

LECTURE 10 — LISTS PART 2

18.1 Topics

We will cover the first three of these topics during lecture. The fourth is for your own use:

- List aliasing, lists and functions
- For loops to operate on lists
- Slicing to create copies of lists and to create sublists
- Converting back and forth between strings and lists

18.2 List Aliasing

- Consider the following example Python code:

```
>>> L1 = [ 'RPI', 'WPI', 'MIT' ]
>>> L2 = L1
>>> L3 = [ 'RPI', 'WPI', 'MIT' ]
>>> L2.append( 'RIT' )
>>> L2[1] = 'CalTech'
>>> L1
['RPI', 'CalTech', 'MIT', 'RIT']
>>> L2
['RPI', 'CalTech', 'MIT', 'RIT']
>>> L3
['RPI', 'WPI', 'MIT']
```

- Surprised? This is caused by the creation of what we call an *alias* in computer science:
 - L1 and L2 reference the same list - they are *aliases* of each other and the underlying list - so changes made using either name change the underlying list
 - L3 references a different list that just happens to have the same string values in the same order: there would have been no confusion if the strings in the list had been different.
 - We'll use our memory model for lists to understand what is happening here.
- Python uses aliases for reasons of efficiency: lists can be quite long and are frequently changed, so copying of entire lists is expensive
- This is true for other *container* data types as well.
 - Assignments create an alias for images, lists, tuples, strings and, as we will see later, sets and dictionaries

- * Aliases of strings and tuples do not create the same confusion as other containers because they can not be changed once they are created.
- Fortunately, if we truly want to copy a list, Python provides a `copy()` method for lists. Try the following and see what happens.

```
L1 = [1,2,3]
L2 = L1.copy()
L1.pop()
L2.append( 4 )
print(L1)
print(L2)
```

18.3 Aliasing and Function Parameters

- When a variable is passed to functions, a copy of its value is created if the value is a number or a booleans:

```
def add_two(val1, val2):
    val1 += val2
    return val1

val1 = 10
val2 = 15
print(val1, val2)
print(add_two(val1, val2))
print(val1, val2)
```

- When a list is passed to functions, the parameter becomes an alias for the argument in the function call.
- Here is an example of a function that returns a list containing the two smallest values in its input list:

```
def smallest_two(mylist):
    mylist.sort()
    newlist = []
    if len(mylist) > 0:
        newlist.append(mylist[0])
        if len(mylist) > 1:
            newlist.append(mylist[1])
    return newlist

values = [35, 34, 20, 40, 60, 30]

print("Before function:", values)
print("Result of function:", smallest_two(values))
print("After function:", values)
```

- In class we will discuss what happened.

18.4 What Operations Change a List? What Operations Create New Lists?

- Operations that change lists include
 - `sort`, `insert`, `append`, `pop`, `remove`

- Operations that create new lists
 - Slicing (discussed below), `copy()`, concatenation (+), replication (*) and `list()`

18.5 Part 1 Practice

Students will be given about 5 minutes to work on the first two lecture exercises

18.6 Part 2: For Loops and Operations on List Items

- Although *while* loops allow us to apply an operation to each entry in a list, Python has a construct called a *for* loop that is often easier to use for such operations.
- Our driving example will be the problem of capitalizing a list of names. We'll start with a simple example:

```
animals = ['cat', 'monkey', 'hawk', 'tiger', 'parrot']
cap_animals = []
for animal in animals:
    cap_animals.append( animal.capitalize() )
print(cap_animals)
```

- We can understand what is happening by looking at this piece-by-piece:
 - The keyword `for` signals the start of a loop
 - `animal` is a loop variable that takes on the value of each item in the list (as indicated by the keyword `in`) in succession
 - * This is called *iterating* over the values/elements of the list
 - The `:` signals the start of a block of code that is the “body of the loop”, executed once in succession for each value that `animal` is assigned
 - The body of the loop here is just a single, indented line of code, but in other cases - just as using *while* loops - there may be more than one line of code.
 - The end of the loop body is indicated by returning to the same level of the indentation as the `for ...` line that started the loop.

18.7 Changing the Values in a List

- What if we wanted to change the list? We might consider copying `cap_animals` back to `animals` at the end of the code sequence.
- But this does not work if we wanted a function that capitalized all strings in a list.

```
def capitalize_list( names ):
    cap_names = []
    for n in cap_names:
        cap_names.append( n.capitalize() )
    names = cap_names

animals = ['cat', 'monkey', 'hawk', 'tiger', 'parrot']
capitalize_list(animals)
print(animals)      # Make sure you understand the output!!!
```

- This does not work because `names` is an alias for the list rather than the list itself
- The following does not work either because `n` is an alias for the string in the list

```
def capitalize_list( names ):
    for n in names:
        n = n.capitalize()
```

- So, based on what we know so far to actually change the values in the list we need to use indexing together with a `while` loop:

```
def capitalize_list( names ):
    i = 0
    while i < len(names):
        names[i] = names[i].capitalize()
        i += 1
```

- We can also solve this using a `for` loop and indexing, but for this we need a `range`

18.8 Using `range`

- A `range` “generates” values in a sequence, almost-but-not-quite like a list.

```
for i in range(5):
    print(i)
```

prints the values 0 through 4...

- We can convert a `range` to an actual list:

```
>>> x = list(range(5))
>>> print(x)
[0, 1, 2, 3, 4]
```

- The general form is

```
range( bi, ei, ii )
```

where

- `bi` is the initial value (defaults to 0)
- `ei` is the ending value (never included in the range!)
- `ii` is the increment, added each time (defaults to 1)

- We’ll look at number of examples:

```
list(range(3,10))
list(range(4, 20, 4))
list(range(10, 2, -2))
```

- Using `for` loops on lists, we often use `len()` in combination with `range` to specify the indices that should be used.

```
def capitalize_list( names ):
    for i in range(len(names)):
        names[i] = names[i].capitalize()
```

Now we have our for loop based solution to capitalizing the names in a list.

- Unlike with a while loop there is no need to write code to compare our index / counter variable `i` directly against the bound and no need to write code to increment `i`.
- This use of range to generate an index list is common
 - When we want to change the integer, float or string values of a list.
 - When we want to work with multiple lists at once.

18.9 Part 2 Practice

1. Recall our list

```
co2_levels = [ 320.03, 322.16, 328.07, 333.91, 341.47, \
              348.92, 357.29, 363.77, 371.51, 382.47, 392.95 ]
```

For the purpose of this exercise only, please pretend the Python sum function does not exist, and then write a short section of Python code that uses a for loop to first compute and then print the sum of the values in the `co2_levels` list. You do not need to use indexing.

18.10 Using Indices to “Slice” a List and Create a New List

- Recall

```
co2_levels = [ 320.03, 322.16, 328.07, 333.91, 341.47, \
              348.92, 357.29, 363.77, 371.51, 382.47, 392.95 ]
```

- Now suppose we just want the values at indices 2, 3 and 4 of this in a new list:

```
>>> three_values = co2_levels[2:5]
>>> three_values
[328.07, 333.91, 341.47]
>>> co2_levels
[ 320.03, 322.16, 328.07, 333.91, 341.47, 348.92, 357.29, 363.77,
  371.51, 382.47, 392.95 ]
```

- We give the first index and one more than the last index we want
- If we leave off the first index, 0 is assumed, and if we leave off the second index, the length of the list is assumed.
- Negative indices are allowed — they are just converted to their associated positive values. Some examples:

```
>>> L1 = ['cat', 'monkey', 'hawk', 'tiger', 'parrot']
>>> L1
['cat', 'dog', 'hawk', 'tiger', 'parrot']
>>> L1[1:-1]
['dog', 'hawk', 'tiger']
>>> L1[1:-2]
['dog', 'hawk']
>>> L1[1:-4]
[]
>>> L1[1:0]
[]
```

```
>>> L1[1:10]
['dog', 'hawk', 'tiger', 'parrot']
```

18.11 More on List Slicing

- Specifying indices for slicing and for a range are very similar:
 - A range uses () and is a generator, while slicing using [] and is applied to a list to create a new list.
- The most general form of slicing involves three values

```
L[si:ei:inc]
```

where

- L is the list
- si is the start index
- ei is the end index
- inc is the increment value

Any of the three values is optional

- We'll work through some examples in class to
 - Use slicing to copy an entire list
 - Use negative increments and generate a reversed list
 - Extracting the even indexed values
- Note: L[:] returns a copy of the whole list of L. This is the same using method L.copy() or the function list()

```
>>> L2 = L1[:]
>>> L2[1] = 'monkey'
>>> L1
['cat', 'dog', 'hawk', 'tiger', 'parrot']
>>> L2
['cat', 'monkey', 'hawk', 'tiger', 'parrot']
>>> L3 = list(L1)
>>> L3[1] = 'turtle'
>>> L1
['cat', 'dog', 'hawk', 'tiger', 'parrot']
>>> L2
['cat', 'monkey', 'hawk', 'tiger', 'parrot']
>>> L3
['cat', 'turtle', 'hawk', 'tiger', 'parrot']
```

18.12 Concatentation and Replication

- Just like with strings, concatenation and replication can be applied to lists:

```
>>> v = [1,2,3]+[4,5]
>>> v
[1,2,3,4,5]
```

- and

```
>>> [1]*3
[1,1,1]
```

18.13 Part 3 Practice

1. What is the output of the following?

```
x = [6,5,4,3,2,1] + [7]*2
y = x
x[1] = y[2]
y[2] = x[3]
x[0] = x[1]
print(x)

y.sort()
print(x)
print(y)
```

2. Write a slicing command to extract values indexed by 1, 4, 7, 10, etc from a list L0 .

18.14 Converting Strings to Lists

- Version 1: use the function `list` to create a list of the characters in the string:

```
>>> s = "Hello world"
>>> t = list(s)
>>> print(t)
['H', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd']
```

- Version 2: use the string `split` function, which breaks a string up into a list of strings based on the character provided as the argument.
 - The default is ' ':
 - Other common splitting characters are ',', '|' and '\t'
- We will play with the `s = "Hello world"` example in class.

18.15 Converting Lists to Strings

- What happens when we type the following?

```
>>> s = "Hello world"
>>> t = list(s)
>>> s1 = str(t)
>>> s.split()
```

```
['Hello', 'world']
>>> s = "Hello    worl  d"
>>> s.split()
['Hello', 'worl', 'd']
>>> s.split(' ')
['Hello', '', '', '', '', 'worl', '', '', 'd']
>>> s.split('l')
['He', '', 'o    wor', '  d']
```

This will not concatenate all the strings in the list (assuming they are strings).

- We can write a for loop to do this, but Python provides something simpler that works:

```
>>> L1 = [ 'No', 'one', 'expects', 'the', 'Spanish', 'Inquisition' ]
>>> print(''.join(L1))
NooneexpectstheSpanishInquisition
>>> print(' '.join(L1))
No one expects the Spanish Inquisition
```

Can you infer from this the role of the string that the join function is applied to?

18.16 Indexing and Slicing Strings

- We can index strings:

```
>>> s = "Hello, world!"
>>> print(s[5])
,
>>> print(s[-1])
!
```

- We can apply all of the slicing operations to strings to create new strings:

```
>>> s = "Hello, world!"
>>> s[:len(s):2]
'Hlo ol!'
```

- Unlike lists, however, we can not use indexing to replace individual characters in strings:

```
>>> s[4] = 'c'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

18.17 Part 4 Practice

1. Given a list

```
L = [ 'cat', 'dog', 'tiger', 'lion' ]
```

Rewrite L so that it is a list of lists, with household pets in the 0th (sub)list, zoo animals in the first. Using slicing of L to create this new list and assign L to the result.

2. How can you append an additional list of farm animals (e.g. 'horse', 'pig' and 'cow') to L?

- Write code to remove 'tiger' from the sublist of zoo animals.
- Suppose you have the string

```
>>> s = "cat | dog | mouse | rat"
```

and you'd like to have the list of strings

```
>>> L = [ "cat", "dog", "mouse", "rat"]
```

Splitting the list alone does not solve the problem. Instead, you need to use a combination of splitting, and a loop that strips off the extra space characters from each string and appends to the final result. Write this code. It should be at most 4-5 lines of Python.

18.18 Summary

- Assignment of lists and passing of lists as parameters creates aliases of lists rather than copies.
- We use `for` loops to iterate through a list to work on each entry in the list.
- We need to combine `for` loops with indices generated by a `range` in order to change the contents of a list of integers, floats or strings. These indices are also used to work with multiple lists at once.
- Concatenation, replication and slicing create new lists.
- Most other list functions that modify a list do so without creating a new list: `insert`, `sort`, `append`, `pop`, etc.
- Strings may be indexed and sliced, but indexing may not be used to change a string.
- Conversion of a string to a list is accomplished using either `list` or `split`; conversion of a list of strings to a string uses `join`.

18.19 Additional Review Exercises: What Does Python Output?

- Without typing into the Python interpreter, find the outputs from the following operations:

```
>>> x = ['a', 'b', 'c', 'd', 'e']
>>> print(x)

>>> for item in x:
...     print("{} ".format(item))
...

>>> print(x[3])

>>> x[3] = 3
>>> x

>>> len(x)

>>> x[2]=x[1]
>>> x

>>> x[5]

>>> y = x[1:4]
```

```
>>> y
>>> x
```

2. What about these operations?

```
>>> y = [1, 2, 3]
>>> y.append('cat')
>>> y
>>> y.pop()
>>> y

>>> y.remove(2)
>>> y

>>> y.remove('cat')

>>> z = ['cat', 'dog']
>>> z.insert(1, 'pig')
>>> z.insert(0, 'ant')
>>> z

>>> z.sort()
>>> z

>>> z1 = z[1:3]
>>> z1

>>> z
```

3. Write a function that returns a list containing the smallest and largest values in the list that is passed to it as an argument *without changing the list*? Can you think of several ways to do this?

- (a) Using `min` and `max`
- (b) Using sorting (but remember, you can't change the original list)
- (c) Using a `for` loop that searches for the smallest and largest values.