Lecture1

Unknown Author

October 21, 2013

1 Lists

Lists are a basic data structure in python:

In [1]:
```
names = ['Tom', 'Dick', 'Hary']
```

In [2]:
```
names[0]
```

Out [2]:
```
'Tom'
```

In [5]:
```
for name in names:
    print "Current name is " + name
```

```
Current name is Tom
Current name is Dick
Current name is Hary
```

Notice that we don't have to use explicit indexes in for loops.

This is because a list is considered to be an **iterables** in python. Iterables are just objects with a defenition of what iteration over them is.

List objects have a series of methods on them useful for operating on lists

In [6]:
```
dir(names)
```

Out [6]:
```
['__add__',
 '__class__',
 '__contains__',
 '__delattr__',
 '__delitem__',
 '__delslice__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__getslice__',
 '__gt__',
```

```
 '__hash__',
 '__iadd__',
 '__imul__',
 '__init__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__mul__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__reversed__',
 '__rmul__',
 '__setattr__',
 '__setitem__',
 '__setslice__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 'append',
 'count',
 'extend',
 'index',
 'insert',
 'pop',
 'remove',
 'reverse',
 'sort']
```

In [7]:
```
names.append('John')
```

In [8]:
```
names
```

Out [8]:
```
['Tom', 'Dick', 'Hary', 'John']
```

In [11]:
```
names.sort()
```

In [12]:
```
names
```

Out [12]:
```
['Dick', 'Hary', 'John', 'Tom']
```

In [15]:
```
names.remove('John')
```

In [16]:
```
names
```

Out [16]:
```
['Dick', 'Hary', 'Tom']
```

In [18]:
```
print help(list.remove)
```

```
Help on method_descriptor:

remove(...)
    L.remove(value) -- remove first occurrence of value.
    Raises ValueError if the value is not present.

None
```

In [19]:
```
print help(list.sort)
```

```
Help on method_descriptor:

sort(...)
    L.sort(cmp=None, key=None, reverse=False) -- stable sort *IN
PLACE*;
    cmp(x, y) -> -1, 0, 1

None
```

2 Tuples

Tuples are like lists except they are **immutable**. Immutable just means they can't be changed once they're created.

In [23]:
```
names_tuple = ('Tom', 'Dick', 'Harry')
```

In [24]:
```
names_tuple
```

Out [24]:
```
('Tom', 'Dick', 'Harry')
```

Tuples just switch `[]` for `()`, but behavior is slightly different.

1. Order never changes
2. Length doesn't change unless
3. Can't be changed at all in-place

In [22]:
```
dir(names)
```

Out [22]:
```
['__add__',
 '__class__',
 '__contains__',
 '__delattr__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__getnewargs__',
```

```
'__getslice__',
'__gt__',
'__hash__',
'__init__',
'__iter__',
'__le__',
'__len__',
'__lt__',
'__mul__',
'__ne__',
'__new__',
'__reduce__',
'__reduce_ex__',
'__repr__',
'__rmul__',
'__setattr__',
'__sizeof__',
'__str__',
'__subclasshook__',
'count',
'index']
```

In [27]:
```
names.index('Dick')
```

Out [27]: 1

Dictionaries

Python dictionaries are an implementation of what is called a *hash-table*. A *hash table* is a particular (and efficient) way of creating an associative data structure with key/value pairs. Each key has a value that it returns. Importantly, it requires that each key be unique.

In [36]:
```
bob1 = {'height': 76, 'first_name': 'Bob', 'last_name': 'Waldron', 'gende
```

dictionaries are created by placing `{}` and filling it with `'key': value` pairs. The key must be a hashable, practically this means it should be a unique string for the dictionary `{ 'key1': value, 'key2': value2 }`

In [33]:
```
bob2 = {'height': 76, 'height': 62 }
```

In [34]:
```
bob2
```

Out [34]: {'height': 62}

Be carefull with dictionary keys.

In [37]:
```
bob1.values()
```

Out [37]: ['M', 'Bob', 'Waldron', 76]

In [44]:
```
for k,v in bob1.items():
    print "Key is: " + k + " Value is: " + str(v)
```

```
Key is: gender Value is: M
Key is: first_name Value is: Bob
Key is: last_name Value is: Waldron
Key is: height Value is: 76
```

Items method just places tuples for each key/value pair into a list.

In [45]:
```
bob1.items()
```

Out [45]:
```
[('gender', 'M'),
 ('first_name', 'Bob'),
 ('last_name', 'Waldron'),
 ('height', 76)]
```

Dictionaries can have their own internal structure as well. It is handled as a dictionary inside of a dictionary.

In [54]:
```
client = {'name': "James Goodyear",
          'address': { 'street': '100 Goodguy lane',
                       'city': 'Great Place',
                       'state': 'PA',
                       'zip': 15512
                       }
          }
```

Dictionary keys can be accesses using a `[]` notation to pull out keys and subkeys.

In [55]:
```
client['address']['street']
```

Out [55]:
```
'100 Goodguy lane'
```