Lecture 6 - More on List, String and Tuple Methods

Computer Programming

Robert D. Vincent and Samia Hilal

Marianopolis College

September 24, 2021

Sequences

- Lists, tuples, and strings are all examples of *sequences*.
    - A string is an immutable sequence of characters.
    - A tuple is an immutable sequence of arbitrary values.
    - A list is an mutable sequence of arbitrary values.
- They are also all *iterable*, that is, they can be used in a `for` loop.
- They are also sometimes called *collections*, however, the term collection does not imply predictable ordering, whereas sequence does.

Copying mutable lists

- Remember: Slice expressions make a copy of a mutable list.

```
>>> x = [1, 2, 3]
>>> y = x
>>> z = x[:]
>>> print(x is y)
True
>>> print(x is z)
False
```

- A clearer alternative is the `copy()` method.

```
>>> x = [1, 2, 3]
>>> z = x.copy()
>>> print(x is z)
False
```

Copying immutable sequences

- Python will not copy tuples:

```
>>> t1 = (1, 2, 3)
>>> t2 = t1
>>> t3 = t1[:]
>>> print(t1 is t2, t1 is t3)
>>> True True
```

- or strings:

```
>>> s1 = "123"
>>> s2 = s1
>>> s3 = s1[:]
>>> print(s1 is s2,s1 is s3)
>>> True True
```

- because they are immutable!**no copy method**.

Reversing a list

- Like `sort()` and `sorted()`, there are two ways to reverse a list:

```
>>> x = [1, 2, 3]
>>> x.reverse() # Reverse IN PLACE
>>> print(x)
[3, 2, 1]
```

- The `reversed()` function returns a reversed **object** of the list. Use list() to print.

```
>>> x = [1, 2, 3]
>>> y = reversed(x) # New reversed object
>>> print(list(y))
[3, 2, 1]
>>> print(x)
[1, 2, 3]
```

Assigning lists

- Remember multiple names can refer to a list:

```
>>> x = [1, 2, 3]
>>> y = x
>>> print(y)
[1, 2, 3]
>>> x[0] = 0 # Changing an element in x
>>> print(y)
[0, 2, 3]
>>> x = [] # Assign x to a new empty list
>>> print(y)
[1, 2, 3]
```

- `x` is changed to point to a *new* empty list.
- The old list still exists, but only `y` references it.
- Assigning a list **not** the same as changing a list.

Clearing lists

- Python provides a `clear()` method:

```
>>> x = [1,2,3]
>>> y = x
>>> print(y)
[1,2,3]
>>> x.clear() # Compare with x=[]
>>> print(x,y)
[] []
```

- Empties both objects but keeps them.
- Note that **del** completely destroys an object.

```
>>> del x
>>> print (x) # Error! x no longer exists
>>> print(y) # But y is still there
[]
```

Three more functions on sequences

- `min()` - Find the minimum value of a sequence.
- `max()` - Find the maximum value of a sequence.
- `sum()` - Find the total value of a sequence.

```
>>> L = (3, 4, 9, -1)
>>> print(max(L))
9
>>> print(min(L))
-1
>>> print(sum(L))
15
>>> max(["anaconda","aardvark","zebra"])
'zebra'
```

Other uses of `min()` and `max()`

- Arguments can be several numbers or other ordered values:

```
>>> x,y,z=9,2,7
>>> print(max(x, y, z, 1))
9
>>> print(min(x, y, z, 1))
1
>>> print(min(-1.2, 5.4))
-1.2
>>> print(min("apple", "apricot"))
'apple'
```

A common list pitfall

- Most methods that change the list *don't* return a useful value.
- A common mistake:

```
>>> L = [5, 7, 2, 6, 1]
>>> L = L.sort()
>>> print(L)
None           # Where did L go?
```

- **None** in Python means "nothing" or "no value".
- `append()`, `clear()`, `extend()`, `insert()`, `remove()`, `reverse()`, and `sort()` all modify the list and return **None**.
- **Note:** Above methods not available for strings.

Avoiding this pitfall

- Assuming **L** is a list and **x** is a list element, you can add an element to the list with either:

```
L.append(x) # Modify L in place.
```

- or

```
L = L + [x] # L refers to new list
```

- But any of the following are generally wrong:

```
L.append([x]) # Just use x
L = L.append(x) # L == None
L + [x]        # Doesn't modify L
L = L + x      # x is not a list
```

Summary

- Review the difference between mutable and immutable types.
- Recognize the possible operations on lists.
- Avoid errors when using list methods that return `None`.
- More background on lists: `http://www.greenteapress.com/thinkpython/html/thinkpython011.html`