CSE 231 **Spring 2016**

Computer Project #04

Assignment Overview

This assignment focuses on the design, implementation and testing of Python programs to practice string manipulation, conditionals, and control loops, as described below.

It is worth 40 points (4% of course grade) and must be completed no later than 11:59 PM on Monday, February 8th.

Assignment Deliverable

The deliverable for this assignment is the following file:

`proj04.py` – the source code for your Python program

Be sure to use the specified file name and to submit it for grading via the **handin system** before the project deadline.

Assignment Background

Text editors on modern computers have advanced a long way from the earliest text editors. In this project, you will simulate one of these early text editors, where text could only be edited through a command-line interface. This editor can only modify words one at a time, and can only move through the string one word at a time. There is no point-and-click with a cursor!

Assignment Specifications

1. The program will have a menu loop that will prompt the user for a command. The available commands are as follows:

```
'n': Change to Next word
'p': Change to Previous word
'i': Insert a word
'e': Erase current word
'r': Replace current word
'c': Cut word, move to copy buffer
'v': Paste word from copy buffer to before current word
'l': Load a string
'h': Print available commands (Help)
```

2. Before asking for a command, begin by printing the components of the string. This editor will display the string as three parts: before the current word, the current word, and after the current word. For example, if the string is empty, this would be printed:

```
[][][]
```

If the string is “This is a test”, and if the current word is “a”, this would be printed:

```
[This is][a][test]
```

After the string is printed, ask for a command. Print the string components before prompting for a command.

3. *Navigating the sentence*: Commands ‘n’ and ‘p’ will move to the *next* or *previous* word in the sentence. The current word is retrieved from the respective parts of the line. If the above example was followed with the command ‘n’, the string would be updated as follows:

```
[This is a][test][]
```

If you cannot go further in the string in the given direction, the status does not change.

4. *Editing the current word*: Commands ‘e’ and ‘r’ will modify the current word.

- The command ‘r’ simply replaces the current word by asking the user for input. To keep things simple, when asking for a new word, assume the user will only enter a single word, without spaces.
- The command ‘e’ erases the current word. You must update the current word by taking the next word from the last part of the string (if the string isn’t empty), or from the first part of the string (if that string isn’t empty). The current word should only be an empty string after the erase command if there were no other words available.

5. *Insert word*: The command ‘i’ will insert a word before the current word. This will modify the first part of the string. The current word will stay the same.

6. *Cut and paste*: Commands ‘c’ and ‘v’ will cut and paste a word, respectively. This will utilize what’s called a “buffer”, to hold the word until it is pasted.

- The command ‘c’ will put the current word in a buffer. Once the word is in the buffer, the current word is erased. Use the same procedure for updating the current word as the erase command.
- The command ‘v’ will insert the buffer word before the current word. After this command is completed, reset the buffer to an empty string.

7. *Load a string*: The command ‘l’ will prompt the user for a string and replace all of the text with that string. This will result in the first word of the string being selected as the current word, with the rest of the string in the last part.

8. *Help and invalid commands*: The command ‘h’ will display all available commands with a brief description. If any other commands besides the required commands are entered, display a message telling the user of their error, and continue with the program.

9. All output should be formatted in such a way that it is easy to read. See the sample output for an example.

Assignment Notes

1. Items 1-7 of the Coding Standard will be enforced for this project.

2. You must NOT use lists to complete this project. This includes `string.split()`. Instead, use string methods such as slicing. To detect words, look for spaces using `.find()` and `.rfind()`. For example, `"this is a string".find(" ")` returns 4, and the reverse find `"this is a string".rfind(" ")` returns 9.

3. Consider the situations when spaces must be removed and added to the string components. If a space remains in the first and last components of the string, the `find` and `rfind` methods will detect that space, instead of the space on the other side of the next word. To remove spaces from the beginning and end of a string, use `strip()`. Remember that strings are immutable!

4. Consider when variables must be defined. If a variable is used in multiple iterations through the loop, it likely should be initially defined before the loop begins.

5. While completing most commands, it may be helpful to consider a "normal" case (such as when there is text in the first part of the string when completing a *previous* command), and "special" cases (such as when there are no more words for the *previous* command to find). Code the normal cases first, then consider how to handle special cases.

Suggested Procedure

- *Solve the problem using pencil and paper first.* You cannot write a program until you have figured out how to solve the problem. This first step may be done collaboratively with another student. However, once the discussion turns to Python specifics and the subsequent writing of Python statements, you must work on your own.

- Divide-and-conquer is a good approach. Complete the program for one option at a time. I started with a main loop (which makes more sense: `while` or `for` in this case?). Prompt before the loop and test that the loop stops when 'q' is entered. Then add an option: I think 'l' makes the most sense since it allows you to load a string to work on. I found that 'n' and 'p' were the next easiest. As mentioned above, don't worry at first about working at the ends of the line—those are special cases to handle later.

Sample Output

```
In [2]: runfile('C:/Users/spart_000/Downloads/proj04.bbf.py',
wdir='C:/Users/spart_000/Downloads')

----------------------------------------
Commands available:
    'n': Move to Next word
    'p': Move to Previous word
    'i': Insert a word
    'e': Erase current word
    'r': Replace current word
    'c': Cut word, move to copy buffer
    'v': Paste word from copy buffer to before current word
    'l': Load a string
----------------------------------------


Enter a command (h for menu; q to quit): q

In [3]:
```

```
In [1]: runfile('C:/Users/spart_000/Downloads/proj04.bbf.py',
wdir='C:/Users/spart_000/Downloads')

--------------------------------------
Commands available:
    'n': Move to Next word
    'p': Move to Previous word
    'i': Insert a word
    'e': Erase current word
    'r': Replace current word
    'c': Cut word, move to copy buffer
    'v': Paste word from copy buffer to before current word
    'l': Load a string
--------------------------------------


Enter a command (h for menu; q to quit): l

Input a string: love me do

[   ] [ love ] [ me do ]

Enter a command (h for menu; q to quit): i

Input input string: Beatles

[ Beatles ] [ love ] [ me do ]

Enter a command (h for menu; q to quit): e

[ Beatles ] [ me ] [ do ]

Enter a command (h for menu; q to quit): c

[ Beatles ] [ do ] [  ]

Enter a command (h for menu; q to quit): v

[ Beatles me ] [ do ] [  ]

Enter a command (h for menu; q to quit): n

[ Beatles me do ] [  ] [  ]

Enter a command (h for menu; q to quit): n

[ Beatles me do ] [  ] [  ]

Enter a command (h for menu; q to quit): q

In [2]:
```