Program layout best practices

Seth House <seth@eseth.com>

Utah Django User Group

2011-06-23

Packages and modules

Module skeleton

```
#!/usr/bin/env python
# coding: utf-8
"""Module docstring"""
__author__ = 'Mr. Me <me@example.net>'
__version__ = '1.2.3'
if __name__=='__main__':
    main()
```

Module or class?

Namespacing.

Be mindful of the interface

```
>>> from UserDict import UserDict
>>> from ConfigParser import ConfigParser
>>> from pprint import pprint
>>> from fabric.api import *
```

Executable code

Avoid putting executable statements directly in a module.

Avoid relative imports

```
pkg/
     __init__.py
     moduleA.py
     moduleB.py

# in moduleB
import moduleA
```

Executing modules as scripts

PEP 338:

```
python -m SimpleHTTPServer
```

Executing modules as scripts

```
python -m smtpd -n -c \
        DebuggingServer localhost:1025

>>> import smtplib
>>> mailserver = smtplib.SMTP('localhost:1025')
>>> mailserver.sendmail(
        'me@example.com',
        'you@example.net',
        'O HAI!')
```

Executing modules as scripts

- `python -m unittest mymodule`

Executing modules as scripts

- `python -m unittest mymodule`
- `python -m pdb mymodule`

Executing modules as scripts

- `python -m unittest mymodule`
- `python -m pdb mymodule`
- `python -m timeit -s "range(1000)"`

Executing modules as scripts

```
runpy.run_module()
```

Executing modules as scripts

```
devel/
pkg/
    __init__.py
    moduleA.py
    moduleB.py
    test/
        __init__.py
        test_A.py
        test_B.py
```

Executing modules as scripts

```
python -m pkg.test.test_A
python -m pkg.test.test_B
```

Django

Avoid manage.py

```
django-admin.py
```

Avoid manage.py

```
DJANGO_SETTINGS_MODULE=myproject.settings\
        django-admin.py
```

Avoid local_settings.py

```
try:
    from local_settings import *
except ImportError, exp:
    pass
```

Avoid local_settings.py

```
from myproject.settings import *

INSTALLED_APPS += [
    'djangodebugtoolbar',
]

CACHES['default']['BACKEND'] = \
        'django.core.cache.backends.
            locmem.LocMemCache'
```

Keep sensitive data out of settings

```
import ConfigParser
secret = ConfigParser.ConfigParser()
secret.optionxform = lambda x: x.upper()
secret.read('/path/to/mysecrets.conf')
```

Keep sensitive data out of settings

```
SECRET_KEY = secret.get('MAIN', 'SECRET_KEY')
DATABASES = {
    'default': dict(secret.items('DB_DEFAULT')),
    'slave': dict(secret.items('DB_SLAVE')),
}
```

Keep sensitive data out of settings

```
[MAIN]
SECRET_KEY = SW$RFDEW$TR

[DB_DEFAULT]
ENGINE = django.db.backends.postgresql_psycopg2
NAME = mydbname
USER = mydbuser
```

Use managers

```
class MyModel(models.Model):
    @classmethod
    def something(cls):
        ...

    @classmethod
    def anotherthing(cls):
        ...

    @classmethod
    def somethingelse(cls):
        ...
```

Use managers

```
class MyModelManager(models.Manager):
    def girls(self):
        return self.filter(girls=True)

    def boys(self):
        return self.filter(boys=True)

class MyModel(models.Model):
    objects = MyModelManager()
```

Use managers

```
>>> from django.contrib.auth.models import User
>>> User.objects.create_superuser(...)
>>> User.objects.make_random_password()
```

Use managers

```
class MyModelManager(models.Manager):
    def increment(self):
        return self.update(
            counter=models.F('counter') + 1)
```

Use managers

```
class MyModelManager(models.Manager):
    def heavy_query(self):
        sql = """\
            SQL goes here...
            """ % (self.model._meta.db_table)

        cursor = connection.cursor()
        cursor.execute(sql)

        return dict(cursor.fetchall())
```

Use managers

```
class MyModelManager(models.Manager):
    def active(self):
        return self.filter(pub_date__lte=now())

    def complete(self):
        return self.filter(content__isnull=False

    def public_forms(self):
        return self.active().complete()
```

Use managers

```
from django.db import models
class MyModelQuerySets(models.query.QuerySet):
    pass
class MyModelManager(models.Manager):
    def get_query_set(self):
        return MyModelQuerySets(self.model)
    def __getattr__(self, attr, *args):
        try:
            return getattr(self.__class__,
                           attr, *args)
        except AttributeError:
            return getattr(self.get_query_set(),
                           attr, *args)
```

Command-line apps

The hashbang

```
#!/usr/bin/env python
```

Future-proof your script

```
if __name__ == '__main__':
    import mymodule
    raise SystemExit(mymodule.main())

import some.modules

def main():
    # stuff
```

Interactive interpreter

```
import cmd
from code import InteractiveConsole
```