Lists are fundamental in Python

We construct them in a variety of ways

- Explicit: `my_list = [1,2,3]`
- Computationally: `my_list.append(4)`
- List comprehension: `my_list = [x for x in range(4)]`
- Reading from a file:
  ```python
  with open(filename, 'r') as f:
    my_list = [line.split('n') for line in f]
  ```
Python - Lists, Arrays, and Data Frames

- Manipulating lists: **list slicing**
- My_list[start:stop:increment]
  - Start - inclusive
  - Stop - exclusive
  - Increment - positive or negative!
  - All can be optional
- Some Examples:

  ```python
  >>> my_list = [1,2,3,4,5,6]
  >>> my_list[2:]
  [3, 4, 5, 6]
  >>> my_list[:2]
  [1, 2]
  >>> my_list[:]
  [1, 2, 3, 4, 5, 6]
  >>> my_list[::2]
  [1, 3, 5]
  >>>
  >>> lst = [x for x in range(10)]
  >>> even = lst[::2]
  >>> even
  [0, 2, 4, 6, 8]
  >>> odd = lst[1::2]
  >>> odd
  [1, 3, 5, 7, 9]
  >>>
  >>> lst = [x for x in range(10)]
  >>> rev = lst[::-1]
  >>> rev
  [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
  >>>
  >>>
  >>>
  >>>
  ```

**Definition:** In computer programming, list (array) slicing is an operation that extracts a subset of elements from a list (array) and packages them as another list (array), possibly in a different dimension from the original. (Wikipedia)
We can also assign into list slices:

```python
>>> lst = [x for x in range(10)]
>>> lst
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> lst[2:5] = [0,0,0]
>>> lst
[0, 1, 0, 0, 0, 5, 6, 7, 8, 9]

>>> bit_vec = [1 for i in range(16)]
>>> bit_vec
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
>>> bit_vec[1::2] = [0 for i in range(8)]
>>> bit_vec
[1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0]
```

For more info see:
http://www.i-programmer.info/programming/python/3942-arrays-in-python.html
Python does not have arrays - they can be constructed with lists of lists.

```python
>>> arr = [[1,2,3],
... [4,5,6],
... [7,8,9]]
>>> arr[1]  
[4, 5, 6]
>>> arr[1][1]
5
>>> for row in arr:
...  for e in row:
...   print(e)
...
1
2
3
4
5
6
7
8
9

>>> arr[1][1] = 0
>>> print(arr)
[[1, 2, 3], [4, 0, 6], [7, 8, 9]]

>>> arr = [[0 for j in range(3)] for i in range(3)]
>>> print(arr)
[[0, 0, 0], [0, 0, 0], [0, 0, 0]]
```
However, slicing does **not** work properly on arrays!

```python
>>> arr = [[1,2,3], ... [4,5,6], ... [7,8,8]]
>>> arr[1][::]
[4, 5, 6]
>>> arr[::][1]
[4, 5, 6]
>>> arr[::]
[[1, 2, 3], [4, 5, 6], [7, 8, 8]]
```
Python - Lists, Arrays, and Data Frames

Pandas data frames - 2D arrays specifically designed for data processing!

```python
>>> import pandas
>>> arr = [[1,2,3],
...        [4,5,6],
...        [7,8,8]]
>>> df = pandas.DataFrame(data=arr,columns=['a','b','c'])
>>> df
   a  b  c
0  1  2  3
1  4  5  6
2  7  8  8

>>> df.iloc[1,1] = 0
>>> df
   a  b  c
0  1  2  3
1  4  0  6
2  7  8  8
```
In data frames slicing works as expected!

```python
>>> df
   a  b  c
0  1  2  3
1  4  5  6
2  7  8  8
>>> df.iloc[1, :]
   a  b  c
0  4  5  6
1  5  6  8
>>> df.iloc[:, 1]
   0  2
   1  5
   2  8
```
Programming Exercise

- Your board size should be a parameter so you can try it on different sized boards
- Your ‘number of generations’ should also be a parameter
- Your main data structure should be a Pandas DataFrame
- Implement the ‘New Rules’ for the game (see the slide ‘New Rules’)
- No fancy graphics necessary, just displaying ascii is fine. (see function on next slide)
Programming Exercise

Rules for the Game:

- Any live cell with fewer than two live neighbors dies, as if caused by underpopulation.
- Any live cell with two or three live neighbors lives on to the next generation.
- Any live cell with more than three live neighbors dies, as if by overpopulation.
- Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.
import os
import time

def display_df(df):
    "clear the screen, display the contents of a dataframe, wait for 1sec"
    os.system('clear')

    rows = df.shape[0]
cols = df.shape[1]

    for i in range(rows):
        for j in range(cols):
            print(df.iloc[i,j],end=' ') # no carriage return, space separated
    print()
    time.sleep(1)
Programming Exercise

- Teamwork allowed - see Teams

Team:
TBA