Conway's Game Of Life

June 13, 2018

Preparing for Conway

You will need to install some special software for today's lesson.

1. Find the search bar on your computer.
2. Type "cmd".
3. At the command prompt, run "py -m pip install matplotlib".
4. If you get any error messages, please raise your hand for us to help.

2D Lists

In this lecture, you will need to understand two-dimensional list.

1D Lists

- A list is considered one-dimensional.
  - It represents a sequence of elements.
  - There's a first element.
  - There's a last element.
  - The list has a number of elements.
  - You can index those elements using square brackets.

2D Lists

```
8 6 7
5 3 0
9 4 6
```

This is a grid of elements. Like 1D lists, you can index into the list but now we need two indices. The first index is always the row. The second index is the column.

2D Lists

```
8 6 7
5 3 0
9 4 6
```

The number 8 appears in the first row and the first column, so we say that its location is at “0,0”. The row comes first and the column comes second. Computer scientists begin counting with 0!

2D Lists

```
8 6 7
5 3 0
9 4 6
```

The number 6 appears in the first row and the second column, so we say that its location is at “0,1”.

2D Lists

```
8 6 7
5 3 0
9 4 6
```

The number 5 appears in the second row and the first column, so we say that its location is at “1,0”.

2D Lists

```
8 6 7
5 3 0
9 4 6
```

- What is the location of 3?
- What is the location of 0?
- What is the location of 4?
- What is the location of 6?

2D Lists

```
8 6 7
5 3 0
9 4 6
```

- What is the location of 3? (1,1)
- What is the location of 0? (1,2)
- What is the location of 9? (2,0)
- What is the location of 6? (2,2)

2D Lists

- A 2D list is considered two-dimensional.
  - It represents a grid of elements.
  - It has both rows and columns.
  - The number of elements in the list is the rows times the columns.
  - You need two indices to index into the list.

Numpy

We use a special library for two dimensional lists named “numpy”. Start up Idle and type this:

```
import numpy as np
```

“np” is the common shorthand for “numpy”. “numpy” is used by scientists for manipulating large datasets.

Creating a 2D list

This is the grid that we used in our examples. Type this line. Here, we define an array with 3 rows and 3 columns. We define them one **row** at a time.

```
x = np.array([[8, 6, 7], [5, 3, 0], [9, 4, 6]])
```

Check our work.

Now we can check our work from earlier.

```
x[1,1]
x[1,2]
x[2,0]
x[2,2]
```

Overwriting values.

We can overwrite values in a 2D list using this approach.

```
x[2,2] = 42
x
```

Let's visualize.

Numbers are fun, but visualizations are more fun. Make sure that you have all of the following libraries imported. You should already have the first one.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
```

Let's visualize.

In this example, the elements equal to 255 represent "ON" elements and the ones with 0 represent "OFF".

```
x = np.array([[0, 0, 255], [255, 255, 0], [0, 255, 0]])
plt.imshow(x, interpolation='nearest')
plot.show()
```

What colors does python assign to "ON" and "OFF" using this approach?

Make a smile!

```
x = np.zeros((5,5))
x[1,1] = 255
x[1,3] = 255
x[3,0] = 255
x[4,1] = 255
x[4,2] = 255
x[4,3] = 255
x[3,4] = 255
plt.imshow(x, interpolation='nearest')
plt.show()
```

Let's make more!

This will make a 400 by 400 grid with a randomly chosen value (0 or 255) for each element. How many elements are in the grid?

```
width = 400
height = 400
x = np.random.choice([0,255], (width, height))
plt.imshow(x, interpolation='nearest')
plt.show()
```

Conway's Game of Life

Conway's Game of Life simulates the people in your life.

- A typical person needs friends.
- If you have too few friends, you get lonely.
- If you have too many friends, you get overcrowded.

In this program, each element in a 2D list represents either a living person or an empty cell. RED means living. BLUE means empty.

Conway's Game of Life

A man by the name of John Conway wanted to simulate life using a simple set of rules. Here are all four rules which we will program.

- If a cell is living but has fewer than two friends, it dies of loneliness.
- If a cell is living and has two or three friends, it continues to live.
- If a cell is living but has more than three friends, it dies of being overcrowded.
- If a cell is empty and has exactly three friends, a person is born at that location.

In this simulation, 255 means ON and 0 means OFF.

How many friends do you have?

In order to determine how many friends an element has, we count the neighbors.

```
(i-1, j-1)    (i-1, j)    (i-1, j+1)
(i, j-1)      (i, j)      (i, j+1)
(i+1, j-1)    (i+1, j)    (i+1, j+1)
```

In order to count the neighbors of an element, we count how many of the 8 cells surrounding a cell that are equal to 255. In a 3 by 3 grid, a cell has up to 8 neighbors.

What about the edge of the map?

- In old Atari games (ask your parents), if something passed over the edge of the screen, it usually repeared on the other side of the screen.
- We can do this with the remainder operator. Let's say that we have a 100 by 100 board. N=100
  - For example, if i=99 and j=99, then (i+1, j+1) becomes (100, 100). This is an illegal index.
  - But instead if we say ((i+1)%N, (j+1)%N), we then get (0,0). This is a legal index and on the opposite corner of the map.

If you see a formation move off the right side of the map, it will reappear on the left side.

Basic Steps to Game of Life

1. Initialize the cells in the grid to either 0 or 255.
2. For generation, we will do the following:
   - 2.1 Create a new, empty grid that a copy of our existing grid.
   - 2.2 Update the new grid based on the number of neighbors in each cell (i, j) of the hold grid.
   - 2.3 Move the new grid into the old grid. (Now both grids are the same.)
   - 2.4 Display the grid.

Begin coding.

Let's begin by creating a new Python script named "conway.py". Add these libraries to the beginning.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

ON = 255
OFF = 0
```

Create the update method.

```
def update(frameNum, img, grid, N):
    newGrid = grid.copy()
    for i in range(N):
        for j in range(N):
            ## Continue this on the next slide.
```

Update method continued.

We are inside the "for j in range(N):".

```
total = (  grid[(i-1)%N,(j-1)%N]
         + grid[(i-1)%N,j]
         + grid[(i-1)%N,(j+1)%N]
         + grid[i,(j-1)%N]
         + grid[i,(j+1)%N]
         + grid[(i+1)%N,(j-1)%N]
         + grid[(i+1)%N,j]
         + grid[(i+1)%N,(j+1)%N])/255
```

Update method continued.

We are inside the "for j in range(N):" and under the "total =" line.

```
if grid[i,j] == ON and (total < 2 or total > 3):
    newGrid[i,j] = OFF
elif grid[i,j] == OFF and total == 3:
    newGrid[i,j] = ON
```

All four of John Conway's original rules are in this `if` statement.

Update method continued.

This is after all of the “for” loops, but still inside the “update” function.

```
img.set_data(newGrid)
grid[:] = newGrid[:]
return img,
```

Create the grid.

```
def main():
    N = 400
    grid = np.random.choice([ON, OFF], (N, N))
```

Setup the animation.

We are still inside the `main()` function.

```
fig, ax = plt.subplots()
img = ax.imshow(grid, interpolation='nearest')
ani = animation.FuncAnimation(fig, update,
                              fargs=(img, grid, N, ),
                              frames=1000,
                              save_count=50)
plt.show()
```

Final line.

After all of the functions, add a call to `main()`:

```
main()
```

Run the simulation!

Let's explore.

Find this line in your program. This makes your random grid.

```
grid = np.random.choice([ON, OFF], (N, N))
```

We can change it to this to make fewer ON cells than OFF.

```
grid = np.random.choice([ON, OFF], (N, N), p=[0.2, 0.8])
```

You can use this to see that some clumps stay the same forever. Other clumps alternate back and forth between two shapes. Each of these clumps of cells have a name in the Game of Life history.

Let's explore.

Remove the "grid =" line and replace it with this:

```
grid = np.zeros((N, N))
grid[1,2] = ON
grid[2,3] = ON
grid[3,1] = ON
grid[3,2] = ON
grid[3,3] = ON
```

In this grid, every cell is OFF except for these five.

Without us telling you, try to draw this shape on paper. It's an important shape in the Game of Life called a "Glider". If you run your program, you should see the glider gliding down the screen from the top-left corner to the bottom right.

Review

In this lesson, we've learned...

- How to create and modify 2D lists.
- How to create John Conway's Game of Life.

Philosophy

Conway's Game of Life has been written about in philosophy.

- Are the cells in this simulation alive?
  - Most people would say that they aren't alive, yet they behave like living creatures.
- Is it possible to create a universe of cells with fewer rules than Conway's Game?

Philosophy

Our universe is vastly more complex than this game, yet this game continues to be studied for its complexity.

Using the same program that we've written today, scientists have built entire programming environments. They have created artificial clumps of cells which follow Conway's rules and solve complex problems. Computer scientists have further proven that the clumps of cells can solve any problem that can be solved with a language such as Python.

Further more, scientists have created clumps of cells within the Game of Life that have produced a fully functioning Game of Life on their own. This game, despite how short it is, is wonderfully complicated.

Play it again.

Play the initial version again and ask yourself, “Are these cells alive?”