Data Structures

Contents

- List
- Tuple
- Sequence
- References
- More about Strings

List

- A list is an ordered sequence of Python objects of any type
  - The objects do not have to all be the same type
- A list is constructed by writing its items enclosed in square brackets, with the items separated by commas, e.g.,

```
['Seahawks', 2014, 'CenturyLink Field']
[5, 10, 4, 5]
['spam', 'ni']
```

- Lists are usually assigned to a name

```
>>> team = ['Seahawks', 2014, 'CenturyLink Field']
>>> nums = [5, 10, 4, 5]
>>> words = ['spam', 'ni']
```

- Once created, you can add, remove, or search for items in the list
- A list is a mutable data type

List

- List operations

Function or Method	Example	Value	Description
len	len(words)	2	number of items in list
max	max(numbers)	10	greatest (items must have same type)
min	min(numbers)	4	least (items must have same type)
sum	sum(nums)	24	total (items must be numbers)
count	nums.count(5)	2	number of occurrences of an object
index	nums.index(4)	2	index of first occurrence of an object
reverse	words.reverse()	["ni", "spam"]	reverses the order of the items
clear	team.clear()	[]	[] is the empty list
append	nums.append(7)	[5, 10, 4, 5, 7]	inserts object at end of list
extend	nums.extend([1, 2])	[5, 10, 4, 5, 1, 2]	inserts new list's items at end of list
del	del team[−1]	["Seahawks", 2014]	removes item with stated index
remove	nums.remove(5)	[10, 4, 5]	removes first occurrence of an object
insert	nums.insert(1, "wink")	["spam", "wink", "ni"]	insert new item before item of given index
+	['a', 1] + [2, 'b']	['a', 1, 2, 'b']	concatenation; same as ['a', 1].extend([2,'b'])
*	[0] * 3	[0, 0, 0]	list repetition

List

- Like the characters in a string, items in a list are indexed from the front with positive indices starting with 0, and from the back with negative indices starting with –1

- The value of the item having index `i` can be changed with a statement of the form **`listName[i] = newValue`**

```
>>> words = ['spam', 'ni']
>>> words[1] = 'eggs'
>>> words
['spam', 'eggs']
```

Objects and Classes

- When you write `i = 5`, you are creating an object `i` of class `int`
  - Use `help(int)` to get more information about `int` class
- A class can have methods, i.e., functions defined for use with respect to that class only, e.g.,

```
mylist.append('an item')
```

  will add the string `'an item'` to the end of the list `mylist`
  - We can add any objects to a list, even other lists
- Some methods do things without returning any value

```
>>> a = [3, 2, 4, 1]
>>> print(a.sort())
None
>>> a
[1, 2, 3, 4]
```

Objects and Classes

```
shoplist = ['apple', 'mango', 'carrot', 'banana']

print('I have {0} items to purchase.'.format(len(shoplist)))

print('These items are:', end='')
for item in shoplist:
   print(item, end='')

print('\nI also have to buy rice.')
shoplist.append('rice')
print('My list is now', shoplist)

print('I will sort my shopping list now')
shoplist.sort()
print('Sorted list is', shoplist)

print('The first item I will buy is', shoplist[0])
olditem = shoplist[0]
del shoplist[0]
print('I bought the', olditem)
print('My shopping list is now', shoplist)
```

Objects and Classes

[Run]

```
I have 4 items to purchase.
These items are: apple mango carrot banana
I also have to buy rice.
My list is now ['apple', 'mango', 'carrot', 'banana', 'rice']
I will sort my shopping list now
Sorted list is ['apple', 'banana', 'carrot', 'mango', 'rice']
The first item I will buy is apple
I bought the apple
My shopping list is now ['banana', 'carrot', 'mango', 'rice']
```

- The **`append`** and **`sort`** methods for the list class actually alters the list
- The **`del`** statement removes the specified item form the specified object

Objects and Classes

- More list functions and methods are shown below:

```
>>> team = ['Seahawks', 2014, 'CenturyLink Field']
>>> nums = [5, 10, 4, 5]
>>> words = ['spam', 'ni']
>>> max(nums)
10
>>> min(nums)
4
>>> sum(nums)
24
>>> nums.count(5)
2
>>> nums.index(4)
2
>>> words.reverse()
>>> words
['ni', 'spam']
>>> nums.append(7)
>>> nums
[5, 10, 4, 5, 7]
```

Objects and Classes

```
>>> nums.extend([1,2])
>>> nums
[5, 10, 4, 5, 7, 1, 2]
>>> nums.remove(5)  # removes first occurrence of an object
>>> nums
[10, 4, 5, 7, 1, 2]
>>> del nums[1:3]
>>> nums
[10, 7, 1, 2]
>>> words.insert(1,'wink')  # before item of given index
>>> words
['ni', 'wink', 'spam']
>>> ['a',1] + [2,'b']
['a', 1, 2, 'b']
>>> [0] * 3
[0, 0, 0]
>>>
```

Objects and Classes

- The **`split`** method turns a single string into a list of substrings

```
>>> 'a,b,c'.split(',')
['a', 'b', 'c']
>>> 'a**b**c'.split('**')
['a', 'b', 'c']
>>> 'a**b**c'.split('*')
['a', '', 'b', '', 'c']
>>> 'a\nb\nc'.split()
['a', 'b', 'c']
>>> 'a b c'.split()
['a', 'b', 'c']
>>> 'a b c'.split(',')
['a b c']
```

  - Three commonly used separators are `','`, `'\n'`, and `' '`
  - If no separator is specified, the split method uses whitespace (newline, tab, or space) as the separator

Objects and Classes

- The `join` method turns a list of strings into a single string consisting of the elements of the list concatenated together and separated by a specified separator

```
line = ['To', 'be', 'or', 'not', 'to', 'be.']
string = ' '.join(line)
print(string)
krispies = ['Snap', 'Crackle', 'Pop']
print(', '.join(krispies))
```

[Run]

```
To be or not to be.
Snap, Crackle, Pop
```

Tuple

- Tuples, like lists, are ordered sequences of items but are immutable
  - Tuples have no **`append`**, **`extend`**, or **`insert`** methods
  - All other list functions and methods apply to tuples, and its items can also be accessed by indices
  - Useful if you want to create a sequence that cannot be modified, especially by mistake
- Tuples are written as comma-separated sequences enclosed in parentheses, or they can often be written without the parentheses
  - The following two statements create tuple **`t`** and assign it the same value

```
t = ('a', 'b', 'c')
t = 'a', 'b', 'c'
```

  - **`print`** statement always displays tuples enclosed in parentheses

Tuple

```
zoo = ('python', 'elephant', 'penguin')
print('Number of animals in the zoo is', len(zoo))

new_zoo = 'monkey', 'camel', zoo
print('All animals in new zoo are', new_zoo)
print('Number of cages in the new zoo is', len(new_zoo))
print('Animals brought from old zoo are', new_zoo[2])
print('Last animal brought from old zoo is', new_zoo[2][2])
print('Number of animals in the new zoo is',
    len(new_zoo) - 1 + len(new_zoo[2]))
```

[Run]

```
Number of animals in the zoo is 3
All animals in new zoo are ('monkey', 'camel', ('python',
'elephant', 'penguin'))
Number of cages in the new zoo is 3
Animals brought from old zoo are ('python', 'elephant',
'penguin')
Last animal brought from old zoo is penguin
Number of animals in the new zoo is 5
```

Tuple

- A statement such as

  ```
  (x, y, z) = (5, 6, 7)
  ```

  creates three variables and assigns values to them

  – Can also be written as

  ```
  x, y, z = 5, 6, 7
  ```

  which can be thought of as making three variable assignments with a single statement

```
x = 5
y = 6
x, y = y, x
print(x, y)
```

[Run]

```
6 5
```

Sequence

- Major features of sequences (i.e., strings, lists, and tuples):
  - Membership test
  - Indexing operation
  - Slicing operation: retrieves a part of the sequence

Sequence

```
shoplist = ['apple', 'mango', 'carrot', 'banana']
name = 'swaroop'

# Indexing or 'Subscription' operation #
print('Item 3 is', shoplist[3])
print('Item -1 is', shoplist[-1])
print('Item -2 is', shoplist[-2])
print('Character 0 is', name[0])

# Slicing on a list #
print('Item 1 to 3 is', shoplist[1:3])
print('Item 2 to end is', shoplist[2:])
print('Item 1 to -1 is', shoplist[1:-1])
print('Item start to end is', shoplist[:])

# Slicing on a string #
print('characters 1 to 3 is', name[1:3])
print('characters 2 to end is', name[2:])
print('characters 1 to -1 is', name[1:-1])
print('characters start to end is', name[:])
```

Sequence

[Run]

```
Item 3 is banana
Item -1 is banana
Item -2 is carrot
Character 0 is s
Item 1 to 3 is ['mango', 'carrot']
Item 2 to end is ['carrot', 'banana']
Item 1 to -1 is ['mango', 'carrot']
Item start to end is ['apple', 'mango', 'carrot', 'banana']
characters 1 to 3 is wa
characters 2 to end is aroop
characters 1 to -1 is waroo
characters start to end is swaroop
```

- The index $-k$ refers to the kth last item in the sequence
- The slice returned starts at the *start* position and ends just before the *end* position

Sequence

- The numbers are optional in a slicing operation but the colon isn't
  - If the first number is not specified, Python will start at the beginning of the sequence
  - If the second number is left out, Python will stop at the end of the sequence

- You can also provide a third argument for the slice, which is the *step* for the slicing (by default, the step size is 1)

Sequence

```
>>> shoplist = ['apple', 'mango', 'carrot', 'banana']
>>> shoplist[::1]
['apple', 'mango', 'carrot', 'banana']
>>> shoplist[::2]
['apple', 'carrot']
>>> shoplist[::3]
['apple', 'banana']
>>> shoplist[::-1]
['banana', 'carrot', 'mango', 'apple']
>>> shoplist[::-2]
['banana', 'mango']
>>> shoplist[::]
['apple', 'mango', 'carrot', 'banana']
>>>
```

- Lists of tuples play a prominent role in analyzing data
  (See the next page)

Sequence

- If `L` is a list of tuples, then `L[0]` is the first tuple
  - `L[0][0]` is the first item of the first tuple
  - `L[-1]` (same as `L[len(L) - 1]`) is the last tuple
  - `L[-1][-1]` is the last item of the last tuple

```
regions = [('Northeast', 55.3), ('Midwest', 66.9),
           ('South', 114.6), ('West', 71.9)]
print('The 2010 population of the', regions[1][0], 'was',
      regions[1][1], 'million.')
totalPop = regions[0][1] + regions[1][1] + regions[2][1]
+ regions[3][1]
print('Total 2010 population of the U.S: {0:.1f} million.'
.format(totalPop))
```

[Run]

```
The 2010 population of the Midwest was 66.9 million.
Total 2010 population of the U.S: 236.8 million.
```

Sequence

- The `list` function converts tuples or strings to lists

```
>>> list(('a', 'b'))
['a', 'b']
>>> list("Python")
['P', 'y', 't', 'h', 'o', 'n']
>>> "Python".split()
['Python']
>>> "P y t h o n".split(' ')
['P', 'y', 't', 'h', 'o', 'n']
>>> "P y t h o n".split()
['P', 'y', 't', 'h', 'o', 'n']
```

References

- When a mutable object is assigned to a variable, the variable name just points to the memory where the object is stored

```
print('Simple Assignment')
shoplist = ['apple', 'mango', 'carrot', 'banana']
# mylist is just another name pointing to the same object!
mylist = shoplist
# I purchased the first item, so I remove it from the list
del mylist[0]

print('shoplist is', shoplist)
print('mylist is', mylist)
# Notice that both shoplist and mylist print the same list
# without the 'apple' confirming that they point to the same
# object

print('Copy by making a full slice')
# Make a copy by using a full slice
mylist = shoplist[:]
```

References

```
# Remove first item
del mylist[0]
print('shoplist is', shoplist)
print('mylist is', mylist)
# Notice that now the two lists are different
```

[Run]

```
Simple Assignment
shoplist is ['mango', 'carrot', 'banana']
mylist is ['mango', 'carrot', 'banana']
Copy by making a full slice
shoplist is ['mango', 'carrot', 'banana']
mylist is ['carrot', 'banana']
```

- Slicing operation should be used to make a copy of a sequence
  - If you just assign the variable name to another name, both of them will *refer* to the same object

References

```
>>> x = [1, 2, 3, 4]
>>> y = x                  # y points to x
>>> z = x[:]               # z is a copy of x
>>> x.append(5)
>>> y
[1, 2, 3, 4, 5]
>>> z                      # z is still a copy of previous x
[1, 2, 3, 4]
>>> w = y.copy()           # another way of copying a list
>>> del y[0]
>>> x                      # x and y point to the same thing
[2, 3, 4, 5]
>>> y
[2, 3, 4, 5]
>>> z
[1, 2, 3, 4]
>>> w
[1, 2, 3, 4, 5]
```

References

- When a variable is created with an assignment statement, the value on the right side becomes an object in memory, and the variable references (that is, points to) that object
- When a list is altered after being assigned to a variable, changes are made to the object in the referenced memory location
- However, when a variable whose value is a number, string, or tuple, has its value changed, Python designates a new memory location to hold the new value and the variable references that new object
- We say that lists can be changed in place, but numbers, strings, and tuples cannot
- Objects that can be changed in place are called mutable, and objects that cannot be changed in place are called immutable

More about Strings

- The strings are objects of the class **`str`**
  - The next example shows more string methods that are useful
  - For a complete list of such methods, see **`help(str)`**

```
# This is a string object
name = 'Swaroop'

if name.startswith('Swa'):
    print('Yes, the string starts with "Swa"')

if 'a' in name:
    print('Yes, it contains the character "a"')

if name.find('war') != -1:
    print('Yes, it contains the string "war"')

delimiter = '_*_'
mylist = ['Brazil', 'Russia', 'India', 'China']
print(delimiter.join(mylist))
```

More about Strings

[Run]

```
Yes, the string starts with "Swa"
Yes, it contains the character "a"
Yes, it contains the string "war"
Brazil_*_Russia_*_India_*_China
```