File Reading

Quest

Now that we have all of this knowledge on containers, let's apply it! There are many times when as a programmer, you will want access to outside data. Outside data can come from the user (as is the case with the input function) or it can come from files. Files can be anything from plain text (characters and words on a page) to music (like an mp3 file), but for this class, we are going to focus on plain text.

There are two ways to read a file in Python. Let's look at **Method 1.**

```
with open('mydata.txt') as file:
    for line in file:
        line = line.strip() # optional
```

The `with` keyword creates an association between the name of the file `'mydata.txt'` and the variable file. The `as` keyword helps indicate which variable will be associated. The actual opening of the file is handled by the `open` function which takes in the name/path of the file to be opened.

Once the file is opened, the for-each loop grabs each line in the file as a string with a new line character `'\n'` on the end. Usually, when processing file data, we don't want to deal with a new line character, so we can use the `strip` method to get rid of it.

At the end of the `with` block (when the indentation stops), the file is automatically closed.

Now for **Method 2**.

```
for line in open('mydata.txt'):
    line = line.strip() # optional
```

Method 2 is very similar to Method 1. We merely skip the association step of the file with a variable. This is a newer method that is becoming more and more popular among programmers. One advantage of this is that the code tends to finish faster. Most programs that you've written so far have run almost instantly, but with large files and complex programs, run time becomes more of an issue. The tradeoff is that the file is not automatically closed at the end of the for loop. It will remain open for the rest of the program and then close once the program is complete. This is not an issue if you do not plan to use the file later in the program, so it's up to you which method you would like to use!

Worked Example - get_word_counts

Now that we've seen both methods, let's put one to use! Let's say we want to build a program that counts the number of times each word is used in a particular file and stores it in a dictionary. For example: let's say we had a file like this:

`sample.txt`

> This is a file with some words. This file can be used to
> count words. THIS IS SO COOL!!

We would want to output to be this:

```
{ 'this' : 3, 'is' : 2, 'a' : 1, 'file': 2, 'with': 1, 'some': 1, 'words': 2, 'can': 1,
 'be' : 1, 'used': 1, 'to': 1, 'count' : 1, 'so': 1, 'cool': 1}
```

Let's see how this would be implemented below:

```
PUNCTUATION = '.!?,-:;'

def delete_punctuation(string):
    '''
    Removes punctuation characters from a string and returns the resulting string
    (without punctuation).
    '''
    result = ''
    for char in string:
        # check char is not a punctuation mark
        if char not in PUNCTUATION:
            result += char
    return result


def get_word_counts(file_name):
        '''
        reads file and returns a dictionary with the words in the file and the number of
        occurances of each word
        '''
        counts = {}

        with open(filename) as file:    # open file to read
            for line in file:
                words = delete_punctuation(line).split()
                for word in words:
                    word = word.lower() # make all words lower case
                    if word not in counts:
                        counts[word] = 1
                    else:
                        counts[word] += 1
        return counts


def main():
    print(get_word_counts('sample.txt')


if __name__ == '__main__':
    main()
```

If you haven't already noticed, a lot of file reading is just text or string processing! That is why the first function that we write is `delete_punctuation`:

```
PUNCTUATION = '.!?,-:;'

def delete_punctuation(string):
    '''
    Removes punctuation characters from a string and returns the resulting string
    (without punctuation).
    '''
    result = ''
    for char in string:
        # check char is not a punctuation mark
        if char not in PUNCTUATION:
            result += char
    return result
```

We use the global constant `PUNCTUATION` and check every character in the string to build up `result`. Only characters not in `PUNCTUATION` are added to result effectively removing the punctuation characters.

Then we get to our key function: `get_word_counts`. The first three lines should look pretty familiar. We declare an empty dict, open the file, and start looping over the lines.

```
def get_word_counts(file_name):
    ...
    counts = {}

    with open(filename) as file:    # open file to read
        for line in file:
```

Next, we use our helper function to get rid of the punctuation and obtain a list of the resulting words using the `split` function. One thing to note is that if you don't give the `split` function an argument, it defaults to using whitespace as the separator.

```
words = delete_punctuation(line).split()
```

We loop over each word in the list of words and convert it to lowercase to make sure that our dictionary's keys are not case sensitive (`'THIS'` should not have a separate count from `'This'`).

```
for word in words:
    word = word.lower()
```

Last but not least, we add the key to our counts dictionary if it is not already in there and increment the count by one if it is. Then, we return the completed dictionary.

```
            if word not in counts:
                counts[word] = 1
            else:
                counts[word] += 1
return counts
```

When you break it down into simple steps, file reading becomes simple! All you need is the right text processing tools. Remember that you can always refer back to the **strings section** if you are unsure of what functions you have available.