Genome 373: Intro to Python II

Doug Fowler
Review

- string objects represent a sequence of characters
- characters in strings can be gotten by index, e.g. `myStr[3]`
- substrings can be extracted by slicing, e.g. `myStr[3:7]`
- string objects have specific methods, e.g. `myStr.find("foo")`
- numbers are either type int (2 or 7) or float (2.7 or 3.1415)
- math operations on numbers retain their type, e.g. `int/int -> int`
- operations on mixed types give floats, e.g. `int*float -> float`
s = "python"  # creates a string object
i = 1  # creates an int object
f = 2.1  # creates a float object

object.method(arguments)
Outline – Intro to Python II

• Lists

• Reading and writing files

• Making decisions in a program

• Additional functionality, and some practical advice
To follow along/try stuff out

• Login to the computer (just click)

• Open the Terminal either from the dock or from Applications -> Utilities

• Type “python” to launch the interactive interpreter

• If you have trouble, raise your hand and Matthew will come and help

• If you want get this slide deck to follow along, go to http://goo.gl/wMoRd1
Lists

- A **list** is an object that represents an ordered set of objects

```python
>>> myString = "Pirates"
>>> myList = ["Pirates", "Ninjas", "Jedis"]
```

- Lists are
  - ordered left to right
  - indexed like strings (from 0)
  - Mutable
  - can be heterogeneous (including containing other lists)

```python
>>> 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 strings are similar

```python
>>> s = 'A'+'T'+'C'+'G'
>>> s
'ATCG'
>>> print s[0]
A
>>> print s[-1]
G
>>> print s[2:]
CG
>>> s * 3
'ATCGATCGATCG'

>>> s
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 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
```

### Lists

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

>>> L[1] = "uracil"
>>> print L
['adenine', 'uracil', 'cytosine', 'guanine']

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

```python
>>> L.reverse()
>>> print L
['guanine', 'cytosine', 'uracil', 'adenine']

>>> del L[0]
>>> print L
['cytosine', 'uracil', 'adenine']
```
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']
```

Note that successive list methods actually change the list itself (unlike string methods that returned a new string but didn’t change the original one!)
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]
# slice the list
['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 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
>>> 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]
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)

```python
>>> protein = "ALA PRO ILE CYS"
>>> residues = protein.split()  # split() uses whitespace
>>> print residues
["ALA", "PRO", "ILE", "CYS"]
>>> list(protein)              # list() explodes each char
>>> 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)
```

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

the order might be 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] = 'nucs' # 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

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
Opening files

- The built-in `open()` function returns a `file` object:
  ```python
  <file_object> = open(<filename>, <access type>)
  ```

- Python will read, write or append to a file according to the access type requested:
  - `'r'` = read
  - `'w'` = write (will replace the file if it exists)
  - `'a'` = append (appends to an existing file)
  - `‘U’` = universal new line mode (useful for files from Mac/Windows)

- For example, open for reading a file called "hello.txt":
  ```python
  >>> myFile = open('hello.txt', 'r')
  ```
Reading the whole file

• You can read the entire content of the file into a single string. If the file content is the text “Hello, world!\n”:

```python
>>> myString = myFile.read()
>>> print myString
Hello, world!
```

why is there a blank line here?
Reading the whole file

• Now add a second line to the file ("How ya doin’?\n") and try again.

```python
>>> myFile = open('hello.txt', 'r')
>>> myString = myFile.read()
>>> print myString
Hello, world!
How ya doin’?

>>> 
```
Reading the whole file

• Alternatively, you can read the file into a list of strings, one string for each line:

```python
>>> myFile = open('hello.txt', 'r')
>>> myStringList = myFile.readlines()
>>> print myStringList
['Hello, world!
', 'How ya doin'
']
>>> print myStringList[1]
How ya doin'?
```

notice that each line has the newline character at the end

this file method returns a list of strings, one for each line in the file
Reading one line at a time

• The `readlines()` method puts all the lines into a list of strings.
• The `readline()` method returns only the next line:

```python
>>> myFile = open('hello.txt', 'r')
>>> myString = myFile.readline()
>>> print myString
Hello, world!

>>> myString = myFile.readline()
>>> print myString
How ya doin'?

>>> print myString.strip() # strip the newline off
How ya doin'?  
```

notice that `readline()` automatically keeps track of where you are in the file - it reads the next line after the one previously read.
Writing to a file

- Open a file for writing (or appending):
  ```
  >>> myFile = open('new.txt', 'w') # (or 'a')
  ```

- Use the `<file>.write()` method:
  ```
  >>> myFile.write('This is a new file
')
  >>> myFile.close()
  >>> Ctl-D (exit the python interpreter)
  > cat new.txt
  This is a new file
  ```

always close a file after you are finished reading from or writing to it.

open('new.txt', 'w') will overwrite an existing file (or create a new one)
open('new.txt', 'a') will append to an existing file
`<file>.write()` is a little different from `print`

- `<file>.write()` does not automatically append a new-line character.
- `<file>.write()` requires a string as input.

```python
>>> newFile.write('foo')
>>> newFile.write(1)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: argument 1 must be string or read-only character buffer, not int
>>> newFile.write(str(1))  # str converts to string
```

(also of course `print` goes to the screen and `<file>.write()` goes to a file)
Making decisions in a program

• Often we need to take differing courses of action depending on our input data
• This is known as flow control
Making decisions in a program

• Often we need to take differing courses of action depending on our input data
• This is known as flow control
• Let’s say we want to do something only if two strings are equal

```python
>>> codon1 = "ATG"
>>> codon1 = "TGA"
```
Making decisions in a program

- Often we need to take differing courses of action depending on our input data
- This is known as flow control
- The simplest flow control is an if statement:

```python
>>> codon1 = "ATG"
>>> codon1 = "TGA"
>>> if codon1 == codon2:
...    print "Codons are identical"
```
Making decisions in a program

• Often we need to take differing courses of action depending on our input data
• This is known as flow control
• The simplest flow control is an if statement:

```python
>>> codon1 = "ATG"
>>> codon2 = "TGA"
>>> if codon1 == codon2:
...    print "Codons are identical"
```

== is an operator meaning “is equal to”

Returns a Boolean (True/False), and the instructions inside the if block are executed only if the statement is true.
Making decisions in a program

- Often we need to take differing courses of action depending on our input data
- This is known as flow control
- The simplest flow control is an if statement:

```python
>>> codon1 = "ATG"
>>> codon1 = "TGA"
>>> if codon1 == codon2:
    ...
    print "Codons are identical"
```

== is an operator meaning “is equal to”

Other operators that are useful here are <, >, <=, => and logical statements like “x” in “xylophone” or “j” not in “jedi”
Making decisions in a program

• Often we need to take differing courses of action depending on our input data
• This is known as flow control
• The simplest flow control is an if statement:

```python
>>> codon1 = "ATG"
>>> codon1 = "TGA"
>>> if codon1 == codon2:
...   print "Codons are identical"
```

Note that the if block is defined by the addition of a tab. More non-tabbed-in lines of code below this if block would be executed regardless of the outcome of the if block.
Making decisions in a program

- Often we need to take differing courses of action depending on our input data
- This is known as flow control
- `if/else` statements give you more options:

```python
>>> codon1 = "ATG"
>>> codon1 = "TGA"
>>> if codon1 == codon2:
...     print "Codons are identical"
>>> else:
...     print "Codons are different"
```
Making decisions in a program

• Often we need to take differing courses of action depending on our input data
• This is known as flow control

• for statements enable iteration over strings, lists, etc:

```python
>>> for i in [1,2,3,4,5]:
...   print i
```
Making decisions in a program

• Often we need to take differing courses of action depending on our input data
• This is known as flow control

• for statements enable iteration over strings, lists, etc:

```python
>>> for i in [1,2,3,4,5]:
...    print i
1
2
3
4
5
```
Making decisions in a program

- Often we need to take differing courses of action depending on our input data
- This is known as flow control

- `for` statements enable iteration over strings, lists, etc:

```python
>>> for i in [1,2,3,4,5]:
    ...    print i
```

This is an example of **iteration** a key concept. Iteration enables you to, for example, traverse an entire string (for `x in “foo”`) executing the same set of instructions for each character in the string.
Additional functionality

- Python comes with a bunch of functions, some of which we have already talked about.
- Additional functionality is can be accessed using `import`.
- Before you reinvent the wheel, check to see if a prebuilt option exists!

- [https://docs.python.org/2.7/](https://docs.python.org/2.7/)
- Google, stackoverflow, etc
- Additional Python links from the course website
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.

```bash
> python dna-composition.py ACGTGCCTTAC
2 A’s
3 C’s
3 G’s
3 T’s
```
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 prints the remaining command line arguments (no matter how many of them are given) in reverse order with asterisks in between.

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

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

\[ T = 2 \times (\text{# of A or T nucleotides}) + 4 \times (\text{# of G or C nucleotides}) \]

• 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
Sample problem #4

• Write a program `read-first-line.py` that takes a file name from the command line, opens the file, reads the first line, and prints the line to the screen.

  > python read-first-line.py hello.txt
  Hello, world!

  >
Sample problem #5

• Modify the previous program to print the first line without an extra new line.

> python read-first-line.py hello.txt
Hello, world!
>
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:
```python
sonnet_text = open("sonnet.txt").read()
```

To sort a list of strings use:
```python
string_list.sort()
```