

WASHINGTON STATE UNIVERSITY VANCOUVER

PROGRAM DESIGN AND DEVELOPMENT - CS 121

Assignment 5 Part 2

Professor:
Ben MCCAMISH

October 13, 2020

Overall Assignment

Your code should all be contained in a single file (excluding additional files I provide) titled `assignment5p2.py`. Your program must be written for Python 3.6+.

This week we will continue with the programming of our game. You may need to reuse some of your code from Part 1.

1 Getting started

Create a new file called `assignment5p2.py` and put the a header at the top of that file similar to this:

```
# Assignment 5 Part 2
# Jane Smith
# October 8, 2019
```

Alternatively, you may use the template provided in autolab, but the comment at the top of the file is still required.

2 Displaying the lists graphically

At the moment, your random-evolver for lists displays its interim generations using text only. This part will use Python's graphics to display those lists. In addition, we will use graphics-based input in order to interact with the functions.

The `csplot.py` module contains a function named `show()` that will display lists as patches of color in a window. To use `csplot`, follow these steps:

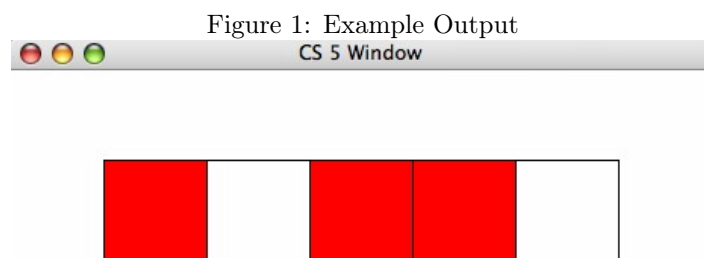
1. Be sure that you move `csplot.py` to the same directory that your `assignment5p2.py` file is located.
2. Find `csplot.py` in the handout folder.
3. Include the line from `csplot` `import *` at the top of your file near the other import lines - this will provide access to the graphics functions in `csplot`.
4. Include the line `show(L)` right above the `print(L)` line that is already in your `runGenerations` function. This is the command to display your list graphically.

2.1 How does it display lists?

Try the function call

```
show( [1,0,1,1,0] )
```

and you should see a window that looks like the following:



3 Accepting user input

The approach to evolving lists thus far is, well, a bit too random. This section will enable the user to guide the process by clicking on the graphical representation of the list. Replace your `evolve` and `setNewElement` functions by copying these at the bottom of your file:

```
def evolve( L ):
    """ evolve takes in a list of integers, L,
    and returns a new list of integers
    considered to be the "next generation"
    """
    N = len(L) # N now holds the size of the list L
    x = sqinput() # Get mouse input from the user
    return [ setNewElement( L, i, x ) for i in range(N) ]

def setNewElement( L, i, x=0 ):
    """ setNewElement returns the NEW list's ith element
    input L: any list of integers
    input i: the index of the new element to return
    input x: an extra, optional input for future use
    """
    if i == x: # if it's the user's chosen column,
        return choice( [0,1] ) # return a random 0 or 1
    else: # otherwise
        return L[i] # return the original
```

Alternatively, you can use the template provided in the handout that contains the updated functions. **Note** that `sqinput()` is a function from the `csplot` module which takes mouse input from the user. It returns the index of the square closest to the mouse click.

Try running `runGenerations([0,0,0,0,0,0])`. Now, the execution should pause and wait for you to click on one of the squares in the window. Note that the square does not simply change from 0 to 1 and 1 to 0 - the call to `choice` randomizes the result.

3.1 Toggling Lights

Change the `setNewElement` function so that the light the user clicks on toggles from 0 to 1 or from 1 to 0, as appropriate.

Hint: if the old value of the light is `L[i]`, what will you get if you subtract that value from 1?

Be sure to test your code by running

```
runGenerations( [0,0,0,0,0,0] )
```

Admittedly, the “game” is not difficult to win in this case, but the next part of the lab adds the wrinkle that makes it much more challenging.

4 Setup Done, now questions

4.1 Question 1 (10 points)

Now, you are ready to implement a full 1D version of “Lights On”. Modify your code so that the game play is as it should be (i.e., when you toggle one light, the lights next to it also toggle). Remember, lights do not “wrap around”. That is, the lights at the edge of the board have only one neighbor. The function `runGenerations(L)` should then play the game from the initial starting list `L`.

Suggestions: Only `setNewElement` needs to change in order to implement this game. You will need to compare the value of `i`, which is checked for each light in the row, and `x`, which is the user’s choice.

4.2 Question 2 (10 points)

Create a function named

```
randBL( N )
```

that takes in a nonnegative integer, `N`, and returns a list of length `N` in which each element is equally likely to be a 0 or a 1. Raw recursion is one way to handle this; list comprehensions are another. This `randBL` function makes it easy to start a new, random game. Try running

```
runGenerations( randBL(9) )
```

to be sure it works.

4.3 Question 3: The final 2D version (40 points)

The `show` command will display 2d grids of lights as well as the 1d rows that this lab has used thus far. For example, try:

```
show( [ [0,1,0], [1,1,0], [0,0,1] ] )
```

You will see a three-by-three grid of lights appear. Each element of the input list to `show` is, itself, a row of the resulting grid of squares. The rows are plotted from low-to-high in the window.

For this question, implement the 2d game of lights out by writing a new function named `runGenerations2d(L)`. Your program should quit and print some victory line once the entire board is filled up (required). Clicking on a square should change its neighbors (if any) to the north, east, west, and south. You will need to rework some of the helper functions in order to implement `run2dGenerations` – rather than write over or redefine your existing functions, create new names for them. For example, you may need

1. `evolve2d`
2. `setNewElement2d`
3. `allOnes2d`
4. `randBL2d`

You will also need the function `sqinput2()` which returns 2d data, as follows:

```
x, y = sqinput2()
```

Warnings:

Make sure you understand exactly how the displayed squares and their coordinates correspond to the positions in the list. If you copy and paste your code (recommended) be sure to change all the function calls within each function to their 2d counterpart.

4.4 Question 4 (20 points) - Letting the computer play...

Don't be selfish – Write new versions of `evolve` and `evolve2d` so that the computer gets to play! That is, have computer choose (randomly) possible indices and the game continues until it's solved. Warning: one-sixth of all the initial binary lists are dead ends! That is, there is no combination of toggled lights that will end up at the all-ones list. So, letting the computer play for a long time on one of those dead-ends will result in Python running out of memory (or reaching its recursion limit).

Commenting/Style (20 points)

Your code will be examined for comments and style. This means that you should include reasonable comments in your code. You might comment and add a small description for each function you create. If there is a particularly complex line of code, then you may comment that single line. Your code should also contain a `main()` function that has all of the

test code that you used. No code should exist outside of a function. **Note:** You must remove all comments and `pass` statements that came with the template to receive full marks.

Autolab Notes:

- Submit only your `assignment5p2.py` file.
- Autograder has its limits, so it cannot test using inputs from the user. Thus, you will have to develop your own tests. Ask the TA or I if you are having trouble coming up with good edge cases. Get started early so you have plenty of time to test!!
- Make sure to include a main function.