CSC231—Assembly

Week #9 — Spring 2017
2 Videos to Watch at a Later Time…

https://www.youtube.com/watch?v=FdMzngWchDk

https://www.youtube.com/watch?v=k2IZ1qsx4CM
https://www.youtube.com/watch?v=CgOcEZInQ2I
A 1-D Version

---

D. Thiebaut, Computer Science, Smith College
A 1-D Version
A 1-D Version

```
#
#
#
#
#
#
#
#
#
```
Rules of Life

Rule 1: 0 neighbors

#
Rule 1: 0 neighbors

Underpopulation
Rules of Life

**Rule 2**: 1 neighbor
Rule 2: 1 neighbor

Right Environment
Rules of Life

**Rule 3**: 2 neighbors

Diagram showing a pattern of Leben's game with 2 neighbors highlighted.
Rules of Life

Rule 3: 2 neighbors

Overpopulation
Problem of the Day(s):
Implement 1D Game of Life in Assembly!
How to Approach This?
#Step 1: Write Algorithm in an More Comfortable Language...
# gameOfLife.py
# D. Thiebaut
# 1-Dimensional Game of Life

from __future__ import print_function
from __future__ import division
import random

def life( dish, N ):
    newGen = ""
    for i in range( 0, N ):
        neighbors = 0
        if i>0 and dish[i-1]!="": neighbors += 1
        if i < N-1 and dish[i+1]!="": neighbors += 1
        if neighbors == 1:
            newGen += "#"
        else:
            newGen += " "
    return newGen

def main():
    N = 40
    dish = (N//2-10)*"#" + 10*" #" + (N//2-10)*" 
    dish = dish[0:N]

    # print first generation
    print( dish )

    # repeat, for some generations
    for generation in range( 20 ):
        newGen = life( dish, N )
        print( newGen )
        dish = newGen

main()
```python
# gameOfLife.py
# D. Thiebaut
# 1-Dimensional Game of Life where cells are maintained
# as arrays of 0s and 1s. 0 means dead, 1 means alive.
# This program uses a neat trick provided by Artemis.
# in class, which recognizes that the fate of a cell is equal to the xor of its neighbors.
# two live neighbors correspond to 1 xor 1 = 0. Cell dies.
# two dead neighbors correspond to 0 xor 0 = 0. Cell dies.
# only one neighbor alive corresponds to 0 xor 1 = 1. Cell lives.
# The other neat trick offered by Emma is to add space (' ')
# to the value of a cell before printing. If a cell is dead,
# adding 0 to ' ' makes it a space. Adding 1 to ' ' makes it '!'

from __future__ import print_function
from __future__ import division

def life( dish, N ):
    newGen = [0]*N
    for i in range( 1, N-1 ):
        fate = dish[i-1] ^ dish[i+1] # ^ is xor
        newGen[i] = fate
    return newGen

def printDish( dish ):
    print( ''.join( [ str(chr(ord(' ') + c)) for c in dish ] ) )

def main():
    N = 40
    dish = (N//2-10)*[1] + 10*[0,1] + (N//2-10)*[0]
    dish = dish[0:N]

    printDish( dish ) # print first generation

    # repeat, for some number of generations
    for generation in range( 20 ):
        newGen = life( dish, N )
        printDish( newGen )
        dish = newGen

main()
```

---

**Game of Life**

Python: V2

Same version but without tests

getcopy GameOfLife_V2.py
Develop Assembly Program as a Class Exercise