

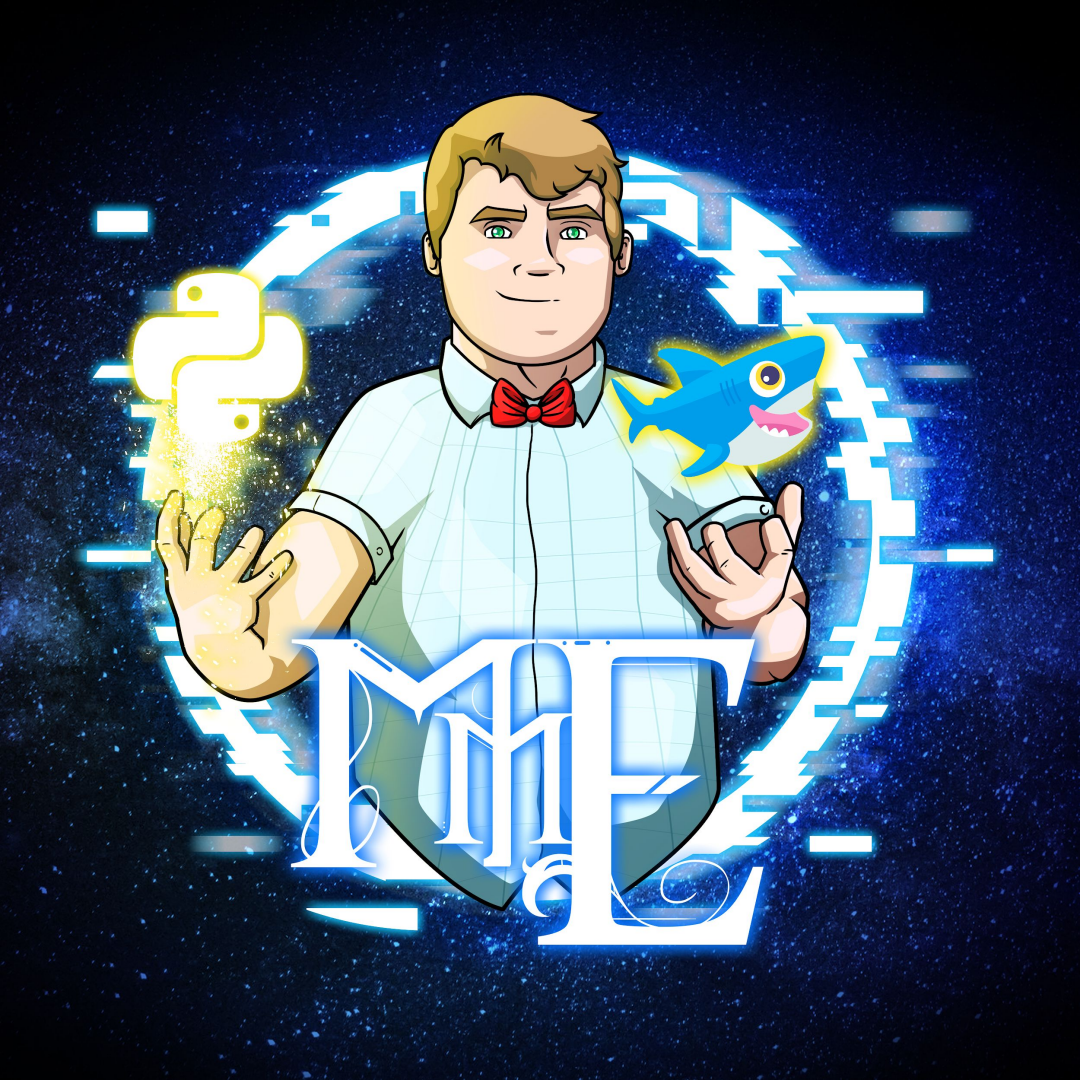
The Enters and Exits of Python's Context Managers

Mason Egger

Developer Advocate

[@masonegger](https://twitter.com/masonegger)

mason@do.co



Who's Seen Something Like This?



```
with open("file.txt", "r") as fh:  
    text = fh.read()
```

Context Managers

- Also lovingly called the **with** block
- Context Managers guarantee that some operation is performed after a block of code, even in the case of an exception, return, or exit.
 - Designed to simplify the **try/finally** pattern
- Allows for reusability, results in cleaner code, and considered “Pythonic”

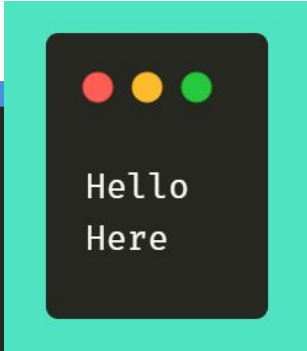
Context Managers Use Cases

- Spin Up/Tear Down
 - File Management
 - Socket connections
 - Database connections
 - Game environments (ppb)
- Managing global state
- Locking
- Mocking and Testing
- Logging
- And more!

Context Managers Class Implementation

- Context managers can be implemented as a class
- Implemented by using the classes `__enter__` and `__exit__` magic methods

```
class MyContextManager:  
    def __enter__(self):  
        return "Here"  
  
    def __exit__(self, exc_type, exc_value, traceback):  
        pass
```



Hello
Here

```
from example02 import MyContextManager  
  
with MyContextManager() as cm:  
    print("Hello")  
  
print(cm)
```

Context Managers `__enter__` method

- Executed at the very beginning, before the `with` block is entered
- Magic method takes only one argument: `self`
- Used to establish connection, modify system functions, setup, etc.
 - Wise to keep an original copy of anything system related you modify so you can change it back
- Must return a value that is stored in the variable specified by `as`

Context Managers `__enter__` Method Examples



```
class YellingText:
    def __enter__(self):
        import sys

        self.stdout = sys.stdout.write
        sys.stdout.write = self.yell
        return "YELLING"

    def yell(self, text):
        self.stdout(text.upper())

    def __exit__(self, exc_type, exc_value, traceback):
        pass
```



```
from example04 import YellingText

with YellingText() as cm:
    print("Hello")

print("I should not be yelling.")
```



```
HELLO
I SHOULD NOT BE YELLING.
```


Context Managers `__exit__` method

- Executed after the body of the `with` statement
- Returns a Boolean flag indicating if any exception that occurred should be suppressed
 - If True is returned the exception will be suppressed. Otherwise the exception will continue propagating up.
- Takes three arguments `exc_type`, `exc_val`, `traceback`
 - `exc_type` – The exception class
 - `exc_val` – The exception instance
 - Sometimes parameters passed to the exception can be found in `exc_val.args`
 - `traceback` – A traceback object

Context Managers `__exit__` Method Example



```
class YellingText:
    def __enter__(self):
        import sys
        self.stdout = sys.stdout.write
        sys.stdout.write = self.yell
        return "YELLING"

    def yell(self, text):
        self.stdout(text.upper())

    def __exit__(self, exc_type, exc_value, traceback):
        import sys
        sys.stdout.write = self.stdout
```



```
from example06 import YellingText

with YellingText() as cm:
    print("Hello")

print("I should not be yelling.")
```



```
HELLO
I should not be yelling.
```

Context Managers `__exit__` Exceptions

- If True is returned the exception will be suppressed. Otherwise the exception will continue propagating up.

```
class YellingText:
    def __enter__(self):
        import sys

        self.stdout = sys.stdout.write
        sys.stdout.write = self.yell
        return "Yelling"

    def yell(self, text):
        self.stdout(text.upper())

    def __exit__(self, exc_type, exc_value, traceback):
        import sys

        sys.stdout.write = self.stdout
        if exc_type is Exception:
            print("There was an exception")
            return True
```

```
from example08 import YellingText


with YellingText() as cm:
    print("Hello")
    raise Exception

print("I should not be yelling.")
```

```
HELLO
There was an exception
I should not be yelling.
```

Context Managers that Take Parameters

- Our very first example takes a parameters, but the `__enter__` method doesn't. How do we pass those?
- The `with` context calls `__init__`, first, then proceeds with the `__enter__` and `__exit__` calls. So you'll need to write your constructor to take these parameters.

A terminal window with a blue border and a black background. At the top left, there are three colored circles: red, yellow, and green. Below them, the following Python code is displayed:

```
with open("file.txt", "r") as fh:  
    text = fh.read()
```

Context Managers that Take Parameters

```
class FileOpen:
    def __init__(self, name, mode):
        self.name = name
        self.mode = mode
        self.file_handle = None

    def __enter__(self):
        self.file_handle = open(self.name, self.mode)
        return self.file_handle

    def __exit__(self, exc_type, exc_value, traceback):
        self.file_handle.close()
```

```
from example06 import FileOpen

with FileOpen("file.txt", "r") as fh:
    text = fh.read()

print(text)
```

Context Managers Function Implementation using `contextlib`

- Context Managers don't have to be implemented as classes. They can also be implemented at the function level with generators
- Use the `contextlib` library and the `@contextlib.contextmanager` decorator to specify
- Use the `yield` function to separate the enter and exit sections

Context Managers Function Implementation using `contextlib`

```
import contextlib

@contextlib.contextmanager
def whisper():
    import sys
    original_write = sys.stdout.write

    def whisper_write(text):
        original_write(text.lower())

    sys.stdout.write = whisper_write
    yield "whisper"
    sys.stdout.write = original_write
```

```
from example07 import whisper

with whisper() as w:
    print("THIS SHOULD BE WHISPERED")

print("THIS IS NOT WHISPERED")
```

```
this should be whispered
THIS IS NOT WHISPERED
```

Context Managers Function Implementation using `try finally`

```
import contextlib

@contextlib.contextmanager
def whisper():
    import sys

    original_write = sys.stdout.write

    def whisper_write(text):
        original_write(text.lower())

    sys.stdout.write = whisper_write
    try:
        yield "whisper"
    except Exception:
        print("Exception happened")
    finally:
        sys.stdout.write = original_write
```

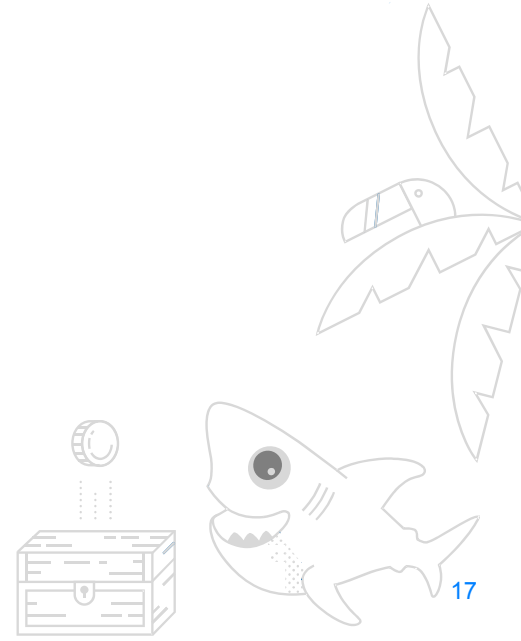
```
from example08 import whisper

with whisper() as w:
    print("THIS SHOULD BE WHISPERED")
    raise Exception

print("THIS IS NOT WHISPERED")
```

```
this should be whispered
exception happened
THIS IS NOT WHISPERED
```


Questions?



That's all for this time!

- All examples are on my GitHub
 - <https://github.com/MasonEgger/context-managers-sample-code>
- Follow me on Twitter [@masonegger](https://twitter.com/masonegger)
- Check out DigitalOcean's Interactive tutorials at do.co/interactive
- Check out DigitalOcean's eBook: [How To Code in Python](https://do.co/ebook-python) at do.co/ebook-python

