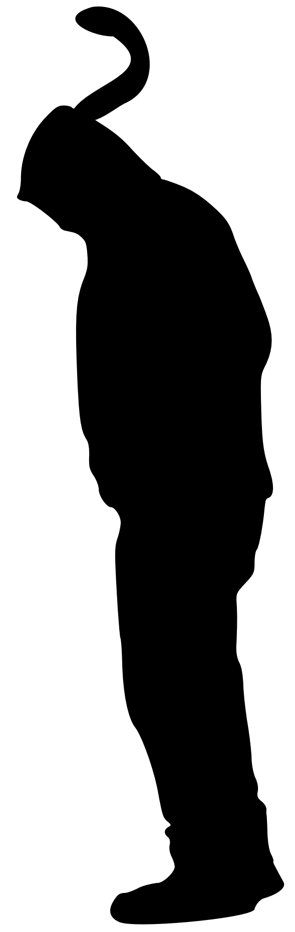
Basket of Random Python Snippets

Armin Ronacher

PyCon Ukraine 2011

Who Am I

- Armin Ronacher
- @mitsuhiko on Twitter/Github
- Part of the Pocoo Team
- Flask, Jinja2, Werkzeug, …



And What is This?

- Random but useful snippets divided by topic
- Give you ideas you might not have had.
- If you have questions: **SHOUT** and interrupt me :-)
- All slides are available for download:
- http://lucumr.pocoo.org/talks/

Iterators
and generators

Iterators are Great

- Tools to deal with them (itertools)
- But not everything speaks the iterator protocol
- How do we get stuff to speak iterator?

Everybody Knows Iterators

```
from itertools import tee, izip


def pairwise(iterable):
    a, b = tee(iterable)
    next(b, None)
    return izip(a, b)


>>> list(pairwise([1, 2, 3, 4]))
[(1, 2), (2, 3), (3, 4)]
```

Iter with Exception Sentinel

```
def iter_except(func, exc_class, first=None):
    try:
        if first is not None:
            yield first()
        while 1:
            yield func()
    except exc_class:
        pass
```

Practical Example

```
>>> elements = set([1, 2, 3, 4, 5])
>>> iterator = iter_except(elements.pop, KeyError)
>>> iterator.next()
1
>>> elements
set([2, 3, 4, 5])
>>> list(iterator)
[2, 3, 4, 5]
>>> elements
set([])
```

Iterator from Calls

```
from greenlet import greenlet
from functools import update_wrapper


def iter_from_func(f, args, kwargs):
    p = greenlet.getcurrent()
    g = greenlet(lambda: f(lambda x: p.switch((x,)), *args, **kwargs), p)
    while 1:
        rv = g.switch()
        if rv is None:
            return
        yield rv[0]

def funciter(f):
    return update_wrapper(lambda *a, **kw: iter_from_func(f, a, kw), f)
```

Example

```
@funciter
def my_enumerate(yield_func, iterable):
    idx = 0
    iterator = iter(iterable)
    while 1:
        yield_func((idx, iterator.next()))
        idx += 1


>>> list(my_enumerate('abc'))
[(0, 'a'), (1, 'b'), (2, 'c')]
```

Iterative Codecs

```
import codecs


def _iter_encode(iterable, func):
    for item in iterable:
        encoded_item = func(item)
        if encoded_item:
            yield encoded_item
    encoded_item = func('', True)
    if encoded_item:
        yield encoded_item

def iter_encode(iterable, codec, errors='strict'):
    cls = codecs.getincrementalencoder(codec)
    return _iter_encode(iterable, cls(errors).encode)

def iter_decode(iterable, codec, errors='strict'):
    cls = codecs.getincrementaldecoder(codec)
    return _iter_encode(iterable, cls(errors).decode)
```

Example Usage

```
>>> u'Foo \N{SNOWMAN}'.encode('utf-8')
'Foo \xe2\x98\x83'
>>> list(iter_decode(_, 'utf-8'))
[u'F', u'o', u'o', u' ', u'\u2603']
```

File Chunks

```
def iter_chunks(fp, chunk_size=4096):
    while 1:
        chunk = fp.read(chunk_size)
        if not chunk:
            break
        yield chunk
```

Lines From Chunks

```
def make_line_iter(chunk_iter):
    buffer = []
    while 1:
        if len(buffer) > 1:
            yield buffer.pop()
            continue
        chunks = chunk_iter.next().splitlines(True)
        chunks.reverse()
        first_chunk = buffer and buffer[0] or ''
        if chunks:
            if first_chunk.endswith('\n') or first_chunk.endswith('\r'):
                yield first_chunk
                first_chunk = ''
            first_chunk += chunks.pop()
        if not first_chunk:
            return
        buffer = chunks
        yield first_chunk
```

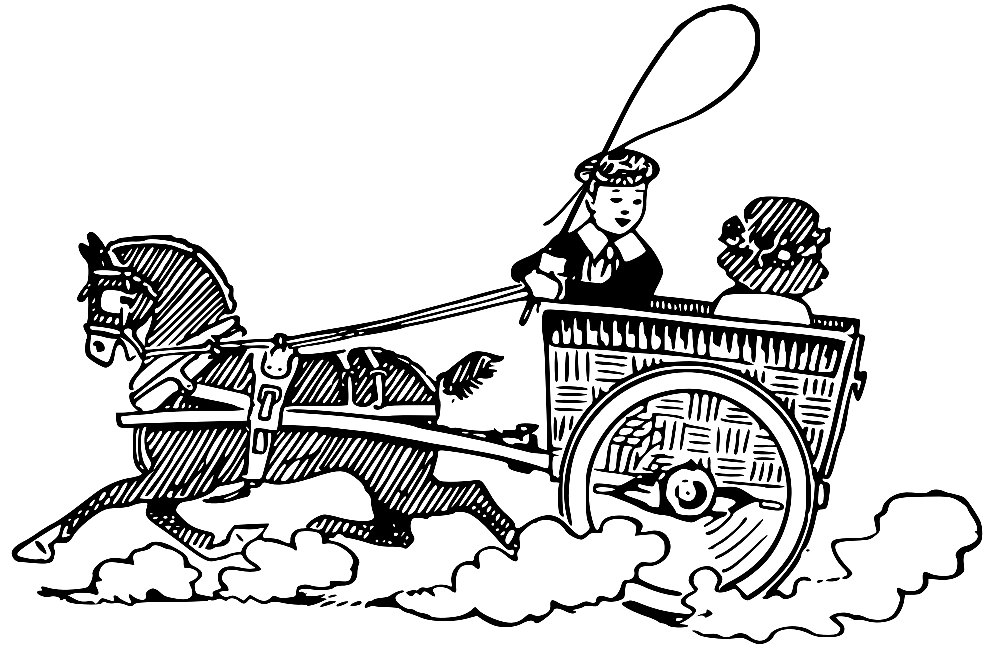
All together now

```
class Response(object):
    ...

    def iter_contents(self, chunk_size=4096):
        chunks = iter_chunks(self.fp, chunk_size=chunk_size)
        if self.transfer_encoding:
            chunks = iter_decode(chunks, self.transfer_encoding)
        if self.content_encoding:
            chunks = iter_decode(chunks, self.content_encoding)
        return chunks

    def iter_lines(self, chunk_size=4096):
        return make_line_iter(self.iter_contents(chunk_size))

    def get_contents(self):
        return ''.join(self.iter_contents())
```

Generator Send

- Don't use it.
- Close to impossible to forward in 2.x (would require yield from)
- If you think you need it, use greenlets instead.

Decorators



Decorators

- Decorators, decorator factories
- on functions, methods and classes
- Source of anger and frustration but soooo neat :-)

Decorators 101

```
@EXPR
def add(a, b):
    return a + b

-->

def add(a, b):
    return a + b
add = EXPR(add)
```

As such …

```
@EXPR(ARG)
def add(a, b):
    return a + b

-->

def add(a, b):
    return a + b
add = EXPR(ARG)(add)
```

Good Decorators

```
def register_a_function(func):
    a_collection.add(func)
    return func
```

Okay Decorators

```
from functools import update_wrapper


def change_function(func):
    def new_function(*args, **kwargs):
        do_something_with(args, kwargs)
        return func(*args, **kwargs)
    return update_wrapper(new_function, func)
```

Bad Decorators

```python
def change_function(func):
    def new_function(*args, **kwargs):
        do_something_with(args, kwargs)
        return func(*args, **kwargs)
    return new_function
```

Method Decorators

Do ***not*** *magically make decorators work on functions and methods. It seems to work until the point where you chain them.*

Better: have a decorator that makes function to method decorators

Make Method Decorator

```
class MethodDecoratorDescriptor(object):

    def __init__(self, func, decorator):
        self.func = func
        self.decorator = decorator

    def __get__(self, obj, type=None):
        return self.decorator(self.func.__get__(obj, type))


def method_decorator(decorator):
    def decorate(f):
        return MethodDecoratorDescriptor(f, decorator)
    return decorate
```

Cached Instance-only

```
class MethodDecoratorDescriptor(object):

    def __init__(self, func, decorator):
        self.func = func
        self.decorator = decorator

    def __get__(self, obj, type=None):
        if obj is None:
            return self
        rv = obj.__dict__.get(self.func.__name__)
        if rv is None:
            rv = self.decorator(self.func.__get__(obj, type))
            obj.__dict__[self.func.__name__] = rv
        return rv
```

Example

```
from functools import update_wrapper
from framework import View, redirect, url_for


def login_required(f):
    def decorated_function(request, *args, **kwargs):
        if request.user is None:
            return redirect(url_for('login', next=request.url))
        return f(request, *args, **kwargs)
    return update_wrapper(decorated_function, f)


class MyClassBasedView(View):

    @method_decorator(login_required)
    def get(self):
        ...
```

Descriptors

What are Descriptors?

- __get__
- __set__
- __delete__
- Common descriptors: functions, properties

Basic Descriptor Lookup

```
>>> class Foo(object):
...  def my_function(self):
...   pass
...
>>> Foo.my_function
<unbound method Foo.my_function>
>>> Foo.__dict__['my_function']
<function my_function at 0x1002e1410>
>>> Foo.__dict__['my_function'].__get__(None, Foo)
<unbound method Foo.my_function>
>>>
>>> Foo().my_function
<bound method Foo.my_function of <__main__.Foo object at 0x1002e2710>>
>>> Foo.__dict__['my_function'].__get__(Foo(), Foo)
<bound method Foo.my_function of <__main__.Foo object at 0x1002e2750>>
```

Non Data Descriptors

```
>>> class Foo(object):
...  def foo(self):
...   pass
...
>>> hasattr(Foo.foo, '__get__')
True
>>> hasattr(Foo.foo, '__set__')
False
>>> hasattr(Foo.foo, '__delete__')
False
```

Cached Properties

```
missing = object()


class cached_property(object):

    def __init__(self, func):
        self.func = func
        self.__name__ = func.__name__
        self.__doc__ = func.__doc__
        self.__module__ = func.__module__

    def __get__(self, obj, type=None):
        if obj is None:
            return self
        value = obj.__dict__.get(self.__name__, missing)
        if value is missing:
            value = self.func(obj)
            obj.__dict__[self.__name__] = value
        return value
```

New-Style Properties

```
class Foo(object):

    @property
    def username(self):
        """Docstring"""
        return self._username

    @username.setter
    def username(self, value):
        self._username = value
```

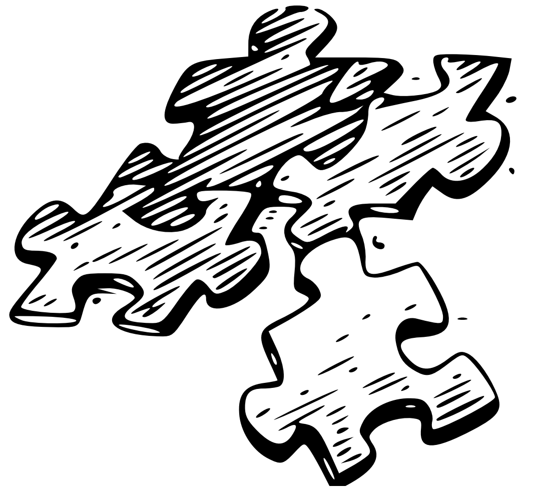
Still My Preferred Way

```
class Foo(object):

    def _get_username(self):
        """Docstring"""
        return self._username

    def _set_username(self, value):
        self._username = value

    username = property(_get_username, _set_username)
    del _get_username, _set_username
```

Alternatively

```
class Foo(object):

    @apply
    def username():
        """Docstring"""
        def getter(self):
            return self._username
        def setter(self, value):
            self._username = value
        return property(getter, setter, doc=username.__doc__)
```

ABCs and Mixins



Embrace MI

```
class Request(BaseRequest, AcceptMixin, ETagRequestMixin,
              UserAgentMixin, AuthorizationMixin,
              CommonRequestDescriptorsMixin):
    pass


class Response(BaseResponse, ETagResponseMixin,
               ResponseStreamMixin,
               CommonResponseDescriptorsMixin,
               WWWAuthenticateMixin):
    pass
```

ABCs embrace it

```
class Mapping(Sized, Iterable, Container):
    ...

class Set(Sized, Iterable, Container):
    ...

class Sequence(Sized, Iterable, Container):
    ...
```

Large MRO is not bad

```
class OrderedDict(MutableMapping)
 |  Dictionary that remembers insertion order
 |
 |
 |  Method resolution order:
 |      OrderedDict
 |      MutableMapping
 |      Mapping
 |      Sized
 |      Iterable
 |      Container
 |      object
```

ABCs not just inheritance

```
>>> from collections import Iterator
>>> class Foo(object):
...  def __iter__(self):
...   return self
...  def next(self):
...   return 42
...
>>> foo = Foo()
>>> isinstance(foo, Iterator)
True
>>> foo.next()
42
>>> foo.next()
42
```

But Also Inheritance

```
from collections import Mapping

class Headers(Mapping):

    def __init__(self, headers):
        self._headers = headers

    def __getitem__(self, key):
        ikey = key.lower()
        for key, value in self._headers:
            if key.lower() == ikey:
                return value
        raise KeyError(key)

    def __len__(self):
        return len(self._headers)

    def __iter__(self):
        return (key for key, value in self._headers)
```

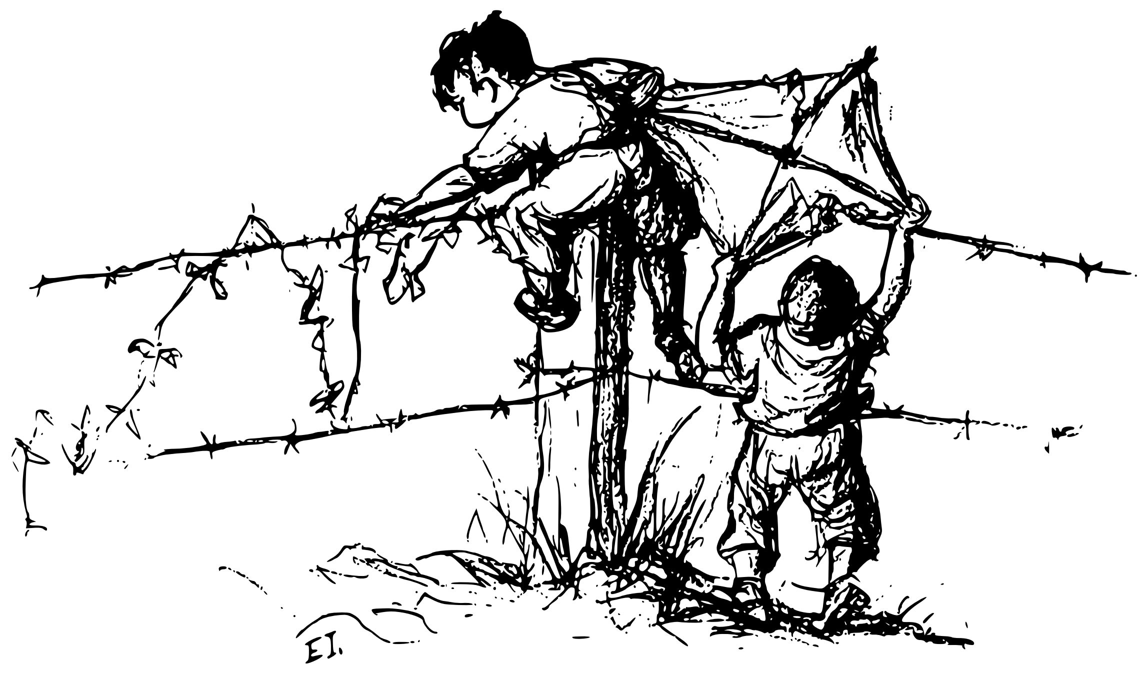
Example

```
>>> headers = Headers([('Content-Type', 'text/html')])
>>> headers['Content-type']
'text/html'
>>> headers.items()
[('Content-Type', 'text/html')]
>>> headers.values()
['text/html']
>>> list(headers)
['Content-Type']
```

New Rules

```
callable(x)          ->  isinstance(x, Callable)
tryexcept(hash(x))   ->  isinstance(x, Hashable)
tryexcept(iter(x))   ->  isinstance(x, Iterable)
tryexcept(len(x))    ->  isinstance(x, Sized)
tryexcept(hasattr(x, ‘__contains__’))
                     ->  isinstance(x, Container)

                     ->  isinstance(x, Mapping)
                         isinstance(x, Set)
                         isinstance(x, Sequence)
                         isinstance(x, MutableMapping)
                         isinstance(x, MutableSet)
                         isinstance(x, MutableSequence)
```

With Blocks



Overview

- They are not Ruby Blocks
- They can execute things before and after a block
- They do not introduce a new scope
- They can control what happens with exceptions that happen in the block

Assert Raises

```
class MyTestCase(TestCase):
    def assert_raises(self, exc_type):
        return _ExceptionCatcher(self, exc_type)


class _ExceptionCatcher(object):
    def __init__(self, test_case, exc_type):
        self.test_case = test_case
        self.exc_type = exc_type
    def __enter__(self):
        return self
    def __exit__(self, exc_type, exc_value, tb):
        exception_name = self.exc_type.__name__
        if exc_type is None:
            self.test_case.fail('Expected exception of type %r' %
                                exception_name)
        elif not issubclass(exc_type, self.exc_type):
            raise exc_type, exc_value, tb
        return True
```

Example

```
class DictTestCase(MyTestCase):

    def test_empty_dict_raises_errors(self):
        d = {}
        with self.assert_raises(KeyError):
            d[42]
```

Inspiration: OpenGL etc.

```
glPushMatrix()
glRotate3f(45.0, 1, 0, 0)
glScalef(0.5, 0.5, 0.5)
glBindTexture(texture_id)
draw_my_object()
glBindTexture(0)
glPopMatrix()
```

```
with Matrix(), \
     Rotation(45.0, 1, 0, 0), \
     Scale(0.5, 0.5, 0.5), \
     texture:
    draw_my_object()
```

Inspiration: Flask

```
from flask import request


with app.test_request_context('http://localhost/'):
    # everything here has access to a fake test request context
    # it's bound to the current thread/greenlet etc.
    assert_equal(request.url, 'http://localhost/')
    ...
```

Design APIs around it

```
from requests import session

with session() as sess:
    resp = sess.request('http://www.example.com/')
    ...
```

Small Things

String Formatting

```
>>> 'Hello {0}!'.format('World')
'Hello World!'

>>> 'Hello {0} {1}!'.format('Mr', 'World')
'Hello Mr World!'

>>> 'Hello {1}, {0}!'.format('Mr', 'World')
'Hello World, Mr!'

>>> 'Hello {name}!'.format(name='World')
'Hello World!'
```

But Better

```
>>> from datetime import datetime
>>> 'It\'s {0:%H:%M}'.format(datetime.today())
"It's 09:22"

>>> from urlparse import urlparse
>>> url = urlparse('http://pocoo.org/')
>>> '{0.netloc} [{0.scheme}]'.format(url)
'pocoo.org [http]'
```

Abuse Itertools

```
from itertools import izip, repeat


def batch(iterable, n):
    return izip(*repeat(iter(iterable), n))
```

How does it work?

```
>>> def debug(*args):
...   print args
...
>>> debug(*repeat(iter([1, 2, 3, 4]), 2))
(<listiterator object at 0x100491e50>,
 <listiterator object at 0x100491e50>)


>>> iterator = iter([1, 2, 3, 4])
>>> zip(iterator, iterator)
[(1, 2), (3, 4)]
```

Catch all the Things

```
BAD:

    try:
        ...
    except:
        ...


GOOD:

    try:
        ...
    except Exception:
        ...
```

Reraise all the Things

```
BAD:

    try:
        ...
    except Exception, e:
        ...
        raise e

GOOD:

    try:
        ...
    except:
        ...
        raise
```

Fight the GC

```python
from threading import Lock
from contextlib import contextmanager

lock = Lock()

@contextmanager
def disabled_gc():
    gc.collect()
    obj_count = len(gc.get_objects())
    was_enabled = gc.isenabled()
    gc.disable()
    try:
        with lock:
            yield
        if obj_count != len(gc.get_objects()):
            raise AssertionError('Section has cycles, requires GC')
    finally:
        if was_enabled:
            gc.enable()
```

Example Usage

```
def application(environ, start_response):
    with disabled_gc():
        return real_wsgi_app(environ, start_response)
```

Libraries

You didn't know you would need

Blinker

```
>>> from blinker import Namespace
>>> signals = Namespace()
>>> siga = signals.signal('siga')
>>> def connected(sender, **kwargs):
...  print sender, kwargs
...  return 'return value'
...
>>> siga.connect(connected)
<function connected at 0x100424320>
>>> siga.send('sender', foo=42)
'sender' {'foo': 42}
[(<function connected at 0x100424320>, 'return value')]
```

It's Dangerous

```
>>> from itsdangerous import URLSafeSerializer
>>> s = URLSafeSerializer('secret-key')
>>> s.dumps([1, 2])
'WzEsMl0.9HVDLVKBQFb0jaw0IeBzjCI7nZA'
>>> s.loads('WzEsMl0.9HVDLVKBQFb0jaw0IeBzjCI7nZA')
[1, 2]

>>> s.loads('WzEsMl0. 9HVDLVKBQFb0jaw0IeBzjCI7nZB')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
itsdangerous.BadSignature:
  Signature "9HVDLVKBQFb0jaw0IeBzjCI7nZB" does not match
```

MarkupSafe

```
>>> from markupsafe import Markup
>>> Markup.escape('<hacker>')
Markup(u'&lt;hacker&gt;')
>>> Markup('<i>%s</i>') % '<script>alert("hacker");</script>'
Markup(u'<i>&lt;script&gt;alert(&#34;hacker&#34;);&lt;/script&gt;</i>')
```

Requests

```
import re
import requests


def _get_params(resp):
    return dict(re.findall(r'<input.*?name="(.*?)".*?value="(.*?)"',
                           resp.content))

def xbl_auth(sess, email, password):
    resp = sess.get('http://live.xbox.com/en-US/friendcenter')
    action = re.findall(r'srf_uPost=\'(.*?)\'', resp.content)[0]
    params = dict(_get_params(resp), login=email, passwd=password)
    params = _get_params(sess.post(action, data=params))
    sess.post('http://live.xbox.com/en-US/friendcenter/Friends', data=params)


with requests.session() as sess:
    xbl_auth(sess, 'your-email@example.com', 'the-password')
    resp = sess.get('http://live.xbox.com/en-US/...')
```

PBKDF2

```
import hmac
from hashlib import sha1
from math import ceil
from struct import pack


def pbkdf2(data, salt, iterations=1000, keylen=24, hashfunc=sha1):
    _pseudorandom = lambda x: hmac.new(data, x, hashfunc).digest()
    def _produce(block):
        rv = u = _pseudorandom(salt + pack('>i', block))
        for i in xrange(iterations - 1):
            u = _pseudorandom(u)
            rv = ''.join([chr(ord(a) ^ ord(b)) for a, b in zip(rv, u)])
        return rv
    blocks = int(ceil(float(keylen) / hashfunc().digest_size))
    return ''.join(map(_produce, xrange(1, blocks + 1)))[:keylen]
```

Augmenting Logging

```
from flask import request


class RequestInfoFilter(Filter):

    def filter(self, record):
        if not request:
            record.request_remote_addr = ''
            record.request_url = ''
            record.request_method = ''
        else:
            record.request_remote_addr = request.remote_addr
            record.request_url = request.url
            record.request_method = request.method
        return True
```

Fighting The State

Rules of Thumb

- Avoid all avoidable global state
- If you need it, at least make it local to an implicit context
- Avoid unnecessary local state

Things To Avoid

- os.chdir() — use absolute paths instead
- socket.setdefaulttimeout() — use per socket timeouts
- “settings” modules

Reasons

- Global state breaks threading
- Global state makes unittesting harder than it has to be
- Global state can change at any point anywhere

Avoidable Global State

```
from yourapplication import settings, some_helper_using_settings

settings.MY_CONFIG_KEY = 'my config value'


def some_function():
    ...
    some_helper_using_settings()
```

Solution A:

```python
from yourapplication import global_settings, some_helper_using_settings


def some_function():
    settings = global_settings.copy()
    settings.MY_CONFIG_KEY = 'my config value'
    some_helper_using_settings(settings=settings)
```

Solution B:

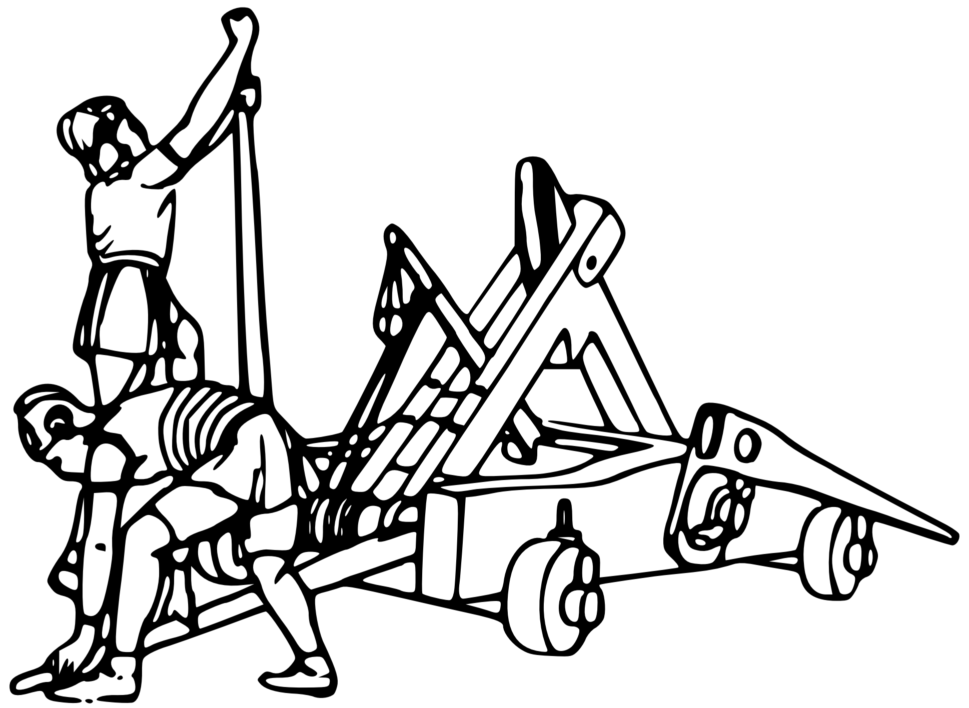
```python
from yourapplication import Settings


def some_function():
    something = Something(something='my config value')
    something.some_helper()
```

Solution C:

```python
from yourapplication import Settings


def some_function():
    with Settings(something='my config value'):
        some_helper_using_settings()
```

And Remember:

Hackernews and Reddit are Evil



!Q&A?

lucumr.pocoo.org/talks/