

CSE 503

Introduction to Computer Science for Non-Majors

Dr. Eric Mikida

epmikida@buffalo.edu

208 Capen Hall

Day 18

Read from CSV Files

Announcements

- Lab #2 due tonight at 11:59PM
- Lab #3 will be released today
- Project will be released today (probably)

Recap

- Opening and reading from text files in Python...

Opening Files in Python

The `open(...)` function is usually used with a [with...as](#) statement:

```
with open("test1.txt", "r") as f:  
    # do something with the file...
```

`f` is a variable. It refers to a [file object](#).

The `with...as` statement ensures that the file is automatically closed at the end of the suite of statements, no matter what happens.

File Objects and Iteration

File objects support iteration so we can use a for loop to iterate over each line in a file:

```
with open("test1.txt", "r") as f:  
    for line in f:  
        # do something with each line...  
        print(line)
```

Exercise

Define a function that takes a filename as an argument, and returns a dictionary of character counts for the file.

For example, if the file contains the character "a" 12 times, and "e" 17 times, the returned dictionary would have `"a":12` and `"e":17`, in addition to the counts of the other characters in the file.

Break It Down

We want to count all of the characters in a text file. We know that we process files line by line, so:

Break It Down

We want to count all of the characters in a text file. We know that we process files line by line, so:

We want to **count the characters on every line of the file...**

Break It Down

We want to count all of the characters in a text file. We know that we process files line by line, so:

We want to **count the characters on every line of the file...**

To do that we must be able to **count all the characters from a single line**

Single Line Sub-Problem

```
def countCharsInLine(line):  
    dict = {}  
    for c in line:  
        dict[c] = dict.get(c, 0) + 1  
  
    return dict
```

Putting It Together

```
def charCount(filename):  
    d = {}  
    with open(filename, "r") as f:  
        for line in f:  
            line = line.rstrip("\n").lower()  
            result = countCharsInLine(line)  
            # What do we do with the result?  
    return d
```

Putting It Together

```
def charCount(filename):
```

```
    d = {}
```

```
    with open(filename, "r") as f:
```

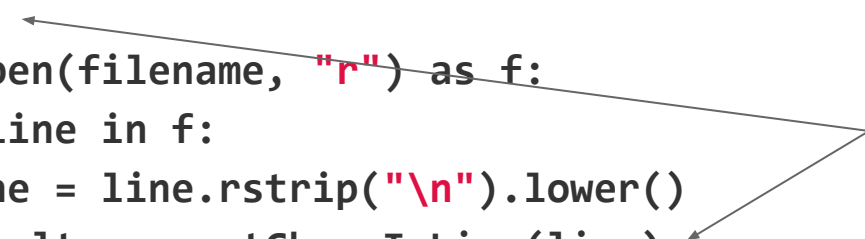
```
        for line in f:
```

```
            line = line.rstrip("\n").lower()
```

```
            result = countCharsInLine(line)
```

```
            # What do we do with the result?
```

```
    return d
```



We want to "add" the result from our subproblem to our overall result

Putting It Together

```
def charCount(filename):
```

```
    d = {}
```

```
    with open(filename, "r") as f:
```

```
        for line in f:
```

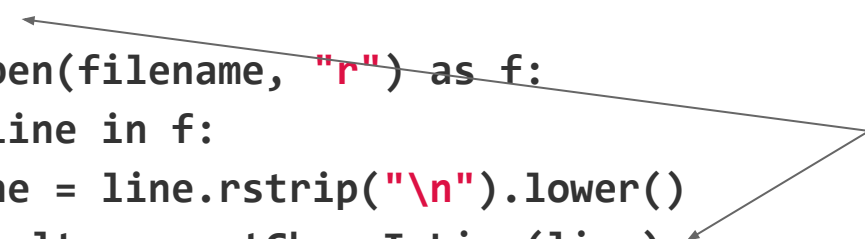
```
            line = line.rstrip("\n").lower()
```

```
            result = countCharsInLine(line)
```

```
            # What do we do with the result?
```

```
    return d
```

We want to "add" the result from our subproblem to our overall result



One option would be to write a function to add two dictionaries...see Lab #3

Putting It Together

```
def charCount(filename):  
    d = {}  
    with open(filename, "r") as f:  
        for line in f:  
            line = line.rstrip("\n").lower()  
            result = countCharsInLine(line)  
            # What do we do with the result?  
    return d
```

We want to "add" the result from our subproblem to our overall result

One option would be to write a function to add two dictionaries...see Lab #3

Option 2: What if we don't start counting from scratch for each new line?

Single Line Sub-Problem

```
def countCharsInLine(line):
```

```
    dict = {}
```

```
    for c in line:
```

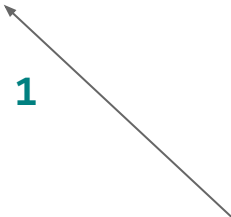
```
        dict[c] = dict.get(c, 0) + 1
```

```
    return dict
```

This means we'll start counting from scratch for each line...

Single Line Sub-Problem

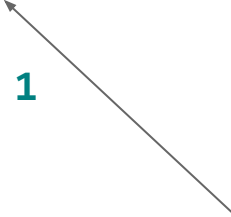
```
def countCharsInLine(line, dict = {}):  
    for c in line:  
        dict[c] = dict.get(c, 0) + 1  
  
    return dict
```



Now we can pass a dictionary in as a starting point, and if none is passed in, we'll just start from scratch with {}

Single Line Sub-Problem

```
def countCharsInLine(line, dict = {}):  
    for c in line:  
        dict[c] = dict.get(c, 0) + 1  
  
    return dict
```

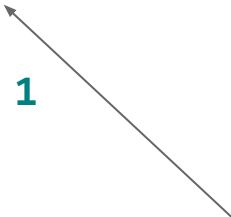


Now we can pass a dictionary in as a starting point, and if none is passed in, we'll just start from scratch with {}

Does anything in this function constrain us to characters in a string? What if instead we passed in a list of numbers instead?

Single Line Sub-Problem

```
def countCharsInLine(line, dict = {}):  
    for c in line:  
        dict[c] = dict.get(c, 0) + 1  
  
    return dict
```



Now we can pass a dictionary in as a starting point, and if none is passed in, we'll just start from scratch with {}

Does anything in this function constrain us to characters in a string? What if instead we passed in a list of numbers instead?
Let's change the naming to make it more general

Single Line Sub-Problem

```
def countSequence(seq, dict = {}):  
    for x in seq:  
        dict[x] = dict.get(x, 0) + 1  
  
    return dict
```

Does anything in this function constrain us to characters in a string? What if instead we passed in a list of numbers instead?
Let's change the naming to make it more general

The Full Solution

```
def countSequence(seq, dict = {}):  
    for x in seq:  
        dict[x] = dict.get(x, 0) + 1  
    return dict  
  
def charCount(filename):  
    d = {}  
    with open(filename, "r") as f:  
        for line in f:  
            line = line.rstrip("\n").lower()  
            d = countSequence(line, d)  
    return d
```

Exercise V2 (words)

What if instead we want to count words instead of characters? What do we have to do?

Exercise V2 (words)

What if instead we want to count words instead of characters? What do we have to do?

We first need to convert a line from a string (sequence of characters) to a list of words.

Exercise V2 (words)

What if instead we want to count words instead of characters? What do we have to do?

We first need to convert a line from a string (sequence of characters) to a list of words (...and define what a "word" is).

Exercise V2 (words)

What if instead we want to count words instead of characters? What do we have to do?

We first need to convert a line from a string (sequence of characters) to a list of words (...and define what a "word" is).

Python has many functions for manipulating strings, many of which are defined in the Regular Expression library: `re`.

Basic Strategy

```
def wordCount(filename):  
    d = {}  
    with open(filename, "r") as f:  
        for line in f:  
            line = line.rstrip("\n").lower()  
            # Convert line to list of words?  
            # Count words  
    return d
```

Basic Strategy

```
def wordCount(filename):  
    d = {}  
    with open(filename, "r") as f:  
        for line in f:  
            line = line.rstrip("\n").lower()  
            # Convert line to list of words?  
            # Count words  
    return d
```

How do we do this?



Regular Expressions

How do we define a word? What are the words in the following sentence?

Sally's new puppy is named Rover. Rover's tail was wagging. Rover was happy!

Regular Expressions

How do we define a word? What are the words in the following sentence?

Sally's new puppy is named Rover. Rover's tail was wagging. Rover was happy!

(One possible) definition of a word is a sequence of characters that are:

a-z, A-Z, or ' '

Regular Expressions

How do we define a word? What are the words in the following sentence?

Sally's new puppy is named Rover. Rover's tail was wagging. Rover was happy!

(One possible) definition of a word is a sequence of characters that are:

a-z, A-Z, or ' '

As a regular expression:

`"[a-zA-Z']+"`

Any characters between the [], repeated one or more times (+),

Basic Strategy

```
def wordCount(filename):  
    d = {}  
    with open(filename, "r") as f:  
        for line in f:  
            line = line.rstrip("\n").lower()  
            # Convert line to list of words?  
            # Count words  
    return d
```

Basic Strategy

```
import re
```

← Import the regular expression library

```
def wordCount(filename):
```

```
    d = {}
```

```
    with open(filename, "r") as f:
```

```
        for line in f:
```

```
            line = line.rstrip("\n").lower()
```

```
            words = re.split("[^a-zA-Z]+", line)
```

```
            # Count words
```

```
    return d
```

← Our pattern adds ^, which negates the set we have given it

← Call split, which breaks up the line by removing anything that matches the pattern

Basic Strategy

```
import re
```

```
def wordCount(filename):
```

```
    d = {}
```

```
    with open(filename, "r") as f:
```

```
        for line in f:
```

```
            line = line.rstrip("\n").lower()
```

```
            words = re.split("[^a-zA-Z]+", line)
```

```
            # Count words
```

```
    return d
```

← We already have a solution for this!

Basic Strategy

```
import re

def wordCount(filename):
    d = {}
    with open(filename, "r") as f:
        for line in f:
            line = line.rstrip("\n").lower()
            words = re.split("[^a-zA-Z]+", line)
            d = countSequence(words, d)
    return d
```

Full Solution

```
def countSequence(seq, dict = {}):  
    for x in seq:  
        dict[x] = dict.get(x, 0) + 1  
    return dict
```

```
def wordCount(filename):  
    d = {}  
    with open(filename, "r") as f:  
        for line in f:  
            line = line.rstrip("\n").lower()  
            words = re.split("[^a-zA-Z]+", line)  
            d = countSequence(words, d)  
    return d
```

CSV Files

Comma-separated values

In computing, a **comma-separated values (CSV)** file is a delimited text file that uses a comma to separate values. A CSV file stores tabular data (numbers and text) in plain text. Each line of the file is a data record. Each record consists of one or more fields, separated by commas. The use of the comma as a field separator is the source of the name for this file format.

Excerpt from https://en.wikipedia.org/wiki/Comma-separated_values

CSV Files

Example

```
Month,Budget,Actual  
January,200,190  
February,200,210  
March,150,185  
April,100,110  
May,50,40  
June,50,15  
July,50,12  
August,50,14  
September,50,35  
October,100,78  
November,150,125  
December,200,167
```

CSV Files

Can be imported/exported with spreadsheet programs like excel/numbers/Google sheets

	A	B	C
1	Month	Budget	Actual
2	January	200	190
3	February	200	210
4	March	150	185
5	April	100	110
6	May	50	40
7	June	50	15
8	July	50	12
9	August	50	14
10	September	50	35
11	October	100	78
12	November	150	125
13	December	200	167
14			
15			

```
Month,Budget,Actual
January,200,190
February,200,210
March,150,185
April,100,110
May,50,40
June,50,15
July,50,12
August,50,14
September,50,35
October,100,78
November,150,125
December,200,167
```

Heating

Month	Budget	Actual
January	200	190
February	200	210
March	150	185
April	100	110
May	50	40
June	50	15
July	50	12
August	50	14
September	50	35
October	100	78
November	150	125
December	200	167

Reading CSV Files

Let's write a program to read the data in our csv file into a dictionary. We'll use the month as a key, and put the rest of the data into a list.

For example:

```
{ 'Month': ['Budget', 'Actual'],  
  'January': ['200', '190'],   'February': ['200', '210'],  
  'March': ['150', '185'],    'April': ['100', '110'],  
  'May': ['50', '40'],        'June': ['50', '15'],  
  'July': ['50', '12'],       'August': ['50', '14'],  
  'September': ['50', '35'],  'October': ['100', '78'],  
  'November': ['150', '125'], 'December': ['200', '167'] }
```

Reading CSV Files

```
import csv
```

```
def readBudget(filename):  
    budget = {}  
    # Read data from file...
```

```
    return budget
```

Reading CSV Files

```
import csv ← Import library for reading csv files
```

```
def readBudget(filename):  
    budget = {}  
    # Read data from file...
```

```
    return budget
```


Reading CSV Files

```
import csv

def readBudget(filename):
    budget = {}
    with open(filename, newline='') as f:
        reader = csv.reader(f)

    return budget
```

Reading CSV Files

```
import csv
```

Open the file *mostly* the same as before (newline is required for CSV file reading)

```
def readBudget(filename):  
    budget = {}  
    with open(filename, newline='') as f:  
        reader = csv.reader(f)
```

```
    return budget
```

Reading CSV Files

```
import csv
```

```
def readBudget(filename):
```

```
    budget = {}
```

```
    with open(filename, newline='') as f:
```

```
        reader = csv.reader(f)
```

← Create a `csv.reader` object from our file

```
    return budget
```

Reading CSV Files

```
import csv
```

```
def readBudget(filename):  
    budget = {}  
    with open(filename, newline='') as f:  
        reader = csv.reader(f)  
        for line in reader:  
            # Do something with each line  
  
    return budget
```

We can iterate over the lines in the file just like we did with textfiles.

But now, instead of a line of text, **line** is a list of values

Reading CSV Files

```
import csv
```

```
def readBudget(filename):  
    budget = {}  
    with open(filename, newline='') as f:  
        reader = csv.reader(f)  
        for line in reader:  
            key = line[0]                # The month will be our key  
            value = [line[1], line[2]] # [budget,actual] will be our value  
            budget[key] = value  
    return budget
```

Reading CSV Files

```
import csv
```

```
def readBudget(filename):  
    budget = {}  
    with open(filename, newline='') as f:  
        reader = csv.reader(f)  
        next(reader) # skip the first line  
        for line in reader:  
            key = line[0] # The month will be our key  
            # [budget,actual] will be our value (convert values to ints)  
            value = [int(line[1]), int(line[2])]  
            budget[key] = value  
    return budget
```

Skip over the header line



Convert values to integers



Exercises

1. Define a function, **overspent**, which takes a dictionary like that produced by the **readBudget** function, and returns a dictionary of the months in which expenditures were over budget, along with the difference (as a negative value).
2. Define a function, **underspent**, which takes a dictionary like that produced by the **readBudget** function, and returns a dictionary of the months in which expenditures were under budget, along with the difference (as a positive value).