Defining Context Managers

Austin Bingham
@austin_bingham
austin@sixty-north.com

Presenter

Robert Smallshire
@robsmallshire
rob@sixty-north.com

pluralsight
hardcore dev and IT training

context manager

an object designed to be used in a with-statement

```
with context-manager:
    body
```

context manager

an object designed to be used in a with-statement

```
with context-manager:
    context-manager.begin()
    body
    context-manager.end()
```

context manager

an object designed to be used in a with-statement

```
with context-manager:
    setup()
    body
    teardown()
```

context manager

an object designed to be used in a with-statement

```
with context-manager:
    construction()
    body
    destruction()
```

context manager

an object designed to be used in a with-statement

```
with context-manager:
    allocation()
    body
    deallocation()
```

context manager

an object designed to be used in a with-statement

```
with context-manager:
    enter()
    body
    exit()
```

A context-manager ensures that resources are properly and automatically managed

`enter()` prepares the manager for use

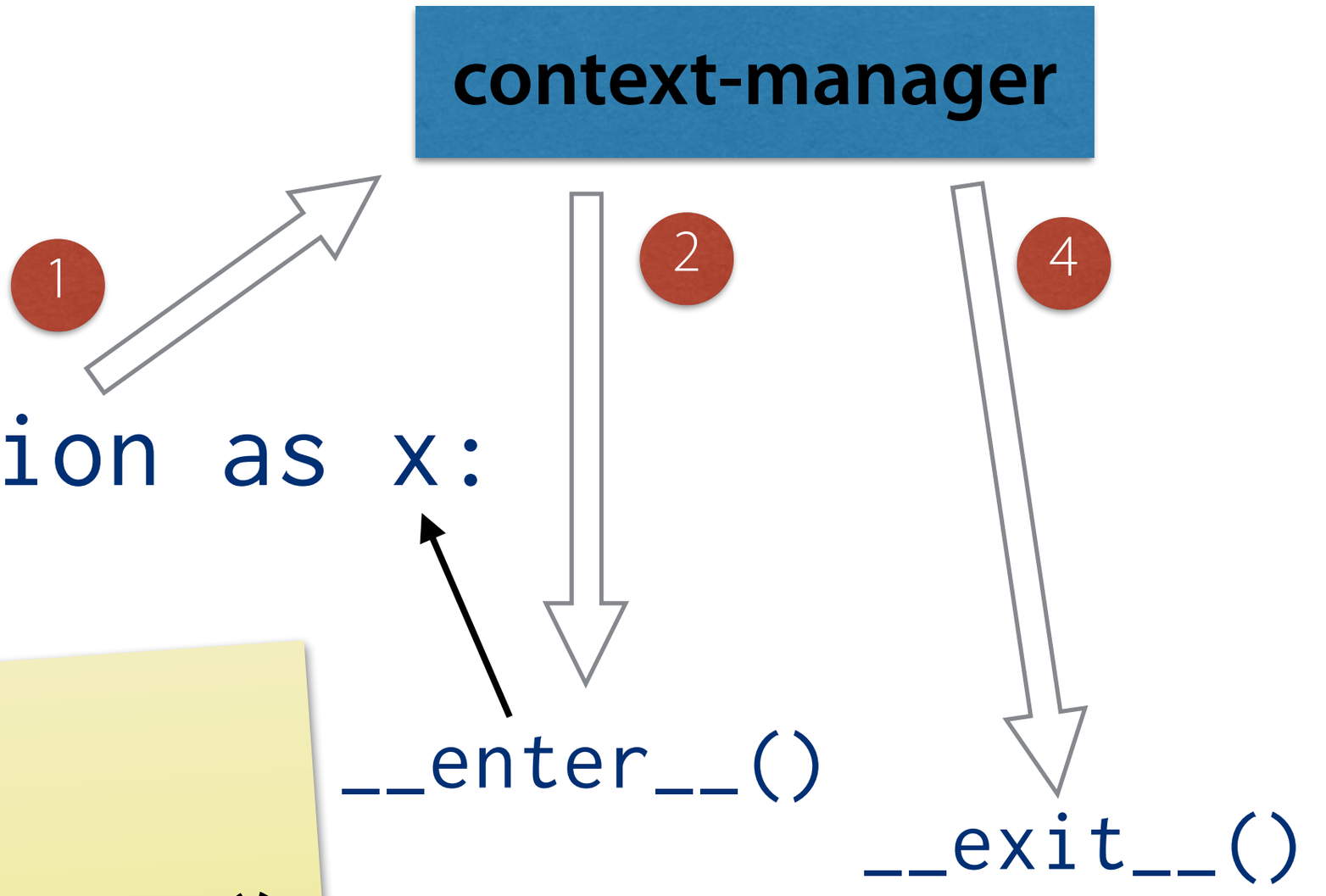
`exit()` cleans it up

Context-manager Protocol

```
__enter__(self)

__exit__(self,
         exc_type,
         exc_val,
         exc_tb)
```

Context-manager Protocol



Important!

The value of

`expression.__enter__()`

is bound to `x`, not the value of

`expression`

`__enter__()`

- called before entering with-statement body
- return value bound to `as variable`
- can return value of any type
- commonly returns context-manager itself

`__exit__()`

called when with-statement body exits

```
__exit__(self, exc_type, exc_val, exc_tb)
```

- `exc_type`: exception type
- `exc_val`: exception object
- `exc_tb`: exception traceback

`__exit__()` **can check** `type` **for** `None` **to see if an exception was thrown**

By default `__exit__()` propagates exceptions thrown from the with-statement's code block

If `__exit__()` returns `False`, the exception is **propagated**

`__exit__()` answers the question

"should the with-statement swallow exceptions?"

By default functions return `None`.

`None` evaluates to `False`.

`__exit__()` should never explicitly re-raise exceptions

`__exit__()` **should never explicitly re-raise exceptions**

`__exit__()` **should only raise exceptions if it fails itself**

PEP343

the PEP that defines context managers

www.python.org/dev/peps/pep-0343/

with-statement Expansion

```
mgr = (EXPR)
exit = type(mgr).__exit__  # Not calling it yet
value = type(mgr).__enter__(mgr)
exc = True
try:
    try:
        VAR = value  # Only if "a
        BLOCK
    except:
        # The exceptional case is handled here
        exc = False
        if not exit(mgr, *sys.exc_info()):
            raise
        # The exception is swallowed if exit() returns true
finally:
    # The normal and non-local-goto cases are handled here
    if exc:
        exit(mgr, None, None, None)
```

Restating what we've covered

contextlib

standard library module for working with context managers

contextlib.contextmanager

a decorator you can use to create new context managers

contextmanager

a decorator you can use to create new context managers

```python
@contextlib.contextmanager
def my_context_manager():
    # <ENTER>
    try:
        yield [value]
        # <NORMAL EXIT>
    except:
        # <EXCEPTIONAL EXIT>
        raise

with my_context_manager() as x:
    # . . .
```

`contextmanager` **lets you define context-managers with simple control flow**

`contextmanager` **lets you define context-managers with simple control flow**

It allows you to leverage the statefulness of generators

`contextmanager`

- Use standard exception handling to propagate exceptions

- Explicitly re-raise – or don't catch – to propagate exceptions

- Swallow exceptions by not re-raising them

multiple context managers

with-statements can use as many context-managers as you need

```
with cm1() as a, cm2() as b, ...:
    BODY
```

```
with cm1() as a, cm2() as b:
    BODY
```

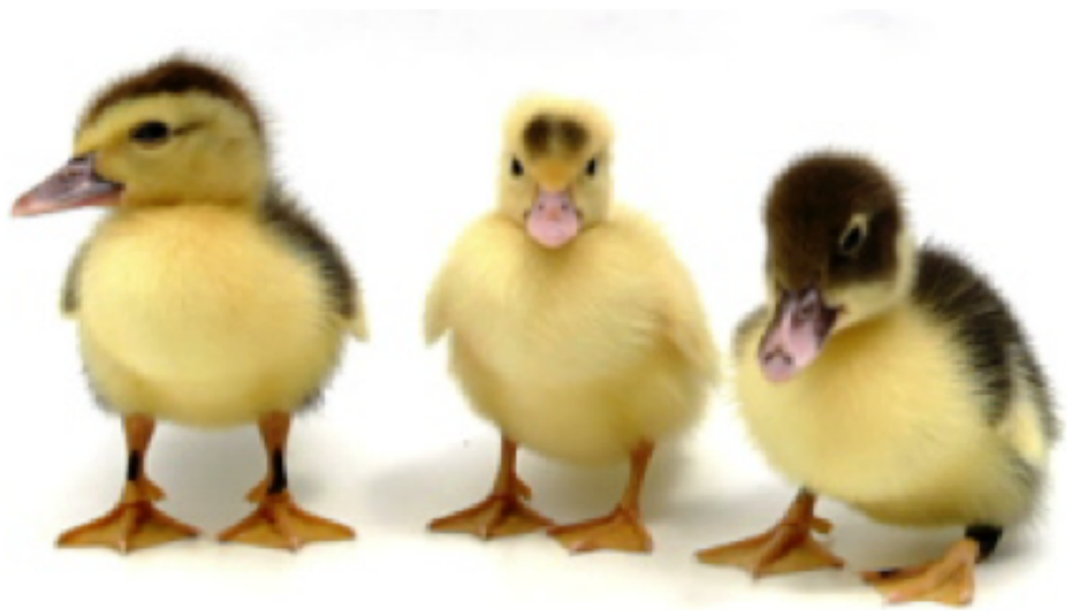
is the same as

```
with cm1() as a:
    with cm2() as b:
        BODY
```

Exceptions propagated from inner context managers will be seen by outer context managers

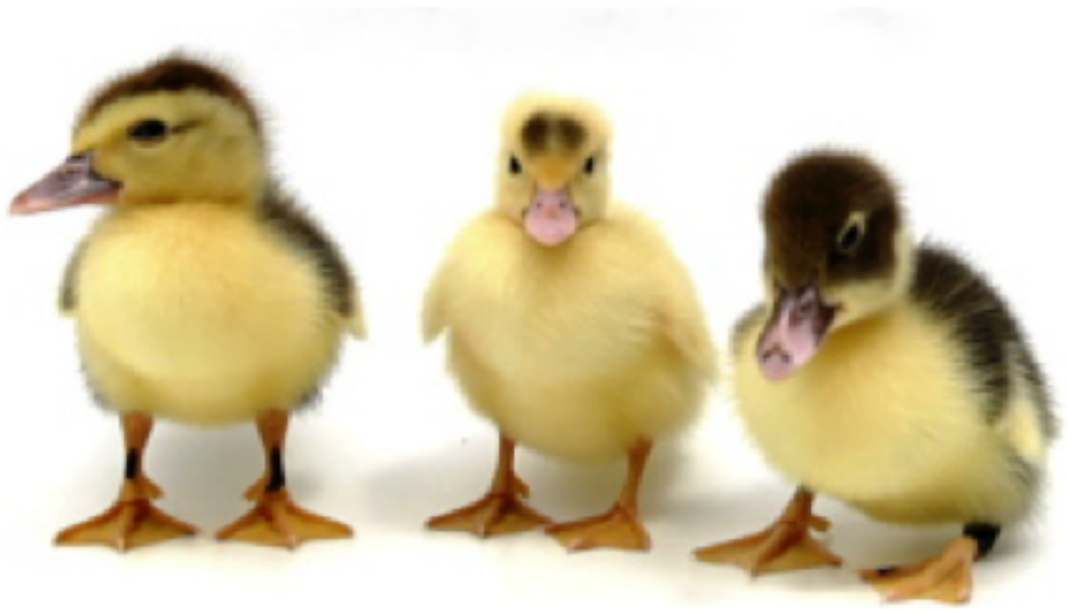
Tales of real-world Python
– better in practice than in theory –
just like Duck Typing.

Duck Tails



Duck Tails

Context Managers for Transactions



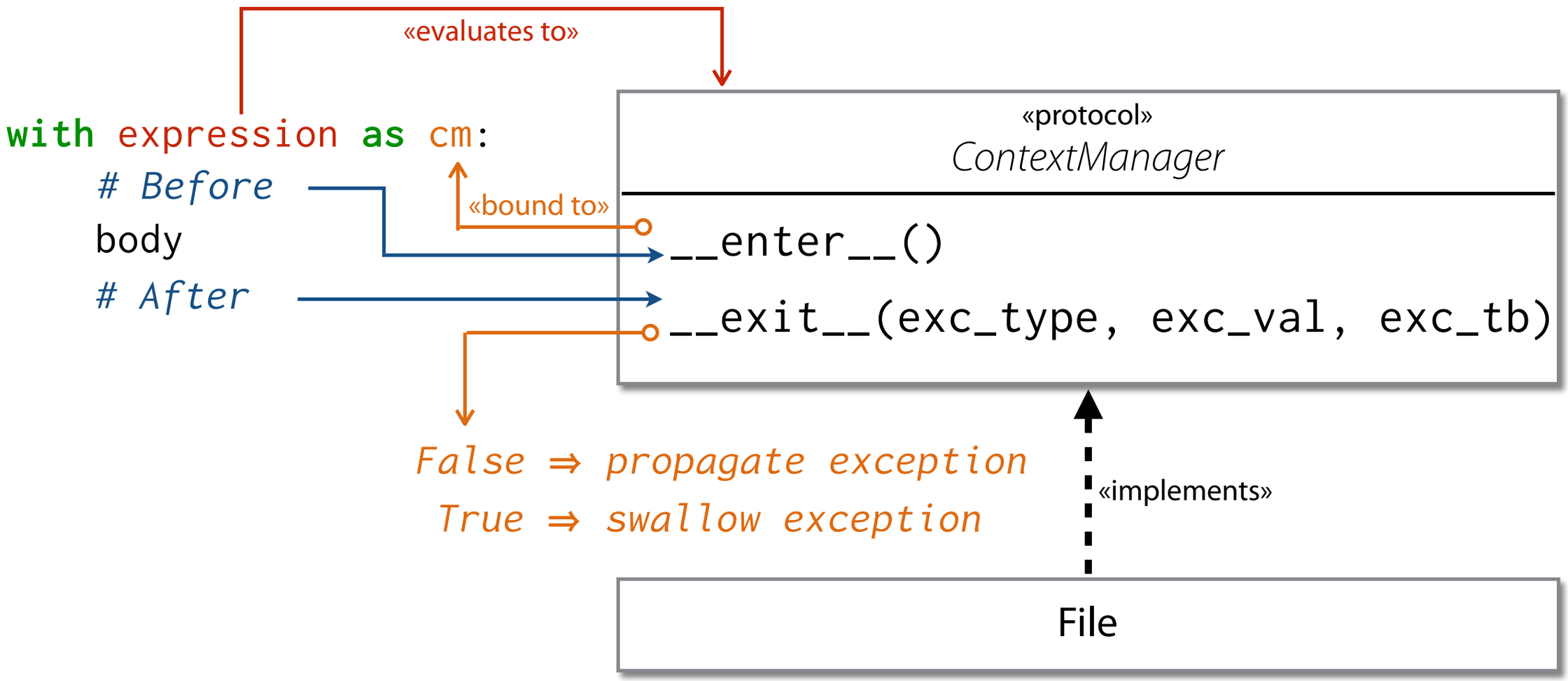
Duck Tails

Let's model database transactions!

Is easy!

Creating context managers

Defining Context Managers



All defined in PEP 343

Defining Context Managers

```
@contextlib.contextmanager
def generator_function():
    #   <ENTER>
    try:
        yield as_variable
        # <NORMAL EXIT>
    except:
        # <EXCEPTIONAL EXIT>
        raise
```

Defining Context Managers

```
with cm1() as a:
    with cm2() as b:
        BODY
```

```
with cm1() as a, cm2() as b:
    BODY
```

```
with cm1() as a,\
     cm2() as b:
    BODY
```