CS 120: Intro to Computer Programming II

In-Class Activity - 01 Python Review - Day 3

Activity 1 - Turn in this one

Write a file named `ica_01_3_act1.py`

Your code must:

- Read the name of a file, from the keyboard
- Open that file
- Read each line one at a time, either by writing a `for` loop, or calling the `readlines()` method on the file object.
- Print out the lines of the file, along with some metadata about each line.

See the testcases to see what metadata you must print out.

The trick about this ICA is that you **must not add extra blank lines in the output.** Discuss with your table why the extra blank lines tend to appear, and how you can avoid them.

Solution: There are blank lines between the output because `print()` automatically adds a newline at the end of any print operation - but the strings we are reading from the file already contain their own newlines. Thus, when we print, we get two sequential newlines - the first takes us from the text, to the start of a new line, and the second takes us one more line down, thus leaving a blank line behind.

```
filename = input()

fobj = open(filename)
for line in fobj:
    print(f"Line length: {len(line)}")
    print(line.rstrip('\n'))

    # line.strip('\n') is also OK; I used rstrip() because I knew
    # searching the head of the stirng was pointless.
    #
    # line.strip() also works, but would remove whitespace other than
    # newlines.  Maybe your program wants that, maybe not.
```

Activity 2 - Turn in this one

Write a file named `ica_01_3_act2.py`

Suppose that you are given a dictionary. It's easy to search through the keys - that's what dictionaries are best at. But how do you search based on **values???**

Write a function `max_key_by_val(data)` that takes a dictionary as its only parameter. Build a loop that iterates through the keys. Look at the value for each one. Keep track of the **key** which has the maximum **value**. Return that key from the function.

EXAMPLE:

```
Key       Value
---       -----
foo       1234
bar       888888
baz       -1
fred      256
```

The key associated with the maximum value, from the example above, would be `"bar"`, because it's associated with the value 888888. The key associated with the second would be `"foo"`, which is associated with 1234.

Solution:

```
def max_key_by_val(data):
    # can't initilaize to 0, since the max might be negative.  Or
    # might not even be an integer!  Instead, use a flag to initialize
    # inside the loop.
    #
    # ADVANCED: Once you know what None does, you could use None
    #           instead - instead of having a separate flag.

    first = True
    for key in data:
        if first:
            best_key = key
            max_val  = data[key]
            first = False
        else:
            if data[key] > max_val:
                best_key = key
                max_val = data[key]

    # TODO: use an 'assert' to enforce that the dictionary wasn't empty
    return best_key
```

(activity continues on next page)

Challenge Activity - Do not turn in this one

As a second step, take your code, and add a little bit to it, so that it can also find the key associated with the **second** largest value. Return both keys, like this:

```
return first,second
```

You can call a function like this, which returns two values, and save the results like this:

```
a,b = function_that_returns_two_things(some_param)
```

Challenge Activity - Do not turn in this one

Write a function, named `second_favorite_word()`. It must take exactly one parameter, and that is the name of a file to read. Inside the function, open that file.

Read each of the lines. Break each one into words. Then, for each word, use `lower()` to convert it into lowercase.

Use a dictionary to count how many times each word showed up in the file. When you reach the end of the file, hunt through the dictionary and find the second-most-common word. Return that word from the function.

Solution: VERSION 1: Finds the most common, efficiently.

```
def second_favorite_word(filename):
    words = {}
    for line in open(filename):
        for word in line.split():
            word = word.lower()
            # TODO: It would be nice to strip punctuation, too!  Some other time...

            if word not in words:
                words[word] = 1
            else:
                words[word] += 1


    # search for the second word.  This works the same as Activity 3
    # above, except:
    #    - It keeps track of the best 2 values.
    #    - It uses 'None' as an implicit "not found yet" flag, to
    #      make the code simpler.

    # but WHEW!  The if() statements here are hairy!  -- Russ :(

    first_key  = None
    second_key = None
    first_val  = None
    second_val = None

    for key in words:
        val = words[key]    # save some CPU time by caching this, don't
                            # have to look it up multiple times.
```

```
            if first_key is None or val > first_val:
                # shift the first down into the 2nd slot.  Note that this
                # code runs on the *VERY FIRST* iteration of the loop.
                second_key = first_key
                second_val = first_val

                first_key = key
                first_val = val

            elif second_key is None or val > second_val:
                second_key = key
                second_val = val

            # else this isn't interesting, move on...

        return second_key
```

VERSION 2: Takes more CPU time (sorting is O(n lg n)), but is a lot easier to read.

```
        # this picks up right after the initial loop; the words{} dictionary
        # has been filled, and now we need to find the second most common
        # word.
        words2 = []    # will be (count,word) pairs
        for key in words:
            words2.append( (key,words[key]) )
        words2 = sorted(words2)

        return words2[1][1]     # 2nd most common word; read field [1] of the tuple
```