# Laboratory 09 — Slices and Nested Loops

Topics covered:

* Slices on Sequences
* Nested Loops

## preparation

Lab attendance is compulsory. You will receive 1 mark for being present at the start of the lab and staying at least until the tutor has finished introducing the lab and has signed your attendance sheet.

## Exercises

The following exercises must be completed during your allocated laboratory time. You must show your work to the laboratory tutor who will sign off when the work is completed correctly.

### Exercise 9.1

[1 mark] Write the find\_longest() function which takes a string as the only parameter and returns the length of the longest line. It can tell the end of a line using “\n”.

**Arguments:** a string

**Returns:** the length of the longest line in the string

**Sample Tests:**

|  |
| --- |
| >>> find\_longest("the cat walks\naround the house\nclockwise\n")  16  >>> find\_longest("the cat walks around the house clockwise\n")  40 |

### Exercise 9.2

[1 mark] Write the find\_num\_lines() function which takes a string as the only parameter and returns the total number of lines. It can tell the end of a line using “\n”.

**Arguments:** a string

**Returns:** the total number of lines in the string

**Sample Tests:**

|  |
| --- |
| >>> find\_num\_lines("the cat walks\naround the house\nclockwise\n")  3  >>> find\_num\_lines("the cat walks around the house clockwise\n")  1 |

### Exercise 9.3

[1 mark] Write the print\_line\_length() function which takes a string as the only parameter and prints out the length of each line. It can tell the end of a line using “\n”.

**Arguments:** a string

**Returns:** prints the length of each line in the string

**Sample Tests:**

|  |
| --- |
| >>> print\_line\_length("the cat walks\naround the house\nclockwise\n")  13  16  9  >>> print\_line\_length("the cat walks around the house clockwise\n")  40 |

### Exercise 9.4

[6 marks] Write the pad\_lines() function which takes 2 parameters: a string, and a padding character and returns a string where all the lines are the same length, by adding padding character(s) to all the lines that are shorter than the longest. You can call the find\_longest() function you wrote in Ex 9.1 to find the length of the longest line in the string.

**Arguments:** a string and a padding character

**Returns:** the string with all the lines the same length

**Sample Test:**

|  |
| --- |
| >>> pad\_lines(" 1 @ #\n2 $ % \n3 ^ & \n", "\*")  ' 1 @ #\*\*\*\*\n2 $ % \*\*\n3 ^ & \n' |

## Homework Exercises

### The following exercises must be completed prior to the start of your Lab 10 session. Include all the exercises in a single module (file), named "Lab09\_Homework.py". Your file must include a docstring at the top of the file containing your name, UPI and ID number. You must submit the file containing your exercises using the Assignment Dropbox before the start of Lab 10.

### Exercise 9.5

Write the rotate() function which reads a file containing ascii art, prints it, then rotates it and prints it out again. Use the “dragon.txt” and “ascii\_small.txt” files provided to test your function. You may like to try finding more pictures at the url “http://artscene.textfiles.com/asciiart/”. You will need to call the functions find\_longest(), find\_num\_lines() , and pad\_lines() that you wrote in the lab. You may like to use the rotate\_matrix() function on Slide 18 Lecture 22 as a template for your function.

**Arguments:** an input filename

**Prints:** the ascii art file in its original form then in its rotated form

**Sample contents of input file:**

|  |
| --- |
| 1 @ #  2 $ %  3 ^ &  4 \* ( |

**Sample Test:**

|  |
| --- |
| >>> rotate("ascii\_small.txt")  1 @ #  2 $ %  3 ^ &  4 \* (    1234    @$^\*    #%&( |

### Exercise 9.6

[5 marks] Write the solve\_sudoku() function which solves a sudoku puzzle. This should work for a 9x9 board. The key to solving the puzzle is to only insert a new number when you have determined that there is only 1 possible number which can be entered into the combination of “row”, “column” and “square” which meet at this cell.

**Arguments:** a list containing the board of the sudoku puzzle to solve

**Prints:** a list representing the solved board.

**Other knowledge**: Use the description and sudoku functions from Lecture 24. Other functions are also provided in the file “my\_sudoku.py” which you can find in the Lab09 folder.

**Sample Test:**

|  |
| --- |
| >>> solve\_sudoku([0, 9, 8, 2, 7, 0, 0, 3, 0, 3, 0, 4, 0, 0, 6, 0, 0, 7, 1, 0, 0, 5, 0, 0, 2, 0, 4, 0, 0, 5, 0, 0, 0, 6, 0, 1, 7, 1, 0, 0, 0, 0, 0, 8, 9, 4, 0, 3, 0, 0, 0, 5, 0, 0, 6, 0, 1, 0, 0, 8, 0, 0, 3, 2, 0, 0, 1, 0, 0, 4, 0, 8, 0, 3, 0, 0, 4, 2, 7, 1, 0])  Solution: [5, 9, 8, 2, 7, 4, 1, 3, 6, 3, 2, 4, 9, 1, 6, 8, 5, 7, 1, 7, 6, 5, 8, 3, 2, 9, 4, 9, 8, 5, 3, 2, 7, 6, 4, 1, 7, 1, 2, 4, 6, 5, 3, 8, 9, 4, 6, 3, 8, 9, 1, 5, 7, 2, 6, 4, 1, 7, 5, 8, 9, 2, 3, 2, 5, 7, 1, 3, 9, 4, 6, 8, 8, 3, 9, 6, 4, 2, 7, 1, 5] |

## Advanced Exercises (optional)

### Exercise 9.7

[0 marks] Extend the three\_letter\_anagram() function given in Lecture 23 Slide 13 so that it a) checks that the anagram word is in the dictionary and b) that the list of words returned does not have any repeated words. Looking at the unique\_list() function from Lab 5 will help. Also using the dictionary as in Lecture 21 Slide 12 for the semordnilap() function will help.

**Arguments:** the filename of the file containing the dictionary (string), and the word to find anagrams of (string).

**Returns:** a list of anagrams of the string

**Sample Tests:**

|  |
| --- |
| >>> three\_letter\_anagram\_extended("unixdict.txt","cat")  ['act', 'cat']  >>> three\_letter\_anagram\_extended("unixdict.txt","see")  ['see'] |

### Exercise 9.8

[0 marks] Write the spelling\_corrector() function which takes 2 parameters: the name of a file which contains a dictionary, and a string with the word you are trying to correct, and returns a list of words from the dictionary that are an edit distance of 1 from the misspelled word. This function will call the following 3 functions : split\_word(), insert\_letter(), delete\_letter() which you write in the next 3 exercises (Ex. 9.9, 9.10, and 9.11)

**Arguments:** the filename of a dictionary, and a string containing a mispelled word

**Returns**: list of possible corrections

**Sample tests:**

|  |
| --- |
| >>> spelling\_corrector("unixdict.txt","see")  ['see', 'se', 'seed', 'seek', 'seem', 'seen', 'seep']  >>> spelling\_corrector("unixdict.txt","say")  ['say', 'sa', 'shay', 'slay', 'spay', 'stay', 'sway'] |

### Exercise 9.9

[0 marks] Write the split\_word() function which takes the word you are trying to split as a parameter and returns a list of all possible splits of this word.

**Arguments:** a string

**Returns**: list of word pairs. Each pair is a list of 2 strings.

**Sample tests:**

|  |
| --- |
| >>> split\_word("see")  [['see', ''], ['se', 'e'], ['s', 'ee'], ['', 'see']]  >>> split\_word("say")  [['say', ''], ['sa', 'y'], ['s', 'ay'], ['', 'say']] |

### Exercise 9.10

[0 marks] Write the insert\_letter() function which takes 1 parameter: a list of pairs of split words. It returns a list of possible words which have been formed by inserting an additional letter in the original string.

**Arguments:** a list of pairs of splits

**Returns**: list of strings

**Sample tests:**

|  |
| --- |
| >>> insert\_letter([['see', ''], ['se', 'e'], ['s', 'ee'], ['', 'see']])  ['zsee', 'szee', 'seze', 'seez', 'ysee', 'syee', 'seye', 'seey', 'xsee', 'sxee', 'sexe', 'seex', 'wsee', 'swee', 'sewe', 'seew', 'vsee', 'svee', 'seve', 'seev', 'usee', 'suee', 'seue', 'seeu', 'tsee', 'stee', 'sete', 'seet', 'ssee', 'ssee', 'sese', 'sees', 'rsee', 'sree', 'sere', 'seer', 'qsee', 'sqee', 'seqe', 'seeq', 'psee', 'spee', 'sepe', 'seep', 'osee', 'soee', 'seoe', 'seeo', 'nsee', 'snee', 'sene', 'seen', 'msee', 'smee', 'seme', 'seem', 'lsee', 'slee', 'sele', 'seel', 'ksee', 'skee', 'seke', 'seek', 'jsee', 'sjee', 'seje', 'seej', 'isee', 'siee', 'seie', 'seei', 'hsee', 'shee', 'sehe', 'seeh', 'gsee', 'sgee', 'sege', 'seeg', 'fsee', 'sfee', 'sefe', 'seef', 'esee', 'seee', 'seee', 'seee', 'dsee', 'sdee', 'sede', 'seed', 'csee', 'scee', 'sece', 'seec', 'bsee', 'sbee', 'sebe', 'seeb', 'asee', 'saee', 'seae', 'seea']  >>> insert\_letter([['say', ''], ['sa', 'y'], ['s', 'ay'], ['', 'say']])  ['zsay', 'szay', 'sazy', 'sayz', 'ysay', 'syay', 'sayy', 'sayy', 'xsay', 'sxay', 'saxy', 'sayx', 'wsay', 'sway', 'sawy', 'sayw', 'vsay', 'svay', 'savy', 'sayv', 'usay', 'suay', 'sauy', 'sayu', 'tsay', 'stay', 'saty', 'sayt', 'ssay', 'ssay', 'sasy', 'says', 'rsay', 'sray', 'sary', 'sayr', 'qsay', 'sqay', 'saqy', 'sayq', 'psay', 'spay', 'sapy', 'sayp', 'osay', 'soay', 'saoy', 'sayo', 'nsay', 'snay', 'sany', 'sayn', 'msay', 'smay', 'samy', 'saym', 'lsay', 'slay', 'saly', 'sayl', 'ksay', 'skay', 'saky', 'sayk', 'jsay', 'sjay', 'sajy', 'sayj', 'isay', 'siay', 'saiy', 'sayi', 'hsay', 'shay', 'sahy', 'sayh', 'gsay', 'sgay', 'sagy', 'sayg', 'fsay', 'sfay', 'safy', 'sayf', 'esay', 'seay', 'saey', 'saye', 'dsay', 'sday', 'sady', 'sayd', 'csay', 'scay', 'sacy', 'sayc', 'bsay', 'sbay', 'saby', 'sayb', 'asay', 'saay', 'saay', 'saya'] |

### Exercise 9.11

[0 marks] Write the delete\_letter() function which takes 1 parameter: the list of pairs of split words. It returns a list of strings which have one letter deleted from the original string.

**Arguments:** a list of pairs of splits of a word

**Returns**: list of strings

**Sample tests:**

|  |
| --- |
| >>> >>> delete\_letter([['see', ''], ['se', 'e'], ['s', 'ee'], ['', 'see']])  ['ee', 'se', 'se', 'see']  >>> delete\_letter([['say', ''], ['sa', 'y'], ['s', 'ay'], ['', 'say']])  ['ay', 'sy', 'sa', 'say'] |

## ASSESSMENT

Name:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Lab day and time: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Check list for laboratory exercises (to be completed by Lab tutor)**

|  |  |
| --- | --- |
| On time: 🞎 (1 mark)  Exercise 9.1: 🞎 (1 marks)  Exercise 9.2: 🞎 (1 marks)  Exercise 9.3: 🞎 (1 marks) | Exercise 9.4: 🞎 (6 marks)  Teaching Assistant: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Total mark: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/10 Tutor: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Marking Scheme

|  |  |
| --- | --- |
| Marks | Feedback |
| 0.5 | Includes a docstring at the top of the file containing your name, UPI and ID number. |
| 0.5 | Uses good descriptive variable names. |
| 1 | Include all the exercises in a single file. |
| 1 | The rotate() function is defined correctly. |
| 3 | Well done! Your rotate() function passed test case 1, 2, and 3 |
| 1 | The solve\_sudoku() function is defined correctly. |
| 3 | Well done! Your solve\_sudoku() function passed test case 1, 2 and 3. |