

Python Data Structures

By Greg Felber

Lists

- An ordered group of items
- Does not need to be the same type
 - Could put numbers, strings or donkeys in the same list
- List notation
 - `A = [1, "This is a list", c, Donkey("kong")]`

Methods of Lists

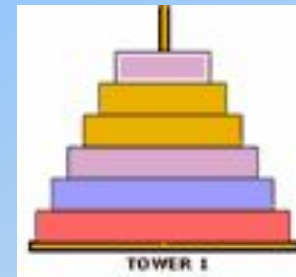
- `List.append(x)`
 - adds an item to the end of the list
- `List.extend(L)`
 - Extend the list by appending all in the given list L
- `List.insert(l,x)`
 - Inserts an item at index l
- `List.remove(x)`
 - Removes the first item from the list whose value is x

Examples of other methods

- `a = [66.25, 333, 333, 1, 1234.5]` //Defines List
 - `print a.count(333), a.count(66.25), a.count('x')` //calls method
 - `2 1 0` //output
- `a.index(333)`
 - //Returns the first index where the given value appears
 - `1` //output
- `a.reverse()` //Reverses order of list
 - `a` //Prints list a
 - `[333, 1234.5, 1, 333, -1, 66.25]` //Output
- `a.sort()`
 - `a` //Prints list a
 - `[-1, 1, 66.25, 333, 333, 1234.5]` //Output

Using Lists as Stacks

- The last element added is the first element retrieved
- To add an item to the stack, `append()` must be used
 - `stack = [3, 4, 5]`
 - `stack.append(6)`
 - Stack is now `[3, 4, 5, 6]`
- To retrieve an item from the top of the stack, `pop` must be used
 - `Stack.pop()`
 - 6 is output
 - Stack is now `[3, 4, 5]` again



Using Lists as Queues

- First element added is the first element retrieved
- To do this `collections.deque` must be implemented



```
>>> from collections import deque
>>> queue = deque(["Eric", "John", "Michael"])
>>> queue.append("Terry")           # Terry arrives
>>> queue.append("Graham")        # Graham arrives
>>> queue.popleft()               # The first to arrive now leaves
'Eric'
>>> queue.popleft()               # The second to arrive now leaves
'John'
>>> queue                          # Remaining queue in order of arrival
deque(['Michael', 'Terry', 'Graham'])
```

List Programming Tools

- Filter(function, sequence)
 - Returns a sequence consisting of the items from the sequence for which function(item) is true

```
>>> def f(x): return x % 2 != 0 and x % 3 != 0
...
>>> filter(f, range(2, 25))
[5, 7, 11, 13, 17, 19, 23]
```

- Computes primes up to 25

Map Function

- Map(function, sequence)
 - Calls function(item) for each of the sequence's items

```
>>> def cube(x): return x*x*x
...
>>> map(cube, range(1, 11))
[1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
```

- Computes the cube for the range of 1 to 11

Reduce Function

- Reduce(function, sequence)
 - Returns a single value constructed by calling the binary function (function)

```
>>> def add(x, y): return x+y
...
>>> reduce(add, range(1, 11))
55
```

- Computes the sum of the numbers 1 to 10

The del statement

- A specific index or range can be deleted

```
>>> a = [-1, 1, 66.25, 333, 333, 1234.5]
>>> del a[0]
>>> a
[1, 66.25, 333, 333, 1234.5]
>>> del a[2:4]
>>> a
[1, 66.25, 1234.5]
>>> del a[:]
>>> a
[]
```

Tuples

- Tuple
 - A number of values separated by commas
 - Immutable
 - Cannot assign values to individual items of a tuple
 - However tuples can contain mutable objects such as lists

```
>>> t = 12345, 54321, 'hello!'
>>> t[0]
12345
>>> t
(12345, 54321, 'hello!')
>>> # Tuples may be nested:
... u = t, (1, 2, 3, 4, 5)
>>> u
((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
```

- Single items must be defined using a comma
 - Singleton = 'hello',

Sets

- An unordered collection with no duplicate elements
- `Basket = ['apple', 'orange', 'apple', 'pear']`
- `Fruit = set(basket)`
- `Fruit`
 - `Set(['orange', 'apple', 'pear'])`

Dictionaries

- Indexed by keys
 - This can be any immutable type (strings, numbers...)
 - Tuples can be used if they contain only immutable objects

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127
>>> tel
{'sape': 4139, 'guido': 4127, 'jack': 4098}
>>> tel['jack']
4098
>>> del tel['sape']
>>> tel['irv'] = 4127
>>> tel
{'guido': 4127, 'irv': 4127, 'jack': 4098}
>>> tel.keys()
['guido', 'irv', 'jack']
>>> 'guido' in tel
True
```

Looping Techniques

- `iteritems()`:
 - for retrieving key and values through a dictionary

```
>>> knights = {'gallahad': 'the pure', 'robin': 'the brave'}
>>> for k, v in knights.iteritems():
...     print k, v
...
gallahad the pure
robin the brave
```

Looping Techniques

- Enumerate():
 - for the position index and values in a sequence

```
>>> for i, v in enumerate(['tic', 'tac', 'toe']):  
...     print i, v  
...  
0 tic  
1 tac  
2 toe
```


- Zip():
 - for looping over two or more sequences

```
>>> questions = ['name', 'quest', 'favorite color']
>>> answers = ['lancelot', 'the holy grail', 'blue']
>>> for q, a in zip(questions, answers):
...     print 'What is your {0}?  It is {1}.'.format(q, a)
...
What is your name?  It is lancelot.
What is your quest?  It is the holy grail.
What is your favorite color?  It is blue.
```


Comparisons

- Operators “in” and “not in” can be used to see if an item exists in a sequence
- Comparisons can be chained
 - $a < b == c$
 - This tests whether a is less than b and that b equals c