Overview

This lesson explains how to write **Sphinx-style docstrings** for Python files and functions. You will learn how to write short descriptions, document parameters and return values, and use type hints directly in function definitions.

Important Information

A **docstring** is a multi-line string written directly under a file header or a function definition. It explains what the code does in clear, simple language.

Every docstring starts with a **short description**. This is one sentence that explains the purpose of the file or function.

Sphinx docstrings use special tags that begin with a colon. The most common tags used in this course are `:param:`, `:return:`, `:raises:`, `:note:`, and `:warning:`.

File-Level (Module) Docstring

A file-level docstring appears at the very top of a Python file and describes the entire file.

```
"""
simple_math.py

Provides basic math functions for integers and decimals.

Author: Alex Student
Date: 2026-01-25
Version: 1.0
Description: Used for small math operations in beginner programs.
"""
```

This docstring explains the purpose of the file and includes useful background information.

Using Type Hints in Function Definitions

Type hints are written directly in the function definition. They show what type of data each parameter expects and what type of value is returned.

```
def multiply(a: int, b: int) -> int:
    return a * b
```

The types are written after each parameter name, and the return type is written after `->`.

Basic Function Docstring

A function docstring is written directly under the function definition.

```
def is_positive(number: int) -> bool:
    """
    Checks if a number is positive.
    """
    return number > 0
```

The first line should always be a short description.

Documenting Parameters With `:param`

Each parameter is documented using `:param`.

```
def is_even(number: int) -> bool:
    """
    Checks if a number is even.

    :param number: The number to check.
    """
    return number % 2 == 0
```

The parameter name must match the function definition exactly.

Documenting Return Values With `:return:`

If a function returns a value, document what that value represents.

```
def square(number: int) -> int:
    """
    Squares a number.

    :param number: The number to square.
    :return: The squared value.
    """
    return number * number
```

Multiple Parameters

Each parameter gets its own `:param` entry.

```
def add_numbers(a: int, b: int) -> int:
    """
    Adds two numbers together.

    :param a: The first number.
    :param b: The second number.
    :return: The sum of the two numbers.
    """
    return a + b
```

Functions With No Return Value

If a function does not return anything, do not include a return tag.

```
def print_message(message: str) -> None:
    """
    Prints a message to the screen.

    :param message: The message to display.
    """
    print(message)
```

Documenting Errors With `:raises:`

Use `:raises:` when a function can cause an error.

```
def divide(a: float, b: float) -> float:
    """
    Divides one number by another.

    :param a: The number being divided.
    :param b: The number to divide by.
    :return: The result of the division.
    :raises ZeroDivisionError: If b is zero.
    """
    return a / b
```

If You Have Been Working Ahead

Hello! Mr. Forsyth here. The next sections of these lessons will start covering some libraries that we will use for projects. For Structured Programming 1, these first 20 lessons have covered most of the important things you need to understand in terms of general programming. If you understand these concepts well, you **do not** need any additional general python knowledge...

...however...

...you have been working ahead...

...which probably means that you find this course at least *somewhat* interesting.

Again, what we have covered so far is enough to get you full marks for Structured Programming 1. But you have shown yourself to be someone who has taken an interest in computer science. If you are still ahead, you may find it interesting to cover the Structured Programming 2 Python Lessons starting with Lesson 01 - Selection - If Statements and continuing through until 15 - Iteration - Nesting Loops.

So far in this course we have covered general python syntax and **sequence** control structures (the concept that code runs in order). The Structured Programming 2 Lessons cover **selection** (being able to run code based on conditions) and **iteration** (looping over code to have it repeat).

These control structures **are not necessary** for Structured Programming 1, *but* they will let you write code that can do more. A lot more. So much more that you can pretty much do whatever you want with code.

If you are interested in creating projects that go above and beyond, my recommendation is that you skip ahead to those lessons, and then come back after to make your projects. Hopefully, *hopefully*, you'll thank me later.