Numbers, lists and tuples

Genome 559: Introduction to Statistical and Computational Genomics

Prof. James H. Thomas

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
```

integer → float

- 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.

watch out - integer divisions are truncated rather than rounded

Formatting numbers

- The % operator formats a number.
- The syntax is `<format> % <number>` where format is a string

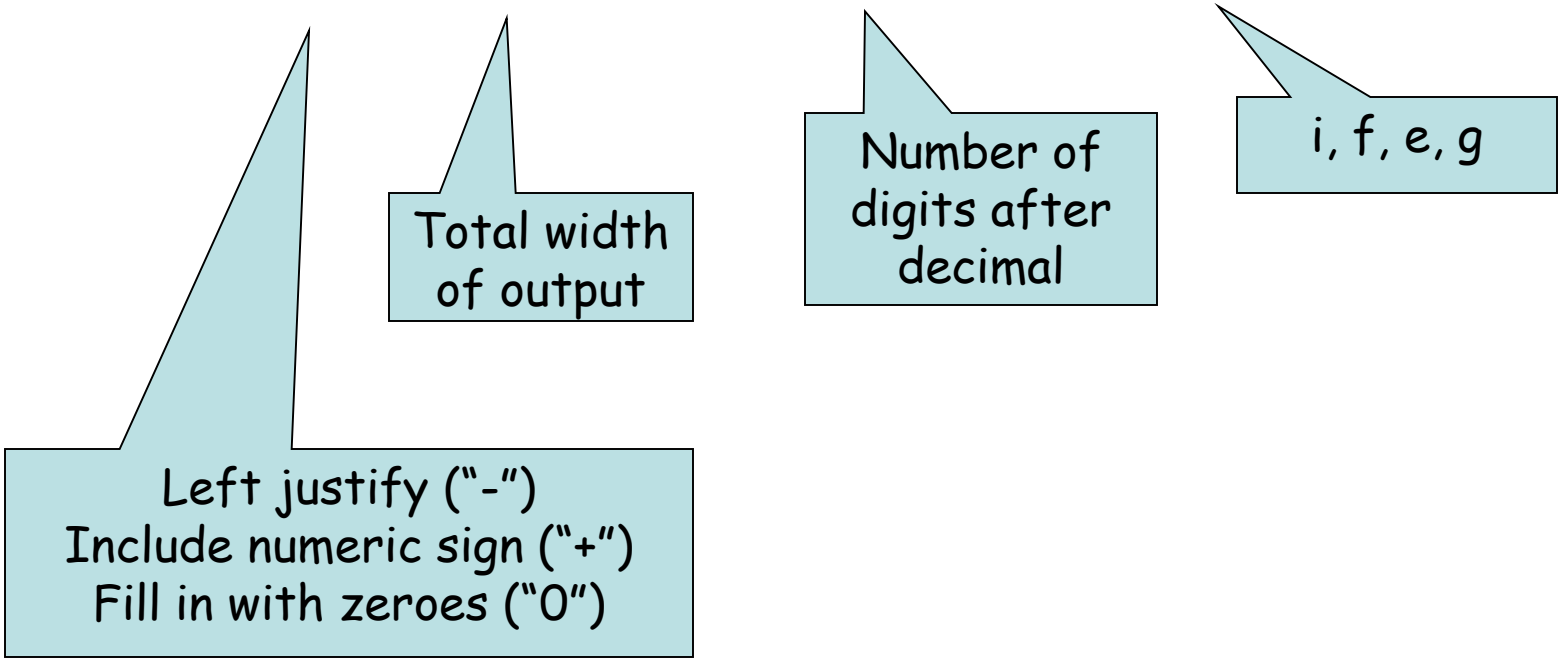
```
>>> "%f" % 3   # print as float
'3.000000'
>>> "%.2f" % 3 # print as float with
'3.00'         # 2 digits after decimal
>>> "%5.2f" % 3 # width 5 characters
' 3.00'
```

Formatting codes

- `%i` = integer (or %d)
- `%f` = float value (decimal number)
- `%e` = scientific notation
- `%g` = general, easily readable notation (uses decimal notation unless there are too many zeroes, then switches to scientific notation)

More complex formats

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



Examples (review later)

```
>>> x = 7718
>>> "%i" % x
'7718'
>>> "%-6i" % x
'7718  '
>>> "%06i" % x
'007718'
>>> x = 1.23456789
>>> "%i" % x
'1'
>>> "%f" % x
'1.234568'
>>> "%e" % x
'1.234568e+00'
>>> "%g" % x
'1.23457'
>>> "%g" % (x * 10000000)
'1.23457e+07'
```

Read as "use the preceding code to format the following number"

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.

It sure looks like Greek to me.

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)

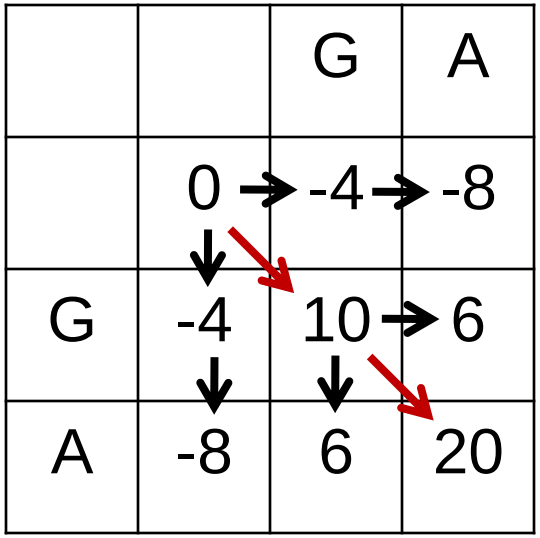
```
>>> list1 = [0, 1, 2]
>>> list2 = ['A', 'B', 'C']
>>> list3 = ['D', 'E', 3, 4]
>>> list4 = [list1, list2, list3]  # WHAT?
>>> list4
[[0, 1, 2], ['A', 'B', 'C'], ['D', 'E', 3, 4]]
```

Lists and dynamic programming

```
# 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 (review later)

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

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

concatenate

```
>>> s
'ATCG'

>>> print s[0]
A
>>> print s[-1]
G
>>> print s[2:]
CG
>>> s * 3
'ATCGATCGATCG'
```

index

slice

multiply

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

```
>>> L = ["adenine", "thymine"] +
["cytosine", "guanine"]
>>> L
['adenine', 'thymine', 'cytosine',
'guanine']
>>> print L[0]
adenine
>>> print L[-1]
guanine
>>> print L[2:]
['cytosine', 'guanine']
>>> L * 3
['adenine', 'thymine', 'cytosine',
'guanine', 'adenine', 'thymine',
'cytosine', 'guanine', 'adenine',
'thymine', 'cytosine', 'guanine']
>>> L[9]
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
IndexError: list index out of range
```

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

Lists can be changed; strings cannot.

Strings

```
>>> 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

```
>>> 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']
```

reassign element value

reverse order

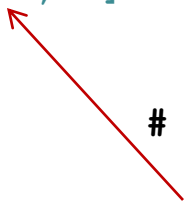
delete element

More list operations and methods

```
>>> L = ["thymine", "cytosine", "guanine"]
>>> L.insert(0, "adenine")                          # insert before position 0
>>> print L
['adenine', 'thymine', 'cytosine', 'guanine']
>>> L.insert(2, "uracil")                           # insert before position 2
>>> print L
['adenine', 'thymine', 'uracil', 'cytosine', 'guanine']
>>> print L[:2]                                     # slice
['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 element that is 'C'
2
>>> L.remove('C')         # remove first element that is 'C'
>>> print L
['A', 'T', 'G']
```

Methods for expanding lists

```
>>> data = []                          # make an empty list
>>> print data
[]
>>> data.append("Hello!")              # append means "add to the end"
>>> print data
['Hello!']
>>> data.append(5)
>>> print data
['Hello!', 5]
>>> data.append([9, 8, 7])             # append a list to end of the list
>>> print data
['Hello!', 5, [9, 8, 7]]
>>> data.extend([4, 5, 6])             # extend means append each element
>>> print 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

`str.split()` or `list(str)`

```
>>> protein = "ALA PRO ILE CYS"
>>> residues = protein.split()      # split() uses whitespace
>>> print residues
['ALA', 'PRO', 'ILE', 'CYS']
>>> list(protein)                   # list() explodes each char
['A', 'L', 'A', ' ', 'P', 'R', 'O', ' ', 'I', 'L',
  'E', ' ', 'C', 'Y', 'S']
>>> print protein.split()           # the list hasn't changed
['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`:

`<delimiter>.join(L)`

```
>>> L1 = ["Asp", "Gly", "Gln", "Pro", "Val"]
>>> print "-".join(L1)
Asp-Gly-Gln-Pro-Val
>>> print "".join(L1)
AspGlyGlnProVal
>>> L2 = "\n".join(L1)
>>> L2
'Asp\nGly\nGln\nPro\nVal'
>>> print L2
Asp
Gly
Gln
Pro
Val
```

the order might be confusing.
- string to join with is first.
- list to be joined is second.

Tuples: immutable lists

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

Tuples support operations but not methods.

```
>>> 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')
>>> y = list(x)                    # converts a tuple to a list
>>> print y.reverse()
('eight', '5', '1')
>>> z = tuple(y)                   # converts a list to a tuple
```

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] = 'nucs'              # delete a slice
L2 + L                           # concatenate
L2*3                             # repeat list
L[x:y]                           # take a slice from 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 – by single character
list(T)                          # convert a tuple to list
```

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 index i
L.remove(x)                      # delete first occurrence of x
L.pop(i)                         # extract element i
L.reverse()                      # reverse list in place – returns None
L.sort()                         # sort list in place – returns None
```

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.

Sample problem #1

- Write a program called dna-composition.py that takes a DNA sequence as the first command line argument and prints the number of *A*'s, *C*'s, *G*'s and T's.

```
> python dna-composition.py ACGTGCGTTAC
2 A's
3 C's
3 G's
3 T's
```

Solution #1

```
import sys
sequence = sys.argv[1].upper()
print sequence.count('A'), "A's"
print sequence.count('C'), "C's"
print sequence.count('G'), "G's"
print sequence.count('T'), "T's"
```

Note - this uses the trick that you can embed single quotes inside a double-quoted string (or vice versa) without using an escape code.

Sample problem #2

- The object `sys.argv` is a list of strings.
- Write a program reverse-args.py that removes the program name from the beginning of this list and then prints the remaining command line arguments (no matter how many of them are given) in reverse order with asterisks in between.

```
> python reverse-args.py 1 2 3
3*2*1
> python reverse-args.py A B C D E
E*D*C*B*A
```

Solution #2

```
import sys
args = sys.argv[1:]
args.reverse()
print "*".join(args)
```

Sample problem #3

- The melting temperature (*C*) of a primer sequence (with its exact reverse complement) can be estimated as:

  T = 2 * (# of A or T nucleotides) + 4 * (# of G or C nucleotides)

- Write a program melting-temperature.py that computes the melting temperature of a DNA sequence given as the first argument.

```
> python melting-temperature.py ACGGTCA
22 degrees C
```

Solution #3

```
import sys
sequence = sys.argv[1].upper()
numAs = sequence.count('A')
numCs = sequence.count('C')
numGs = sequence.count('G')
numTs = sequence.count('T')
temp = (2 * (numAs + numTs)) + (4 * (numGs + numCs))
print temp, 'degrees C'
```

Challenge problem

Download the file "sonnet.txt" from the course web site. Read the entire file contents into a string, divide it into a list of words, sort the list of words, and print the list. Make the words all lower case so that they sort more sensibly (by default all upper case letters come before all lower case letters).

Tips:

To read the file as a single string use:

```
sonnet_text = open("sonnet.txt").read()
```

To sort a list of strings use:

```
string_list.sort()
```

Challenge problem solution

```
sonnet_text = open("sonnet.txt").read()
# next line optional, just gets rid of common punctuation
sonnet_text = sonnet_text.replace(",","").replace(".","")
sonnet_text = sonnet_text.lower()
wordList = sonnet_text.split()
wordList.sort()
print wordList
```

Reading

- Chapters 10 and 12 of *Think Python* by Downey.