Lists

- List is an ordered collection of objects
  - can contain any sort of object: numbers, strings, and even other lists
  - unlike strings, lists may be changed in place. (Lists are mutable)

- + concatenates list
- * repeats list

```
% python
>>> len([1, 2, 3])                    # Length
3
>>> [1, 2, 3] + [4, 5, 6]             # Concatenation
[1, 2, 3, 4, 5, 6]
>>> ['Ni!'] * 4                       # Repetition
['Ni!', 'Ni!', 'Ni!', 'Ni!']
```

1

List Iteration and Comprehensions

```
>>> 3 in [1, 2, 3]                    # Membership
True
>>> for x in [1, 2, 3]:
...     print(x, end=' ')              # Iteration (2.X uses: print x,)
...
1 2 3
```

```
>>> res = [c * 4 for c in 'SPAM']     # List comprehensions
>>> res
['SSSS', 'PPPP', 'AAAA', 'MMMM']
```

```
>>> res = []
>>> for c in 'SPAM':                  # List comprehension equivalent
...     res.append(c * 4)
...
>>> res
['SSSS', 'PPPP', 'AAAA', 'MMMM']
```

2

Indexing, Slicing, and Matrixes

```
>>> L = ['spam', 'Spam', 'SPAM!']
>>> L[2]                          # Offsets start at zero
'SPAM!'
>>> L[-2]                         # Negative: count from the right
'Spam'
>>> L[1:]                         # Slicing fetches sections
['Spam', 'SPAM!']
```

```
>>> matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> matrix[1]
[4, 5, 6]
>>> matrix[1][1]
5
>>> matrix[2][0]
7
>>> matrix = [[1, 2, 3],
...           [4, 5, 6],
...           [7, 8, 9]]
>>> matrix[1][1]
5
```

3

Changing Lists in Place

- List support operations that change a list object *in place*. The operations modify the list object directly—overwriting its former value—without requiring that you make a new copy

```
>>> L = ['spam', 'Spam', 'SPAM!']
>>> L[1] = 'eggs'                 # Index assignment
>>> L
['spam', 'eggs', 'SPAM!']

>>> L[0:2] = ['eat', 'more']      # Slice assignment: delete+insert
>>> L                             # Replaces items 0,1
['eat', 'more', 'SPAM!']

>>> L = [1, 2, 3]
>>> L[1:2] = [4, 5]               # Replacement/insertion
>>> L
[1, 4, 5, 3]
>>> L[1:1] = [6, 7]               # Insertion (replace nothing)
>>> L
[1, 6, 7, 4, 5, 3]
>>> L[1:2] = []                   # Deletion (insert nothing)
>>> L
[1, 7, 4, 5, 3]
```

4

Changing Lists in Place

```
>>> L = [1]
>>> L[:0] = [2, 3, 4]          # Insert all at :0, an empty slice at front
>>> L
[2, 3, 4, 1]
>>> L[len(L):] = [5, 6, 7]     # Insert all at len(L):, an empty slice at end
>>> L
[2, 3, 4, 1, 5, 6, 7]
>>> L.extend([8, 9, 10])       # Insert all at end, named method
>>> L
[2, 3, 4, 1, 5, 6, 7, 8, 9, 10]
```

5

List method calls

- list.append(*x*)
  - Add an item to the end of the list. Equivalent to a[len(a):] = [x].
- list.extend(*L*)
  - Extend the list by appending all the items in the given list. Equivalent to a[len(a):] = L.
- list.insert(*i*, *x*)
  - Insert an item at a given position. The first argument is the index of the element before which to insert, so a.insert(0, x) inserts at the front of the list, and a.insert(len(a), x) is equivalent to a.append(x).
- list.remove(*x*)
  - Remove the first item from the list whose value is *x*. It is an error if there is no such item.

6

List method calls

- list.pop([*i*])
  - Remove the item at the given position in the list, and return it. If no index is specified, a.pop() removes and returns the last item in the list. (The square brackets around the *i* in the method signature denote that the parameter is optional)
- list.clear()
  - Remove all items from the list. Equivalent to del a[:].
- list.index(*x*)
  - Return the index in the list of the first item whose value is *x*. It is an error if there is no such item.
- list.count(*x*)
  - Return the number of times *x* appears in the list.

7

List method calls

- list.sort(*key=None*, *reverse=False*)
  - Sort the items of the list in place (the arguments can be used for sort customization).
- list.reverse()
  - Reverse the elements of the list in place.
- list.copy()
  - Return a shallow copy of the list. Equivalent to a[:].

8

List method calls

```
>>> L = ['eat', 'more', 'SPAM!']
>>> L.append('please')                # Append method call: add item at end
>>> L
['eat', 'more', 'SPAM!', 'please']
>>> L.sort()                          # Sort list items ('S' < 'e')
>>> L
['SPAM!', 'eat', 'more', 'please']
```

- By default sorts in ascending order.

```
>>> L = ['abc', 'ABD', 'aBe']
>>> L.sort()                                # Sort with mixed case
>>> L
['ABD', 'aBe', 'abc']
>>> L = ['abc', 'ABD', 'aBe']
>>> L.sort(key=str.lower)                   # Normalize to lowercase
>>> L
['abc', 'ABD', 'aBe']
>>>
>>> L = ['abc', 'ABD', 'aBe']
>>> L.sort(key=str.lower, reverse=True)     # Change sort order
>>> L
['aBe', 'ABD', 'abc']
```

9

List method calls

- L=L.append(X)
- Pythons as a builtin function, which sorts any collection (not just lists) and returns a new list for the result (instead of in-place changes):

```
>>> L = ['abc', 'ABD', 'aBe']
>>> sorted(L, key=str.lower, reverse=True)          # Sorting built-in
['aBe', 'ABD', 'abc']

>>> L = ['abc', 'ABD', 'aBe']
>>> sorted([x.lower() for x in L], reverse=True)    # Pretransform items: differs!
['abe', 'abd', 'abc']
```

10

List method calls

```
>>> L = [1, 2]
>>> L.extend([3, 4, 5])          # Add many items at end (like in-place +)
>>> L
[1, 2, 3, 4, 5]
>>> L.pop()                      # Delete and return last item (by default: -1)
5
>>> L
[1, 2, 3, 4]
>>> L.reverse()                  # In-place reversal method
>>> L
[4, 3, 2, 1]
>>> list(reversed(L))            # Reversal built-in with a result (iterator)
[1, 2, 3, 4]

>>> L = []
>>> L.append(1)                  # Push onto stack
>>> L.append(2)
>>> L
[1, 2]
>>> L.pop()                      # Pop off stack
2
>>> L
[1]
```


List method calls

```
>>> L = ['spam', 'eggs', 'ham']
>>> L.index('eggs')              # Index of an object (search/find)
1
>>> L.insert(1, 'toast')         # Insert at position
>>> L
['spam', 'toast', 'eggs', 'ham']
>>> L.remove('eggs')             # Delete by value
>>> L
['spam', 'toast', 'ham']
>>> L.pop(1)                     # Delete by position
'toast'
>>> L
['spam', 'ham']
>>> L.count('spam')              # Number of occurrences
1

>>> L = ['spam', 'eggs', 'ham', 'toast']
>>> del L[0]                     # Delete one item
>>> L
['eggs', 'ham', 'toast']
>>> del L[1:]                    # Delete an entire section
>>> L                            # Same as L[1:] = []
['eggs']
```

Dictionaries

- Dictionaries are unordered collections; in which items are stored and fetched by *key*, instead of by positional offset
- When coded as a literal expression, a dictionary is written as a series of *key*:*value* pairs, separated by commas, enclosed in curly braces.
- An empty dictionary is an empty set of braces
- You can nest dictionaries by simply coding one as a value inside another dictionary, or within a list or tuple

13

Basic Dictionary Operations

```
% python
>>> D = {'spam': 2, 'ham': 1, 'eggs': 3}     # Make a dictionary
>>> D['spam']                                # Fetch a value by key
2
>>> D                                        # Order is "scrambled"
{'eggs': 3, 'spam': 2, 'ham': 1}
```

```
>>> len(D)                                   # Number of entries in dictionary
3
>>> 'ham' in D                               # Key membership test alternative
True
>>> list(D.keys())                           # Create a new list of D's keys
['eggs', 'spam', 'ham']
```

14

Changing Dictionaries in Place

```
>>> D
{'eggs': 3, 'spam': 2, 'ham': 1}

>>> D['ham'] = ['grill', 'bake', 'fry']          # Change entry (value=list)
>>> D
{'eggs': 3, 'spam': 2, 'ham': ['grill', 'bake', 'fry']}

>>> del D['eggs']                                # Delete entry
>>> D
{'spam': 2, 'ham': ['grill', 'bake', 'fry']}

>>> D['brunch'] = 'Bacon'                        # Add new entry
>>> D
{'brunch': 'Bacon', 'spam': 2, 'ham': ['grill', 'bake', 'fry']}
```

15

Basic Dictionary Operations

- The dictionary values and items methods return all of the dictionary's values and (*key*,*value*) pair tuples, respectively

```
>>> D = {'spam': 2, 'ham': 1, 'eggs': 3}
>>> list(D.values())
[3, 2, 1]
>>> list(D.items())
 [('eggs', 3), ('spam', 2), ('ham', 1)]
```

- get method returns a default value—None, or a passed-in default—if the key doesn't exist

```
>>> D.get('spam')                 # A key that is there
2
>>> print(D.get('toast'))         # A key that is missing
None
>>> D.get('toast', 88)
88
```

16

Basic Dictionary Operations

- The update method provides something similar to concatenation for dictionaries

```
>>> D
{'eggs': 3, 'spam': 2, 'ham': 1}
>>> D2 = {'toast':4, 'muffin':5}          # Lots of delicious scrambled order here
>>> D.update(D2)
>>> D
{'eggs': 3, 'muffin': 5, 'toast': 4, 'spam': 2, 'ham': 1}
```

- The pop method deletes a key from a dictionary and returns the value it had.

```
# pop a dictionary by key
>>> D
{'eggs': 3, 'muffin': 5, 'toast': 4, 'spam': 2, 'ham': 1}
>>> D.pop('muffin')
5
>>> D.pop('toast')                        # Delete and return from a key
4
>>> D
{'eggs': 3, 'spam': 2, 'ham': 1}
```

17