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

"""
=====================================
PEP 20 (The Zen of Python) by example
=====================================

Usage: %prog

:Author: Hunter Blanks, hblanks@artifex.org / hblanks@monetate.com
:Date: 2011-02-08 for PhillyPUG/philly.rb, revised 2011-02-10

Sources:

    - http://artifex.org/~hblanks/talks/2011/pep20_by_example.pdf
    - http://artifex.org/~hblanks/talks/2011/pep20_by_example.html
    - http://artifex.org/~hblanks/talks/2011/pep20_by_example.py.txt

Dependencies for PDF output:

    - Pygments 1.4
    - pdflatex & the usual mess of latex packages
"""

from __future__ import with_statement
import sys
```

```
################################ preface ###############################

"""
   "In his wisdom and in his Molisan poverty, Officer Ingravallo,
    who seemed to live on silence... , in his wisdom, he sometimes
    interrupted this silence and this sleep to enunciate some
    theoretical idea (a general idea, that is) on the affairs of men,
    and of women. At first sight, or rather, on first hearing, these
    seemed banalities. They weren't banalities. And so, those rapid
    declarations, which crackled on his lips like the sudden
    illumination of a sulphur match, were revived in the ears of people
    at a distance of hours, or of months, from their enunciation: as if
    after a mysterious period of incubation. 'That's right!' the person
    in question admitted, 'That's exactly what Ingravallo said to me.'"

        - Carlo Emilio Gadda, *That Awful Mess on the Via Merulana*
"""
```

```
################################# text #################################

"""
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
"""
```

```
################################## 1 #################################

"""
Give me a function that takes a list of numbers and returns only the
even ones, divided by two.
"""

#----------------------------------------------------------------------

halve_evens_only = lambda nums: map(lambda i: i/2, filter(lambda i: not i%2, nums))

#----------------------------------------------------------------------

def halve_evens_only(nums):
    return [i/2 for i in nums if not i % 2]

#----------------------------------------------------------------------

print 'Beautiful is better than ugly.'
```

```
################################## 2 ###################################

"""
Load the cat, dog, and mouse models so we can edit instances of them.
"""

def load():
    from menagerie.cat.models import *
    from menagerie.dog.models import *
    from menagerie.mouse.models import *

#-----------------------------------------------------------------------

def load():
    from menagerie.models import cat as cat_models
    from menagerie.models import dog as dog_models
    from menagerie.models import mouse as mouse_models

#-----------------------------------------------------------------------

print 'Explicit is better than implicit.'
```

```python
################################### 3 ##################################

"""
Can you write out these measurements to disk?
"""

measurements = [
    {'weight': 392.3, 'color': 'purple', 'temperature': 33.4},
    {'weight': 34.0, 'color': 'green', 'temperature': -3.1},
    ]

#-----------------------------------------------------------------------

def store(measurements):
    import sqlalchemy
    import sqlalchemy.types as sqltypes

    db = sqlalchemy.create_engine('sqlite:///measurements.db')
    db.echo = False
    metadata = sqlalchemy.MetaData(db)
    table = sqlalchemy.Table('measurements', metadata,
        sqlalchemy.Column('id', sqltypes.Integer, primary_key=True),
        sqlalchemy.Column('weight', sqltypes.Float),
        sqlalchemy.Column('temperature', sqltypes.Float),
        sqlalchemy.Column('color', sqltypes.String(32)),
        )
    table.create(checkfirst=True)

    for measurement in measurements:
        i = table.insert()
        i.execute(**measurement)

#-----------------------------------------------------------------------

def store(measurements):
    import json
    with open('measurements.json', 'w') as f:
        f.write(json.dumps(measurements))

#-----------------------------------------------------------------------
```

```
print 'Simple is better than complex.'
```

```
################################### 4 ##################################

"""
Can you write out those same measurements to a MySQL DB? I think we're
gonna have some measurements with multiple colors next week, by the way.
"""

#-----------------------------------------------------------------------

def store(measurements):
    import sqlalchemy
    import sqlalchemy.types as sqltypes

    db = create_engine(
        'mysql://user:password@localhost/db?charset=utf8&use_unicode=1')
    db.echo = False
    metadata = sqlalchemy.MetaData(db)
    table = sqlalchemy.Table('measurements', metadata,
        sqlalchemy.Column('id', sqltypes.Integer, primary_key=True),
        sqlalchemy.Column('weight', sqltypes.Float),
        sqlalchemy.Column('temperature', sqltypes.Float),
        sqlalchemy.Column('color', sqltypes.String(32)),
        )
    table.create(checkfirst=True)

    for measurement in measurements:
        i = table.insert()
        i.execute(**measurement)

#-----------------------------------------------------------------------

def store(measurements):
    import MySQLdb
    db = MySQLdb.connect(user='user', passwd="password", host='localhost', db="db")

    c = db.cursor()
    c.execute("""
        CREATE TABLE IF NOT EXISTS measurements
          id int(11) NOT NULL auto_increment,
          weight float,
          temperature float,
          color varchar(32)
          PRIMARY KEY id
          ENGINE=InnoDB CHARSET=utf8
          """)

    insert_sql = (
        "INSERT INTO measurements (weight, temperature, color) "
        "VALUES (%s, %s, %s)")

    for measurement in measurements:
        c.execute(insert_sql,
            (measurement['weight'], measurement['temperature'], measurement['color'])
            )

#------------------------------------------------------------------------

print 'Complex is better than complicated.'
```

```
################################### 5 ##################################

"""Identify this animal. """

#-----------------------------------------------------------------------

def identify(animal):
    if animal.is_vertebrate():
        noise = animal.poke()
        if noise == 'moo':
            return 'cow'
        elif noise == 'woof':
            return 'dog'
    else:
        if animal.is_multicellular():
            return 'Bug!'
        else:
            if animal.is_fungus():
                return 'Yeast'
            else:
                return 'Amoeba'

#-----------------------------------------------------------------------

def identify(animal):
    if animal.is_vertebrate():
        return identify_vertebrate()
    else:
        return identify_invertebrate()

def identify_vertebrate(animal):
    noise = animal.poke()
    if noise == 'moo':
        return 'cow'
    elif noise == 'woof':
        return 'dog'

def identify_invertebrate(animal):
    if animal.is_multicellular():
        return 'Bug!'
    else:
        if animal.is_fungus():
            return 'Yeast'
        else:
            return 'Amoeba'

#-----------------------------------------------------------------------

print 'Flat is better than nested.'
```

```
################################### 6 ##################################

""" Parse an HTTP response object, yielding back new requests or data. """

#------------------------------------------------------------------------

def process(response):
    selector = lxml.cssselect.CSSSelector('#main > div.text')
    lx = lxml.html.fromstring(response.body)
    title = lx.find('./head/title').text
    links = [a.attrib['href'] for a in lx.find('./a') if 'href' in a.attrib]
    for link in links:
        yield Request(url=link)
    divs = selector(lx)
    if divs: yield Item(utils.lx_to_text(divs[0]))

#------------------------------------------------------------------------

def process(response):
    lx = lxml.html.fromstring(response.body)

    title = lx.find('./head/title').text

    links = [a.attrib['href'] for a in lx.find('./a') if 'href' in a.attrib]
    for link in links:
        yield Request(url=link)

    selector = lxml.cssselect.CSSSelector('#main > div.text')
    divs = selector(lx)
    if divs:
        bodytext = utils.lx_to_text(divs[0])
        yield Item(bodytext)

#------------------------------------------------------------------------

print 'Sparse is better than dense.'
```

```
################################### 7 ##################################

""" Write out the tests for a factorial function. """

#-----------------------------------------------------------------------

def factorial(n):
    """
    Return the factorial of n, an exact integer >= 0.

    >>> [factorial(n) for n in range(6)]
    [1, 1, 2, 6, 24, 120]

    >>> factorial(30)
    265252859812191058636308480000000L

    >>> factorial(-1)
    Traceback (most recent call last):
        ...
    ValueError: n must be >= 0
    """
    pass

if __name__ == '__main__' and '--test' in sys.argv:
    import doctest
    doctest.testmod()

#-----------------------------------------------------------------------

import unittest

def factorial(n):
    pass

class FactorialTests(unittest.TestCase):
    def test_ints(self):
        self.assertEqual(
            [factorial(n) for n in range(6)], [1, 1, 2, 6, 24, 120])

    def test_long(self):
        self.assertEqual(
            factorial(30), 265252859812191058636308480000000L)

    def test_negative_error(self):
        with self.assertRaises(ValueError):
            factorial(-1)

if __name__ == '__main__' and '--test' in sys.argv:
    unittest.main()

#--------------------------------------------------------------------------

print 'Readability counts.'
```

```
################################# 8 & 9 ################################

"""
Write a function that returns another functions. Also, test floating point.
"""

#-----------------------------------------------------------------------

def make_counter():
    i = 0
    def count():
        """ Increments a count and returns it. """
        i += 1
        return i
    return count

count = make_counter()
assert hasattr(count, '__name__') # No anonymous functions!
assert hasattr(count, '__doc__')


assert float('0.20000000000000007') == 1.1 - 0.9 # (this is platform dependent)
assert 0.2 != 1.1 - 0.9 # Not special enough to break the rules of floating pt.
assert float(repr(1.1 - 0.9)) == 1.1 - 0.9

#-----------------------------------------------------------------------

def make_adder(addend):
    return lambda i: i + addend # But lambdas, once in a while, are practical.

assert str(1.1 - 0.9) == '0.2' # as may be rounding off floating point errors
assert round(0.2, 15) == round(1.1 - 0.9, 15)

#-----------------------------------------------------------------------

print "Special cases aren't special enough to break the rules."
print 'Although practicality beats purity.'
```

```
################################ 10 & 11 ###############################

""" Import whatever json library is available. """

try:
    import json
except ImportError:
    try:
        import simplejson as json
    except:
        print 'Unable to find json module!'
        raise

#-----------------------------------------------------------------------

print 'Errors should never pass silently'
print 'Unless explicitly silenced.'
```

```
################################## 12 ##################################

""" Store an HTTP request in the database. """

def process(response):
    db.store(url, response.body)

#-----------------------------------------------------------------------

def process(response):
    charset = detect_charset(response)
    db.store(url, response.body.decode(charset))

print 'In the face of ambiguity, refuse the temptation to guess.'
```

```
################################## 13 ##################################

# Example 1
assert hasattr(__builtins__, 'map') # ('map' in __builtins__) raises TypeError
assert not hasattr(__builtins__, 'collect')

# Example 2
def fibonacci_generator():
    prior, current = 0, 1
    while current < 100:
        yield prior + current
        prior, current = current, current + prior

sequences = [
    range(20),
    {'foo': 1, 'fie': 2},
    fibonacci_generator(),
    (5, 3, 3)
    ]

for sequence in sequences:
    for item in sequence: # all sequences iterate the same way
        pass

#-----------------------------------------------------------------------

print 'There should be one, and preferably only one way to do it.'
print "Although that way may not be obvious at first unless you're Dutch."
```

```
################################## 14 ##################################

def obsolete_func():
    raise PendingDeprecationWarning

def deprecated_func():
    raise DeprecationWarning

print 'Now is better than never'
print 'Although never is often better than *right* now.'
```

```
################################### 15 ###################################

def hard():

    # Example 1
    try:
        import twisted
        help(twisted) # (this may not be as hard as I think, though)
    except:
        pass

    # Example 2
    import xml.dom.minidom
    document = xml.dom.minidom.parseString(
        '''<menagerie><cat>Fluffers</cat><cat>Cisco</cat></menagerie>''')
    menagerie = document.childNodes[0]
    for node in menagerie.childNodes:
        if node.childNodes[0].nodeValue== 'Cisco' and node.tagName == 'cat':
            return node


def easy(maybe):

    # Example 1
    try:
        import gevent
        help(gevent)
    except:
        pass

    # Example 2
    import lxml
    menagerie = lxml.etree.fromstring(
        '''<menagerie><cat>Fluffers</cat><cat>Cisco</cat></menagerie>''')
    for pet in menagerie.find('./cat'):
        if pet.text == 'Cisco':
            return pet

print "If the implementation is hard to explain, it's a bad idea."
print 'If the implementation is easy to explain, it may be a good idea.'
```

```
################################## 16 ##################################

def chase():
    import menagerie.models.cat as cat
    import menagerie.models.dog as dog

    dog.chase(cat)
    cat.chase(mouse)

print "Namespaces are one honking great idea -- let’s do more of those!"
```

```
############################### Readings ###############################

"""
    - Peters, Tim. PEP 20, "The Zen of Python".

    - Raymond, Eric. *The Art of Unix Programming*.
      (http://www.catb.org/~esr/writings/taoup/)

    - Alchin, Marty. *Pro Python*.

    - Ramblings on
      http://stackoverflow.com/questions/228181/the-zen-of-python

"""
```

```
############################## main block ##############################

from optparse import OptionParser

import os
import re
import subprocess
import sys

parser = OptionParser(usage=__doc__.strip())
parser.add_option('-v', dest='verbose', action='store_true',
    help='Verbose output')

header_pat = re.compile(r'^\\PY\{c\}\{' + (r'\\PYZsh\{\}' * 8))

def yield_altered_lines(latex):
    """
    Adds page breaks and page layout to our pygments file. Blah.
    """
    for line in latex.splitlines():
        if line == r'\documentclass{article}':
            yield line
            yield r'\usepackage{geometry}'
            yield r'\geometry{letterpaper,landscape,margin=0.25in}'
        elif line == r'\begin{document}':
            yield line
            yield r'\large'
        elif header_pat.search(line):
            yield r'\end{Verbatim}'
            yield r'\pagebreak'
            yield r'\begin{Verbatim}[commandchars=\\\{\}]'
            yield line
        else:
            yield line

if __name__ == '__main__':
    print
    options, args = parser.parse_args()
    if options.verbose:
        errout = sys.stderr
    else:
```

```
    errout = open('/tmp/pep20.log', 'w')

try:
    # TODO: pygmentize in Python instead of farming it out
    p = subprocess.Popen(
        ('pygmentize', '-f', 'latex', '-l', 'python',
            '-O', 'full', sys.argv[0]),
        stdout=subprocess.PIPE, stderr=errout)
    output, err = p.communicate()
    assert p.returncode == 0, 'pygmentize exited with %d' % p.returncode

    p2 = subprocess.Popen(
        ('pygmentize', '-f', 'html', '-l', 'python',
            '-O', 'full', '-o', 'pep20_by_example.html', sys.argv[0]),
        stdout=errout, stderr=errout)
    p2.communicate()
    assert p2.returncode == 0, 'pygmentize exited with %d' % p2.returncode

except OSError, e:
    print >> sys.stderr, 'Failed to run pygmentize: %s' % str(e)
except AssertionError, e:
    print e

altered_output = '\n'.join(l for l in yield_altered_lines(output))

try:
    p = subprocess.Popen(('pdflatex',),
        stdin=subprocess.PIPE, stdout=errout, stderr=errout)
    p.communicate(altered_output)
    assert p.returncode == 0, 'pdflatex exited with %d' % p.returncode
except OSError, e:
    print >> sys.stderr, 'Failed to run pygmentize: %s' % str(e)
except AssertionError, e:
    print e

os.rename('texput.pdf', 'pep20_by_example.pdf')

errout.close()
```