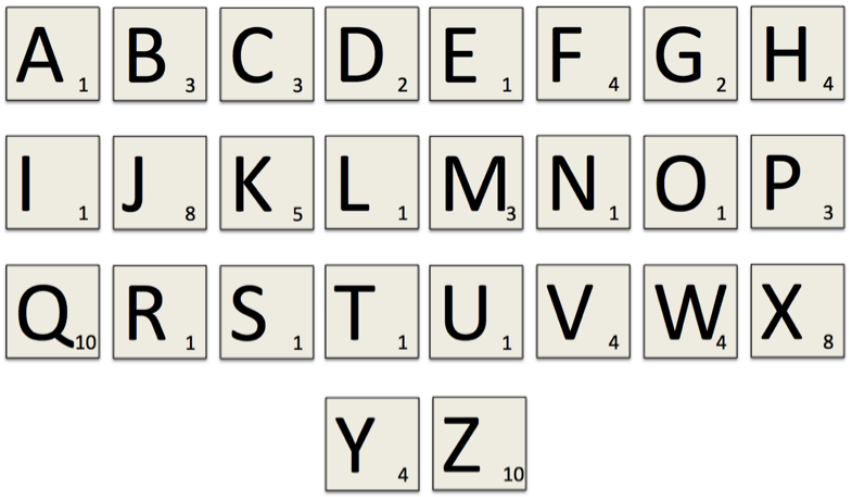
Classes, Error Handling and I/O (Week 6)

Scrabble Scoring

In the game *Scrabble*, points are scored by summing the value of the letters in a word. Different letters are assigned different values, based on how difficult it is to use the letter in a word. Below is a table that maps each letter to it's corresponding value. (Click the image for a larger version.)



(img/ScrabbleMappings.png)

1. Download the file scrabble_scores.txt (scrabble_scores.txt). This file contains each of the letters in the alphabet with their corresponding scrabble value. Write a function `read_scores` which uses this file to create a dictionary that maps letters to their scrabble score.

   Why can't we create a dictionary that does the reverse — that is, maps a scrabble score to the letter which gains that score?

2. Write a function `get_score(scores, word)`, that takes as input the scores dictionary and a string containing a word, and returns the word's scrabble score. (You may assume that the input string only consists of lowercase letters.) For example:

```
>>> scores = read_scores('scrabble_scores.txt')
>>> get_score(scores, 'quack')
20
```

Reading Configuration Files

When an application has to store information about how it's configured (for example, a user's preferences), it can do it by writing the information to a file, which can later be retrieved. When reading the configuration file, the application must translate the file into a suitable format, such as a dictionary.

Download the file config.txt (config.txt), which contains the following:

```
[user]
name=Eric Idle
email=e.idle@pythons.com
mobile=0412345678
[notifications]
email=yes
sms=no
```

In this format, each piece of data has a name (e.g. `email`) and a value (e.g. `e.idle@pythons.com`). The names/values are grouped under a heading (such as `user` or `notifications`). Each line in the file contains either a heading (surrounded by `[]` brackets), or a name/value pair (separated by an `=`).

Write a function `read_config` which takes a configuration file such as this, and returns a dictionary representation of the data, as in this example:

```
>>> read_config('config.txt')
{'user': {'name': 'Eric Idle',
          'email': 'e.idle@pythons.com',
          'mobile': '0412345678'},
 'notifications': {'email': 'yes', 'sms': 'no'}}
```

Also write a function `get_value` which takes the above dictionary, and the dot-separated name of a setting (e.g. 'user.mobile'), and returns the appropriate value ('0412345678' in this case). It is safe to assume the inputs are valid.

```
>>> config = read_config('config.txt')
>>> get_value(config, 'user.mobile')
'0412345678'
>>> get_value(config, 'notifications.email')
'yes'
```

Modify your `read_config` function so that it raises a `ValueError` if the file is invalid; that is, if the file contains a line which does not look like `[...]` or `...=...` , or if the file contains any name/value pairs before the first heading. You may wish to test your code on the following files: bad_config1.txt (bad_config1.txt), bad_config2.txt (bad_config2.txt).

Throughout this exercise, it is safe to assume that the headings/names/values in the file do not contain the characters `[ ] =` , and that the headings/names do not contain ' `.` '