Python Workshop
Built-In Data Structures

C.D. Langevin

U.S. Geological Survey
Reston, Virginia, USA

USGS National Groundwater Workshop, August 2012

Outline

Concepts

- Data Structure – A way to store and organize data in a computer
- Mutability – An *immutable* object is one whose property or state cannot be changed, whereas, *mutable* objects can change state

Object Terminology

Everything is an object in Python

- Object – An instance of a data structure
- Function – Works on an object that is passed into it
  e.g. run(model)
- Method – A set of instructions that works on itself
  e.g. model.run()
- Member – A piece of information, such as an integer, that is part of an object
  e.g. model.nlay

Numbers

- integer (int)
- float/double precision (float)
- long integer (long)
- complex (complex)

```
In [2]: type(2)
Out[2]: <type 'int'>

In [3]: type(2.0)
Out[3]: <type 'float'>

In [4]: type(2**100)
Out[4]: <type 'long'>

In [5]: type(2j)
Out[5]: <type 'complex'>
```

Strings

- type 'str'
- An immutable sequence of characters
- Individual characters are accessed using zero-based indexing

```
In [69]: s = 'modflow'

In [70]: s.upper()
Out[70]: 'MODFLOW'

In [71]: s.capitalize()
Out[71]: 'Modflow'

In [72]: s[0]
Out[72]: 'm'

In [73]: s[-1]
Out[73]: 'w'

In [74]: s[0:4]
Out[74]: 'modf'

In [75]: len(s)
Out[75]: 7

In [76]: 'Modflow' + '-88'
Out[76]: 'Modflow-88'
```

USGS science for a changing world

Lists

- A mutable collection of objects
- List members are accessed using a zero-based indexing scheme
- A list can contain different types

```
In [79]: l = []

In [80]: l.append('first')

In [81]: l.append(2)

In [82]: l.append(3.0)

In [83]: l
Out[83]: ['first', 2, 3.0]

In [84]: l[0]
Out[84]: 'first'

In [85]: l[1]
Out[85]: 2

In [86]: l[2]
Out[86]: 3.0

In [87]: len(l)
Out[87]: 3
```

USGS science for a changing world

List Methods

- append
- count
- extend
- index
- insert
- pop
- remove
- reverse
- sort

```
In [115]: l
Out[115]: ['mf.dis', 'mf.bas', 'mf.pcg', 'mf.lpf']

In [116]: l =
['mf.dis', 'mf.bas', 'mf.lpf', 'mf.pcg']

In [117]: l.index('mf.bas')
Out[117]: 1

In [118]: l.remove('mf.pcg')

In [119]: l.append('mf.sip')

In [120]: l
Out[120]: ['mf.dis', 'mf.bas', 'mf.lpf', 'mf.sip']
```

Tuples

- type 'tuple'
- An immutable collection of objects
- Tuple members are accessed using a zero-based indexing scheme
- A tuple can contain different types

```
In [1]: point1 = (0, 10, 0)

In [2]: point2 = (10, 0, 0)

In [3]: point3 = (0, 0, 0)

In [4]: triangle = (point1, point2, point3)

In [5]: type(triangle)
Out[5]: <type 'tuple'>

In [6]: triangle[0]
Out[6]: (0, 10, 0)

In [7]: triangle[-1]
Out[7]: (0, 0, 0)

In [8]: point1 = (-1, -1, -1)

In [9]: triangle
Out[9]: ((0, 10, 0), (10, 0, 0), (0, 0, 0))
```

USGS science for a changing world

Dictionaries

- type 'dict'
- A mutable collection of keys and values where d[key] = value
- Useful for quickly looking up a value based on a key
- Keys must be immutable
- There is no guarantee that the order of the dictionary will be the same as the creation order

```
In [21]: d = {}

In [22]: d[1] = 'January'

In [23]: d[2] = 'February'

...

In [33]: d[12] = 'December'

In [34]: d
Out[34]:
{1: 'January',
 2: 'February',
 3: 'March',
 4: 'April',
 5: 'May',
 6: 'June',
 7: 'July',
 8: 'August',
 9: 'September',
 10: 'October',
 11: 'November',
 12: 'December'}
```

USGS science for a changing world

Accessing Dictionary Keys

- A value can be looked up in the dictionary by referencing d[key]

```
In [59]: d[1]
Out[59]: 'January'

In [60]: d[12]
Out[60]: 'December'

In [61]: s = 'It is the month of ' + d[8]

In [62]: s
Out[62]: 'It is the month of August'

In [63]: l = d.keys()

In [64]: l
Out[64]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

Key Types

- Keep in mind that all types cannot be keys

```
In [73]: d =
{ (1,1,1):'cell 1', (2,1,1):'cell 2' }

In [74]: s = 'this is cell: ' + d[(2,1,1)]

In [75]: s
Out[75]: 'this is cell: cell 2'

In [76]: d =
{ [1,1,1]:'cell 1', (2,1,1):'cell 2' }
---------------------------------------------------
TypeError
Traceback (most recent call last)

TypeError: unhashable type: 'list'
```

Iterating through a Dictionary

- It is possible to iterate through all the key:value pairs in a dictionary

```
In [89]: for k, v in d.iteritems():
   ....:     print 'key: ', k
   ....:     print 'value: ',v
   ....:
   ....:
key:  (1, 1, 1)
value:  cell 1
key:  (2, 1, 1)
value:  cell 2
```

Others

- Sets: A set is an unordered collection with no duplicates.

```
In [12]: s = set(['one', 'one', 'two', 'three'])

In [13]: s
Out[13]: set(['one', 'three', 'two'])
```

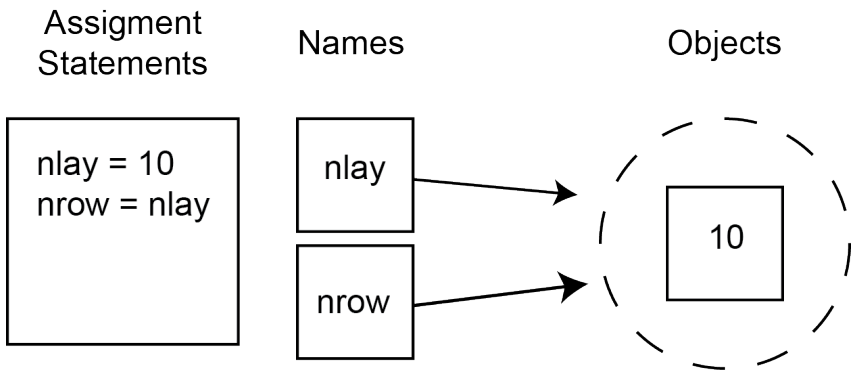
- Boolean: value of True or False (no quotes, and capitalized first letter)

```
In [14]: done = False

In [15]: if not done: print 'still working...'
   ....:
still working...
```

- None (without quotes and capitalized first letter) indicates the absence of a value

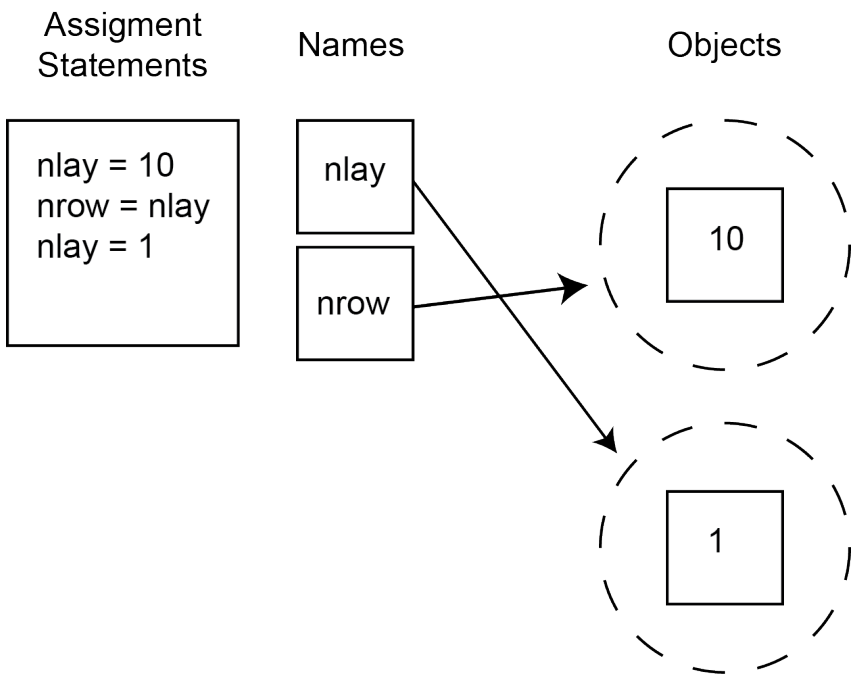
Shared References



- Variable nlay and nrow both point to the same object in memory (an integer value of 10)

See pages 116-121 in Learning Python, Third Edition

Shared References



- The new assigment of nlay creates a new object (an integer value of 1)

Shared References

```
In [30]: model1 = ['mf.bas', 'mf.dis', 'mf.lpf', 'mf.wel', 'mf.pcg']

In [31]: model2 = model1

In [32]: model1
Out[32]: ['mf.bas', 'mf.dis', 'mf.lpf', 'mf.wel', 'mf.pcg']

In [33]: model2
Out[33]: ['mf.bas', 'mf.dis', 'mf.lpf', 'mf.wel', 'mf.pcg']

In [34]: model1 = None

In [35]: model1

In [36]: model2
Out[36]: ['mf.bas', 'mf.dis', 'mf.lpf', 'mf.wel', 'mf.pcg']
```

In-Place Changes

```
In [39]: model1 = ['mf.bas', 'mf.dis', 'mf.lpf', 'mf.wel', 'mf.pcg']

In [40]: model2 = model1

In [41]: model2[3] = 'mf2.wel'

In [42]: model1
Out[42]: ['mf.bas', 'mf.dis', 'mf.lpf', 'mf2.wel', 'mf.pcg']

In [43]: model2
Out[43]: ['mf.bas', 'mf.dis', 'mf.lpf', 'mf2.wel', 'mf.pcg']
```

- Both lists contain 'mf2.wel'
- Here we changed a component of the model2 object and not the list object itself
- This can cause undesired behavior

In-Place Changes

```
In [44]: model1 = ['mf.bas', 'mf.dis', 'mf.lpf', 'mf.wel', 'mf.pcg']

In [45]: model2 = model1[:]

In [46]: model2[3] = 'mf2.wel'

In [47]: model1
Out[47]: ['mf.bas', 'mf.dis', 'mf.lpf', 'mf.wel', 'mf.pcg']

In [48]: model2
Out[48]: ['mf.bas', 'mf.dis', 'mf.lpf', 'mf2.wel', 'mf.pcg']
```

- By using model2 = model1[:] we created a copy of model1

Testing for Equality

- '==' tests to see if the two objects have the same value
- 'is' tests to see if the two variables point to the same object

```
In [50]: a = [0, 1, 2, 3]

In [51]: b = a

In [52]: c = a[:]

In [53]: a == b
Out[53]: True

In [54]: a == c
Out[54]: True

In [55]: b == c
Out[55]: True

In [56]: a is b
Out[56]: True

In [57]: a is c
Out[57]: False

In [58]: b is c
Out[58]: False
```

Zip

- zip provides a way to combine two lists, element by element, into a new list containing a tuple for each pair

```
In [8]: alist = ['a1', 'a2', 'a3', 'a4']

In [9]: blist = ['b1', 'b2', 'b3', 'b4']

In [10]: zip(alist, blist)
Out[10]: [('a1', 'b1'), ('a2', 'b2'), ('a3', 'b3'), ('a4', 'b4')]

In [11]: for a, b in zip(alist, blist):
   ....:     print a, b
   ....:
   ....:
a1 b1
a2 b2
a3 b3
a4 b4
```

Summary