Console-Based Editor - Basic - 2025

```
>
>iA simple, basic editor by CSC1002
A simple, basic editor by CSC1002
>.
A simple, basic editor by CSC1002
>w
A simple, basic editor by CSC1002
>w
A simple, basic editor by CSC1002
>w
A simple, basic editor by CSC1002
>
```

OVERVIEW

In this assignment, you are going to design and develop a simple, basic console-based editor. Unlike the modern, advanced editor which provides a sophisticated editing environment, utilizing the high-resolution of the graphical screen together with the mouse and the keyboard to position and adjust any text and figures displayed on the screen, giving us the WYSIWYG (What-You-See-Is-What-You-Get) experience.

In the early days, lacking access to a rich graphical display and mouse, the functionality of editors was limited, providing only a much simpler user interface, usually console-based. Editing was carried out based on simple text commands entered via the keyboard, commands such as inserting (i) and appending (a) a text string, positioning the editor cursor one character position to the left (h), one character position to the right (l), one-word position forwards (w), one-word position backwards (b), and so on.

```
A simple, basic editor by CSC1002
>h
A simple, basic editor by CSC1002
>h
A simple, basic editor by CSC1002
>h
A simple, basic editor by CSC1002
>h
A simple, basic editor by CSC1002
>l
A simple, basic editor by CSC1002
>l
A simple, basic editor by CSC1002
>$
A simple, basic editor by CSC1002
>^
A simple, basic editor by CSC1002
>$
A simple, basic editor by CSC1002
>a,Kinley/SSE
A simple, basic editor by CSC1002,Kinley/SSE
```

SCOPE

1. Complete all the following editor commands:

```
? – display this help info
. – toggle row cursor on and off
h – move cursor left
l – move cursor right
^ – move cursor to beginning of the line
$ – move cursor to end of the line
w – move cursor to beginning of next word
b – move cursor to beginning of previous word
i – insert <text> before cursor
a – append <text> after cursor
x – delete character at cursor
dw – delete word and trailing spaces at cursor
u – undo previous command
r – repeat last command
s – show content
q – quit program
```

   NOTE: Refer to the section “Specific Spec” for more information on particular requirements.

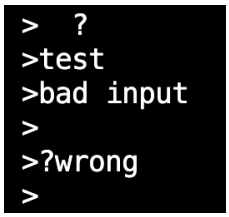
2. Case-sensitive commands - all editor commands are case-sensitive, for example, the capital letter ‘A’ does not equal the lowercase letter ‘a’.

3. Command types - most commands are single-letter (in lower case) commands (such as ?, $, x, ^, …etc), while some are two-letter (such as dw). Most commands do not require extra input, while a few do, such as insert (i) and append (a).

4. Command prompt (>) - The prompt is a single character string ‘>’. See the screenshots on the first page.

5. Command Syntax - “Command[Text]”, where “Command” is one of the commands shown in step 1, and “Text” applies only to commands requiring extra input such as insert (i) and append (a). Note: any commands whose description includes a substring enclosed in “<>” brackets require extra input “Text”.

6. Command Parsing - the user types a single command and then presses the return key to continue. Parse each command string according to “Command Syntax” to ensure that the input string matches EXACTLY one of the commands from step 1, including the extra input “Text” if required. When invalid input is entered, simply display another prompt as illustrated in the following screenshot.



a. Examples of valid command input:
    i. “$”
    ii. “^”
    iii. “h”
    iv. “x”
    v. “ahello world”
    vi. “i hello world “
b. Examples of invalid command input:
    i. “ $”
    ii. “   ?”
    iii. “? “
    iv. “ ahello world”
    v. “i”

7. Command Execution - the editor will repeatedly prompt the user to enter an editor command, execute the command, and then output the updated content on the display console (except for commands ‘?’ and ‘q’, see Note follows). After the updated content is displayed, the editor will display a new prompt on a new line. Refer to the section “Sample Output” for more examples.

   Note: when the help command (?) is entered, output only the help menu as shown in step 1; when the quit command (‘q’) is entered, terminate the program.

NOTE:

- Keep your entire source code in ONE SINGLE file.
- Use only Python modules as specified in the “Permitted Modules” section.
- In your design stick ONLY to functions, in other words, no class objects of your own.
    - Furthermore, the lines of code containing the sub-function(s) defined within another function will be counted as part of the parent function.
    - NOTE: Failure to adhere to the instructions outlined in the assignment handout will result in a 50% reduction in the coding style score.

SPECIFIC SPEC

1. Editor content - the editor shows its content, if any, as a single line of text string constructed by one or multiple Insert/Append commands. If the row cursor is enabled, it shows its position in a color such as green. When the editor program initially starts, its content is empty. Refer to the section "Sample Output" for more examples.

2. Row cursor - it's used to show where the cursor is on the current row if not empty. In other words, the cursor will appear on printable characters including space. The cursor is shown by wrapping a character with a pair of escape character strings such as "\033[42m" and "\033[0m". For example, given a string "hello world", to show the green cursor at the position of the letter 'e', this is the string to print: "h" + "\033[42m" + "e" + "\033[0m" + "llo world".

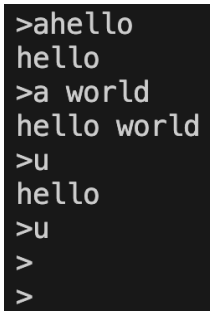
3. Insert - the given string "Text" will be inserted to the left of the cursor and the cursor position will be changed to the beginning of the "Text" string.

4. Append - the given string "Text" will be inserted to the right of the current cursor position and the cursor position will be changed to the end of the "Text" string.

5. Delete word - delete all characters from the cursor position to the beginning of the next word or to the end of the line.

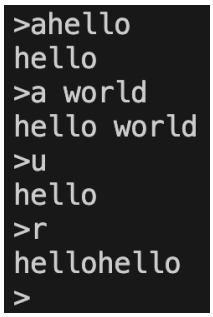
6. Cursor left and right - when repositioning the cursor to the left or right, one or multiple positions, and if the cursor is already at the far left or far right position, leave the cursor where it is.

7. Undo - it's used to reverse the change(s) made to the editor content including the row cursor positions based on the most recent commands. If multiple consecutive undo commands are executed, each will undo one command at a time in the reverse order that the commands were originally executed. For example, given the last 2 valid commands are "ahello" followed by "a world", the first undo command will reverse the "a world", and the second consecutive undo command will reverse "ahello". Refer to the following figure for an illustration.



8. Repeat - The "Repeat" command is used to re-execute the last valid command and it offers the convenience of sparing the user from retyping it again. The "Repeat" command is not applicable to the Undo and Help commands. For example, consider the command sequence: "ahello", "a world", "?", and "u". If the command "r" is subsequently entered multiple times, each Repeat command will always re-execute "ahello". Refer to step "Undo followed by Repeat" for another illustration.

9. Undo followed by Repeat - In this case, the “Undo” is not considered as the last command and the "Repeat" command is used to target the command immediately preceding the "Undo" command, not the most recent action performed. Any command entered prior to the "Undo" will be re-executed upon triggering the "Repeat" command. Refer to the following figure for an illustration.



ASSUMPTIONS

1. The goal of this assignment is to illustrate the benefits of “Problem Decomposition”, “Clean Coding” and “Refactoring”, all together achieving high code readability to ease logic expansion and keep high maintainability, therefore, it’s not aimed at designing a complex, general-purpose editor for handling large editing content.
2. It’s assumed that the length of each line is kept within a reasonable length so that each line can be stored directly using the standard Python ‘str’ type. The number of lines is also kept within a reasonable number so that all lines can be kept in one standard Python list and the lines can be efficiently updated using the standard list and str operations such as append, insert, slicing, cloning, …etc.
3. It is assumed that the user will not input a command that consumes excessive memory and leads to a buffer overflow (also called memory overflow or overrun) at runtime, such as inserting a very long string like “ihello …….. world.” In other words, all test cases executed against your program will be based on the commands from step 1 (Scope) with a short“Text” string.
4. Each test case is designed to evaluate the functionality and correctness of your program, rather than its speed, performance and memory usage. Each test case consists of multiple editing commands with short “Text”.
5. The text editor is required to handle only regular English characters, thus additional unicode support, if any, is unnecessary.

Startup Options

Not applicable

Skills

In this assignment, you will be trained on the use of the followings:

- Refactoring - logic reuse or simplification based on the existing logic.
- Variable scope: global, local and function parameters.
- Coding Styles (naming convention, meaningful names, comments, doc_string, …etc)
- Problem Decomposition, Clean Code, Top-Down Design
- Functions (with parameters and return) for program structure and logic decomposition
- Standard objects (strings, numbers & lists)
- Variable Scope

Permitted Modules

Only the following Python module(s) is allowed to be used:

- re (regular expression)

Deliverables

Program source code (A1_School_StudentID.py), where School is SSE, SDS, SME, HSS, FE, LHS, MED and StudentID is your 9-digit student ID.

For instance, a student from SME with student ID “119010001” will name the Python file as follows:

- A1_SME_119010001.py:

Ensure that your source file is saved in standard, regular UTF-8 encoding format. On the status bar of Visual Studio Code, you can view the current encoding format as follows:

On an occasion, the encoding scheme is set to UTF-8 with BOM as follows:

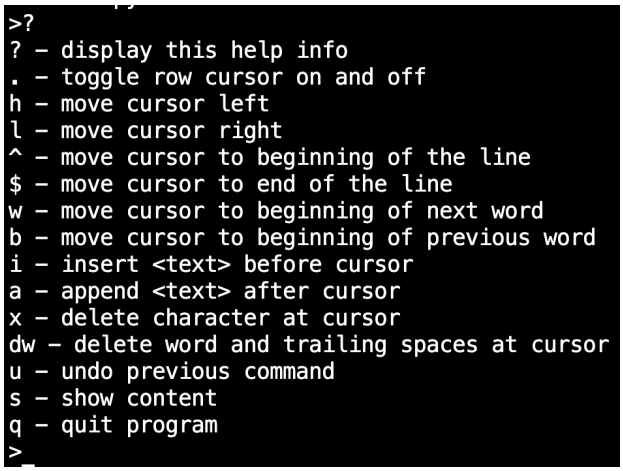
The presence of the Byte Order Mark (BOM) could be due to copying from websites, older version of editor, file conversion from other sources, default encoding setting, and so on.

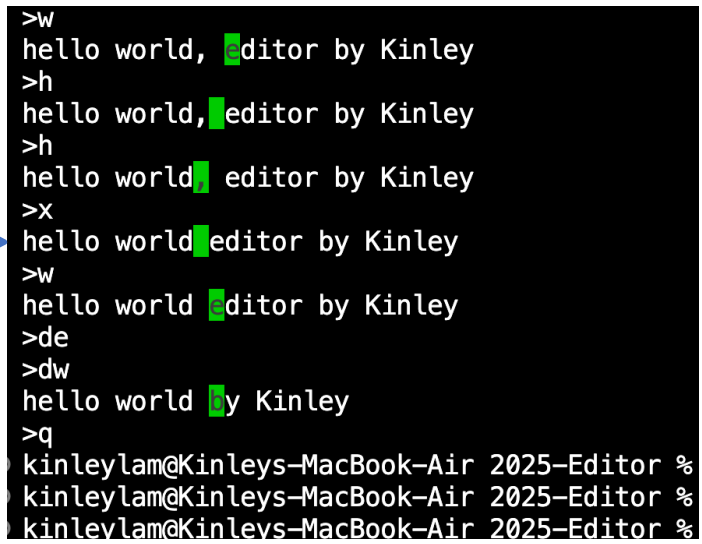
Confirm the encoding scheme is UTF-8 and the file name is correct, then submit the plain program file to the corresponding assignment folder. A deduction of 5% will be penalized if the file is incorrectly named or in wrong encoding format.

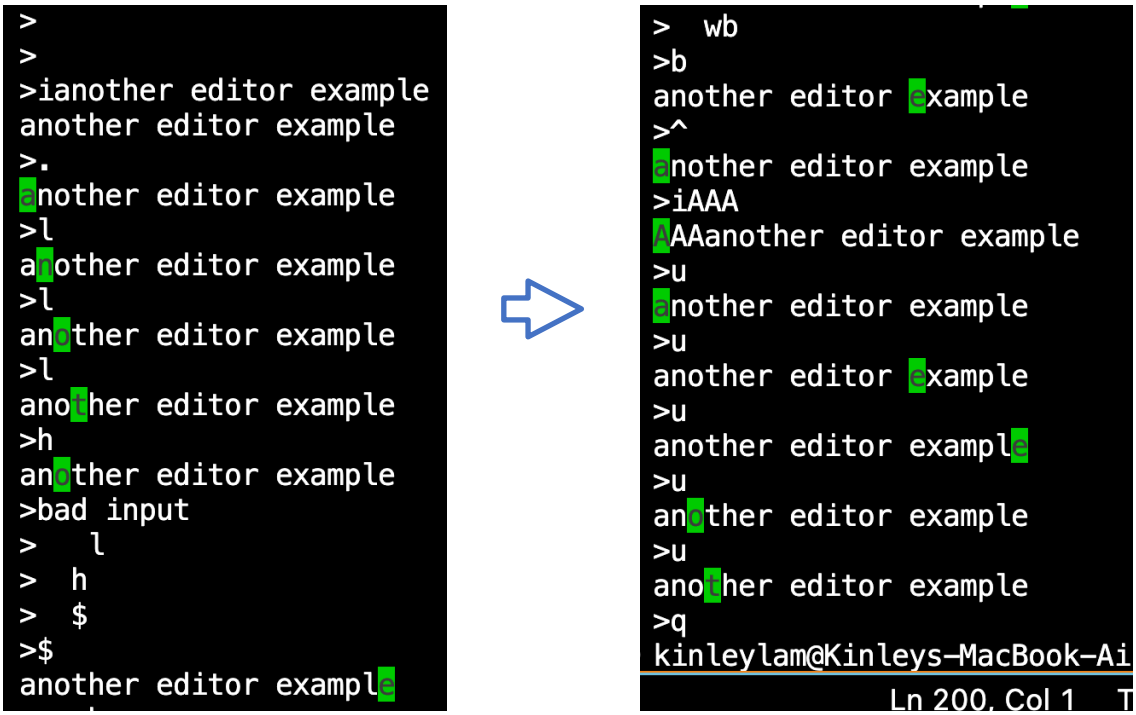
TIPS & HINTS

- Apply problem decomposition, Clean Code and Refactoring as illustrated during classes.
- Beware of variable scope as you might keep a few variables as global such as current editor content, cursor position, undo buffer, and so on.
- Refer to Python website for program styles and naming conventions (PEP 8)

SAMPLE OUTPUT






>
>
>ianother editor example
another editor example
>.
another editor example
>l
another editor example
>l
another editor example
>l
another editor example
>h
another editor example
>bad input
> l
> h
> $
>$
another editor example
> wb
>b
another editor example
>^
another editor example
>iAAA
AAAanother editor example
>u
another editor example
>u
another editor example
>u
another editor example
>u
another editor example
>u
another editor example
>q
kinleylam@Kinleys-MacBook-Ai
Ln 200, Col 1

Marking Criteria

- Coding Styles – overall program structure including layout, comments, white spaces, naming convention, variables, indentation, functions with appropriate parameters and return.
- Program Correctness – whether or the program works 100% as per Scope.
- User Interaction – how informative and accurate information is exchanged between your program and the player.
- Readability counts – programs that are well structured and easy to follow using functions to break down complex problems into smaller cleaner generalized functions are preferred over a function embracing a complex logic with many nested conditions and branches! In other words, a design with a clean architecture and high readability is the predilection for the course objectives over efficiency. The logic in each function should be kept simple and short, and it should be designed to perform a single task and be generalized with parameters as needed.
- KISS approach – Keep It Simple and Straightforward.
- Balance approach – you are not required to come up with a very optimized solution. However, take a balance between readability and efficiency with good use of program constructs.

Items	Percentage	Remarks
Coding Styles	30%-40%	0% if program doesn't run
Functionality	60%-70%	Refer to Scope

Due Date

March 2nd, 2025, 11:59:59PM