Writing context managers

1. Writing context managers

00:00 - 00:06

Now that you know how to use context managers, I want to show you how to write a context manager for other people to use.

2. Two ways to define a context manager

00:06 - 00:20

There are two ways to define a context manager in Python: by using a class that has special __enter__() and __exit__() methods or by decorating a certain kind of function.

3. Two ways to define a context manager

00:20 - 00:29

This course is focused on writing functions, and some of you may not have been introduced to the concept of classes yet, so I will only present the function-based method here.

4. How to create a context manager

00:29 - 00:58

There are five parts to creating a context manager. First, you need to define a function. Next, you can add any setup code your context needs. This is not required though. Third, you must use the "yield" keyword to signal to Python that this is a special kind of function. I will explain what this keyword means in a moment. After the "yield" statement, you can add any teardown code that you need to clean up the context.

5. How to create a context manager

00:58 - 01:22

Finally, you must decorate the function with the "contextmanager" decorator from the "contextlib" module. You might not know what a decorator is, and that's ok. We will discuss decorators in-depth in the next chapter of this course. For now, the important thing to know is that you write the "at" symbol, followed by "contextlib.contextmanager" on the line immediately above your context manager function.

6. The "yield" keyword

01:22 - 02:10

The "yield" keyword may also be new to you. When you write this word, it means that you are going to return a value, but you expect to finish the rest of the function at some point in the future. The value that your context manager yields can be assigned to a variable in the "with" statement by adding "as <variable name>". Here, we've assigned the value 42 that my_context() yields to the variable "foo". By running this code, you can see that after the context block is

done executing, the rest of the my_context() function gets run, printing "goodbye". Some of you may recognize the "yield" keyword as a thing that gets used when creating generators. In fact, a context manager function is technically a generator that yields a single value.

7. Setup and teardown

02:10 - 02:32

The ability for a function to yield control and know that it will get to finish running later is what makes context managers so useful. This context manager is an example of code that accesses a database. Like most context managers, it has some setup code that runs before the function yields. This context manager uses that setup code to connect to the database.

8. Setup and teardown

02:32 - 02:43

Most context managers also have some teardown or cleanup code when they get control back after yielding. This one uses the teardown section to disconnect from the database.

9. Setup and teardown

02:43 - 02:59

This setup/teardown behavior allows a context manager to hide things like connecting and disconnecting from a database so that a programmer using the context manager can just perform operations on the database without worrying about the underlying details.

10. Yielding a value or None

02:59 - 03:25

The database() context manager that we've been looking at yields a specific value - the database connection - that can be used in the context block. Some context managers don't yield an explicit value. in_dir() is a context manager that changes the current working directory to a specific path and then changes it back after the context block is done. It does not need to return anything with its "yield" statement.

11. Let's practice!

03:25 - 03:30

Now it's your turn to practice writing context managers.