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

```
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)
```