Numbers and lists

Genome 373
Genomic Informatics
Elhanan Borenstein
Numbers

• Python defines various types of numbers:
  - Integer (1234)
  - Floating point number (12.34)
  - Octal and hexadecimal number (0177, 0x9gff)
  - Complex number (3.0+4.1j)

• You will likely only use the first two.
Conversions

>>> 6/2
3

>>> 3.0/4.0
0.75

>>> 3/4.0
0.75

>>> 3*4.0
12.0

>>> 3*4
12

>>> 3/4
0

• The result of a mathematical operation on two numbers of the same type is a number of that type.

• The result of an operation on two numbers of different types is a number of the more complex type.
Formatting numbers

• The `%` operator formats a number.
• The syntax is `<format> % <number>`

```python
>>> print 3 # no formatting
3

>>> print "%f" % 3    # print as float
3.000000

>>> print "%0.2f" % 3  # print as float with
3.00                  # 2 digits after decimal

>>> print "%6.2f" % 3 # width 5 characters
3.00
```
Formatting codes

- `%d` = integer (d as in digit?)
- `%f` = float value (decimal number)
- `%e` = scientific notation
- `%g` = easily readable notation (i.e., use decimal notation unless there are too many zeroes, then switch to scientific notation)
More complex formats

%[flags][width][.precision][code]

- Total width of output
- Number of digits after decimal
- Left justify (“-“)
  - Include numeric sign (“+“)
  - Fill in with zeroes (“0“)

- d, f, e, g
Examples

```python
>>> x = 7718
>>> print "%d" % x
7718
>>> print "%-6d" % x
7718_ _
>>> print "%-06d" % x
007718
>>> x = 1.23456789
>>> print "%f" % x
1.234568
>>> print "%e" % x
1.234568e+00
>>> print "%g" % x
1.23457
>>> print "%g" % (x * 10000000)
1.23457e+07
```

Don't worry if this all looks like Greek - you can figure out how to do these when you need them in your programs. After a while they are pretty easy.
Lists

• A list is an ordered set of objects

>>> myString = "Hillary"
>>> myList = ["Hillary", "Barack", "John"]

• Lists are
  - ordered left to right
  - indexed like strings (from 0)
  - mutable
  - possibly heterogeneous (including containing other lists)

>>> list1 = [0, 1, 2]
>>> list2 = ['A', 'B', 'C']
>>> print list2[1]
B
>>> list3 = ['D', 'E', 3, 4]
>>> list4 = [list1, list2, list3]  # WHAT?
>>> print list4
[[0, 1, 2], ['A', 'B', 'C'], ['D', 'E', 3, 4]]
Lists and dynamic programming

```python
# program to print scores in a DP matrix
dpm = [ [0,-4,-8], [-4,10,6], [-8,6,20] ]
print dpm[0][0], dpm[0][1], dpm[0][2]
print dpm[1][0], dpm[1][1], dpm[1][2]
print dpm[2][0], dpm[2][1], dpm[2][2]
```

> python print_dpm.py
```
0 -4 -8
-4 10 6
-8 6 20
```

this is called a 2-dimensional list
(or a matrix or a 2-dimensional array)
More readable output

```python
# program to print scores in a matrix
dpm = [ [0,-4,-8], [-4,10,6], [-8,6,20] ]
print "%3d" % dpm[0][0], "%3d" % dpm[0][1], "%3d" % dpm[0][2]
print "%3d" % dpm[1][0], "%3d" % dpm[1][1], "%3d" % dpm[1][2]
print "%3d" % dpm[2][0], "%3d" % dpm[2][1], "%3d" % dpm[2][2]
```

> python print_dpm.py

```
0   -4   -8
-4   10   6
-8   6   20
```

print integers with 3 characters each (default is right-justified)
Lists and strings are similar

**Strings**

```python
>>> s = 'A'+'T'+'C'+'G'
```

```python
>>> s = 'ATCG'
```

```python
>>> print s[0]
A
```

```python
>>> print s[-1]
G
```

```python
>>> print s[2:]
CG
```

```python
>>> s * 3
'ATCGATCGATCG'
```

```python
>>> s[9]
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
IndexError: string index out of range
```

(you can think of a string as an **immutable list** of characters)

**Lists**

```python
>>> L = ["adenine", "thymine"] +
["cytosine", "guanine"]
```

```python
>>> L = ["adenine", "thymine",
"cytosine", "guanine"]
```

```python
>>> print L[0]
adenine
```

```python
>>> print L[-1]
guanine
```

```python
>>> print L[2:]
['cytosine', 'guanine']
```

```python
>>> L * 3
['adenine', 'thymine', 'cytosine', 'guanine', 'adenine', 'thymine', 'cytosine', 'guanine', 'adenine', 'thymine', 'cytosine', 'guanine']
```

```python
>>> L[9]
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
IndexError: list index out of range
```
Lists can be changed; strings are immutable.

**Strings**

```python
>>> s = "ATCG"

>>> print s
ATCG

>>> s[1] = "U"
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: object doesn't support item assignment

>>> s.reverse()
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
AttributeError: 'str' object has no attribute 'reverse'
```

**Lists**

```python
>>> L = ['adenine', 'thymine', 'cytosine', 'guanine']

>>> print L
['adenine', 'thymine', 'cytosine', 'guanine']

>>> L[1] = 'uracil'

>>> print L
['adenine', 'uracil', 'cytosine', 'guanine']

>>> L.reverse()

>>> print L
['guanine', 'cytosine', 'uracil', 'adenine']

>>> del L[0]

>>> print L
['cytosine', 'uracil', 'adenine']
```
More list operations and methods

```python
>>> L = ["thymine", "cytosine", "guanine"]

>>> L.insert(0, "adenine")  # insert before position 0
>>> print L
['adenine', 'thymine', 'cytosine', 'guanine']

>>> L.insert(2, "uracil")
>>> print L
['adenine', 'thymine', 'uracil', 'cytosine', 'guanine']

>>> print L[:2]
['adenine', 'thymine']

>>> L[:2] = ['A', 'T']  # replace elements 0 and 1
>>> print L
['A', 'T', 'uracil', 'cytosine', 'guanine']

>>> L[:2] = []  # replace elements 0 and 1 with nothing
>>> print L
['uracil', 'cytosine', 'guanine']

>>> L = ['A', 'T', 'C', 'G']
>>> L.index('C')  # find index of first list element that is the same as 'C'
2

>>> L.remove('C')  # remove first element that is the same as 'C'
>>> print L
['A', 'T', 'G']
```
Methods for expanding lists

```python
globals()['data'] = [] # make an empty list
data
[]
data.append("Hello!") # append means "add to the end"
data
['Hello!']
data.append(5)
data
['Hello!', 5]
data.append([9, 8, 7]) # append a list to end of the list
data
['Hello!', 5, [9, 8, 7]]
data.extend([4, 5, 6]) # extend means append each element
data
['Hello!', 5, [9, 8, 7], 4, 5, 6]
print data[2]
[9, 8, 7]
print data[2][0] # data[2] is a list - access it as such
9
```

notice that this list contains three different types of objects: a string, some numbers, and a list.
Turn a string into a list

`string.split(x)`

```python
>>> protein = "ALA PRO ILE CYS"
>>> residues = protein.split()  # split() uses whitespace
>>> print residues             # the string hasn't changed
['ALA', 'PRO', 'ILE', 'CYS']

>>> print protein
ALA PRO ILE CYS

>>> protein2 = "HIS-GLU-PHE-ASP"
>>> protein2.split("-")      # split at every "-" character
['HIS', 'GLU', 'PHE', 'ASP']
```
**Turn a list into a string**

`join` is the opposite of `split`:

```python
<delimiter> . join(L)
```

```python
>>> L1 = ["Asp", "Gly", "Gln", "Pro", "Val"]

>>> print ".-" . join(L1)
Asp-Gly-Gln-Pro-Val

>>> print ".**" . join(L1)
Asp**Gly**Gln**Pro**Val

>>> L2 = "\n" . join(L1)

>>> print L2
Asp
Gly
Gln
Pro
Val
```

the order is confusing.
- string to join with is first.
- list to be joined is second.
Basic list operations:

\[
L = ['dna', 'rna', 'protein']
\]
# list assignment

\[
L2 = [1, 2, 'dogma', L]
\]
# list hold different objects

\[
L2[2] = 'central'
\]
# change an element (mutable)

\[
L2[0:2] = 'ACGT'
\]
# replace a slice

\[
del L[0:1]
\]
# delete a slice

\[
L2 + L
\]
# concatenate

\[
L2*3
\]
# repeat list

\[
L[x:y]
\]
# define the range of a list

\[
len(L)
\]
# length of list

\[
''.join(L)
\]
# convert a list to string

\[
S.split(x)
\]
# convert string to list - x delimited

\[
list(S)
\]
# convert string to list - explode

\[
list(T)
\]
# converts a tuple to list

Methods:

\[
L.append(x)
\]
# add to the end

\[
L.extend(x)
\]
# append each element from x to list

\[
L.count(x)
\]
# count the occurrences of x

\[
L.index(x)
\]
# give element location of x

\[
L.insert(i, x)
\]
# insert at element x at element i

\[
L.remove(x)
\]
# delete first occurrence of x

\[
L.pop(i)
\]
# extract element I

\[
L.reverse()
\]
# reverse list in place

\[
L.sort()
\]
# sort list in place
Reminder - linked from the course web site is a Python cheat sheet that contains most of the basic information we are covering in a short reference format.
Tuples: immutable lists

Tuples are immutable. Why? Sometimes you want to guarantee that a list won’t change.

Tuples support operations but not methods.

```python
>>> T = (1, 2, 3, 4)
>>> T*4
(1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4)
>>> T + T
(1, 2, 3, 4, 1, 2, 3, 4)
>>> T
(1, 2, 3, 4)
>>> T[1] = 4
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: object doesn't support item assignment
>>> x = (T[0], 5, "eight")
>>> print x
(1, 5, 'eight')
>>> print x
(1, 5, 'eight')
>>> y = list(x)  # converts a tuple to a list
>>> print y.reverse()
('eight', '5', '1')
>>> z = tuple(y)  # converts a list to a tuple
```