Boring Object Orientation

Moshe Zadka – https://cobordism.com

2021

Acknowledgement of Country

Belmont (in San Francisco Bay Area Peninsula)
Ancestral homeland of the Ramaytush Ohlone

Python and object oriented programming

Everything is an object

Why OO design principles?

Guidelines to code that is easy to maintain

Do OO design principles work?

Yes

Do OO design principles work?

Yes ...but

Principles

Make your objects more boring! The simple tricks that they don't want you to know!

- ▶ Declare interfaces

Principles

Make your objects more boring! The simple tricks that they don't want you to know!

- ▶ Declare interfaces
- ▶ Simplify initialization

Principles

Make your objects more boring! The simple tricks that they don't want you to know!

- ▶ Declare interfaces
- ▶ Simplify initialization
- ▶ Avoid mutation

Principles

Make your objects more boring! The simple tricks that they don't want you to know!

- ▶ Declare interfaces
- ▶ Simplify initialization
- ▶ Avoid mutation
- ▶ Avoid hiding

Principles

Make your objects more boring! The simple tricks that they don't want you to know!

- ▶ Declare interfaces
- ▶ Simplify initialization
- ▶ Avoid mutation
- ▶ Avoid hiding
- ▶ Avoid methods

Principles

Make your objects more boring! The simple tricks that they don't want you to know!

- ▶ Declare interfaces
- ▶ Simplify initialization
- ▶ Avoid mutation
- ▶ Avoid hiding
- ▶ Avoid methods
- ▶ Avoid inheritance

Why declare interfaces?

Explicit is better than implicit

Declaring interfaces with zope.interface

```python
from zope import interface

class ISprite(interface.Interface):

    bounding_box = interface.Attribute(
        "The bounding box"
    )

    def intersects(box):
        "Does this intersect with a box"
```

Testing for interface provision

```python
from zope.interface import verify

def test_implementation():
    sprite = make_sprite()
    verify.verifyObject(ISprite, sprite)
```

Interesting constructors

```python
class Stuff:

    def __init__(self, fname):
        # Create a new object
        self.destination = Destination()
        # Call a system call
        self.finput = open(fname)
```

Boring constructors

```python
class Stuff:

    def __init__(self, finput, destination):
        self.destination = destination
        self.finput = finput

    @classmethod
    def from_name(cls, name):
        # Create a new object
        destination = Destination()
        # Call a system call
        finput = open(fname)
        return cls(finput, destination)
```

Why boring constructors?

- Testing

Why boring constructors?

- Testing
- Correctness

Why boring constructors?

- Testing
- Correctness
- Async

Why boring constructors?

- Testing
- Correctness
- Async

Using attrs

```python
import attr

@attr.s(auto_attribs=True)
class Stuff:
    finput: Any
    destination: Any
```

Immutable objects

```python
@attr.s(auto_attribs=True, frozen=True)
class Stuff:
    destination: Any
    finput: Any


my_stuff = Stuff(Destination(), io.StringIO())


try:
    my_stuff.finput = io.StringIO()
except Exception as exc:
    print(repr(exc))


FrozenInstanceError()
```

Immutablity as bug avoidance

```python
def some_function(some_list=[]):
    pass
```

Immutablity as interface simplifying

No variation, no invariant breakage!

Change without mutation

```python
@attr.s(auto_attribs=True, frozen=True)
class Point:
    x: float
    y: float


origin = Point(0, 0)


up = attr.evolve(origin, y=1)


origin, up


(Point(x=0, y=0), Point(x=0, y=1))
```

Private methods

```python
class HTTPSession:
    def _request(self, method, url):
        pass
    def get(self, url):
        return self._request('GET', url)
    def head(self, url):
        return self._request('HEAD', url)
```

Private methods to separate class

```python
class RawHTTPSession:
    def request(self, method, url):
        pass
```

Private attribute instance

```python
class HTTPSession:
    _raw: RawHTTPSession
    def get(self, url):
        return self._raw.request('GET', url)
    def head(self, url):
        return self._raw.request('HEAD', url)
```

Methods

```python
@attr.s(auto_attribs=True, frozen=True)
class Point2D:
    x: float
    y: float

    def distance_from_origin(self):
        return (self.x**2 + self.y**2) ** 0.5
```

Methods

```python
@attr.s(auto_attribs=True, frozen=True)
class Point3D:
    x: float
    y: float
    z: float

    def distance_from_origin(self):
        return (self.x**2 +
                self.y**2 +
                self.z**2)
```

Why not methods?

Bloats classes

Methodless Polymorphism

```python
@attr.s(auto_attribs=True, frozen=True)
class Point2D:
    x: float;
    y: float


@attr.s(auto_attribs=True, frozen=True)
class Point3D:
    x: float
    y: float
    z: float
```

Methodless Polymorphism

```python
import functools


@functools.singledispatch
def distance_from_origin(pt):
    raise NotImplementedError(point)


@distance_from_origin.register(Point2D)
def distance_2d(pt):
    return (pt.x**2 + pt.y**2) ** 0.5


@distance_from_origin.register(Point3D)
def distance_3d(pt):
    return (pt.x**2 + pt.y**2 + pt.z**2) ** 0.5
```

Inheritance as API: Twisted

```python
# From the Twisted tutorial
class FingerProtocol(basic.LineReceiver):
    def lineReceived(self, user):
        self.transport.write(b"No such user\r\n")
        self.transport.loseConnection()
```

Inheritance as API: Django

```python
# From the Django tutorial
class IndexView(generic.ListView):
    template_name = 'polls/index.html'
    context_object_name = 'latest_question_list'

    def get_queryset(self):
        """Return the last five published questions
        return Question.objects.order_by('-pub_date
```

Inheritance as API: Jupyter

```python
# From the Jupyter documentation

class EchoKernel(Kernel):
    implementation = 'Echo'
    implementation_version = '1.0'
    language = 'no-op'
    language_version = '0.1'
    language_info = {
        'name': 'Any text',
        'mimetype': 'text/plain',
        'file_extension': '.txt',
    }
    banner = "Echo kernel - as useful as a parrot"

    def do_execute(self, code, silent, store_history
                   allow_stdin=False):
        ...
```

Inheritance as API: Issues

"Shared everything"

Composition

- Define *interface*
- Useful behavior in *referred class*

Composition: Example

```python
class IMovable(interface.Interface):
    x_position = interface.Attribute("x coordinate"
    y_position = interface.Attribute("y coordinate"
    def tick():
        pass
```

Composition: Example

```python
@interface.implementer(IMovable)
@attr.s(auto_attribs=True)
class StraightLine:
    dx: float
    dy: float
    x_position: float
    y_position: float
    def tick(self):
        self.x_position += dx
        self.y_position += dy
```

Composition: Example

```python
@interface.implementer(IMovable)
@attr.s(auto_attribs=True)
class Sprite:
    _movable: IMovable
    @property
    def x_position(self):
        return self._movable.x_position
    @property
    def y_position(self):
        return self._movable.y_position
    def tick(self):
        return self._movable.tick()
```

Python: Language of the free

Diamond inheritance with overriddable constructors as mandatory interface? Sure!

Python: With Great Power

Diamond inheritance with overriddable constructors as mandatory interface?

Maybe not...

Opaque lessons learned

- Do as we say, not as we do
- Big systems, big headaches

Less interesting code

Be dumb as possible when writing code.