PyCon 2011

Seth House <seth@eseth.com>

Utah Python User Group

2011-03-17

PyCon 2011

2011-03-18

Inspiration

Hands on Intermediate Python

- Matt Harrison
- http://panela.blog-city.com/

Useful Namespaces: Context Managers and Decorators

- Jack Diederich
- http://blip.tv/file/4881235

Inspiration
Hands on Intermediate Python
Matt Harrison
http://panela.blog-city.com/
Useful Namespaces: Context Managers and Decorators
Jack Diederich
http://blip.tv/file/4881235

Decorators

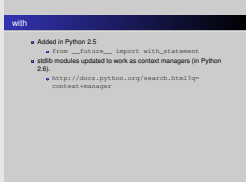
```
import functools

def my_decorator(func):
    @functools.wraps(func)
    def wrapper(*args, **kwargs):
        return func(*args, **kwargs)
    return wrapper
```


Decorators
import functools
def my_decorator(func):
@functools.wraps(func)
def wrapper(*args, **kwargs):
return func(*args, **kwargs)
return wrapper

with

- Added in Python 2.5
  - `from __future__ import with_statement`
- stdlib modules updated to work as context managers (in Python 2.6).
  - `http://docs.python.org/search.html?q=context+manager`



tempfile.NamedTemporaryFile

```
with tempfile.NamedTemporaryFile() as myfile:
    myfile.readlines()
```


tempfile.NamedTemporaryFile
with tempfile.NamedTemporaryFile() as myfile:
myfile.readlines()

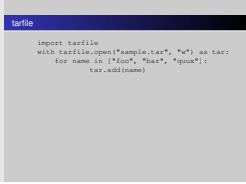
sqlite3

```
import sqlite3
with sqlite3.connect(":memory:") as con:
    con.execute("create table ...")
```


sqlite3
import sqlite3
with sqlite3.connect(":memory:") as con:
con.execute("create table ...")

tarfile

```
import tarfile
with tarfile.open("sample.tar", "w") as tar:
    for name in ["foo", "bar", "quux"]:
            tar.add(name)
```

contextlib

```
import contextlib

@contextlib.contextmanager
def tag(name):
    print "<%s>" % name
    yield
    print "</%s>" % name
```


contextlib

```
import contextlib

@contextlib.contextmanager
def tag(name):
    print "<%s>" % name
    yield
    print "</%s>" % name
```

contextlib

```
from contextlib import closing
import urllib

with closing(urllib.urlopen(
        'http://www.python.org')) as page:
    for line in page:
        print line
```


contextlib

```
from contextlib import closing
import urllib

with closing(urllib.urlopen(
        'http://www.python.org')) as page:
    for line in page:
        print line
```

Multiple contexts

```
with A() as a, B() as b:
    # stuff
```


Multiple contexts
with A() as a, B() as b:
stuff

Writing your own

```
class MyManager(object):
    def __init__(self, *args, **kwargs):
        pass

    def __enter__(self):
        return self  # set to ``as`` target

    def __exit__(self, exc_type, exc_value,
            traceback):
        return True  # suppress all exceptions
```


Writing your own
class MyManager(object):
def __init__(self, *args, **kwargs):
pass
def __enter__(self):
return self # set to ``as`` target
def __exit__(self, exc_type, exc_value,
traceback):
return True # suppress all exceptions

Monkeypatching your mom would approve of

```
with patched_object("smtplib", "SMTP",
        MockSMTP):
    s = smtplib.SMTP()
```


Monkeypatching your mom would approve of
with patched_object("smtplib", "SMTP",
MockSMTP):
s = smtplib.SMTP()

Explicit and Pythonic monkeypatching

Creating a context manager and a decorator

- `__call__`
- `__enter__`
- `__exit__`

http://code.activestate.com/recipes/577273-decorator-and-context-manager-from-a-single-api/

collections.counter

```
c = Counter()
c['test'] += 1
c.most_common()
c.elements()
```


collections.counter
c = Counter()
c['test'] += 1
c.most_common()
c.elements()

- Added in 2.7. Backport for 2.5: `http://code.activestate.com/recipes/576611/`
- Multisets (set operations)

collections.namedtuple

Old:

```
# (name, age, num. kids)
employee = ('frank', 36, 3)
```

New:

```
EmployeeStats = namedtuple('EmployeeStats',
        'name, age, num_kids')
employee = EmployeeStats('frank', 36, 3)
```


collections.namedtuple
Old:
# (name, age, num. kids)
employee = ('frank', 36, 3)
New:
EmployeeStats = namedtuple('EmployeeStats',
'name, age, num_kids')
employee = EmployeeStats('frank', 36, 3)

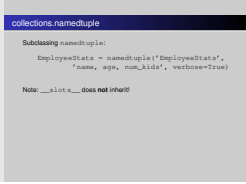
- Makes code self-documenting.
- space cost vs. tuples is zero.
- Official docs have a good example of reading csv values into a namedtuple.
- `_as_dict()`
- `_replace()`
- Underscore is to preverse the namespace for user stuffs.

collections.namedtuple

Subclassing `namedtuple`:

```
EmployeeStats = namedtuple('EmployeeStats',
        'name, age, num_kids', verbose=True)
```

Note: `__slots__` does **not** inherit!



functools.lru_cache

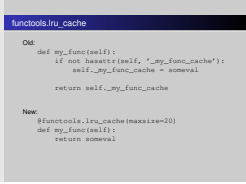
Old:

```
def my_func(self):
    if not hasattr(self, '_my_func_cache'):
        self._my_func_cache = someval

    return self._my_func_cache
```

New:

```
@functools.lru_cache(maxsize=20)
def my_func(self):
    return someval
```

- New in Python 3.2
- Backport by Raymond Hettinger:
  `http://code.activestate.com/recipes/498245/`