CSI2100-01 Spring 2020 Lab 11

Contents

1 General Information

- Starting with Lab 11, the specifications are provided in LaTeX format. In some places and to facilitate reading, we use "␣" to depict a space (blank) character, and "¤" for a "\n" (newline) character.
- Our lab problems are gradually outgrowing the capabilities of the OnlinePythonTutor. If you need to investigate a particular issue in your code, it is best to cut down the code to the issue you are interested in. For example, if you expect some data to show up in a list, then extract only the list updates of your program into a working example and copy/paste it into the OnlinePythonTutor.
- As an alternative to the OnlinePythonTutor, you may consider the PyCharm debugger for debugging your programs. We've used the PyCharm debugger in the lecture in week 10. The video lecture may be helpful to get you started.
- For the time being, no problems from Lab 11 can be tested with Hyeongjae Python. (Please check back regularly on YSCEC for updates on this issue.)
- **Because of our coronavirus precautions, you are provided with an excerpt of the relevant sections of the textbook together with this lab assignment.**

2 Programming Problems

Problem 1:

We're going to program a cellular automaton. A cellular automaton is a two-dimensional (2D) grid of square cells, each of which is in one of two possible states: alive or dead. Every cell interacts with its eight neighbours, which are the cells that are horizontally, vertically, or diagonally next to it. At each step in time, the following transitions occur:

1. Any live cell with fewer than two live neighbours dies, as if caused by under-population.
2. Any live cell with two or three live neighbours lives on to the next generation.
3. Any live cell with more than three live neighbours dies, as if by overcrowding.
4. Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.

Please download the "CellularAutomatonTemplate.py" file from the "Labs" board on YSCEC. This file contains a code-template that you should use for your implementation. Additional information is as follows:

(*a*) We use a two-dimensional grid of cells. The default dimensions are 20x20 grid cells. The user can override the dimension, but the grid is always square, and the minimum dimensions are 20x20 (see the below input example where the user just presses 'enter' and the default 20x20 grid is thus selected). If the user would input 25, the grid would be 25x25 cells. If the user inputs a value smaller than 20, your program should use the default 20x20 grid. If the user input is faulty, your program should use the default 20x20 grid (but please note that your program is not allowed to crash on faulty user input!).

The second input from the user specifies the number of generations to be computed. You should compute and output each generation of cells from generation 0 up-to-and-including a limit that the user inputs:

```
Grid␣sidelength␣(default␣20):␣¤
Max␣generation:␣10¤
```

(program output omitted).

If the user inputs 10 for the maximum generation, your program should print generations 0...10 (11 generations in total).

The user's input for the number of generations must be an integer larger than 0. Your program should keep prompting the user if the input is faulty (note again that your program is not allowed to crash on faulty user input).

Hint: Because your program is not allowed to crash on faulty user input, you must wrap your code that handles user input with a try-block and an appropriate error handler.

(*b*) Each generation should be output in the following format. An "x" represents a live cell, a blank character represents a dead cell, and "-" and "|" are used to delimit the boundary of our grid. After each line of the 2D grid, the corresponding row number is printed. For example:

```
---------------------
| x                 | row 0
| x                 | row 1
| x                 | row 2
|                   | row 3
|                   | row 4
|                   | row 5
|                   | row 6
|                   | row 7
|                   | row 8
|                   | row 9
|         xxx       | row 10
|         x         | row 11
|         xxx       | row 12
|                   | row 13
|                   | row 14
|                   | row 15
|                   | row 16
|                   | row 17
|                   | row 18
|                   | row 19
---------------------
```

(*c*) You must decide on a data-structure to represent your 2D grid of cells. One way is to use a list of 20 elements, where each element represents one row of the grid. Each element is therefore a sublist of 20 elements, which represent the cells of one row.

(*d*) You must create and initialize your data-structure. Please consider the "*" operator which we used already in Lab 10 to initialize an empty list of 26 counters. Please make sure that you do not introduce aliases. You must create 20 different sublists and add them to the top-level list.

To represent individual cells, you can, for example, use the value 1 for a living cell, and 0 for an empty grid-element.

(*e*) You must initialize your automaton to the layout of generation 0 shown above (please download the file SampleOutput.py from YSCEC, which provides you with generation 0 and generation 1 (generation 1 you can use to compare with the output of your program, for debugging purposes). Generation 0 is the initial generation that your program should output. From Generation 0, your program should compute generation 1, generation 2 and so on. Each generation is computed from the previous generation according to the rules (a)–(d) stated at above.

Please note: if the user selects a larger grid, e.g., 25x25, then you should still use the provided generation 0 for initialization. It means that the additional grid space must be initialized to empty cells. After several generations, the cellular automation may grow into this additional space.

(*f*) The template contains an infinite loop, for you to observe the successions of generations. With your submission, you should replace this infinite loop by a loop that computes the number of generations asked by the user.

(*g*) The template uses the `sleep()` function from the time module to pause for one second. Arrange your program in such a way that between the output of any two generations, the program pauses for one second.

(*h*) Please pay special attention to the cells next to the boundary of the 2D grid. Such cells have fewer neighbor cells. Avoid out-of-bounds accesses of your 2D grid!

(*i*) The template uses the `size` variable to set the size of the 2D grid. Your program must use this variable. In particular, your program must work correctly if the value of this variable is changed. You may assume that we do not shrink the grid below 20x20 cells.

(*j*) **40% Extra Lab 11 Credit**: you may hand in an **additional** cellular automation implementation, which displays all output in a Python Turtle window. Your program should display a generation, pause for one second, display the following generation, and so on.

You are also allowed to use a different GUI than the Python Turtle module. However, in this case you must hand in your code **and** demonstrate the program on your laptop or lab-PC during the Lab 11 inspection to receive the extra credit. Only implementations based on open-source libraries will be accepted.

Problem 2:

Modify Lab 10 Problem 2 such that the function returns `-1` if any one of the files f1, f2 or f3 cannot be opened. Otherwise, the function should return `0`.

Hint: Your function must be able to catch the `OSError` exception that will occur with the `open()` function in the error-case.

```
def copyFiles(f1, f2, f3):
  """ Copies files f1 and f2 onto file f3.

      If f1, f2 or f3 cannot be opened, -1 is returned.
      Otherwise, the copy operation is performed and 0 is returned.
  """
```

Problem 3:

Problem M6 on page 334 of the textbook.

(*a*) **NOTE: the program on pages 312–313 of the textbook has a bug.** Therefore you must refer to the version provided on YSCEC. The provided version is the one we discussed in the lecture (please refer to the Lecture 8 slide stack).

(*b*) Your program must count all the individual words in the file. Counting is case-**in**sensitive. For example, “foo”, ”Foo”, “FOo” etc. all stand for the same word.

(*c*) After counting, your program must output an alphabetically sorted list of the words and their counts in the requested file.

(*d*) If the input filename is “wordtest.txt”, the output filename is “wordtest.wc”. You can assume that the user will always provide a file with a 3-character extension after the dot as the input file.

(*e*) Please write all counts in lowercase to the file. For example:
```
ada:␣1¤
beta:␣22¤
```

```
brown:␣1¤
foo:␣1¤
xerox:␣1¤
zeta:␣22¤
```

Problem 4:

Problem M7 on page 334 of the textbook.
To solve this problem, you need to do a bit of data-engineering. Real-world data is frequently not perfectly suited for processing by a computer. The following list gives you several hints on how to "engineer" the data to process it with your program.

(*a*) Please note that the air pollution data stated with Problem M7 in the textbook is not in CSV format. From the data stated in the textbook, you should prepare your own file. The name of your file must be Air_Pollution_Data.csv. In your CSV file, you're not allowed to omit any of the US states or change the order of the states. Other than that, you can come up with your own CSV format for this file. Please hand in your CSV file with your code (see "Deliverables" below). Your program is only required to work with your own CSV file.

(*b*) Please note that the data items in the air pollution data set and the lung cancer data set do not match 100%. If a data item x occurs only in one data set, then your program should drop that data item from the correlation computation.

(*c*) Note also that the air pollution data set is given in a different order than the lung cancer data set. You must make sure that you correlate the corresponding data items (New York with New York, Montana with Montana, . . .).

(*d*) Please find the lung cancer and the smokers data sets in the "Labs" folder on YSCEC. The filenames are CDC_Lung_Cancer_Data.csv and CDC_Cigarette_Smoking_Data.csv.

(*e*) Please find the source-code (file Smoking_Cancer_Correlation.py) of the smoker-cancer correlation program on YSCEC. Please adapt the program greeting according to your own will and taste. When grading, we will only consider the computed r_value.

Please do keep the code "`print('r_value = ', correlation)`", **including the blank after** '='.

(*f*) To read data, you can prompt the user as follows:
Enter␣lung␣cancer␣data␣from␣filename:␣
Enter␣air␣pollution␣data␣from␣filename:␣

(*g*) **Note:** the provided source-code does not close files. You should fix this in your solution.

3 Marking Criteria and Plagiarism

Marking Criteria

- Score is only given to programs that compile and produce the correct output with Python version 3.
- Points are deducted for programs that are named wrongly. See the list of deliverables for the required file names.
- Points are deducted for programs that produce warnings.
- Points deductions on programming style:
  - Please provide comments in your code.
  - Please provide docstrings with all your functions. It is not necessary to provide docstrings for functions in code provided to you. If the lab specification already provides a docstring for a function you are asked to implement, you can use the provided docstring.
- Please pay particular attention to the requested output format of your programs. Deviating from the requested output format results in points deductions.
- Use of modules:
  - You are allowed to use the `sys` module with all programing problems.
  - Please always use an exit code of 0 with the `exit()` function of the `sys` module: `sys.exit(0)`.
  - You are allowed/asked to use modules stated with individual programming problems and contained in provided code (if any).
  - No other modules are allowed.

Plagiarism (Cheating)

- This is an individual assignment. All submissions are checked for plagiarism.
- Once detected, measures will be taken for **all** students involved in the plagiarism incident (including the "source" of the plagiarized code).

4 Deliverables and Due Date

Problem	File name	Comment
1	lab11_p1.py	
	lab11_p1_GUI.py	(optional, only for extra credit)
2	lab11_p2.py	
3	lab11_p3.py	
4	lab11_p4.py Air_Pollution_Data.csv	

- Please submit all files in a ZIP-file named lab11_<student_id>.zip
- Please submit your archive on YSCEC by Wednesday, June 10, 23:00.