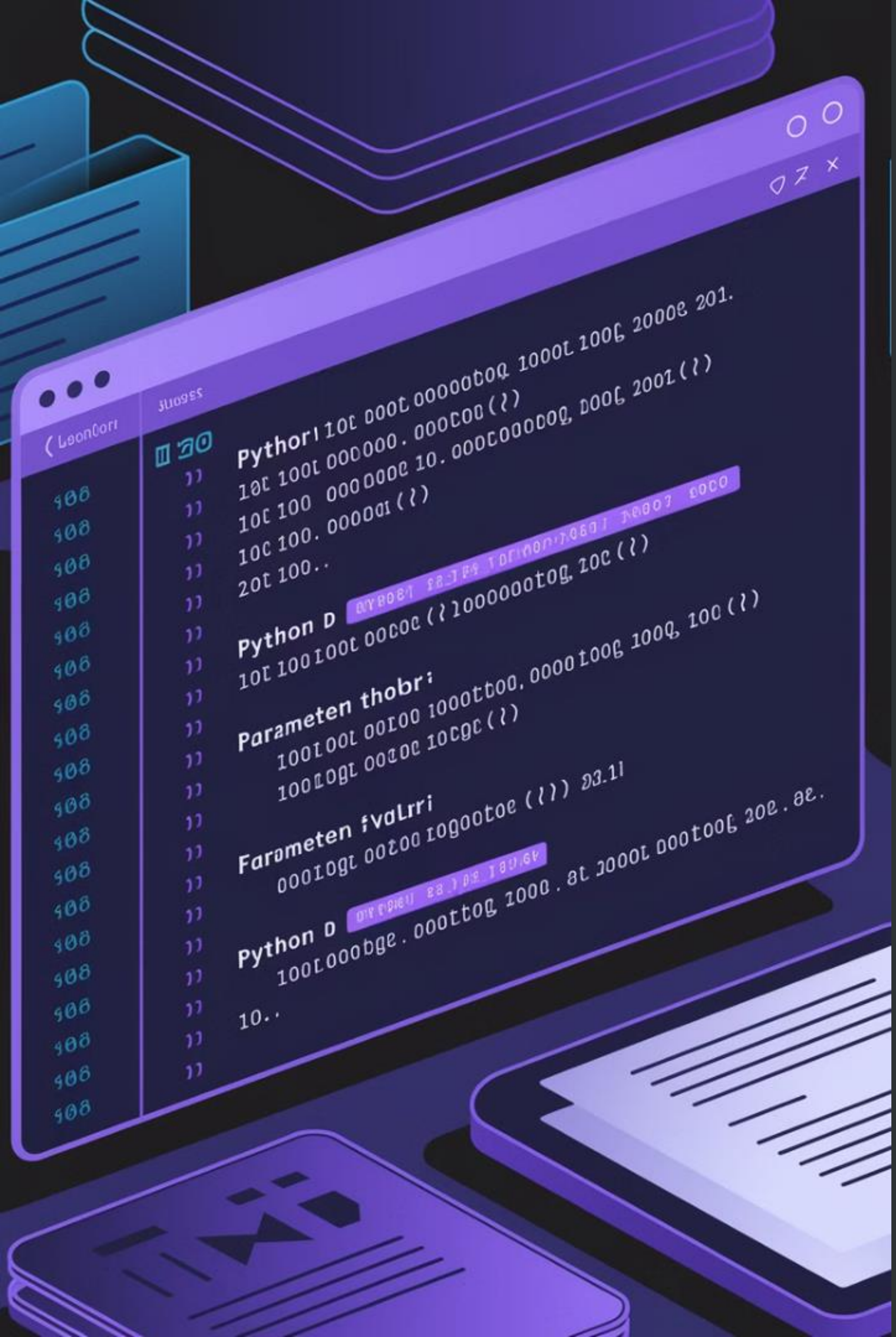
Problem Solving

Breaking complex problems into simpler simpler cases

Example: Factorial Calculation



```python
def factorial(n):
    """Return the factorial of n (n!)."""
    if n == 0:
        return 1
    return n * factorial(n - 1)

print(factorial(5))  # Output: 120
```



Documenting Functions

Docstrings

Use triple quotes to describe what your function does, its parameters, and return values.

Example

```python
def multiply(a, b):
    """
    Multiply two numbers and return the product.

    Args:
    a (int): The first number.
    b (int): The second number.

    Returns:
    int/float: The product of a and b.
    """
    return a * b
```

Importance

Enhances code readability and helps others (or future you) understand the function's purpose.

Best Practices

Keep Functions Focused

Each function should perform one task.

Meaningful Names

Use clear, descriptive names for functions and parameters.

Avoid Side Effects

Functions should, ideally, not alter state outside their scope.