

PPALMS Unit Test Procedure

Authors:

1. Al Yaqdhan Al Maawali
2. Fahia Tabassum
3. Mulki Yusuf
4. Ahmed Al Raisi

Group: 04

Test Case 1 - Exclude Lines

Purpose: Excludes lines of code that the user selects

Expected Results: The selected lines will be removed correctly. A new view of the code with the remaining lines will be provided for the user in the terminal.

Test Data: Source code file. In our example, sc.txt and sample.txt.

Test Tools: PPALMS implementation code: ppalms.py

Dependencies: A source code text file with at least one line of code to exclude must be provided.

Initialization: The ppalms.py file is running and the terminal window is open to view the results of the exclusions of the source code lines.

Description: Selected lines in the source code are excluded from the file and the remaining lines are shown on the terminal.

For the unit test, an array of letters representing lines of code is used.

This is the test array used for the test results below:

```
test_lines = ["#a", "b", "c", "d", "#e"]
```

Exclude Test Cases	
<p>Case 1: No exclusions.</p> <p>Expected result: The terminal shows all of the elements with no removal of lines.</p>	<p>Result:</p> <pre> (base) MacBook-Air-153:Csci5801-P4 mulkiyusuf\$ python -u "/Users/mulkiyusuf/Desktop/Csci5801-P4/tests.py" Enter val of lines to exclude with space between vals or press enter for no exclusion: ----- 0 : #a 1 : b 2 : c 3 : d 4 : #e </pre>
<p>Case 2: Exclude lines of code in between the bounds of the indices of the test list</p> <p>Expected result: The terminal shows the remaining elements only. The selected lines have been excluded.</p>	<pre> (base) MacBook-Air-153:Csci5801-P4 mulkiyusuf\$ python -u "/Users/mulkiyusuf/Desktop/Csci5801-P4/tests.py" Enter val of lines to exclude with space between vals or press enter for no exclusion: 1 3 ----- 0 : #a 1 : c 2 : #e </pre>
<p>Case 3: Exclude lines of code out of the bounds of the indices of the test list</p> <p>Expected result: The terminal prints out “val of lines out of index bounds” and shows the initial list with no exclusions.</p>	<pre> (base) MacBook-Air-153:Csci5801-P4 mulkiyusuf\$ python -u "/Users/mulkiyusuf/Desktop/Csci5801-P4/tests.py" Enter val of lines to exclude with space between vals or press enter for no exclusion: 8 ----- val of lines out of index bounds 0 : #a 1 : b 2 : c 3 : d 4 : #e </pre>

Test Case 2: Grouping Lines

Purpose: Grouping lines of the code that the user selects

Expected Results: The selected lines will be grouped together correctly. The lines which are supposed to be grouped together will be placed in the subsequent row with sub-indices. The indices for the remaining lines will be modified according to the grouping of the lines.

Test Data: Source code file. In our example, sc.txt/sample.txt

Test Tools: PPALMS implementation code: ppalms.py

Dependencies: A source code text file with at least one line of code must be provided, the starting and ending bounds of the lines in the file should be maintained as well.

Initialization: The ppalms.py file is running and the terminal window is open to view the results of the grouping of the source code lines.

Description: Selected lines in the source code are grouped from the file and the remaining lines are shown on the terminal.

For the unit test, an array of letters representing lines of code is used.

Group Line Test Cases	
<p>Case 1: No Grouping of the lines</p> <p>Expected result: The terminal shows all the elements as they were without grouping any of the lines together.</p>	<pre> ----- 0 : #a 1 : b 2 : c 3 : d 4 : #e ----- group lines? (yes/no): no ----- 0 : #a 1 : b 2 : c 3 : d 4 : #e ----- remove comments? (yes/no): </pre>
<p>Case 2: Group lines together in between the bounds of the indices mentioned by the user.</p> <p>Expected Result: The lines within the bound will be grouped together and the rest of the lines will have an adjusted indices according to the grouping of the other lines. The grouped lines will have sub-indices based on the first index user had mentioned.</p>	<pre> group lines? (yes/no): yes ----- enter num of start of line to group: 0 ----- enter num of end of line to group: 2 ----- 0.0 : #a 0.1 : b 0.2 : c 1 : d 2 : #e ----- group more lines? (yes/no): </pre>

<p>Case 3: Bound Error</p> <p>Expected Result: If the user enters wrong bound information then the error message saying “bounds error” will be printed out and the terminal will ask the user if he/she still wants to group more lines.</p>	<pre>----- group more lines? (yes/no): yes ----- enter num of start of line to group: 4 ----- enter num of end of line to group: 0 ----- --bounds error-- ----- 0.0 : #a 0.1 : b 0.2 : c 1 : d 2 : #e ----- group more lines? (yes/no): </pre>
<p>Case 4: Grouping more lines</p> <p>Expected Result: The terminal will ask the user if he/she wants to group more lines.</p>	<pre>----- group more lines? (yes/no): yes -----</pre>

Test Case 3 - Remove Comments

Purpose: Removes comments from the source code file

Expected Results: User is prompted with whether or not they want comments removed from the source code text file. If yes is the answer, all comments in the source code text file are removed and the remaining lines are displayed on the terminal. Otherwise, the source code with the comments is displayed.

Test Data: Source code file.

Test Tools: PPALMS implementation code: ppalms.py

Dependencies: A source code text file with at least one line of code to remove must be provided.

Initialization: The ppalms.py file is running and the terminal window is open to view the results of the removal of the comments.

Description: Comments in the source code are excluded from the file and the remaining lines are shown on the terminal.

For the unit test, an array of letters representing lines of code is used.

These are the two test arrays:

```
test_lines = ["#a", "b", "c", "d", "#e"]  
test_lines_comments = ["b", "c", "d"]
```

Exclude Test Cases	
---------------------------	--

<p>Case 1: No comment removal</p> <p>Expected Result: In this case, the user answers no to the prompt so the comments are not removed. The lines with the comments included are displayed on the terminal.</p>	<p>Result:</p> <pre>remove comments? (yes/no): no ----- 0 : #a 1 : b 2 : c 3 : d 4 : #e -----</pre>
<p>Case 2: Remove the comments</p> <p>Expected result: The comments are removed. In this example, “#a” and “#e” are removed and the rest of the elements are displayed.</p>	<pre>remove comments? (yes/no): yes ----- 0 : b 1 : c 2 : d -----</pre>
<p>Case 3: No comments in the source code text</p> <p>Expected result: In this case, there are no comment lines in the list, so the contents of the lines are displayed on the terminal with no removal of any kind. For example, using would test_lines_comments = ["b","c","d"], as</p>	<pre>remove comments? (yes/no): yes ----- 0 : b 1 : c 2 : d -----</pre>

the test list prompts no removal since there are no comments.	
---	--

Test Case 4: Remove Empty Lines

Purpose: Removing the empty lines of the code from the file.

Expected Results: The empty lines of the file will be removed.

Test Data: Source code file.

Test Tools: PPALMS implementation code: ppalms.py

Dependencies: A source code text file with at least one line of code to exclude must be provided.

Initialization: The ppalms.py file is running and the terminal window is open to view the results of the grouping of the source code lines.

Description: The empty lines in the file will be removed after running the `rem_empty_lines(lines)` function.

For the unit test, an array of letters representing lines of code is used. Here the `test_lines_empty` array has been used to test the `rem_empty_lines(lines)` function.

```
test_lines = ["#a","b","c","d","#e"]
test_lines_comments = ["b","c","d"]
test_lines_empty = ["#a","b","c","","d","#e",""]
```

Remove Empty Lines	
<p>Case 1: Removing Empty lines</p> <p>Expected result: The terminal shows all the empty lines to be removed from the code file. If the user enters yes then all the empty lines(“”) will be removed from the code file.</p>	<pre>remove empty lines? (yes/no): yes ----- 0 : #a 1 : b 2 : c 3 : d 4 : #e</pre>
<p>Case 2: Not removing the empty lines</p> <p>Expected result: The terminal shows all the elements as they were without removing any of the lines from the file. So, if the user enters no then the will print out the code lines as before without changing anything.</p>	<pre>remove empty lines? (yes/no): no ----- 0 : #a 1 : b 2 : c 3 : 4 : d 5 : #e 6 :</pre>

Test Case 5 - Import the File

Purpose: Imports the file, opens it, and reads the contents.

Expected Results: User enters the file name, if it exists, the file is opened, read, and edited. Otherwise, a file not found message is displayed.

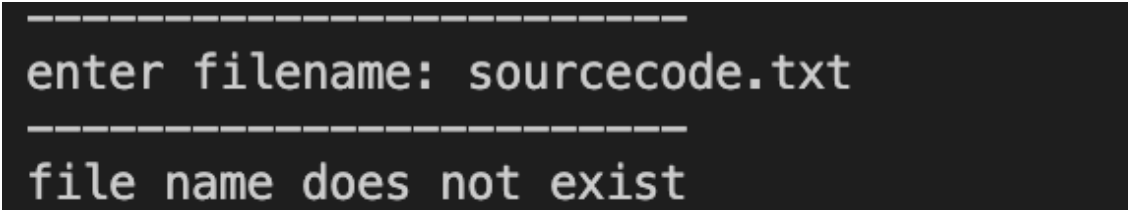
Test Data: Source code file.

Test Tools: PPALMS implementation code: ppalms.py

Dependencies: An existing source code file name.

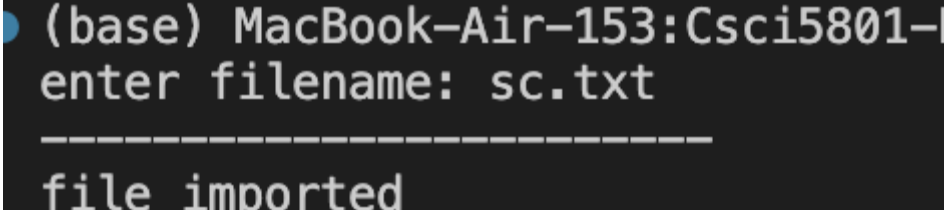
Initialization: The ppalms.py file is running and the terminal window is open to view the results of the test.

Description: User enters a filename to open, read, and make changes to.

Exclude Test Cases	
<p>Case 1: Filename does not exist</p> <p>Expected Result: The terminal displays: “file name does not exist”.</p>	<p>Result:</p>  <pre> ----- enter filename: sourcecode.txt ----- file name does not exist </pre>

Case 2: Filename exists

Expected result: The terminal displays: “file imported”.

A terminal window with a black background and white text. The prompt is "(base) MacBook-Air-153:Csci5801-". The user enters "enter filename: sc.txt". A dashed line separates the input from the output. The output is "file imported".

```
(base) MacBook-Air-153:Csci5801-  
enter filename: sc.txt  
-----  
file imported
```