# Python - Lists, Arrays, and Data Frames

- 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:

```
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:

**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)

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

```
>>> 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]
>>>
```

# Python - Lists, Arrays, and Data Frames

We can also assign into list slices:

```
>>> 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 - Lists, Arrays, and Data Frames

Python does not have arrays - they can be constructed with lists of lists.

```
>>> 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]]
>>>
```

# Python - Lists, Arrays, and Data Frames

However, slicing does not work properly on arrays!

```
>>> 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!

We will have much more to say about data frames later on

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

# Python - Lists, Arrays, and Data Frames

In data frames slicing works as expected!

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

# Python – Classes and Objects

- Classes are dynamic objects in the spirit of Python: variables become defined when they appear in the program text.
- It matters where they appear!
- No protection mechanisms – everything is globally visible!
- Classes also support inheritance (I let you explore that…)

In [16]:
```
class Dog:

    kind = 'canine'             # class variable shared by all instances

    def __init__(self, name):   # constructor function -- automatically called
        self.name = name        # instance variable unique to each instance
        self.tricks = []        # another instance variable!

    def __str__(self):          # function to compute string representation of object
        return "{} can do the following tricks: {}".format(self.name,self.tricks)

    def add_trick(self, trick):
        self.tricks.append(trick)
```

# Python – Classes and Objects

```
In [12]: Dog.kind

Out[12]: 'canine'

In [13]: fido = Dog('Fido')
         buddy = Dog('Buddy')
         fido.add_trick('roll over')
         buddy.add_trick('play dead')

In [14]: fido.tricks

Out[14]: ['roll over']

In [15]: print(buddy)

         Buddy can do the following tricks: ['play dead']
```

Note: this is in Jupyter Notebook style – In is a program statement – Out is the interpreter output

# Programming Exercise

- You are to implement Conway's Game of Life in Python:
  en.wikipedia.org/wiki/Conway's_Game_of_Life
- 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 an array – or two if you use *double buffering* (recommended)
- No fancy graphics necessary, just displaying ascii is fine. (see function on next slides)

# 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.

# Programming Exercise

In [24]:
```
import os
import time

def display_array(ar):
    "clear the screen, display the contents of an array, wait for 1sec"
    os.system('clear')

    rows = len(ar)    # grab the rows

    if rows == 0:
        raise ValueError("Array contains no data")

    cols = len(ar[0]) # grab the columns - indices start at 0!

    for i in range(rows):
        for j in range(cols):
            print(ar[i][j],end=' ') # no carriage return, space separated
        print()

    time.sleep(1)
```

n [19]:
```
ar = [[1,2,3],
      [4,5,6],
      [7,8,9]]

display_array(ar)
```

```
1 2 3
4 5 6
7 8 9
```

n [21]:
```
board = [[' ','*',' '],
         ['*',' ','*'],
         [' ','*',' ']]

display_array(board)
```

```
  *
*   *
  *
```

# Programming Exercise

In [19]:
```
ar = [[1,2,3],
      [4,5,6],
      [7,8,9]]

display_array(ar)
```

```
1 2 3
4 5 6
7 8 9
```

In [21]:
```
board = [[' ','*',' '],
         ['*',' ','*'],
         [' ','*',' ']]

display_array(board)
```

```
  *
*   *
  *
```

# Programming Exercise

- Teamwork allowed - see Teams

Team:
Team 0: ['Alber', 'Alexander', 'Shamal']
Team 1: ['David', 'Matt', 'Najib']
Team 2: ['Evelyn', 'Peter', 'Cory']
Team 3: ['Joe', 'Kermalyn', 'Baez']
Team 4: ['Geron', 'Harout', 'Susallin']
Team 5: ['Christopher', 'Aguilar', 'Gabe']
Team 6: ['Aakash', 'Kevin', 'David']
Team 7: ['Ben']