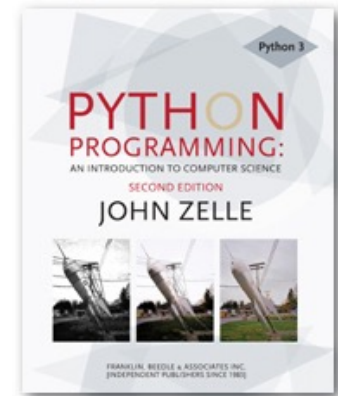




CS 177



Data Collections: Advanced Lists and Tuples



Lists, Lists, Lists... so many lists

- List structures and values
- Accessing elements
- List length and membership
- List operations and slicing
- Changing and manipulating Lists
- Lists and looping
- List methods
- Lists to Strings → Strings to Lists



List structures and values

It's important to consider not only what we're going to store in our List, but how it will be organized

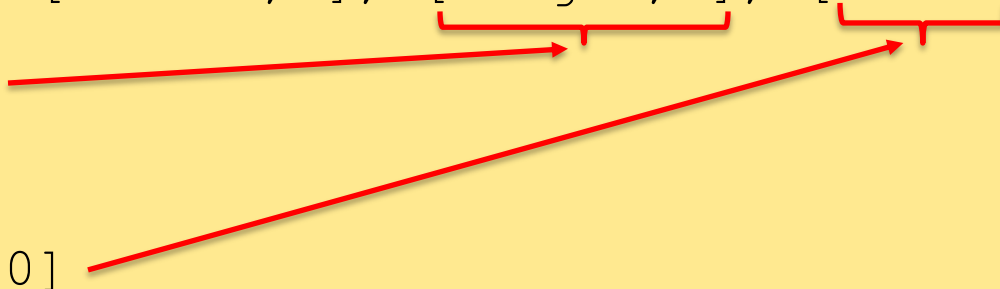
```
fnames = ['Will', 'John', 'Yolanda', 'Zeb']
lnames = ['Carson', 'Wilhelm', 'Brown', 'Indiano']
for j in range(len(fnames)):
    print(lnames[j] + ', ' + fnames[j])
```



Accessing List elements (index)

When List elements contains smaller Lists, it provides a 2-dimensional data storage effect. Values can be addressed using indexing.

```
>>> myPets = [ ['Cats',2], ['Dogs',3], ['Fish',4] ]
>>> myPets[1]
['Dogs', 3]
>>> myPets[2][0]
'Fish'
```



The smaller List elements may contain any data type



Accessing List elements (loop)

It is common to use a loop variable as a List index

```
horsemen = ["war", "famine", "pestilence", "death"]
for i in [0, 1, 2, 3]:
    print(horsemen[i])
```

A for loop can also use the List elements as a variable

```
horsemen = ["war", "famine", "pestilence", "death"]
for h in horsemen:
    print(h)
```



List length and membership

Using the `len()` function to determine the length of a List allows our program to access the List index when we don't already know their length

```
horsemen = ["war", "famine", "pestilence", "death"]
for i in range(len(horsemen)):
    print(horsemen[i])
```



List operations (+)

- Operators are the symbols used in mathematical operations
- When used with Lists, operators can be used to manipulate their content

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> c
[1, 2, 3, 4, 5, 6]
```

- The + operator concatenates Lists



List operations (*)

- Operators are the symbols used in mathematical operations
- When used with Lists, operators can be used to manipulate their content

```
>>> [0] * 4
[0, 0, 0, 0]
>>> [1, 2, 3] * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

- The * operator repeats a List n times



List slicing

Slicing operations also work with Lists

```
>>> a_list = ["a", "b", "c", "d", "e", "f"]
>>> a_list[1:3]
['b', 'c']
>>> a_list[:4]
['a', 'b', 'c', 'd']
>>> a_list[3:]
['d', 'e', 'f']
>>> a_list[:]
['a', 'b', 'c', 'd', 'e', 'f']
```



Changing and manipulating Lists

- Lists are mutable using the List index values

```
>>> fruit = ["banana", "apple", "quince"]
>>> fruit[0] = "pear"
>>> fruit[2] = "orange"
>>> fruit
['pear', 'apple', 'orange']
>>> my_list = ["T", "E", "S", "T"]
>>> my_list[2] = "X"
>>> my_list
['T', 'E', 'X', 'T']
```



Changing and manipulating Lists

- Using the slice operator, we can update a sublist and even remove elements

```
>>> a_list = ["a", "b", "c", "d", "e", "f"]
>>> a_list[1:3] = ["x", "y"]
>>> a_list
['a', 'x', 'y', 'd', 'e', 'f']
>>> a_list[1:3] = []
>>> a_list
['a', 'd', 'e', 'f']
```



Changing and manipulating Lists

- Using the slice operator, we can also *squeeze* (insert) new elements within the List

```
>>> a_list = ["a", "d", "f"]
>>> a_list[1:1] = ["b", "c"]
>>> a_list
['a', 'b', 'c', 'd', 'f']
>>> a_list[4:4] = ["e"]
>>> a_list
['a', 'b', 'c', 'd', 'e', 'f']
```



Lists and looping (*again*)

for variable in List:

This usage of a List as the variable in a for loop reads much like natural language

```
friends = ["Joe", "Zoe", "Brad", "Angelina"]
for friend in friends:
    print(friend)
```

"For (every) friend in (the list of) friends, print (the name of the) friend."



Lists have many methods used to manipulate and manage their elements

<code>append()</code>	Add Single Element
<code>extend()</code>	Add Elements of a List to Another List
<code>insert()</code>	Inserts Element
<code>remove()</code>	Removes Element
<code>index()</code>	returns smallest index of element
<code>count()</code>	returns occurrences of element
<code>pop()</code>	removes Element at Given Index
<code>reverse()</code>	reverses a List
<code>sort()</code>	sorts elements of a list
<code>len()</code>	returns Length
<code>max()</code>	returns largest element
<code>min()</code>	returns smallest element
<code>reversed()</code>	returns reversed of List
<code>sorted()</code>	returns sorted list
<code>sum()</code>	returns the sum of all items



Strings → Lists and Lists → Strings

It is easy to convert a String to a List

```
>>> song = "The rain in Spain..."
>>> wds = song.split()
>>> wds
['The', 'rain', 'in', 'Spain...']
```

The `split()` method divides a String by its whitespaces

We can also specify the value used to `split()` a String



Strings → Lists and Lists → Strings

We must 'glue' a List together using the `join()` method to build a String using the List elements

```
>>> wds = ['The', 'rain', 'in', 'Spain...']
>>> glue = ";"
>>> s = glue.join(wds)
>>> s
'The;rain;in;Spain...'
```

In this case, ";" is the glue between the List elements



Strings → Lists and Lists → Strings

We can also use empty Strings to glue a List into a String

```
>>> wds = ['The', 'rain', 'in', 'Spain...']  
>>> "".join(wds)  
'TheraininSpain...'
```



The *Tuple*: An ordered sequence of values

- Tuples are similar to lists
- The primary difference is that *Tuples* are immutable.
- A tuple is defined by using parentheses

```
a = (1, "a", 45.2)
```

```
myTuple = ("Pete", "Rowdy")
```

Tuples are used to “pack and unpack” multiple values

```
X = (23,45,67)
X = 23,45,67
X = tuple([2,3,4])
```

← Tuple Definition

```
X[0:2]
```

← Indexing works, and so does slicing

```
X = ()
X = (23,)
```

← Tuples of length 0 and 1
Tuple Definitions for length 1 must be followed by a comma

This is how multiple parameters are passed to and from functions



You've already been using *Tuples* this semester!

These should look familiar:

```
def make(x, y):  
    win = GraphWin('win', x, y)  
    c = Circle(Point(x/2, y/2), 20)  
    c.draw(win)  
    return win, c  
  
def main():  
    win, c = make(200, 300)
```

All of these examples utilize a *Tuple* to manipulate, pass or refer to multiple variables simultaneously