



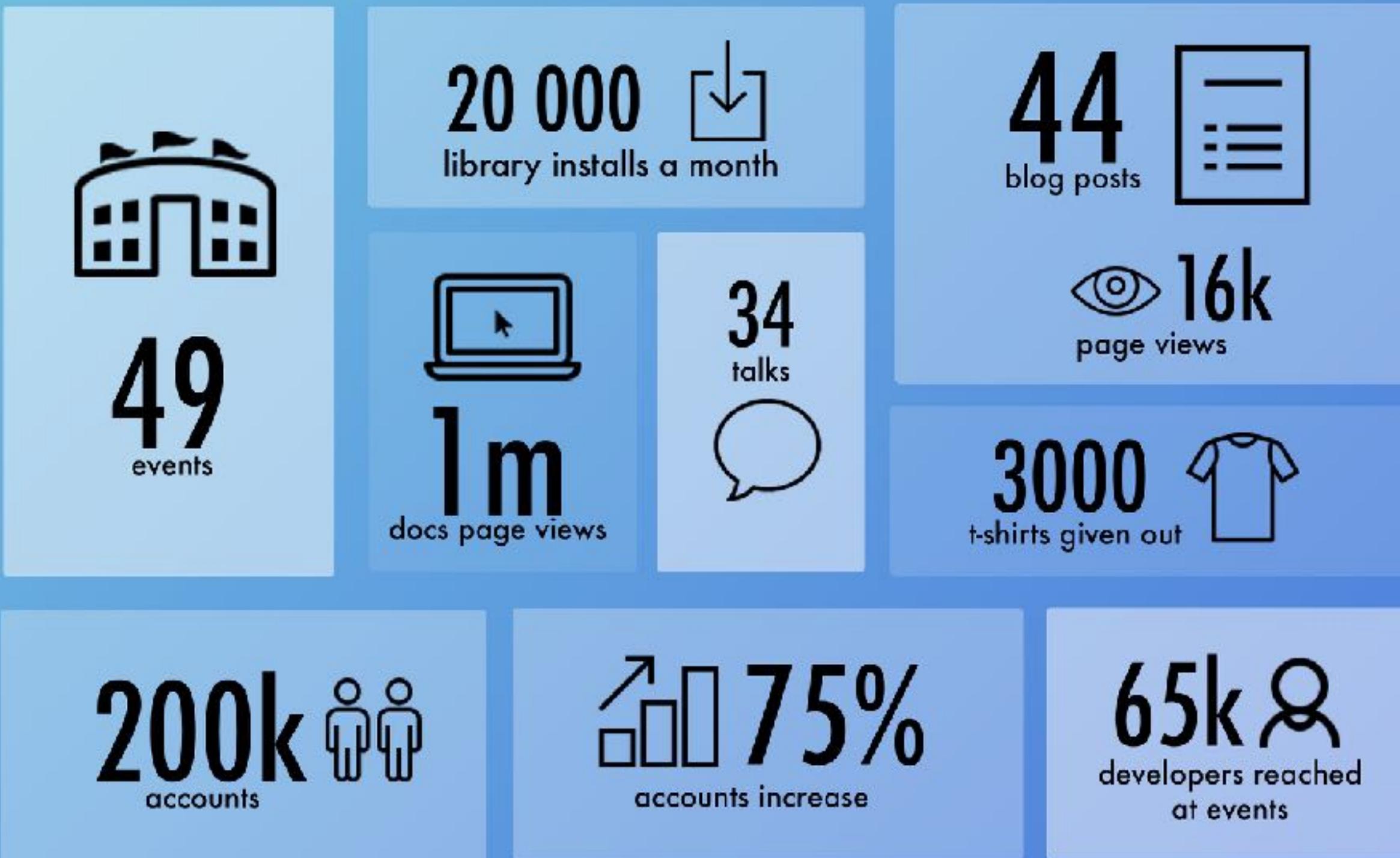
# Pythonic Refactoring

& protecting your users from change

# Mark Smith

- Developer Advocate @ Nexmo
- Python Edinburgh
- @judy2k
- Stupid Python
- Not a Viking

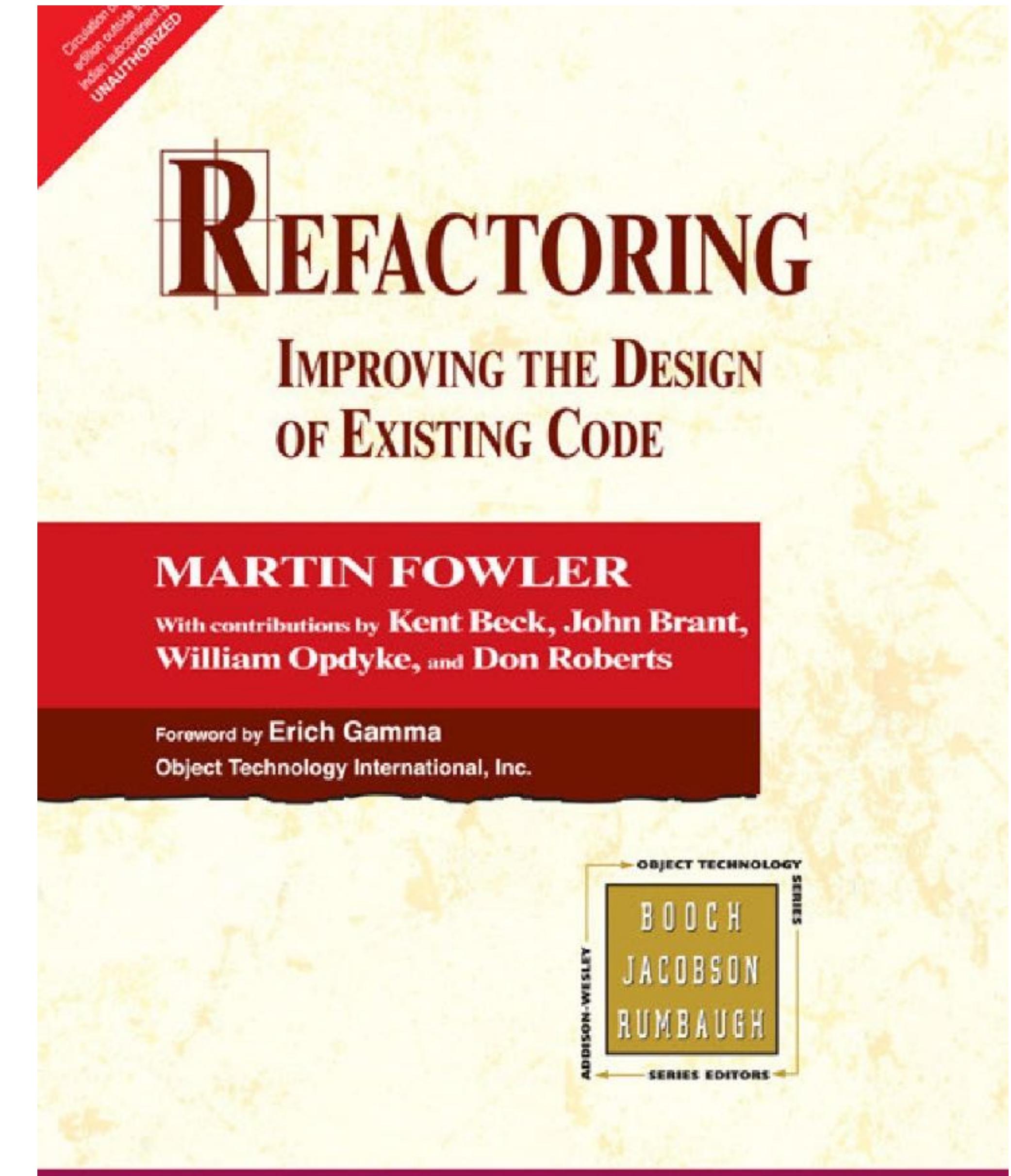


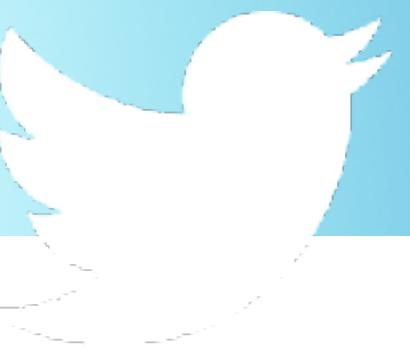


ONE YEAR OF DEVREL @ NEXMO

Me & Nexmo ❤

This is **not** actually a talk about refactoring

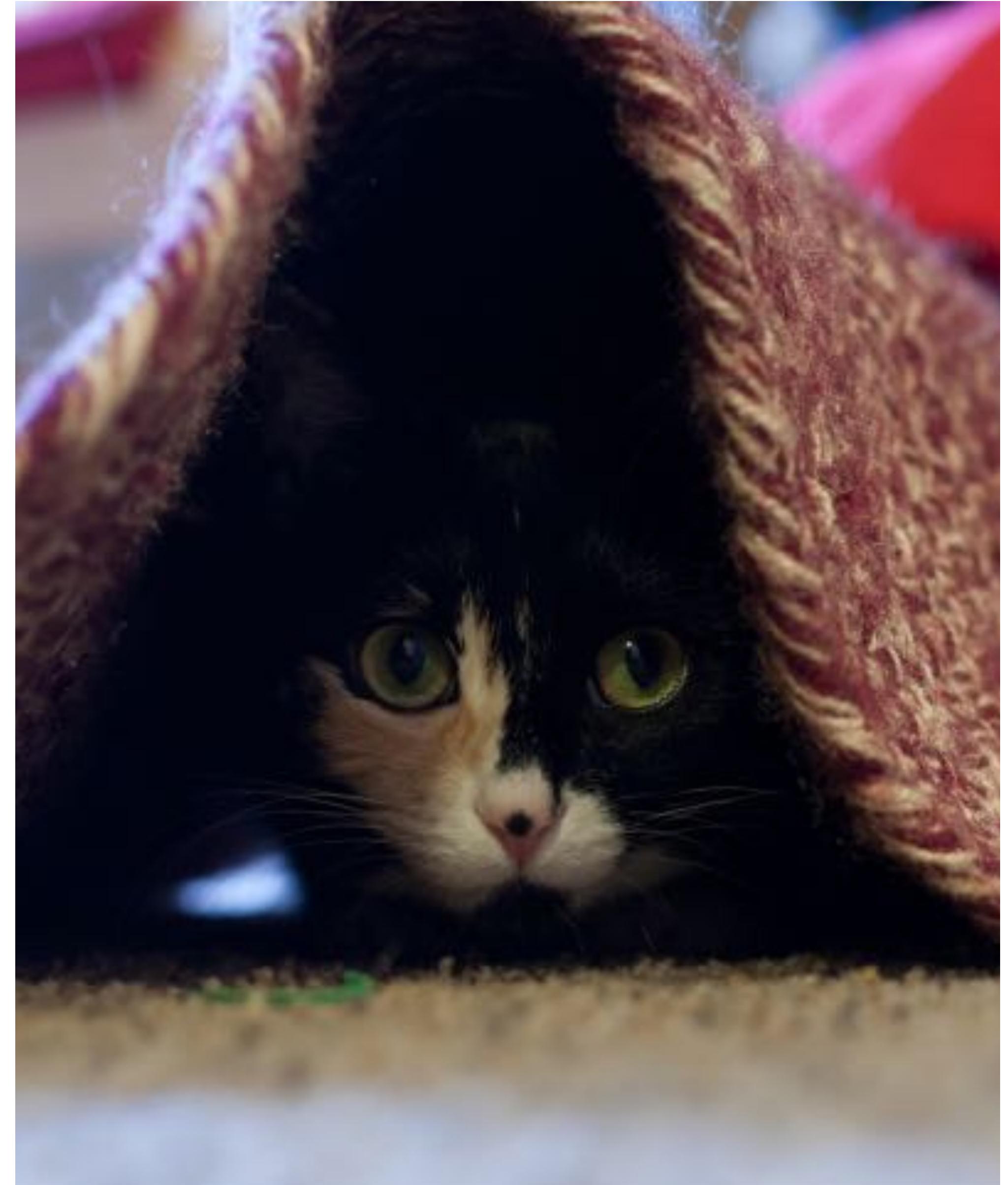


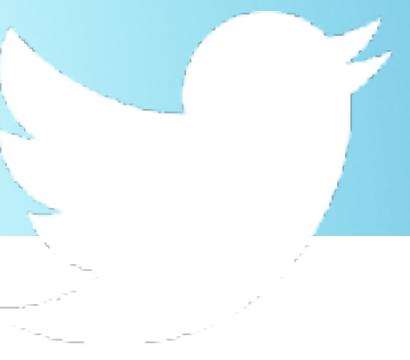


@judy2k

This is a talk about **change**

... and about **hiding**  
**change** from your  
users





@judy2k

# Ensure Your Interface is Stable



# What *is* your Interface?

**Structure**

**Bytecode**

**Classes**

**Exceptions**

**Modules**

**Functions  
& Methods**

**Global Variables**

**Parameters**



Interfaces should be  
as **strict** as possible



# Things that are like this

- REST/OpenAPI/Swagger (*2000*)
- J2EE (*1999*)
- SOAP (*1998*)
- CORBA (*1991*)



Python is not like this

```
>>> import requests
>>> dir(requests)
['ConnectTimeout', 'ConnectionError', 'DependencyWarning',
 ' FileModeWarning', 'HTTPError', 'NullHandler', 'PreparedRequest',
 'ReadTimeout', 'Request', 'RequestException', 'Response', 'Session',
 'Timeout', 'TooManyRedirects', 'URLRequired', '__author__', '__build__',
 '__builtins__', '__cached__', '__copyright__', '__doc__', '__file__',
 '__license__', '__loader__', '__name__', '__package__', '__path__',
 '__spec__', '__title__', '__version__', '_internal_utils', 'adapters',
 'api', 'auth', 'certs', 'codes', 'compat', 'cookies', 'delete',
 'exceptions', 'get', 'head', 'hooks', 'logging', 'models', 'options',
 'packages', 'patch', 'post', 'put', 'request', 'session', 'sessions',
 'status_codes', 'structures', 'utils', 'warnings']
```

```
import requests

def stupid_request(*args, **kwargs):
    print("Don't be stupid!")

requests.get = stupid_request
```

```
import requests
requests.get("https://www.reallycoolsite.com/")
```

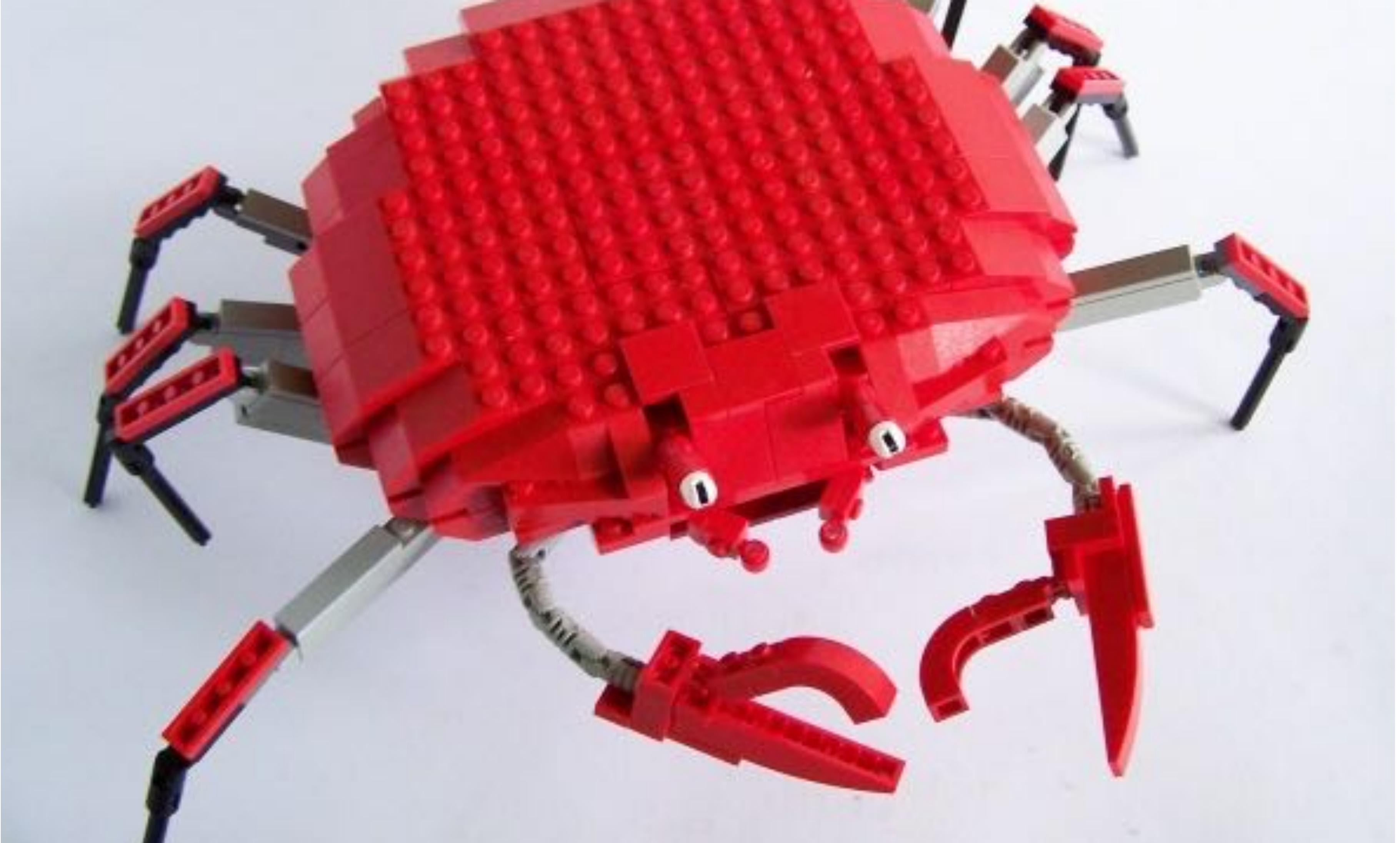
# Don't be stupid!



What we would like



# Python



Pythonic



# Making your interface knowable

# Underscore Prefix

```
class HammerTime(object):  
    def _dont_touch_this(self):  
        print("Stop!")
```

```
mc = HammerTime()  
mc._dont_touch_this()  
# Stop!
```



# Double-Underscores

```
class HammerTime(object):  
    def __cant_touch_this(self):  
        print("Stop!")
```

```
mc = HammerTime()  
mc._HammerTime__cant_touch_this()  
# Stop!
```



# Structure

Hide your private code in submodules



# Documentation

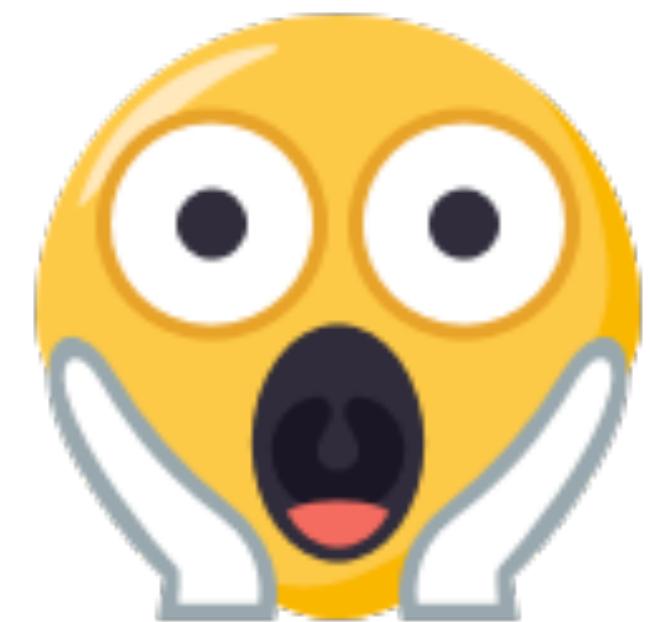


The Python ecosystem has  
**excellent** documentation



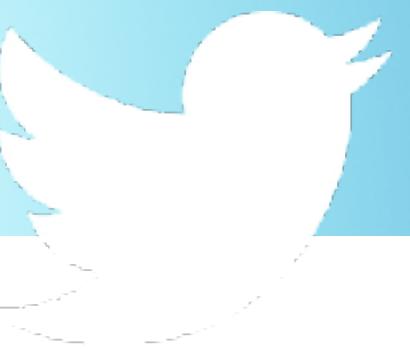
Your users will use the interface  
**that you document**

If you **don't** write documentation...  
...users will **read your code**





If you **don't** write documentation...  
...users will **guess** the interface  
... and they'll get it **wrong**



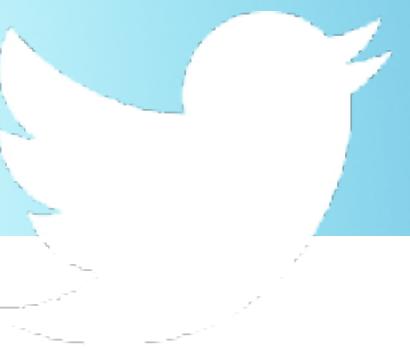
@judy2k

# Use Sphinx or MkDocs

Don't auto-generate your docs!



*Should you use type hints?*



**Q:** What is your interface?

**A:** Code that is documented



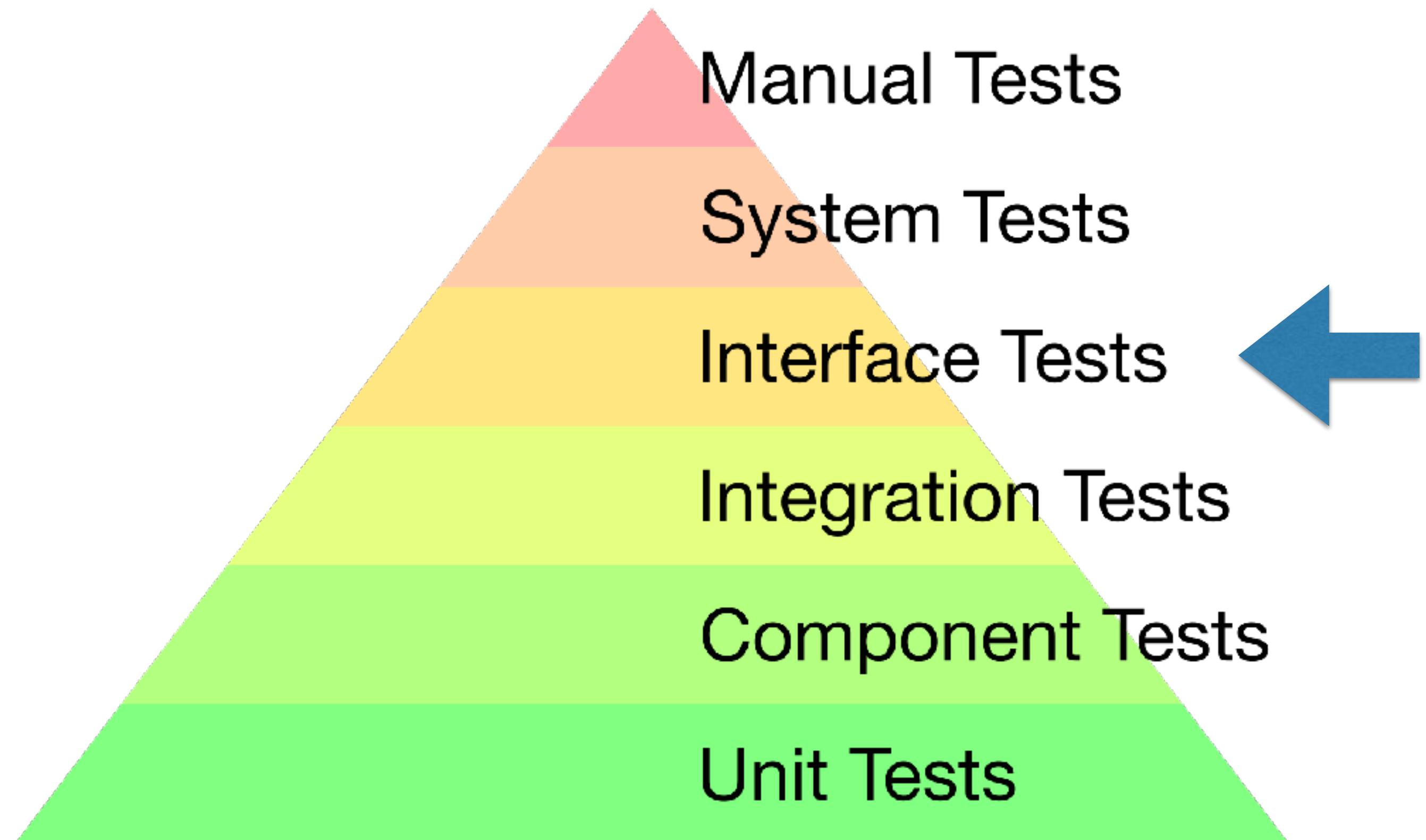
# Testing



"Code without tests  
is broken by design."

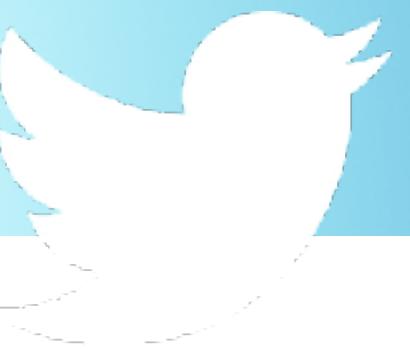
—Anonymous

# Test Types





# Versioning



# Semantic Versioning

Major.Minor.Patch

- **Patch** - No interface change
- **Minor** - Additions, backwards-compatible changes
- **Major** - Modifications of existing interface. Incompatible



# Release Notes

- Added
  - **Changed**
  - Deprecated
  - **Removed**
  - Fixed
  - Security
- keepachangelog.com**



It *is* okay to break compatibility  
**OCCASIONALLY!**



# Add & Deprecate

... instead of modifying



# Good Engineering Practices

protect your users from change



# Technical Solutions

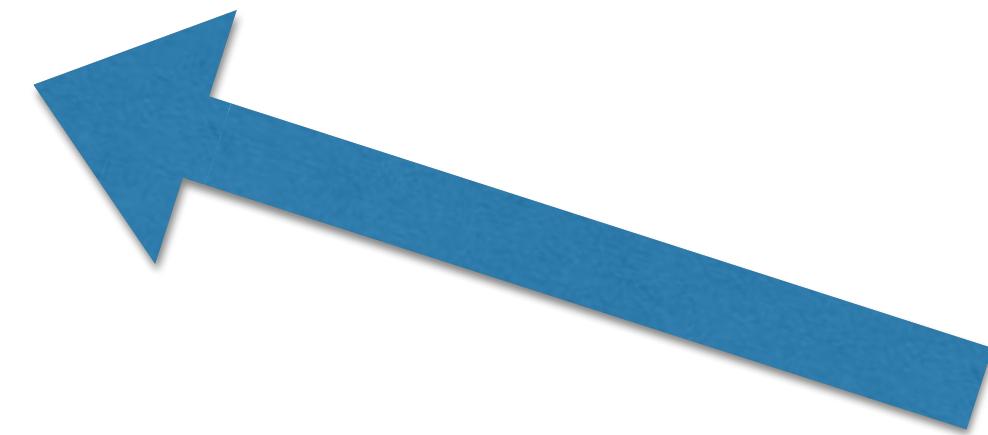
Finally, some *Python!*



# Object Attribute to Property

# Property: Example

```
class EncryptionParameters:  
    salt = 4
```



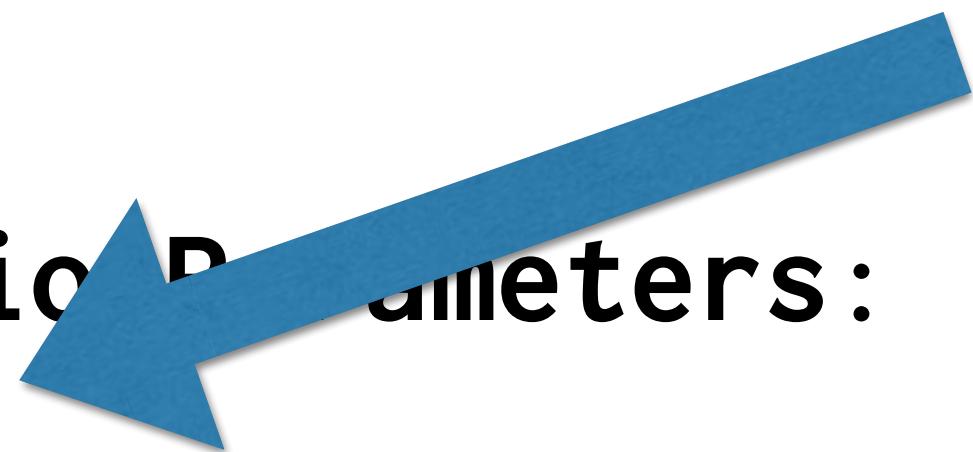


@property

# Property: Example

```
import time

class EncryptionParameters:
    @property
    def salt(self):
        return time.time()
```





# Singletons & factories

# Factory: Example

```
class Config(object):
    def __init__(self, path):
        # Read config in from path here
    pass
```

```
config_a = Config("a.txt")
config_a2 = Config("a.txt")
```



Implement new

# Factory: Example

```
class Config(object):
    cache = {}

    def __new__(cls, path):
        if path in cls.cache:
            return cls.cache[path]
        else:
            return cls.cache.setdefault(
                path,
                super().__new__(cls))
```



# Functions to Methods

Default Instance Refactoring



# Default Instance: Example

```
from support import Config

config = None

def load_config(path):
    global config
    config = Config(path)

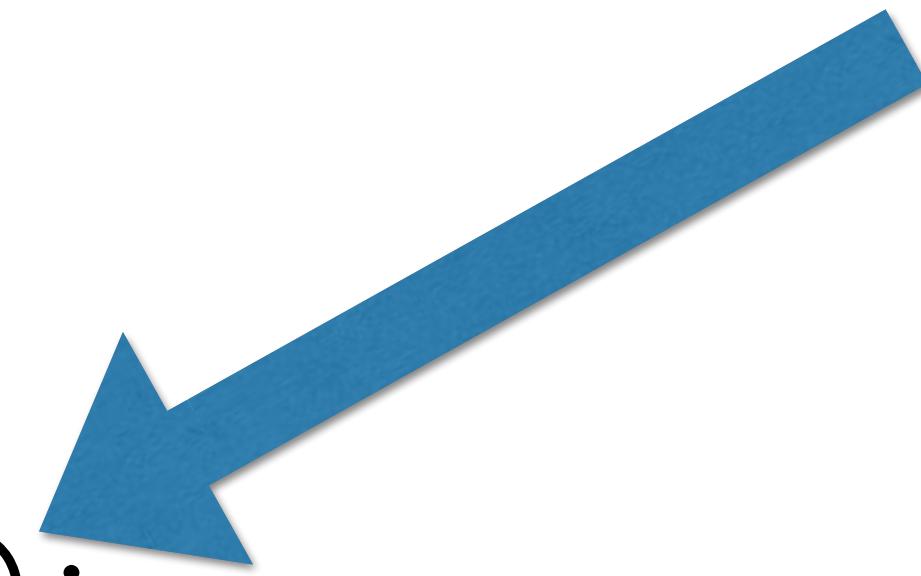
def do_the_thing():
    print(
        "Doing the thing with config from:",
        config.path)
```



# Default Instance: Example

```
class App(object):
    def do_the_thing(self):
        pass

default = App()
do_the_thing = default.do_the_thing
```





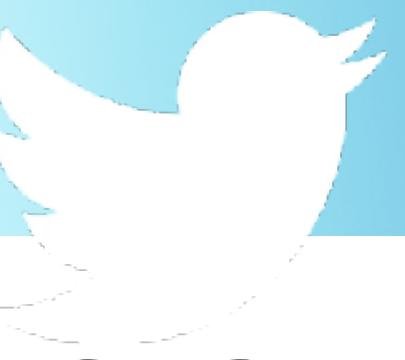
# Module to Object



# Module Object: Example

```
# important.py
```

```
salt = 4
```



# Module Object: Mechanics

1. Create a class
2. Make each of your dynamic variables a **property**
3. Create an instance of your class
4. Replace the just-imported module with the instance

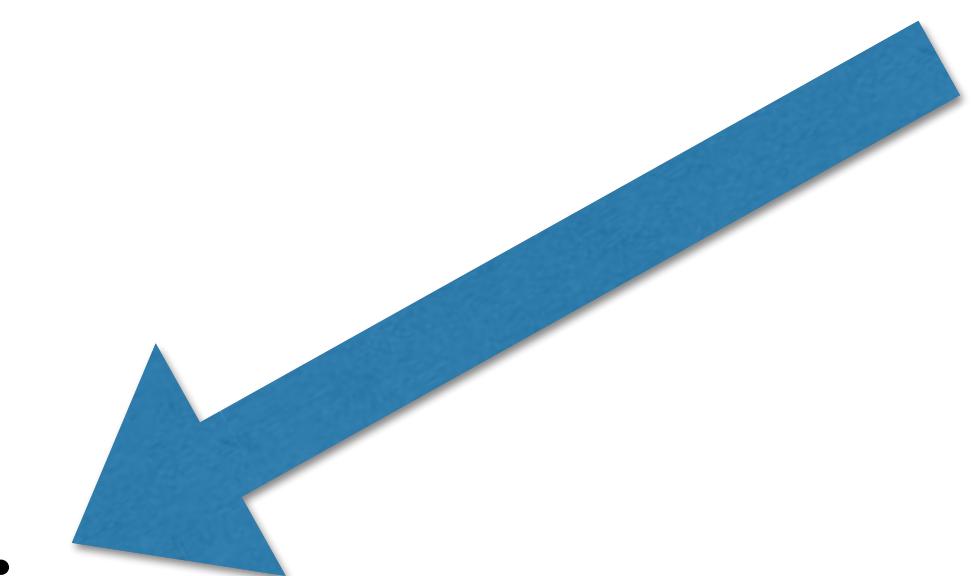


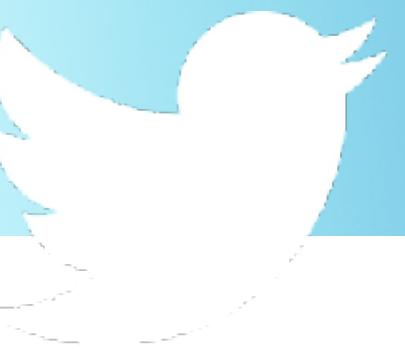
# Module Object: Example

```
# fake.py
import sys

class FakeModule(object):
    @property
    def salt(self):
        return time.time()

sys.modules['__name__'] = FakeModule()
```





# Further Techniques

- Class creation/initialisation (`__init__`, `__new__`)
- Objects that behave like iterators (`__iter__`, `__next__`)
- Making method calls look like attribute access (`__get__`)
- Classes that act like functions (`__call__`)
- Manually extract params from args & kwargs

<http://www.diveintopython3.net/special-method-names.html> for more



# Dangers



All clever tricks have  
**risks** attached



# Exceptions



# Type Assumptions

# Monkey Patching



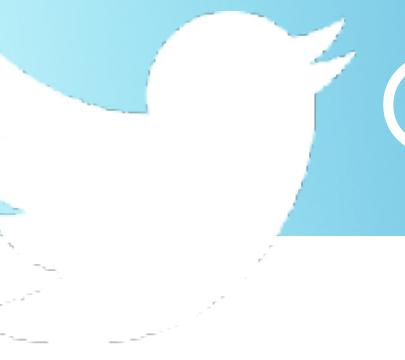


There is no **silver bullet**  
... you *will* break client code

# Strong & Stable

- Know your interface
- Document your interface
- Test your interface
- Have a deprecation plan
- Know some tricks for switching code





@judy2k

# Thank You!

**<http://bit.ly/pyconie17-refactoring>**

Email: **judy@judy.co.uk**

Twitter: **@judy2k**

Talk to me about Nexmo!

