1. **Reading file data**

   (a) Write a function that takes as input the name of a text file and returns the total number of words in the file. You can test your function with:

   http://stanford.edu/~danfrank/cme193/data/course_description.txt

```
def num_words(file_name):
    with open(file_name, 'r') as f:
        total = 0
        for line in f:
           total += len(line.split())
        return total

print num_words('course_description.txt')
```

   The above prints 136.

   (b) Write a function that takes as input the name of a text file and returns the most frequently occurring word in the file. You can test your function with:

   http://stanford.edu/~danfrank/cme193/data/bill_of_rights.txt

   Hint: The Python `sorted()` function might be useful.

```
def most_common(file_name):
    with open(file_name, 'r') as f:
        words = {}
        for line in f:
           line_words = line.split()
           for word in line_words:
               if word in words:
                   words[word] += 1
               else:
                   words[word] = 1

        return sorted(words.items(),
                      key=lambda x: x[1],
                      reverse=True)[0]

print most_common('bill_of_rights.txt')
```

   The above prints `('the', 39)`.

(c) Suppose that we have scraped some web data from reddit. We have a text file where each row contains the base web page of the post, the number of comments, and the number of "upvotes" the post has received. Suppose that the data for each post is separated by `|`. For example, here are 5 data points:

```
imgur.com | 345 | 5060
i.imgur.com | 1228 | 4215
quickmeme.com | 434 | 4312
scientificamerican.com | 185 | 2412
bbc.co.uk | 191 | 726
```

Write a function that computes the average number of comments (number of comments is the right-most column) of the data points. Assume that the name of the data file is the function argument.

The above data is available at:

http://stanford.edu/~danfrank/cme193/data/reddit_data.txt

```
def avg_comments(file_name):
    with open(file_name, 'r') as f:
        total = 0
        count = 0
        for line in f:
           total += int(line.split('|')[-1])
           count += 1
        return float(total) / count

print avg_comments('reddit_data.txt')
```

The above prints 3345.0.

(d) Write a function that computes the average number of upvotes for posts on imgur and average number of upvotes for posts not on imgur.

```
def avg_upvotes(file_name):
    with open(file_name, 'r') as f:
        imgur_total = 0
        imgur_count = 0
        other_total = 0
        other_count = 0
        for line in f:
           data = line.split('|')
           if data[0].find('imgur') != -1:
               imgur_total += int(data[1])
               imgur_count += 1
           else:
               other_total += int(data[1])
               other_count += 1
        return (float(imgur_total) / imgur_count,
                float(other_total) / other_count)

print avg_upvotes('reddit_data.txt')
```

The above prints (786.5, 270.0).

2. **Cartesian points**

   (a) In the second homework, you represented (x, y, z) points using tuples and lists. Write a class called `CartPoint` that contains x, y, and z points as member variables. The constructor for the class should take an (x, y, z) tuple.

   (b) Add a `magnitude()` function to `CartPoint` that computes the magnitude of the point ($\sqrt{x^2 + y^2 + z^2}$).

   (c) Create a subclass of `CartPoint` called `CartPointTime` that represents an (x, y, z, t) vector, t representing time. The constructor for the class should take an (x, y, z, t) tuple.

   (d) Add a `magnitude_and_time()` to `CartPointTime` function that returns a $(magnitude, time)$ tuple.

```python
import math

class CartPoint:
  def __init__(self, coords):
    self.x, self.y, self.z = coords

  def magnitude(self):
    return math.sqrt(self.x ** 2 + self.y ** 2 + self.z ** 2)

p1 = CartPoint((1, 1, 0))
print p1.magnitude()

class CartPointTime(CartPoint):
  def __init__(self, coords):
    CartPoint.__init__(self, (coords[0], coords[1], coords[2]))
    self.t = coords[3]

  def magnitude_and_time(self):
    return (self.magnitude(), self.t)

p2 = CartPointTime((1, 1, 0, 0.5))
print p2.magnitude_and_time()
```

The above prints:

```
1.41421356237
(1.4142135623730951, 0.5)
```

3. **Exploring NumPy**

(a) What does the function `numpy.arange()` do? How is it different from `range()`?
`numpy.arange(n)` returns a numpy array that looks like a vector with entries $0, 1, \ldots, n-1$.

This is similar to `range()`, but `range()` returns a list.

(b) Create an identity matrix with `I = np.eye(3)`. What is the difference between the following two sequence of commands:

```
>>> x = I[0, [0, 1]]
>>> x[0] = 3.0
>>> x[1] = 2.0
```

```
>>> x = I[0, 0:2]
>>> x[0] = 3.0
>>> x[1] = 2.0
```

In the latter sequence, `x` is called a *view*. Why?

The difference is that the first sequence of code creates a copy of the elements of `I` while in the second code, `x` is a reference to entries in `I`. After executing the second sequence of code, `I` will be modified. In the latter case, we say that `x` is a view because it is still referencing (looking at) `I`.

(c) Using what you learned in part (b), write a function that takes as input a 2-D ndarray and scales the last row and column by 2.5 (the last entry should be scaled by 2.5^2).

An example:

```
def scale(A):
  A[-1, :] *= 2.5
  A[:, -1] *= 2.5

# make a call to scale()
M = np.array([[1, 2], [3, 4]])
scale(M)
```

(d) Explain what the following command does: `A = np.random.rand(6, 3) * np.arange(3)`.

`np.random.rand(6, 3)` creates a 6×3 matrix with random entries. `np.arange(3)` creates a vector of length 3 (see part (a)). The multiplication operator here does matrix vector multiplication, so A is a vector of length 6.