

---

# The Python List Data Structure

- A list is an ordered sequence of items of any type.
  - A list can even contain other lists!



---

# Similarities with Strings

- concatenate/+ (but only of lists)
  - repeat/\*
  - indexing (the [ ] operator)
  - slicing ([:])
  - membership (the in operator)
  - len (the length operator)
-

---

# Differences Between Lists and Strings

- Lists can contain a mixture of any python objects
    - 1, "bill", 1.2345, True
  - Lists are mutable; their values can be changed while strings are immutable.
  - Lists are designated with [ ], with elements separated by commas; strings use "".
-

---

# Indexing

- Can be a little confusing - what does the [ ] mean, a list or an index?

`[1, 2, 3][1] ⇒ 2`

- Context solves the problem. An index always comes at the end of an expression and is preceded by something (a variable, a sequence).
-

---

# List of Lists

```
myLst = ['a', [1, 2, 3], 'z']
```

- What is the second element (index 1) of that list?
- Another list:

```
myLst[1][0] # apply left to right
```

```
myLst[1] ⇒ [1, 2, 3]
```

```
[1, 2, 3][0] ⇒ 1
```

---

---

# List Functions

- `len(lst)`: Number of elements in list (top level).  
`len([1, [1, 2], 3]) ⇒ 3`
  - `min(lst)`: Minimum element in the list. If list of lists, looks at first element of each list.
  - `max(lst)`: Max element in the list.
  - `sum(lst)`: Sum the elements, numeric only.
-

---

# Iteration

```
for element in [1, [1, 2], 'a', True]:  
    print element
```

- Gives us

1

[1, 2]

'a'

True

---

---

# Change an Object's Contents

- Strings are immutable. Once created, the object's contents cannot be changed. New objects can be created to reflect a change, but the object itself cannot be changed:

```
myStr = 'abc'
```

```
myStr[0] = 'z'      # cannot do!
```

```
# instead, make new str
```

```
newStr = myStr.replace('a','z')
```

---

---

# Lists are Mutable

- Unlike strings, lists are mutable. You can change the object's contents!

```
myLst = [1, 2, 3]
```

```
myLst[0] = 127
```

```
print myLst ⇒ [127, 2, 3]
```



---

# List Methods

- Remember, a function is a small program (such as `len`) that takes some arguments, the stuff in the parenthesis, and returns some value.
  - A method is called in a special way, the “dot call”. It is called in the context of an object (or a variable holding an object).
-

---

# Again, Lists have Methods

```
myList = ['a', 1, True]
```

```
myList.append('z')
```

← arguments to  
the method

← the name of  
the method

↑ the object that  
we are calling the  
method with

---

---

# Some New Methods that Modify the List

- `myList[0]='a'`
    - Index assignment
  - `myList.append(x)`
    - Appends `x` to end of `myList`
  - `myList.extend(C)`
    - Takes a collection (like another list) and add each of the collection's elements to the end of `myList`
-

---

# Some New Methods that Modify the List

- `myList.pop()`
    - Removes the element at the end of `myList` and returns that element.
  - `myList.insert(i,x)`
    - Inserts element `x` at position `i` into `myList`.
  - `myList.remove(x)`
    - Removes element `x` from `myList`
-

---

# Some New Methods that Modify the List

- `myList.sort()`
    - Sorts `myList`. If sorting a list of lists, only the first element in each list is considered in comparison operations.
  - `myList.reverse()`
    - Reverses the elements in `myList`
-

---

# Some New Methods that do not Modify the List

- `myList.index(x)`
  - Returns index value of element `x` in `myList`.
- `myList.count(x)`
  - Returns the number of times `x` appears in `myList`.

---

# Lab 10

- We played with and re-implemented many of these list methods as our own functions in lab 10.
- Let's take a look.

---

# More about List Methods

- Most of these methods do not return a value.
- This is because lists are mutable so the methods modify the list directly; there is no need to return anything.



# Sorting

- Only lists have a built-in sorting method. Thus you often convert your data to a list if it needs sorting:

```
myLst = list('xyzabc')
```

```
myLst ⇒ ['x','y','z','a','b','c']
```

```
myLst.sort()
```

```
# convert back to a string
```

```
sortStr = ''.join(myLst)
```

```
⇒ 'abcxyz'
```

---

# Sorted Function

- The sorted function will break a sequence into elements and sort the sequence, placing the results in a list:

```
sortLst = sorted('hi mom')
```

```
⇒ [' ', 'h', 'i', 'm', 'm', 'o']
```



---

# Unusual Results

```
myLst = [4, 7, 1, 2]
```

```
myLst = myLst.sort()
```

```
myLst ⇒ None    # what happened?
```

What happened was the sort operation changed the order of the list in place (right side of assignment). Then the sort method returned None, which was assigned to the variable. The list was lost and None is now the value of the variable.

---

---

# Anagram Example

- Anagrams are words that contain the same letters in a different order. For example: 'iceman' and 'cinema.'
  - A strategy to identify anagrams is to take the letters of a word, sort those letters, then compare the sorted sequences.
  - Anagrams should have the same sequence.
-

---

# Anagram Program Algorithm

1. Input 2 words to examine.
2. Sort the letters of each word into a new string.
3. Compare the resulting sorted strings