Fundamentals Problem Set #3

Covid-19 Computational Biology Workshop

May 12, 2020

1 Reading CSV Files

Working with files in the Comma-separated-values (CSV) format occurs often in bioinformatics. This is one specific case of a ”delimited” file, one with a specific pattern of information separated by a ”delimiter” character. In CSV files, the delimiter is a comma (,) symbol.

For instance, an example line you might see in a CSV file is:

```
1,apples,03/27/1956,17.5
```

This line would be intended to communicate the separate data items: ”1”, ”apples”, ”03/27/1956”, and ”17.5”. You may be wondering: ”If that’s a ’1’ or a ”17.5”, then why are there quotes around it?” which would be a very good question. CSV files by default carry information as strings. Therefore it’s up to the user to identify and process pieces of data in the correct way.

Quiz: What is the best Python data type for each of the above pieces of data?

As an example, let’s go through a loop to open and read a csv file in python in the simplest way, opening it as with any other text file:

```
my_csv_file_path = "path/to/file.csv"
with open(my_csv_file_path) as my_csv_file:
    for line in my_csv_file:
        #Do something with variable: "line" here.
```

Quiz: What data type is variable ”line”?

As a reminder, the above code opens the file at path ”path/to/file.csv,” and reads through the file line by line. Let’s say the above example line is the first line of a csv file. Therefore we could read the items in the line with:

```
my_csv_file_path = "path/to/file.csv"
with open(my_csv_file_path) as my_csv_file:
    for line in my_csv_file:
        line_items = line.split(',')
```

The ”str().split(delim)” function takes a string, finds every instance of the substring in delim, and returns a list of all the strings between those substrings. AKA, the file is ”split” by the delimiter string. For the first line, this will therefore give you the following list of strings:

```
print(line_items)
    --> ["1", "apples", "03/27/1956", and "17.5"]
```

Quiz: how do I access only column 4?

Remember, these are strings. To work with the data types as they are we have to convert the data. For example, to work with column 1 as a number, we have to convert the string to an integer using the int() function:

```
my_csv_file_path = "path/to/file.csv"
with open(my_csv_file_path) as my_csv_file:
    for line in my_csv_file:
        line_items = line.split(',')
        col_1_as_int = int(line_items[0])
```

Quiz (/Warning): What if we try to access column 4 in a line that only has 3 columns? How could we prevent this?

2 Writing CSV Files

We've covered how to separate a CSV line into a list of items, now how do we do the reverse? We do the similar operations, in reverse! Let's start with a new list of items that we want to write to a CSV line:

```
my_list = [4, "pears", "04/05/1972", 9.3]
```

How do we do so? The str().join(my_iter) method is useful for taking a list of strings and combining it into one long string, separated by the character(s) given in str().
Some examples:

```
test_list = ["a","b","c","d"]
print("".join(test_list))  # Note, this joins using an empty string
    --> "abcd"

print("*".join(test_list))
    --> "a*b*c*d"

print("---".join(test_list))
    --> "a---b---c---d"
```

Quiz: what might happen if I use the join method on my_list without changing anything first?

Our list contains many different data types, so we first have to make a list containing only strings.

```
my_list = [4, "pears", "04/05/1972", 9.3]
new_list = []
for item in my_list:
    new_list.append(str(item))
print(new_list)
    --> ["4", "pears", "04/05/1972", "9.3"]

#OR, using list comprehensions:
new_list = [str(item) for item in my_list]
print(new_list)
    --> ["4", "pears", "04/05/1972", "9.3"]
```

Now we can use the join method, with the delimiter of a comma to create our comma-delimited line:

```
comma_line = ",".join(new_list)
print(comma_line)
    --> "4,pears,04/05/1972,9.3"
```

Putting this all together, and writing this single line to a file:

```
my_list = [4, "pears", "04/05/1972", 9.3]
new_list = []
for item in my_list:
    new_list.append(str(item))
comma_line = ",".join(new_list)

out_csv_path = "path/to/out.csv"
with open(out_csv_path, 'w') as out_csv:
    out_csv.write(comma_line + '\n')
    # Don't forget the newline '\n' character at the end of each written line!
```

3 CSV Homework

Posted with this homework is a file: "sequence_info.csv"
This is a CSV file where each line contains:

1. seq-id
2. sequence
3. Date-identified
4. count
5. Percent-"C"-nucleotides

6. Importance-factor

Using the methods described above, the homework assignment is to:

1. Read the input CSV file
2. Count the number of lines in the input CSV, and print the result.
3. Print if the motif "AGTC" is contained within the nucleotide sequence
4. Print the percentage of C nucleotides out of 100% (IE not a fraction).
5. Calculate the "weighted count" of each sequence: $weighted\_count = (count * importance\_factor)$
6. Append the weighted count to each line, and write the resulting line to a new output CSV file called "sequence_info_weighted.csv"

"Extra Credit:"

1. Implement each of the above tasks using a function.
2. In the output file, replace commas (",") with tab-characters ("\t") as the delimiter.

Comments: There is flexibility in how the above tasks are performed, and so there are potentially many correct answers. Some methods are faster than others, and some are more readable than others, but the first goal is getting something that works *right* for the given question.