Why Change?

Life is Better Without It

Moshe Zadka – https://cobordism.com

2020

Shared Mutable State is Bad

Options:

- Don't share
- Don't mutate

Avoid Sharing?

What about

- Modules
- Function defaults
- Class variables
- Arguments

Avoid Mutating!

Much better.

Digression: Are Squares Rectangles?

...and why would anyone care?

What's a Rectangle (in Python)

```
class IRectangle(Interface):
    def get_height() -> float:
        """Return height"""
    def get_width() -> float:
        """Return width"""
    def set_height(height: float):
        """Set height"""
    def set_width(width: float):
        """Set width"""
```

What's a Square (in Python)

```
@implementer(IRectangle)
@attr.s(auto_attribs=True)
class Square:
    _side: float
    def get_height(self) -> float: return self._sid
    def get_width(self) -> float: return self._side
try: verifyClass(IRectangle, Square)
except Exception as exc:
    print(textwrap.dedent(str(exc).split(":")[1]
          ).strip().replace("__main__.", ""),
          file=sys.stderr)

The IRectangle.set_height(height) attribute was not
The IRectangle.set_width(width) attribute was not p
```

What's a Square (in Python) (Fixed)

An easy mistake – we forgot a couple of methods.
Let's fix that.

```
@implementer(IRectangle)
@attr.s(auto_attribs=True)
class Square:
    _side: float
    def get_height(self) -> float: return self._sid
    def get_width(self) -> float: return self._side
    def set_height(self, height: float):
        self._side = height # ???
    def set_width(self, width: float):
        self._side = width # ???
```

What Do You Do With a Shape (in Python)

```
def area(rectangle: IRectangle) -> float:
    return (rectangle.get_height() *
            rectangle.get_width())
def double_height(rectangle: IRectangle) -> float:
    rectangle.set_height(
        2 * rectangle.get_height())
```

What Do You Do With a Shape (in Python) (Cont.)

```
x = Square(side=5)
print(area(x))
double_height(x)
print(area(x))
```

```
25
100
```

Let's Stop Mutating

```
class IRectangle(Interface):
    def get_height() -> float:
        """Return height"""
    def get_width() -> float:
        """Return width"""
    def with_height(height: float) -> IRectangle:
        """Rectangle with same width, new height"""
    def with_width(width: float) -> IRectangle:
        """Rectangle with same height, new width"""
```

The Immutable Rectangle

```
@implementer(IRectangle)
@attr.s(auto_attribs=True, frozen=True)
class Rectangle:
    _height: float
    _width: float
    def get_height(self) -> float:
        return self._height
    def get_width(self) -> float:
        return self._width
    def with_height(self, height) -> float:
        return attr.evolve(self, height=height)
    def with_width(self, width) -> float:
        return attr.evolve(self, width=width)
```

The Immutable Square

```
@implementer(IRectangle)
@attr.s(auto_attribs=True)
class Square:
    _side: float
    def get_height(self) -> float:
        return self._side
    def get_width(self) -> float:
        return self._side
    def with_height(self, height: float) -> IRectan
        return Rectangle(width=self._side,
                         height=height)
    def with_width(self, width: float) -> IRectangl
        return Rectangle(height=self._side,
                         width=width)
verifyClass(IRectangle, Square)

True
```

What Do You Do With an Immutable Shape (in Python)

```
def double_height(rectangle):
    return rectangle.with_height(
        2 * rectangle.get_height())
x = Square(side=5)
print(area(x))
print(area(double_height(x)))
```

```
25
50
```

Let's Get Back to Sharing

At some point, someone told you not to do this. Do you remember why?

```
def sum_with_extra(e1, e2, things=[]):
    things.append(e1)
    things.append(e2)
    return sum(things)
```

A Bad Trip Down Memory Lane

```
sum_with_extra(1, 2, [3, 4])

10

sum_with_extra(1, 2)

3

# Whoops!
sum_with_extra(1, 2)

6
```

The Fix is Easy!

```
def sum_with_extra_v2(e1, e2, things=None):
    if things is None:
        things = []
    things.append(e1)
    things.append(e2)
    return sum(things)
```

Everything is Awesome!

```
sum_with_extra_v2(1, 2, [3, 4])
```

```
10
```

```
sum_with_extra_v2(1, 2)
```

```
3
```

```
sum_with_extra_v2(1, 2)
```

```
3
```

```
things = [1, 2]; sum_with_extra_v2(1, 2, things)
```

```
6
```

```
# Whoops!
sum_with_extra_v2(1, 2, things)
```

```
9
```

One Urgent Hot Fix Later...

We got it to work!

```
def sum_with_extra_v3(e1, e2, things=None):
    if things is None:
        things = []
    things = things.copy()
    things.append(e1)
    things.append(e2)
    return sum(things)
```

..Meanwhile, Without Mutation

Let's throw caution to the wind and live our best life.

```
def sum_with_extra_p_v1(e1, e2, things=v()):
    things = things.append(e1)
    things = things.append(e2)
    return sum(things)
```

We Don't Need v2

```
sum_with_extra_p_v1(1, 2)
```

```
3
```

```
sum_with_extra_p_v1(1, 2)
```

```
3
```

```
things = v(1, 2)
sum_with_extra_p_v1(1, 2, things)
```

```
6
```

```
sum_with_extra_p_v1(1, 2, things)
```

```
6
```

But Nested Data Structures Are a Drag?

How do you increase the hits on web_1?

```
stats = m(
    frontend=m(
        web_1=m(hits=53),
        web_2=m(hits=78)),
    backend=m(
        db1=m(queries=23),
        db2=m(queries=11)))

# This doesn't work:
# stats["frontend"]["web_1"]["hits"] += 1
```

Like This

```
new_stats = stats.transform(
    v("frontend", "web_1", "hits"),
    lambda x: x + 1)
pprint.pprint(pyrsistent.thaw(new_stats),
              width=50)

{'backend': {'db1': {'queries': 23},
             'db2': {'queries': 11}},
 'frontend': {'web_1': {'hits': 54},
              'web_2': {'hits': 78}}}
```

Conclusion

- Sharing good
- Mutation bad
- Share more
- Mutate less
- Be happy