CIS 122 Fall 2015 Project 6 read files
Due Monday, Feb 22, 11:59 PM

Briefly

This entire project focuses on reading .txt files into lists and working with those lists.

You'll discover that it takes surprisingly little effort to convert a program from handling a list of 20-some items to a list of 200,000-some items.

For files with several data items per line, you'll use a list of little "sub-lists".

P6_read_words1.py
10 points

Copy the **sowpods_short.txt** file to your program folder.
Open the text file, read each line into a list;
If the filename contains the word "short", print the list of words.

Repeatedly:
Ask user for a word
Tell whether the word was in your list of words

P6_read_words2.py
7 points

Use the same logic exactly as P6_read_words1.py **except,** copy the **sowpods.txt** file to your program folder.
Change the filename to sowpods.txt, but leave everything else the same.

You now have a program that lets you see if the word you entered is in the 267,751 word British list of legal scrabble words.

Notice that you needed almost no change except for filename to handle about 10 thousand times the data volume.

The fancy term for this is **scaling.**

Your program "scales well" when you only need to change the filename to handle thousands of times more data.

P6-read_words_**bonus**.py
3 bonus points

Change your P6_read_program to allow 2 options
M Match the word the user typed exactly
S Starts with – Match all words

Start first with the sowpods_short.txt file.
Starts with will have issues – selecting all words that start with "m" works well on the short file and overwhelms on the long file.

One strategy is to break up selecting matching words from printing matching words into 2 distinct steps.

Before printing in detail, inform your user about how many matching words were found, and ask whether to proceed.

It's also a good idea to ask periodically whether to keep printing.

P6_read_states1.py
5 points

You will copy the **state_data.txt** file to your program folder.
Each line in this file has several data items:

State name followed by a comma
Capital city followed by a comma
Area in square kilometers followed by a comma
Population in millions such as 3.9 followed by a comma

Your program will read the data into a list of states; each item in the list is a details list somewhat like this
`['oregon','salem',345678,3.8]`

Once your program has read the data into the states list, it repeatedly asks the user to type the start of a state name or 'quit'.

It then displays each state or states that start with the letters the user provided.

Your data must line up nicely in neat columns for more than 2 points credit.

1 point bonus

Display the computed **density** of **people per square kilometer** in your state display of data.

P6_read_states2.py
5 points

Also reads and processes the **states_data.txt** file and its comma separated list of values.

User search options are extended to include
S State name **Starts with**
C State name **Contains**
E State name **Ends with**
Q **Quit**

You also will need to line up the data neatly in columns for full credit.

Bonus 2 _ 1 point

Display density (people per square km) and select on density greater than a user-supplied number.

The **week 6 module** has an **example program** with several functions that should prove useful to writing and debugging these programs.

The next page shows some example output from the programs.

P6_read1.py

Sample results

```
DEBUG
ace
apt
baggage
but
dog
god
her
mates
meats
open
opts
pat
pone
post
pots
put
she
spot
steam
stop
the
throw
tub
worth
There are 24 in the word list

Type a word to find in the word list, or $ to exit: teM
Sorry tem is not in the word list

Type a word to find in the word list, or $ to exit: TEAM
Sorry team is not in the word list

Type a word to find in the word list, or $ to exit: steaM
Found steam
```

10 points:

2 point Your program reads each word, strips newlines and skips any line starting with #

1 point Display list of words read in to a word list
1 point Display number of words in word list

2 points User interaction includes a way to quit that does not use a word (for example, "$" to quit).
2 points User-entered word correctly found in list
2 points User-entered word (albania for example) not found in list.

P6_read2.py 5 points

Sample results

```
There are 267751 in the word list

Type a word to find in the word list, or $ to exit: albania
Sorry albania is not in the word list

Type a word to find in the word list, or $ to exit: albatross
Found albatross

Type a word to find in the word list, or $ to exit: zoos
Found zoos

Type a word to find in the word list, or $ to exit: nother
Found nother

Type a word to find in the word list, or $ to exit: retweets
Sorry retweets is not in the word list
```

5 points
-2 points printing all words

3 points Program is same as P6_read1.py except for filename.
1 point Program correctly finds words in large word list
1 point Program quits, but not using any specific word to quit

P6_read_states1.py

Sample results

```
state_capitals_area_pop_v4
Type a state or start of state name or QUIT to end: w

Washington       Olympia          184,661 sq km    7.0 millions
West Virginia    Charleston        62,755 sq km    1.9 millions
Wisconsin        Madison          169,639 sq km    5.7 millions
Wyoming          Cheyenne         253,336 sq km    0.6 millions
Type a state or start of state name or QUIT to end:
```

5 points

1 point Read state data into a states list of sublists.
1 point Select states starting with specified letters
1 point Program converts area and population data into integer and float values
1 point Display state name and capital titled (Oregon, not oregon) in neat uniform width columns
1 point Display area in square kilometers with numbers lining up as shown, with commas in number and population figures line up nicely too as float numbers with one digit after decimal point.

Bonus available only if all prior points in read_states1.py met:
1 point
Display population density per square kilometer correctly:

```
state_capitals_area_pop_v4
Type a state or start of state name or QUIT to end: w

Washington       Olympia          184,661 sq km    7.0 millions     37.9 people/sq km
West Virginia    Charleston        62,755 sq km    1.9 millions     30.3 people/sq km
Wisconsin        Madison          169,639 sq km    5.7 millions     33.6 people/sq km
Wyoming          Cheyenne         253,336 sq km    0.6 millions      2.4 people/sq km
Type a state or start of state name or QUIT to end:
```

P6_read_states2.py

5 points

Also reads and processes the states_data.txt file and its comma separated list of values.

User search options are extended to include
S State name **Starts with**
C State name **Contains**
E State name **Ends with**
Q **Quit**

1 point Display menu of options

1 point Display states whose name starts with user-supplied letters

1 point Display states whose name contain the user-supplied letters

1 point Displays states whose name ends with the user-supplied letters
1 point Data lines up in neat columns for good readability

Starts with example

```
Options:
S Starts with
C Contains
E Ends with
Q Quit
Type the option you want: s

What letters does state name start with? te
Tennessee        Nashville        109,151 sq km    6.5 millions
Texas            Austin           695,621 sq km   26.4 millions
```

Contains example

```
Options:
S Starts with
C Contains
E Ends with
Q Quit
Type the option you want: c

What letters does state name contain? sh
New Hampshire     Concord           24,216 sq km    1.3 millions
Washington        Olympia          184,661 sq km    7.0 millions

Options:
S Starts with
C Contains
E Ends with
Q Quit
Type the option you want:
```

This example shows that after you finish each selection, your program re-displays the options.

Ends with example

```
Options:
S Starts with
C Contains
E Ends with
Q Quit
Type the option you want: e

What letters does state name end with? ia
California        Sacramento       423,970 sq km   38.3 millions
Georgia           Atlanta          153,909 sq km   10.0 millions
Pennsylvania      Harrisburg       119,283 sq km   12.8 millions
Virginia          Richmond         110,785 sq km    8.3 millions
West Virginia     Charleston        62,755 sq km    1.9 millions
```

Bonus 2 - 1 point

Note that you could display density (people per square kilometer); you could see if you can select states based on population density greater than a given number.

Here's an example of displaying density

```
Options:
S Starts with
C Contains
E Ends with
Q Quit
Type the option you want: s

What letters does state name start with? te
Tennessee         Nashville        109,151 sq km    6.5 millions     59.6 people/sq km
Texas             Austin           695,621 sq km   26.4 millions     38.0 people/sq km
```

If your programs selects based on density, it must also display density.