Day 2: Rule 110

We will essentially be creating an ASCII art version of (one kind of) this program:

<https://elife-asu.github.io/wss-modules/modules/1-1d-cellular-automata/>

The program is essentially going to be creating a new list (of the same length) of ▢s and ▩s starting with an initial string and applying certain rules. We are going to represent ▩s as 1s and ▢s as 0s. For example, the strings below mean the same thing.

000010011111001011011000101111

▢▢▢▢▩▢▢▩▩▩▩▩▢▢▩▢▩▩▢▩▩▢▢▢▩▢▩▩▩▩

There is a rule that says that a substring 000 becomes 1 ie.

000

1

That means we would have in the next line

000010011111001011011000101111

1 1

That is, the second (index 1) and 22nd (index 23) characters are both 1. But this is only part of the string. We need to use the rest of the rules to build the rest of the string. Here is the full list of the string generation rules:

000 100 010 110 001 100 010 111

1 0 1 1 0 1 1 1

| **Try this:** Calculate the next string using the rule above. Remember that I’ve given you two already.  000010011111001011011000101111  \_0\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_0\_\_\_\_\_\_\_  *What problems did you encounter?* |
| --- |

The process of generating strings—in this case of ▢s and ▩s—is what’s called a *cellular automaton*. The one we are playing with above is called *Rule 110*. They are a staple of certain branches of theoretical computer science, and generalize to higher dimensions, like in [Conway’s Game of Life](https://playgameoflife.com/).

# Part I: Getting the Input File

We are going to generate a new string of 0s and 1s from a given string in a file called *art\_input.txt*. Make sure you have this file in your CSCI204 folder before you begin. You are going to be writing your code in the “main” branch of your program (the part that runs when you run the python interpreter).

# The Program

if \_\_name\_\_ == "\_\_main\_\_" :

## Introduction

print('''

----------------------------------------------

| Welcome to the Art Generator Device |

| Authors: |

| Date: |

----------------------------------------------

''', end="\n")

## Main ##

<— Don’t forget to indent!

To open a file into a buffer, you can write :

file\_buffer = open("filename.txt", "r")

# do something with the file

file\_buffer.close()

This manually opens and closes (also important) the file buffer with a string representation of the file. The command file\_buffer.readline() converts the *next* line in the file (starting with the first one) to a string. Try running this:

file\_buffer = open("art\_input.txt", "r")

# do something with the file

line = file\_buffer.readline()

print(line)

file\_buffer.close()

Let’s convert this line of text into a string of 1s and 0s (the number kind, not the string kind).

| **Try this:**   * Use the focus\_list = line.split("art\_input.txt") method to split the string into a list of characters. * Use *list comprehension* to convert the list of characters into integers. focus\_list = [int(char) for char in line] * Now print out focus\_list to make sure you did this right! |
| --- |

| **Try this:**   * After you close the file, open up the file “art\_output.txt” in *write* mode (use “w” instead of “r”). Python will create a file with this name if one does not already exist. * Apply the function stringify(item\_list) to focus\_list and write the resulting string to art\_output.txt using file\_buffer.write(focus\_list) * Run that and go find art\_output.txt to check that the right thing happened (you might need to use the terminal to change directory to the right folder). |
| --- |

# Part 2: Complete the apply\_cell\_rule() Function

**Before you continue reading:** Take a bit of time with your group to figure out how you are going to implement the function that creates a new list of 0s and 1s by applying the rules from the beginning of the document.

**Hint:** Start by adding a zero to the left and to the right of the input string.

| **Try this:** Use a for loop and slicing input\_list[i : i + 3] to loop through all the lists of three consecutive digits. Append the 0 or 1 given by the rule to an output list that the function then returns. |
| --- |

# Part 3: Make Art

**Before you continue reading:** Think with your group about how you might do this. Will you

* create a string variable and put your art entirely inside it?
* Or will you append a new line to a file every time you use apply\_cell\_rule()?

How will you make sure that every line of art goes on a new line?

| **Try this:** Use a for loop to repeat the process of calculating a new line in your ASCII art about 20-30 times. Make sure you apply stringify(item\_list) before you append the line. Either create a string variable and put all those lines in there before writing the whole thing to art\_output.txt, or append each line to art\_output.txt file one-by-one. |
| --- |

Bonus

* Try out other rules in the same format as Rule 110.
  + For eg., try out Rule 22   
    
* Allow the user to input their own names for the input and output text files so that the program doesn't always just read from "art\_input.txt" and write to "art\_output.txt".
* Allow the user to start with a random string of 0s and 1s instead of "art\_input.txt".
* Allow the user to change the size of the piece of art.
* Change the output symbols. Copy/paste these if you like (or Google to find your own):

♥︎ ▲ ◈ ◯ ◳