File Writing

Review of match vs. search vs. findall vs. finditer:

Match returns the first match IFF the first match is at the very beginning of the sentence

Search returns the first match of the pattern

list of matched strings

Findall returns every match as a string

list of matched objects

Finditer returns every match as a 'match object', which contains a lot of information

match_obj.start() is the index of the first character of the match

match_obj.end() is the index of the last character of the match

match_obj.group() is the actual string that matched

PotD 21 (using the above review):

```
# Trace through the code by hand
import re

def match_occurrence(string, regex):
    result = []
    number_occurrence = 0

    print('\n' + "=== match_occurrence ===")

    match_objects = regex.match(string)
    if match_objects:
        print("match_objects = " + match_objects.group())
        for occurrence in match_objects.group().split():
            number_occurrence = number_occurrence + 1
            result.append(occurrence)

    return result, number_occurrence


def search_occurrence(string, regex):
    result = []
    number_occurrence = 0

    print('\n' + "=== search_occurrence ===")

    search_objects = regex.search(string)
    if search_objects:
        print("search_objects = " + search_objects.group())
        for occurrence in search_objects.group().split():
            number_occurrence = number_occurrence + 1
            result.append(occurrence)

    return result, number_occurrence


def findall_occurrence(string, regex):
    result = []
```

```
    number_occurrence = 0

    print('\n' + "=== findall_occurrence ===")

    find_objects = regex.findall(string)
    if find_objects:
        # print("find_objects = " + str(find_objects))
        for occurrence in find_objects:
            number_occurrence = number_occurrence + 1
        result = find_objects

    return result, number_occurrence


def finditer_occurence(string, regex):
    result = []
    number_occurence = 0

    print('\n' + "=== finditer_occurence ===")

    find_objects = regex.finditer(string)
    if find_objects:
        # print("find_objects = " + str(find_objects))
        for occurrence in find_objects:
            result.append(occurrence)
            number_occurence = number_occurence + 1

    return result, number_occurence


input_string = "A few small syntax errors is OK. But if you are really off we will
take off points. " \
               "Try to write correct code."
# the entire string is being read--if you add an 'off' in the second line, the
finall and finditer occurrences increase by one
regex = re.compile(r'(.f{2})')
# represents any single character, followed by two 'f's

# each of these functions is printing some formatting stuff, and then the function
named (match, search, or findall)
# returns a list: first, the result of the function as a list, second, the number
of objects the function returned

print(match_occurrence(input_string, regex))
# not at the beginning so gives us [[], 0]]
print(search_occurrence(input_string, regex))
# search should give us one single 'off', so [['off'], 1]
print(findall_occurrence(input_string, regex))
# findall gives us just the match, every time something matches, so [['off',
'off'], 2]
```

```
print(finditer_occurence(input_string, regex))
# finditer gives us a bunch of info about every match (not gonna write it out)
```

So far, we have only been reading files (**'r'**); however, we can also write or append to them
Overview of file writing available here: https://cs1110.cs.virginia.edu/16-addendum.html
Also in slides on the schedule page

Open the file with **'w'** to write over all of it, or with **'a'** to add to the end of it
Remember: ALWAYS close the file after you use it, or your version of the file will get out of sync with other users, and it will take up a bunch of your computer's memory space

When we open a file in write mode, it automatically empties it--this is dangerous!

Example 1:

```
def read_list_of_names(filename):
    names = []
    datafile = open(filename, 'r')
    # this is the default mode, read mode
    outfile = open(filename, 'a')
    # we are overriding the original file now
    for line in datafile:
        # this is the file we are only reading
        line = line.strip()
        # strip our version of the line
        names.append(line)
        # add the stripped line to the list 'names'
        outfile.write(line)
        # add the stripped line to the end of the original file, where we are
appending
    datafile.close()
    outfile.close()
    # close everything
    return names
# we only returned the list of names, but the original file has been changed, as
well
```

(your .py file)

```
# DO NOT pass a name of a file on your computer, or you will delete the file!
```

Function **with open** will close the file automatically at the end of the indented code
Will also auto-close the file if the is an error or the program crashes

Example 2:

```
# create a file full of your friends' favorite cartoon characters
def create_file(new_file_name):
    new_file = open(new_file_name, 'w')
    char = 'Favorite Characters: '
    while char != '':
```

```
        new_file.write(char + '\n')
        char = input('What is your favorite cartoon character? (enter a blank field
to quit) ')
    new_file.close()
    # had to close it ourself
    return new_file

# change this to a with open

def create_file_with_open(new_file_name):
    with open(new_file_name, 'w') as new_file:
        char = 'Favorite Characters: '
        while char != '':
            new_file.write(char + '\n')
            char = input('What is your favorite cartoon character (2)? (enter a
blank field to quit) ')
    # closed automatically
    return new_file

create_file('cartoon.txt')
create_file_with_open('cartoons.txt')
```

Note: I did these as a while loop so you could enter as many characters as you want; the class did a for loop that asked for three characters (no more no less)