Review

- Use of ImageProcessing.py for Assignment 4
- Similarities between objects and functions
- Call Stack (of frames)
- Recursion

Python Lists

- A data structure that can holds a sequence of items
- A list can hold any number of items, of any type
- Syntactically, it is a comma-separated sequence surrounded by square brackets

```
myList = ["a string", 1.234, True]
```

- Lists are mutable (they can be modified in code)
- The len() function returns list length

```
>>> len( myList )
3
```

- Lists can even contain other lists

```
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Python Lists

- The elements of a list can be accessed using square brackets and an index number
- Indexes are 0-based

```
>>> options = ["Create", "Read", "Update", "Delete"]
>>> options[2]
'Update'
>>> options[1] = "Select"
>>> options
['Create', 'Select', 'Update', 'Delete']
```

Python Lists

- Lists provide a variety of functions to manipulate their members
- Use the del keyword to delete a list element

```
>>> items = [0, 1, 2, 3, 4, 5]
>>> del items[2]
>>> items
[0, 1, 3, 4, 5]
```

- Use the '+' operator to concatenate lists

```
>>> ["a", "b", "c"] + ["d", "e"]
['a', 'b', 'c', 'd', 'e']
```

Python Lists

- Accessing all elements in a list using for-in

```
L = ['a', 'b', 'c', 'd', 'e']
for i in L:
    print( i )
>>> a
>>> b
>>> c
>>> d
>>> e
```

Python Lists

- Access list methods using dot-notation

```
L.append(x)       # add an item to the end of L
L.extend(M)       # append all items in M to the end of L
L.insert(i, x)    # insert x into L at position i
L.remove(x)       # remove the first occurrence of x from L
L.pop(i)          # remove the item at position i
                  # and return it (or from end if no i)
L.index(x)        # find the index of the first item
                  # that matches x on L
L.count(x)        # return the number of times x occurs in L
L.sort()          # sort the items in L, in place
L.reverse()       # reverse the order of items in L,
                  # in place.
```

Functional Programming

- Functions are first-class objects in Python

```
print( len )
>>> <built-in function len>

print( print )
>>> <built-in function print>
```

Functional Programming – map()

- The map() function applies a function to all members of a list, and returns a list of the results

```
def rem3( x ):
    return x % 3

L = [0, 1, 2, 3, 4, 5]
result = map( rem3, L )

print( result )

>>> [0, 1, 2, 0, 1, 2]
```

List Comprehensions

- A similar method of producing lists of results is to use List Comprehensions
- Format:

```
[ expression for variable in list ]
```

- Example

```
result = [ x % 3 for x in range(6) ]
print( result )
```

```
>>> [0, 1, 2, 0, 1, 2]
```

Sets

- An unordered collection with no duplicate elements
- Sets are created with the set() function and a sequence of elements

```
s = set( ['aa', 'bb', 'cc', 'aa', 'cc'] )
print( s )

>>> set(['aa', 'bb', 'cc'])
```

Sets

- Compute set length using len()
- Access elements using for-in

```
s = set( ['aa', 'bb', 'cc', 'aa', 'cc'] )
print( len(s) )

>>> 3

for e in s:
    print( e )

>>> aa
>>> bb
>>> cc
```

Sets

Methods

```
s1 = set( ['aa', 'bb', 'cc'] )
s2 = set( ['cc', 'dd'] )

s1.issubset( s2 )        # False
s1.issuperset( s2 )      # False
s1.isdisjoint( s2 )      # False
s1.union( s2 )           # set(['aa', 'bb', 'cc', 'dd'])
s1.difference( s2 )      # set(['aa', 'bb'])
s1.intersection( s2 )    # set(['cc'])
s1.copy()                # set(['aa', 'bb', 'cc'])
```

Dictionaries

- Lists allow its items to be accessed with an integer
- I.e.: A list maps integers to elements

- Dictionaries are data structures in Python that map other objects to elements in the data structure.
  - aka: Associative Arrays

{ *key1*:*val1*, *key2*:*val2*, … **}**

- Example: Phone numbers – mappings strings

```
phone = {'home': '111-222-3333',
         'work': '222-333-4444',
         'cell': '333-444-5555' }
```

Dictionaries

- Dictionary elements are accessed using square brackets and a key

```
phone = { 'home': '111-222-3333',
          'work': '222-333-4444',
          'cell': '333-444-5555' }

pn = phone['work']
print( pn )

>>> 222-333-4444

phone['home'] = '444-555-6666‘

print( phone['home'] )

>>> 444-555-6666
```

Dictionaries

- Using for-in on dictionaries enumerates keys

```
phone = { 'home': '111-222-3333',
          'work': '222-333-4444',
          'cell': '333-444-5555'  }

for k in phone:
    print(k, phone[k])

>>> home 111-222-3333
>>> work 222-333-4444
>>> cell 333-444-5555
```

Nested Data Structures

- List of Lists

```
identity = [[1, 0, 0],
            [0, 1, 0],
            [0, 0, 1]]
```

- List of Dictionaries

```
attrs = [{ 'age': 20, 'IQ': 100, 'pickles': True },
         { 'age': 50, 'IQ': 85,  'pickles': False} ]
```

Nested Data Structures

- Dictionary containing Lists

```
numbers = { 'prime':     [1, 2, 3, 5, 7, 11, 13],
            'fibonacci': [0, 1, 1, 2, 3, 5, 8] }
```

- Dictionary containing Dictionaries

```
attrs = { 'fred':   { 'age': 20, 'IQ': 110 },
          'george': { 'age': 50, 'IQ': 85  } }
```

Nested Data Structures

- List of Lists
- List of Dictionaries
- Dictionary containing Lists
- Dictionary containing Dictionaries
- Set of Lists
- Dictionary of Sets
- Lists of Lists of Lists
- Lists of Dictionaries containing Sets
- …