1 line: Output

print 'Hello, world!'

2 lines: Input, assignment

name = raw\_input('What is your name?\n')

print 'Hi, %s.' % name

3 lines: For loop, built-in enumerate function, new style formatting

friends = ['john', 'pat', 'gary', 'michael']

for i, name in enumerate(friends):

print "iteration {iteration} is {name}".format(iteration=i, name=name)

4 lines: Fibonacci, tuple assignment

parents, babies = (1, 1)

while babies < 100:

print 'This generation has {0} babies'.format(babies)

parents, babies = (babies, parents + babies)

5 lines: Functions

def greet(name):

print 'Hello', name

greet('Jack')

greet('Jill')

greet('Bob')

6 lines: Import, regular expressions

import re

for test\_string in ['555-1212', 'ILL-EGAL']:

if re.match(r'^\d{3}-\d{4}$', test\_string):

print test\_string, 'is a valid US local phone number'

else:

print test\_string, 'rejected'

7 lines: Dictionaries, generator expressions

prices = {'apple': 0.40, 'banana': 0.50}

my\_purchase = {

'apple': 1,

'banana': 6}

grocery\_bill = sum(prices[fruit] \* my\_purchase[fruit]

for fruit in my\_purchase)

print 'I owe the grocer $%.2f' % grocery\_bill

8 lines: Command line arguments, exception handling

# This program adds up integers in the command line

import sys

try:

total = sum(int(arg) for arg in sys.argv[1:])

print 'sum =', total

except ValueError:

print 'Please supply integer arguments'

9 lines: Opening files

# indent your Python code to put into an email

import glob

# glob supports Unix style pathname extensions

python\_files = glob.glob('\*.py')

for file\_name in sorted(python\_files):

print ' ------' + file\_name

with open(file\_name) as f:

for line in f:

print ' ' + line.rstrip()

print

10 lines: Time, conditionals, from..import, for..else

from time import localtime

activities = {8: 'Sleeping',

9: 'Commuting',

17: 'Working',

18: 'Commuting',

20: 'Eating',

22: 'Resting' }

time\_now = localtime()

hour = time\_now.tm\_hour

for activity\_time in sorted(activities.keys()):

if hour < activity\_time:

print activities[activity\_time]

break

else:

print 'Unknown, AFK or sleeping!'

11 lines: Triple-quoted strings, while loop

REFRAIN = '''

%d bottles of beer on the wall,

%d bottles of beer,

take one down, pass it around,

%d bottles of beer on the wall!

'''

bottles\_of\_beer = 99

while bottles\_of\_beer > 1:

print REFRAIN % (bottles\_of\_beer, bottles\_of\_beer,

bottles\_of\_beer - 1)

bottles\_of\_beer -= 1

12 lines: Classes

class BankAccount(object):

def \_\_init\_\_(self, initial\_balance=0):

self.balance = initial\_balance

def deposit(self, amount):

self.balance += amount

def withdraw(self, amount):

self.balance -= amount

def overdrawn(self):

return self.balance < 0

my\_account = BankAccount(15)

my\_account.withdraw(5)

print my\_account.balance

13 lines: Unit testing with unittest

import unittest

def median(pool):

copy = sorted(pool)

size = len(copy)

if size % 2 == 1:

return copy[(size - 1) / 2]

else:

return (copy[size/2 - 1] + copy[size/2]) / 2

class TestMedian(unittest.TestCase):

def testMedian(self):

self.failUnlessEqual(median([2, 9, 9, 7, 9, 2, 4, 5, 8]), 7)

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

14 lines: Doctest-based testing

def median(pool):

'''Statistical median to demonstrate doctest.

>>> median([2, 9, 9, 7, 9, 2, 4, 5, 8])

7

'''

copy = sorted(pool)

size = len(copy)

if size % 2 == 1:

return copy[(size - 1) / 2]

else:

return (copy[size/2 - 1] + copy[size/2]) / 2

if \_\_name\_\_ == '\_\_main\_\_':

import doctest

doctest.testmod()

15 lines: itertools

from itertools import groupby

lines = '''

This is the

first paragraph.

This is the second.

'''.splitlines()

# Use itertools.groupby and bool to return groups of

# consecutive lines that either have content or don't.

for has\_chars, frags in groupby(lines, bool):

if has\_chars:

print ' '.join(frags)

# PRINTS:

# This is the first paragraph.

# This is the second.

16 lines: csv module, tuple unpacking, cmp() built-in

import csv

# write stocks data as comma-separated values

writer = csv.writer(open('stocks.csv', 'wb', buffering=0))

writer.writerows([

('GOOG', 'Google, Inc.', 505.24, 0.47, 0.09),

('YHOO', 'Yahoo! Inc.', 27.38, 0.33, 1.22),

('CNET', 'CNET Networks, Inc.', 8.62, -0.13, -1.49)

])

# read stocks data, print status messages

stocks = csv.reader(open('stocks.csv', 'rb'))

status\_labels = {-1: 'down', 0: 'unchanged', 1: 'up'}

for ticker, name, price, change, pct in stocks:

status = status\_labels[cmp(float(change), 0.0)]

print '%s is %s (%s%%)' % (name, status, pct)

18 lines: 8-Queens Problem (recursion)

BOARD\_SIZE = 8

def under\_attack(col, queens):

left = right = col

for r, c in reversed(queens):

left, right = left - 1, right + 1

if c in (left, col, right):

return True

return False

def solve(n):

if n == 0:

return [[]]

smaller\_solutions = solve(n - 1)

return [solution+[(n,i+1)]

for i in xrange(BOARD\_SIZE)

for solution in smaller\_solutions

if not under\_attack(i+1, solution)]

for answer in solve(BOARD\_SIZE):

print answer

20 lines: Prime numbers sieve w/fancy generators

import itertools

def iter\_primes():

# an iterator of all numbers between 2 and +infinity

numbers = itertools.count(2)

# generate primes forever

while True:

# get the first number from the iterator (always a prime)

prime = numbers.next()

yield prime

# this code iteratively builds up a chain of

# filters...slightly tricky, but ponder it a bit

numbers = itertools.ifilter(prime.\_\_rmod\_\_, numbers)

for p in iter\_primes():

if p > 1000:

break

print p

21 lines: XML/HTML parsing (using Python 2.5 or third-party library)

dinner\_recipe = '''<html><body><table>

<tr><th>amt</th><th>unit</th><th>item</th></tr>

<tr><td>24</td><td>slices</td><td>baguette</td></tr>

<tr><td>2+</td><td>tbsp</td><td>olive oil</td></tr>

<tr><td>1</td><td>cup</td><td>tomatoes</td></tr>

<tr><td>1</td><td>jar</td><td>pesto</td></tr>

</table></body></html>'''

# In Python 2.5 or from http://effbot.org/zone/element-index.htm

import xml.etree.ElementTree as etree

tree = etree.fromstring(dinner\_recipe)

# For invalid HTML use http://effbot.org/zone/element-soup.htm

# import ElementSoup, StringIO

# tree = ElementSoup.parse(StringIO.StringIO(dinner\_recipe))

pantry = set(['olive oil', 'pesto'])

for ingredient in tree.getiterator('tr'):

amt, unit, item = ingredient

if item.tag == "td" and item.text not in pantry:

print "%s: %s %s" % (item.text, amt.text, unit.text)

28 lines: 8-Queens Problem (define your own exceptions)

BOARD\_SIZE = 8

class BailOut(Exception):

pass

def validate(queens):

left = right = col = queens[-1]

for r in reversed(queens[:-1]):

left, right = left-1, right+1

if r in (left, col, right):

raise BailOut

def add\_queen(queens):

for i in range(BOARD\_SIZE):

test\_queens = queens + [i]

try:

validate(test\_queens)

if len(test\_queens) == BOARD\_SIZE:

return test\_queens

else:

return add\_queen(test\_queens)

except BailOut:

pass

raise BailOut

queens = add\_queen([])

print queens

print "\n".join(". "\*q + "Q " + ". "\*(BOARD\_SIZE-q-1) for q in queens)

33 lines: "Guess the Number" Game (edited) from [http://inventwithpython.com](http://inventwithpython.com/)

import random

guesses\_made = 0

name = raw\_input('Hello! What is your name?\n')

number = random.randint(1, 20)

print 'Well, {0}, I am thinking of a number between 1 and 20.'.format(name)

while guesses\_made < 6:

guess = int(raw\_input('Take a guess: '))

guesses\_made += 1

if guess < number:

print 'Your guess is too low.'

if guess > number:

print 'Your guess is too high.'

if guess == number:

break

if guess == number:

print 'Good job, {0}! You guessed my number in {1} guesses!'.format(name, guesses\_made)

else:

print 'Nope. The number I was thinking of was {0}'.format(number)