Lists

Kelly M. Thayer
Week 8 Lecture 7
2017-03-28

Contact info:
kthayer@wesleyan.edu
Office Hours: M 8:30-10:30 AM, T 10:10-11:15 AM, W 5:00-6:00 PM
Office: Exley 322/325

(note: there is no lecture 6)
Warm Up Exercises: Recalling Strings

1. What will Python return?
```python
>>> s='Clearwater, Florida'
```

a) `s[16:]`


c) `(s[:5] + s[11]) *2 + s[5].upper() + s[6:10] + "!"`

2. Write a function `space_2_underscore` that will process each character in a string. If it finds a space, it should replace it with an underscore. Return the result.
Warm Up Exercises: Recalling Strings

1. What will Python return?
   >>> s='Clearwater, Florida'
   a) s[16:]
      >>> s[16:]
      'ida'
      'wear'
   c) (s[:5] + s[11]) *2 + s[5].upper() + s[6:10] + "!"
      >>> (s[:5] + s[11]) *2 + s[5].upper() + s[6:10] + "!
      'Clear Clear Water!'
Warm Up Exercises:
Recalling Strings

Write a function that will process each character in a string. If it finds a space, it should replace it with an underscore. Return the result.

def space_2_underscore(s):
    # signature: str -> str
    # converts spaces to underscore
    acc=""
    for i in s:
        if i==' ': # if it is a space
            acc += '_'
        else:
            acc += i
    return acc

==== RESTART: F:/09_Python_course  S17/01_Lecture/
>>> space_2_underscore("Palm Beach, Florida!")
'Palm_Beach,_Florida!'
Learning Objectives

• Compare and contrast lists and strings
• Learn list syntax
• Perform common tasks similar to those for strings
• Loop over lists
• Mutate lists
• Knowledge integration: Combine new list skills with flow control and functions to write programs
Lists vs. Strings

• Lists and strings bear many similarities
  • They can be manipulated in many ways that parallel strings
  • They may contain multiple elements
  • Their elements can be processed in loops
• They differ in two important ways
  • Lists may have elements of various types; they are not limited to characters.
  • Lists are *mutable*; the contents can be changed after they are created
Why lists are useful:
A preview of lists in action

```python
# program average_with_variables
score1 = 98.5
score2 = 100.0
score3 = 87.4
score4 = 90.1
score5 = 88.7

scoreT=score1+score2+score3+score4+score5
scoreAvg = scoreT / 5

print ('Your average is',scoreAvg,'.')
```

```python
# program average_with_list
scores=[98.5, 100.0, 87.4, 90.1, 88.7]

scoreT=0
for score in scores:
    scoreT += score
scoreAvg = scoreT/len(scores)

print ('Your average is',scoreAvg,'.')
```
Lists: Syntax

- Enclose a list in the square brackets, and separate elements by a comma
- An empty list is just the brackets []
- The index of the items in the list begins at 0

Syntax:
<list> = [ <element_0>, <element_1>, ..., <element_n>]

Examples:

# list containing three integers
>>> [1, 2, 3]
[1, 2, 3]

# list called A contains three strings
>>> A = ['a', 'b', 'c']
>>> A
['a', 'b', 'c']

# list called B contains three strings
>>> B = ['bobcat', 'bluebird', 'bear']
>>> B
['bobcat', 'bluebird', 'bear']

# list called prices contains three floats
>>> prices=[7.99, 4.50, 3.99]
>>> prices
[7.99, 4.5, 3.99]
List Homogeneity

- **Homogeneous list** - all elements of the list are of the same type
- **Inhomogeneous list** – the elements of the list are of mixed type
- The scope of this course is limited to homogeneous lists

Examples:

# Homogeneous lists
Alpha = ['a', 'b', 'c'] # elements of type str
Ints = [12, 14, 16] # elements of type int
Floats = [5.5, 22/7, 4.3 + 7.9]

# Inhomogeneous list
Mix = [4.50, 'boat', 9] # elements of type float, str, and int
Familiar Tasks Overloaded for Lists
List Indexing and Slicing

- Indexing and slicing work analogously to how they work with strings
- Indexing returns a value, whereas slicing returns a list

Indexing Syntax:
<list>[index]

Example:
```python
>>> cities=['Tampa','Orlando','Boca Raton']
>>> cities[0]
'Tampa'
>>> cities[-2]
'Orlando'
```

Slicing Syntax:
<list>[<start_index>:<stop_index>]

Example:
```python
>>> cities=['Tampa','Orlando','Boca Raton']
>>> cities[:2]
['Tampa', 'Orlando']
>>> cities[-2:]
['Orlando', 'Boca Raton']
```
List Addition and Repetition

• The + (concatenation) and * (repetition) operators are overloaded for lists

Concatenation Syntax:
\[ <list1> \ + \ <list2> \]

Repetition Syntax:
\[ <list1> \ * \ <int> \]

Examples:
```python
>>> ['a', 'b', 'c'] + ['d', 'e']
['a', 'b', 'c', 'd', 'e']
>>> [1, 2, 3]*2
[1, 2, 3, 1, 2, 3]
>>> ['cat', 'dog'] * 2
['cat', 'dog', 'cat', 'dog']
```
List Coercion Function

• The `list` function can be used to generate a list out of a string

Syntax:
`list(<string>)`

Example:
```python
>>> list ('Ft. Lauderdale')
['F', 't', '.', ' ', 'L', 'a', 'u', 'd', 'e', 'r', 'd', 'a', 'l', 'e']
```
List Comparisons

• The comparison operators <, >, >=, and <= are overloaded for lists to do comparisons of lists.

Syntax: 

Examples:

```python
<list1> < <list2> >>> [0, 0] < [1, 0]
True

<list1> > <list2> >>> [0,5] > [0,7]
False

<list1> <= <list2> >>> ['Zoo','bark']<=['aardvark','Dog']
True

<list1> >= <list2> >>> ['lion','tiger'] >= ['lion','monkey']
True
```
Length

• `len()` can be used on lists similar to how it is used on strings
• Returns the number of elements in the list

Syntax:

```
len(<list>)
```

Example:

```python
>>> cities = ['Miami', 'Ocala', 'Daytona', 'Tallahassee', 'Pensacola', 'Boca Raton']
>>> len(cities)
6
```
**in Operator**

- **in** can be used on lists similar to how it is used on strings
- Returns a Boolean to indicate if the query is in the list

Syntax:

```python
<query> in <list>
```

Examples:

```python
>>> mylist = ['Miami', 'Tampa', 'Clearwater']
>>> 'Miami' in mylist
True
>>> 'Boston' in mylist
False
```
Looping Over Lists
**for** Loops and Lists

- **for** loops can be used with lists
- The loop variable is assigned to each element in the list in turn

Syntax:

```
for <variable> in <list>:
    <loop_body>
```

Example:

```python
cities = ['Tampa', 'Orlando', 'Boca Raton']

for city in cities:
    print('Visit sunny', city, 'in the sunshine state!')
```

Visit sunny Tampa in the sunshine state!
Visit sunny Orlando in the sunshine state!
Visit sunny Boca Raton in the sunshine state!
Revisiting Average Score Computation

```python
# program average_with_list
scores=[98.5, 100.0, 87.4, 90.1, 88.7]

scoreT=0
for score in scores:
scoreT += score
scoreAvg = scoreT/len(scores)

print ('Your average is',scoreAvg,'.')
```

== RESTART: F:\09_Python_course S17\01_Lecture\lecture6_week8
Your average is 92.94 .
Exercise: for Loops and Lists

Write a function `get_longest` that determines which city in a list of cities has the longest name. The input is a list of cities. The function should return the name of the longest city. If a tie happens to occur, the function should return ‘tie’. A test list is given.

```python
cities = ['Orlando', 'Tampa', 'Boca Raton']
```
Exercise: **for** Loops and Lists

```python
def get_longest(cities):
    # signature: list (str) -> str
    # determines which city has longest name
    # or reports a tie

    longest = 0  # initialize to 0
    for city in cities:
        if len(city) > longest:  # new longest city found
            longest = len(city)  # update longest length
            longcity = city  # update corresponding city
        elif len(city) == longest:  # dealing with a tie
            longcity = 'tie'

    return longcity

cities = ['Orlando', 'Tampa', 'Boca Raton']

print(get_longest(cities))

==== RESTART: F:/09_Python_course
Boca Raton
List Mutability and Aliasing
Assignment vs. Mutation

• The contents of lists can be mutated after they are created
• This differs from assignment and reassignment

```python
>>> zoo=['lion','tiger','monkey']  # list assignment
>>> zoo=['elephant','zebra','kangaroo']  # list reassignment
```

• What if you want to change a specific element or elements?
  – Mutate the list by index assignment or slice assignment
List Mutation with Index Assignment

• Unlike strings, lists can be **mutated** after they are created; they can be changed.
• Use **index assignment** to mutate the list such that the value of an element of a list at the indicated index is changed.

**Syntax:**

```
<list> [<index>] = <new_element>
```

• Example:

```python
>>> cities
['Tampa', 'Orlando', 'Boca Raton']
>>> cities[1] = 'Ft. Lauderdale'
>>> cities
['Tampa', 'Ft. Lauderdale', 'Boca Raton']
```
List Mutability and Slice Assignment

• A slice can be cut out and replaced by new entries using the slice operator.
• The length of what is spliced out does not need to be the same as the length of what is spliced in.

Syntax:
<list> [<start_index>:<stop_index>] = <splice_list>

Example:
```python
>>> cities
['Tampa', 'Ft. Lauderdale', 'Boca Raton']
>>> cities[0:2] = ['Miami', 'Daytona', 'Tallahassee', 'Pensacola']
>>> cities
['Miami', 'Daytona', 'Tallahassee', 'Pensacola', 'Boca Raton']
```
List Mutability and Slice Assignment: Special Cases

- **Insertion:** If the list that is cut has length 0, the effect is an insert at that point

```python
>>> cities = ['Miami', 'Daytona', 'Tallahassee', 'Pensacola', 'Boca Raton']
>>> cities[1:1] = ['Ocala']
>>> cities
['Miami', 'Ocala', 'Daytona', 'Tallahassee', 'Pensacola', 'Boca Raton']
```

- **Deletion:** If the new splice list has length 0, the effect is a deletion of the region to be cut

```python
>>> cities
['Miami', 'Ocala', 'Daytona', 'Tallahassee', 'Pensacola', 'Boca Raton']
>>> cities[2:4] = []
>>> cities
['Miami', 'Ocala', 'Pensacola', 'Boca Raton']
```
What is an Alias?

Definition of ALIAS

: an assumed or additional name that a person (such as a criminal) sometimes uses • a fugitive using several aliases • He checked into the hotel using an alias. • John Smith, who goes by the alias Richard Jones

# Creation of an alias

```python
>>> mylist1 = ['Miami', 'Tampa', 'Clearwater']
>>> mylist2 = mylist1
```

# one list has two names

```
mylist1 = ['Miami', 'Tampa', 'Clearwater']
mylist2 = ['Miami', 'Tampa', 'Clearwater']
```
Aliasing and Mutability

- **Aliasing** occurs when two variables refer to the same list.

# both variables point to the exact same list. Aliasing has occurred.

```python
>>> mylist1 = ['Miami', 'Tampa', 'Clearwater']
>>> mylist2 = mylist1
mylist1 ➔ ['Miami', 'Tampa', 'Clearwater']
mylist2 ➔ ['Miami', 'Tampa', 'Clearwater']
```

# each variable points to its own copy of the list.

```python
>>> mylist1 = ['Miami', 'Tampa', 'Clearwater']
>>> mylist3 = ['Miami', 'Tampa', 'Clearwater']
mylist1 ➔ ['Miami', 'Tampa', 'Clearwater']
mylist3 ➔ ['Miami', 'Tampa', 'Clearwater']
```
Aliasing and Mutability Example

• Consider the following snippet of code.
  • How many copies of the list exist?
  • Which list(s), x, y, or z, point to which one?
    Draw a diagram to show this.
  • What will the lists x, y and z be after the mutation?

```python
>>> x = [1, 2, 3]
>>> y = x
>>> z = [1, 2, 3]
>>> y[1]=0
```
Aliasing and Mutability: Solution

• Consider the following snippet of code.
  • How many copies of the list exist? 2

• Which list(s), x, y, or z, point to which one? Draw a diagram to show this.

```
x  y  z
[1, 2, 3]  [1, 2, 3]  # (Note: shown before the mutation occurs)
```

• What will the lists x, y and z be after the mutation?

```
>>> x = [1, 2, 3]
>>> y = x
>>> z = [1, 2, 3]
>>> y[1]=0

>>> x
[1, 0, 3]
>>> y
[1, 0, 3]
>>> z
[1, 2, 3]
```
Detecting Aliasing

- Using `==` and `!=` compares the values but does not compare the reference, so aliasing will not be detected.

```python
>>> x = [1, 2, 3]
>>> z = [1, 2, 3]
>>> x == z  # not aliased but this isn't detected with ==
True
```

- Use `is` or `is not` to determine whether two references alias the same value.

Syntax:

```
<ref1> is <ref2>
<ref1> is not <ref2>
```

```python
>>> x is z  # aliasing is detected with `is` and `is not`
True
>>> x is not z
False
>>> x is y
True
>>> y is x
True
>>> y is z
False
```

31
def square(data):
    # signature: list(int) -> list(int)
    # squares the elements in data
    newdata = []  # empty list  # a new list is created
    for x in data:
        newdata.append(x**2)
    return newdata  # the new list is returned

data = [2, 3, 5]
print('data before function call:', data)
print('function returns: ', square(data))
print('data after function call:', data)

Will data be mutated by the square function?
Functions and Mutation

def square(data):
    # signature: list(int) -> list(int)
    # squares the elements in data

    newdata = []  # empty list
    for x in data:
        newdata.append(x**2)
    return newdata

# the list called data is the same before...
data = [2, 3, 5]
print('data before function call: ', data)

# ... and after the function call
print('function returns: ', square(data))
print('data after function call: ', data)

#### RESTART: F:/09_Python_course  S17/01_Lecture/

# the list called data is the same before...
data before function call: [2, 3, 5]
function returns: [4, 9, 25]
data after function call: [2, 3, 5]
Functions and Mutation

```python
def square_v2(data):
    # signature: list(int) -> NoneType
    # squares the elements in data

    i = 0
    while i < len(data):
        data[i] = data[i]**2
        i += 1
    return

data = [2, 3, 5]
print('data before function call:', data)
print('function returns: ', square_v2(data))
print ('data after function call:', data)
```

Will `data` be mutated by the `square_v2` function?
Functions and Mutation

```python
def square_v2(data):
    # signature: list(int) -> NoneType
    # squares the elements in data
    i = 0
    while i < len(data):
        data[i] = data[i]**2
        i += 1
    return

data = [2, 3, 5]
print('data before function call:', data)
print('function returns: ', square_v2(data))
print('data after function call:', data)
```

```text
== RESTART: F:/09_Python_course S17/01_Lecture/
data before function call: [2, 3, 5]
function returns:  None
data after function call: [4, 9, 25]
```

the list called `data` before function call...

#... differs from `data` after function call.
Methods Pertaining to Lists
Methods to Add Items to a List

```python
<list>.insert(<insert_index>, <item>)

>>> mylist = ['a', 'b', 'd']
>>> mylist.insert(2, 'c')  # Insert 'c' at element 2
>>> mylist
['a', 'b', 'c', 'd']

<list>.append(<item>)

>>> mylist = ['a', 'b', 'c']
>>> mylist.append('d')
>>> mylist
['a', 'b', 'c', 'd']

<list>.extend(<list_to_add>)

>>> mylist = ['a', 'b', 'c']
>>> mylist.extend(['d', 'e', 'f'])
>>> mylist
['a', 'b', 'c', 'd', 'e', 'f']
```
Methods to Remove Items from a List

```python
<list>.pop (<index>)

>>> mylist
['a', 'b', 'c', 'd', 'e', 'f']
>>> popped = mylist.pop(1)  # pop the item with index 1 from the list
>>> popped
'b'
>>> mylist
['a', 'c', 'd', 'e', 'f']

<list>.remove (<list_item>)

>>> mylist = ['a','b','c','d','e']
>>> mylist.remove('a')  # remove the item with the value 'a' from the list
>>> mylist
['b', 'c', 'd', 'e']
```
**Methods for Splitting and Joining**

```python
str.split(<string>,<delimiter>)

>>> mystring = 'Miami, Tampa, Clearwater'
>>> str.split(mystring)  # split on whitespace by default
['Miami,', 'Tampa,', 'Clearwater']
>>> str.split(mystring,',')  # split on commas
['Miami', 'Tampa', 'Clearwater']

str.splitlines(<string>)

>>> text = 'Line 1 a b c 

Line 2 d e f 

Line 3 g h i'
>>> str.splitlines(text)  # recall: \n is the newline character
['Line 1 a b c ', 'Line 2 d e f ', 'Line 3 g h i']

str.join(<delimiter>,<list>)

>>> mylist = ['Miami','Tampa','Clearwater']
>>> str.join('', mylist)  # join list elements without a delimiter
'MiamiTampaClearwater'
>>> str.join(', ', mylist)  # join list elements separated by a comma and space
'Miami, Tampa, Clearwater'
```