facebook

Context Managers:
You Can Write Your Own!

Daniel Porteous

Production Engineer

dport.me/pycon.pdf

Me!

Me!!! Daniel Porteous!

Tweeter:

@banool1000

Github:

github.com/banool

Website:

dport.me

Agenda

TODO

- What is a context manager?
- Why use context managers?
- Core Python examples.
- Making our own context managers the hard way.
- The easy way!
- Best practices, gotchas, and more.

What are context managers?

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

What are context managers?

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

```
with
```

What are context managers?

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

`with` **`as`**

What are context managers?

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

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

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.
- More!

Why use context managers?

They're pretty and safe is why!

>>> import this

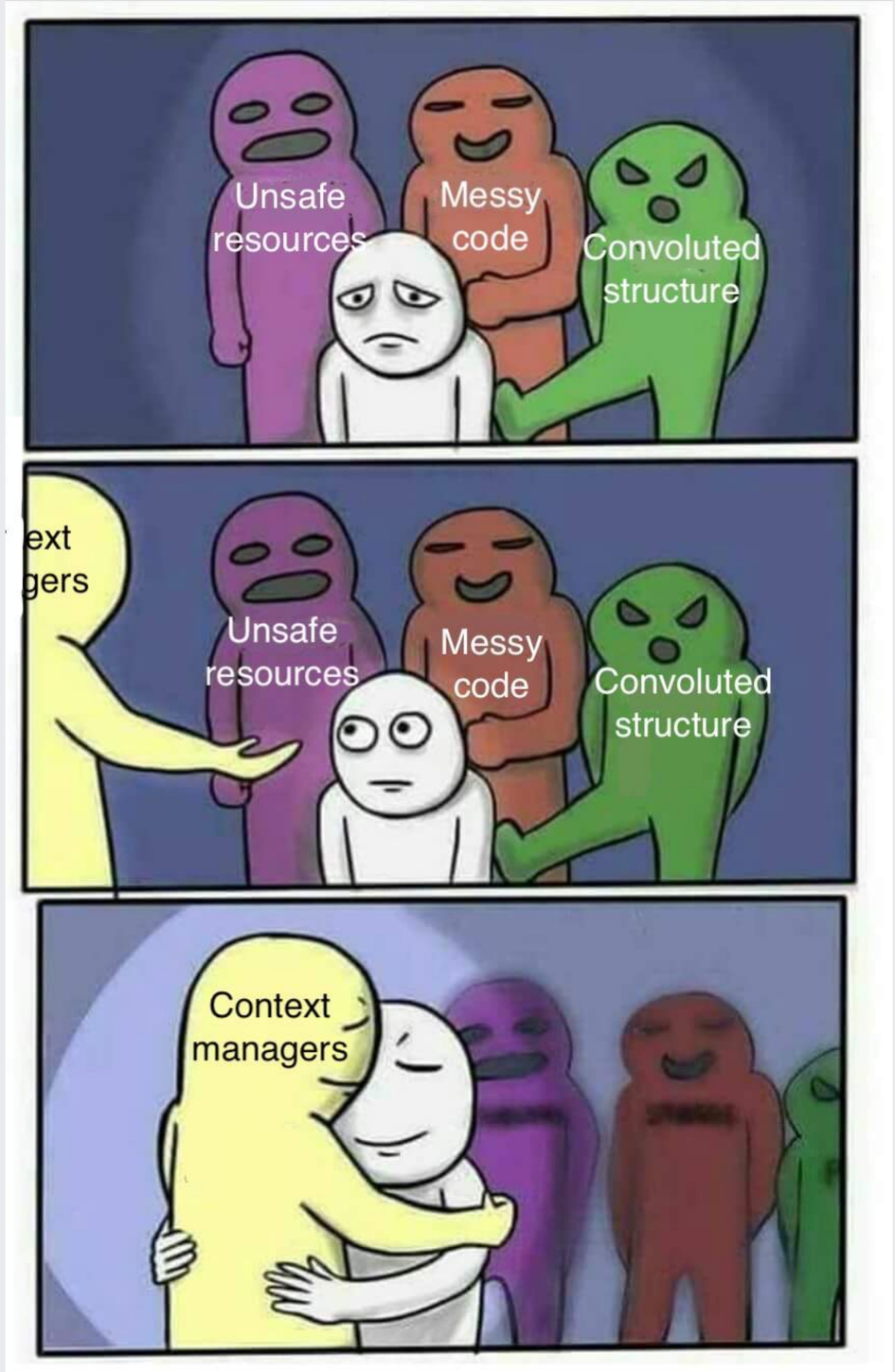
The Zen of Python, by Tim Peters

Beautiful is better than ugly.

Simple is better than complex.

Readability counts.

dport.me/pycon.pdf



Context managers in core Python

contextlib.suppress

TODO make the code look nicer

```
# Old School version
def kill_process(pid):
    try:
        os.kill(pid, signal.SIGKILL)
    except ProcessLookupError:
        pass
```

Context managers in core Python

contextlib.suppress

TODO make the code look nicer

```
# Old School version
def kill_process(pid):
    try:
        os.kill(pid, signal.SIGKILL)
    except ProcessLookupError:
        pass

def kill_process(pid):
    with contextlib.suppress(ProcessLookupError):
        os.kill(pid, signal.SIGKILL)
```

Context managers in core Python

ThreadPoolExecutor - Bad version!

```
from concurrent.futures import ThreadPoolExecutor

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

Context managers in core Python

ThreadPoolExecutor - Good version!

```
from concurrent.futures import ThreadPoolExecutor

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

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

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!")
```

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!
```

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"
```

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"
```

```
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"
```

```
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"
```

"Boy, that sure was a lot of work"

— Me when I first wrote a context manager this 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.

Generators

TODO

- code snippet of generator
- switch to terminal, call next manually to see how it works.
- then put it in for loop

Decorators

TODO

- code snippet of decorator
- show how it just wraps a function.
- syntactic sugar

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!")
```

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!")
```

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!")
```

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!")
```

```
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"
```

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"
```

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"
```

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"
```

All the rest

Some things you should really know

- Scope
- Exceptions in __exit__
- try / finally in generator context managers

Scope with Context Managers

Variables defined inside it still exist!

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

Scope with Context Managers

The thing yielded does too, but it'll be closed.

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

All the rest

Some things you should really know

- Scope
- **Exceptions in __exit__**
- try / finally in generator context managers

dport.me/pycon.pdf

Exceptions in __exit__

Making context managers even more powerful 💪

Exceptions in __exit__

Making context managers even more powerful 💪

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

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


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

Exceptions in __exit__

Making context managers even more powerful 💪

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

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


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

```
def __exit__(
    self,
    exc_type,
    exc,
    exc_tb,
):
    print("Exit!")
```

All the rest

Some things you should really know

- Scope
- Exceptions in __exit__
- **try / finally in generator context managers**

dport.me/pycon.pdf

try / finally

TODO

Best practices

TODO

- Do not explicitly re-raise exceptions in __exit__ methods, return False.
- Know the roles of __init__ and __enter__
  - No side effects in __init__
  - Don’t make __init__ too expensive.

Other possible uses!

So many!

- Enclose an event and log based on what happens.
- More!

Key take aways

Context managers are amazing!

- Use context managers

Questions?

facebook

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!
```

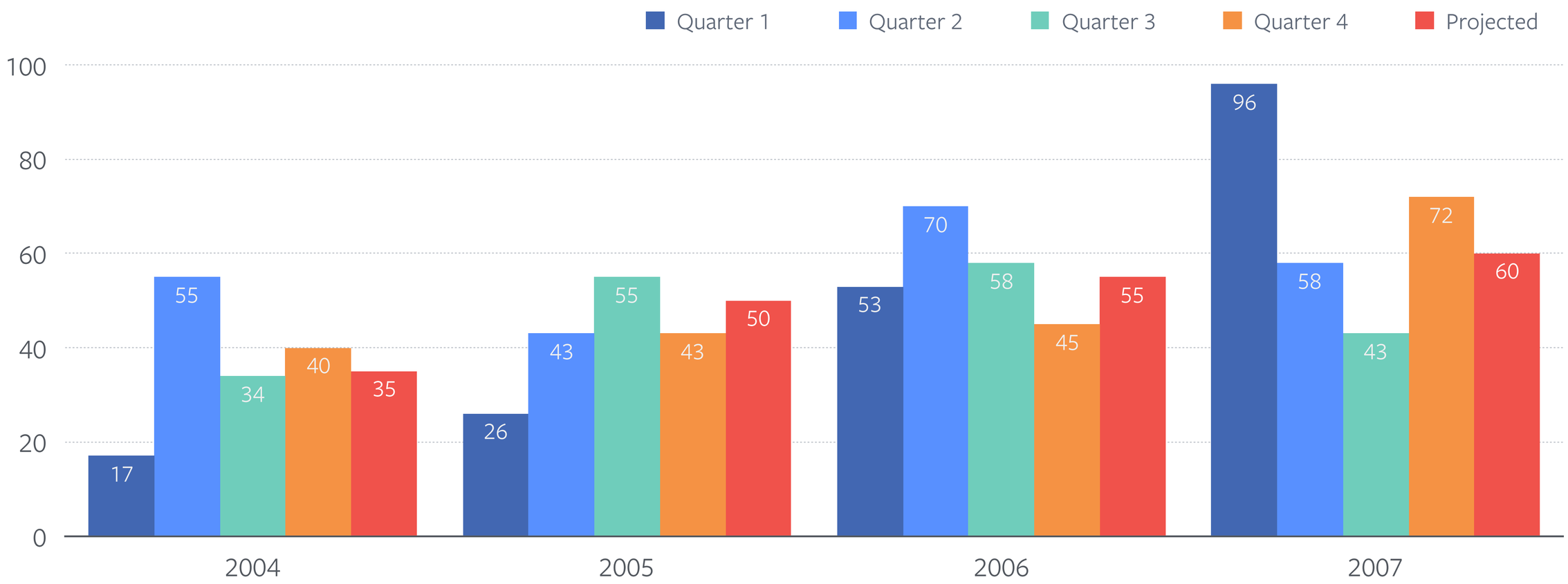
Slide Title

Slide Subtitle

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

Slide Title

Slide Subtitle



Source: sed ut unde omnis