facebook

Context Managers:
You Can Write Your Own!

Daniel Porteous

Production Engineer

<link to slides>

Me!

Daniel Porteous

Tweeter:

@banool1000

Github:

github.com/banool

Website:

dport.me

Picture

<link to slides>

Self, PE at fb. From Townsville but lived in Melbourne for uni. I live in the bay area, CA now. Always found talks on core python features really great, super excited to give my first talk on one of my fave python features: context managers.

What are context managers?

You've seen them, you just don't know it!

<link to slides>

Lots of people I talked to didn't know context managers by name, but everyone has used them.

Anytime you see the with keyword (maybe as), you're looking at a context manager.

Here is the classic, most peoples' first introduction to context managers:

What are context managers?

You've seen them, you just don't know it!

```
with open("myfile.txt") as f:
    content = f.read()
    print(content)
```

<link to slides>

HAVE LINE NUMBERS

Simple example opens a file for you and gives you a handle, `f` as reference.

After `exit`ing with, f is closed.

Context managers, manager your context, in this case a file.

Note on scope: next slide

What are context managers?

You've seen them, you just don't know it!

```
with open("myfile.txt") as f:
    content = f.read()
print(content)
```

<link to slides>

After a context manager, variables defined in it still exist outside of it. Idk if that seems obvious, but it didn't to me so heads up.

What are context managers?

You've seen them, you just don't know it!

```
with open("myfile.txt") as f:
    pass
content = f.read()
print(content)
```

<link to slides>

The thing after the `as` still exists too, but it will be closed after the block.

Why use context managers?

They're pretty and safe is why!

- You can't forget to close resources.
- They can make code much prettier.
- They can make complex logic simpler.

<link to slides>

talk talk talk about points

Let's look at some more examples of context managers.

Why use context managers?

contextlib.suppress

```
def kill_process(pid):
    # Kill a process, ignore if it can't be found.
    with contextlib.suppress(ProcessLookupError):
        os.kill(pid, signal.SIGKILL)

# Nicer than:
def kill_process(pid):
    try:
        os.kill(pid, signal.SIGKILL)
    except ProcessLookupError:
        pass
```

<link to slides>

Why use context managers?

ThreadPoolExecutor - Bad version!

```
from concurrent.futures import ThreadPoolExecutor

# Bad!!!
pool = ThreadPoolExecutor()
for k, v in data.items():
    pool.submit(make_a_sentence, k, v)
# Wait on the results and do something with them.
pool.shutdown()
```

<link to slides>

talk talk talk about points

Let's look at some more examples of context managers.

Why use context managers?

ThreadPoolExecutor - Good version!

```
from concurrent.futures import ThreadPoolExecutor

# Good, safe, context managed!
with ThreadPoolExecutor() as pool:
    for k, v in data.items():
        pool.submit(make_a_sentence, k, v)
    # Wait on the results and do something with them.
```

<link to slides>

talk talk talk about points

Let's look at some more examples of context managers.

Why use context managers?

ThreadPoolExecutor - Good version!

```
data = {
    "Fruit": "spectacular",
    "Dairy": "yucky",
    "Meat": "not cool",
}

def make_a_sentence(noun, adjective):
    return f"{noun} is {adjective}!"
```

```
# Spinach is delicious!
# Fruit is spectacular!
# Dairy is yucky!
# Meat is not cool!
```

<link to slides>

talk talk talk about points

Let's look at some more examples of context managers.

Write your own context managers!

First, the hard way

But not necessarily the wrong way

write a context manager for something we've seen before

hmm open is a builtin and threadpoolexecutor is too complex. suppress would necessitate explanation of exception handling

<link to slides>

Our very own context manager!

Simple!

```
class MyContextManager:
    def __enter__(self):
        print("Enter!")

    def __exit__(self, *exc):
        print("Exit!")


with MyContextManager():
    print("Inside the block!")
```

<link to slides>

what does a context manager look like under the hood?

formally, context manager is any class with an __enter__ and __exit__ method, you must define both. __init__ method optional.

what would this output look like?

Our very own context manager!

Super simple!

```
class MyContextManager:
    def __enter__(self):
        print("Enter!")

    def __exit__(self, *exc):
        print("Exit!")


with MyContextManager():
    print("Inside the block!")
```

Output:

```
Enter!
Inside the block!
Exit!
```

<link to slides>

as you would expect!

you'll notice there is no `as`. let's see how that works:

Our very own context manager!

as neat as it gets

```
class FoodContextManager:
    def __init__(self):
        self.data = {}

    def __enter__(self):
        print(f"Enter: {self.data}")
        return self.data

    def __exit__(self, *exc):
        print(f"Exit: {self.data}")

with FoodContextManager() as data:
    data["vegetables"] = "delicious"
```

thing after `as` is ref to what you return from enter

```python
class FoodContextManager:
    def __init__(self, data):
        self.data = data

    def __enter__(self):
        print(f"Enter: {self.data}")
        return self.data

    def __exit__(self, *exc):
        print(f"Exit: {self.data}")


with FoodContextManager({"dairy": "yuck"}) as data:
    data["fruit"] = "delicious"
```

initialising context manager with data is pretty straight forward

pretty much everything except for some exception stuff with __exit__ (will we get to that?)

"Boy, that sure was a lot of work"

— Me when I first wrote a context manager this way

<link to slides>

wow a class and 2/3 methods just for that? surely there's an easier way!

There is an easier way!

`contextlib.contextmanager`

`@contextlib.contextmanager`

This function is a decorator that can be used to define a factory function for `with` statement context managers, without needing to create a class or separate `__enter__()` and `__exit__()` methods.

<link to slides>

enter contextlib.contextmanager

read the thingo. wow, no enter and exit! pretty neat :D

in summary: decorator you whack on top of generator to make it a context manager

let's look at an example

There is an easier way!

`contextlib.contextmanager`

```
class MyContextManager:
    def __enter__(self):
        print("Enter!")

    def __exit__(self, *exc):
        print("Exit!")


with MyContextManager():
    print("Inside the block!")
```

this is our old example showing how enter and exit work.

let's reforge this with contextmanager

There is an easier way!

contextlib.contextmanager

```
class MyContextManager:
    def __enter__(self):
        print("Enter!")

    def __exit__(self, *exc):
        print("Exit!")


with MyContextManager():
    print("Inside the block!")
```

```
@contextmanager
def MyContextManager():
    print("Enter!")
    yield
    print("Exit!")


with MyContextManager():
    print("Inside the block!")
```

here it is with contextlib.contextmanager (import excluded).

let's step through it piece by piece

There is an easier way!

contextlib.contextmanager

```python
class MyContextManager:
    def __enter__(self):
        print("Enter!")

    def __exit__(self, *exc):
        print("Exit!")


with MyContextManager():
    print("Inside the block!")
```

```python
@contextmanager
def MyContextManager():
    print("Enter!")
    yield
    print("Exit!")


with MyContextManager():
    print("Inside the block!")
```

bit above yield is enter

There is an easier way!

contextlib.contextmanager

```
class MyContextManager:
    def __enter__(self):
        print("Enter!")

    def __exit__(self, *exc):
        print("Exit!")


with MyContextManager():
    print("Inside the block!")
```

```
@contextmanager
def MyContextManager():
    print("Enter!")
    yield
    print("Exit!")


with MyContextManager():
    print("Inside the block!")
```

bit below yield is exit

lets go back to our food context manager!

There is an easier way!

`contextlib.contextmanager`

```python
from contextlib import contextmanager

@contextmanager
def food_context_manager(data):
    print(f"Enter: {data}")
    yield data
    print(f"Exit: {data}")

with food_context_manager({"dairy": "yuck"}) as data:
    data["fruit"] = "delicious"
```

notice same example as before! pretty simple!

There is an easier way!

`contextlib.contextmanager`

```
from contextlib import contextmanager

@contextmanager
def food_context_manager(data):
    print(f"Enter: {data}")
    yield data
    print(f"Exit: {data}")

with food_context_manager({"dairy": "yuck"}) as data:
    data["fruit"] = "delicious"
```

relationship yield and as

There is an easier way!

`contextlib.contextmanager`

```
from contextlib import contextmanager

@contextmanager
def food_context_manager(data):
    print(f"Enter: {data}")
    yield data
    print(f"Exit: {data}")

with food_context_manager({"dairy": "yuck"}) as data:
    data["fruit"] = "delicious"
```

data is initially this

A few extra notes

TODO

- Exceptions in exit
- try / finally in generator contest managers

<link to slides>

talk talk talk about points

Let's look at some more examples of context managers.

Other possible uses!

So many!

- Enclose an event and log it based on what happens.
- They can make code much prettier.
- They can make complex logic simpler.

<link to slides>

talk talk talk about points

Let's look at some more examples of context managers.

Slide Title

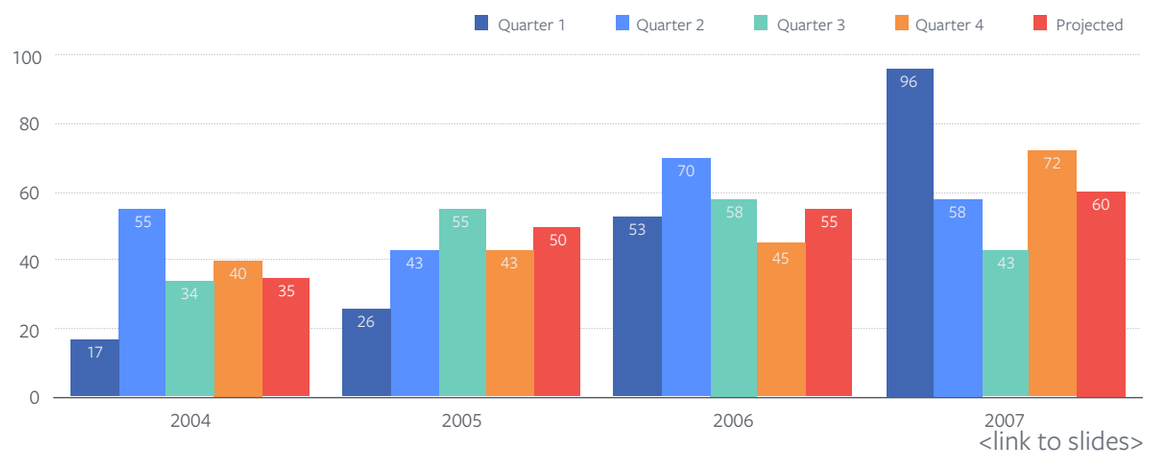
Slide Subtitle

```
with open(myfile, "r") as f:
    f.read()
```

<link to slides>

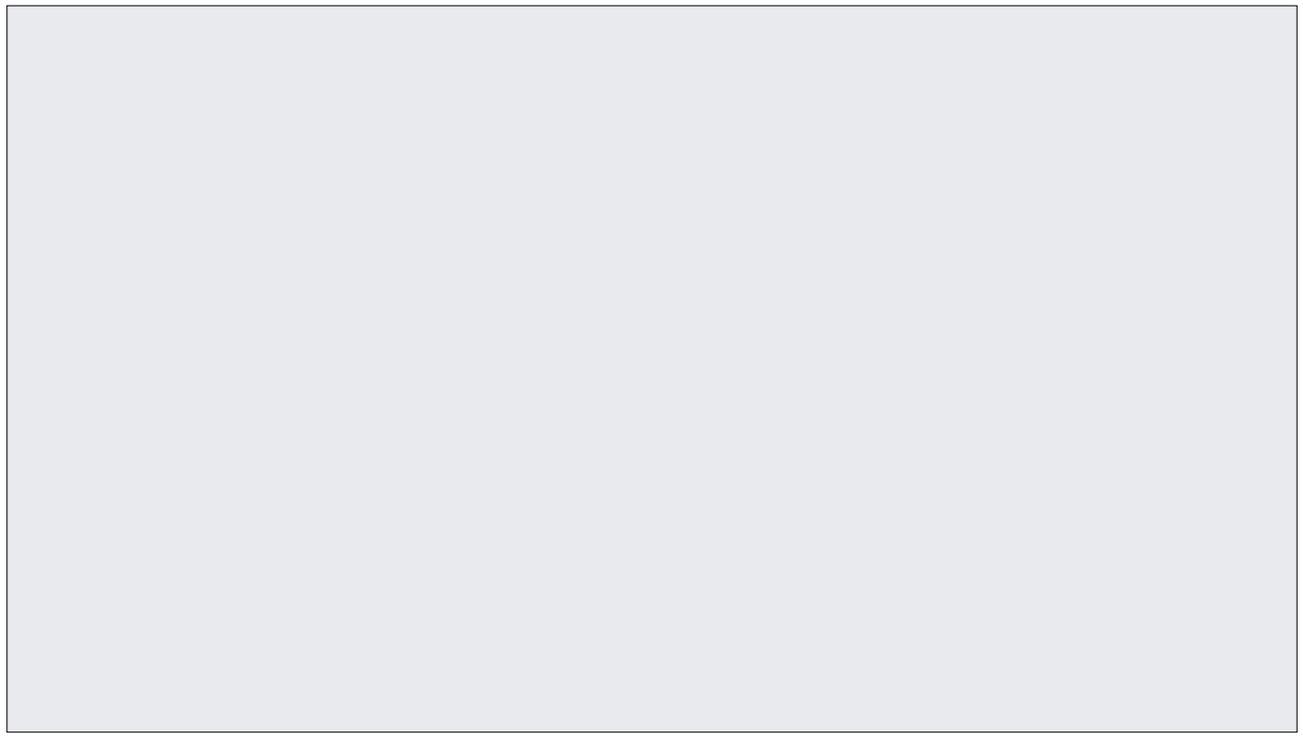
Slide Title

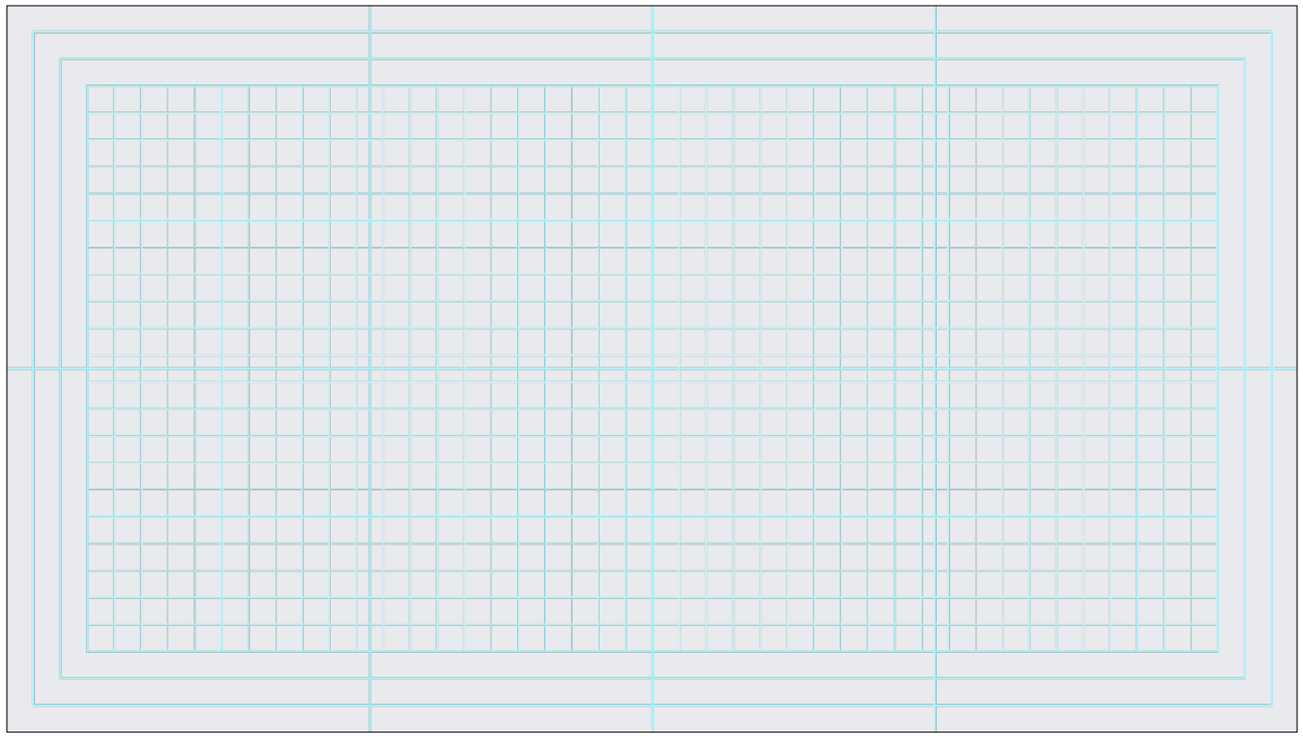
Slide Subtitle



<link to slides>

Source: sed ut unde omnis





facebook