Modules, Sorting, Functions as Arguments

Genome 559: Introduction to Statistical and Computational Genomics

Elhanan Borenstein
A quick review

- **Functions:**
  - Reusable pieces of code (write once, use many)
  - Take arguments, “do stuff”, and (usually) return a value
  - Use to organize & clarify your code, reduce code duplication

- **Defining a function:**
  ```python
def <function_name>(<arguments>):
    <function code block>
    <usually return something>
  ```

- **Using (calling) a function:**
  ```python
<function defined here>
<my_variable> = function_name(<my_arguments>)
  ```
A quick review

- Returning multiple values from a function
  ```python
  return [sum, prod]
  ```

- Pass-by-reference vs. pass-by-value
  - Python passes arguments by reference
  - Can be used (carefully) to edit arguments “in-place”

- Default Arguments
  ```python
  def printMulti(text, n=3):
  ```

- Keyword Arguments
  ```python
  runBlast("my_fasta.txt", matrix="PAM40")
  ```
Modules
Recall your makeDict function:

```python
def makeDict(fileName):
    myFile = open(fileName, "r")
    myDict = {}
    for line in myFile:
        fields = line.strip().split("\t")
        myDict[fields[0]] = float(fields[1])
    myFile.close()
    return myDict
```

This is in fact a very useful function which you may want to use in many programs!

So are other functions you wrote (e.g., makeMatrix)
Modules

- A module is a file that contains a collection of related functions.
- You have already used several built-in modules:
  - e.g.: sys, math
- Python has numerous standard modules
  - Python Standard Library: [http://docs.python.org/library/](http://docs.python.org/library/)

- It is easy to create and use your own modules:
  - JUST PUT YOUR FUNCTIONS IN A SEPARATE FILE!
Importing Modules

- To use a module, you first have to import it into your namespace
- To import the entire module:
  ```python
  import module_name
  ```

```python
my_prog.py

import utils
import sys

Dict1 = utils.makeDict(sys.argv[1])
Dict2 = utils.makeDict(sys.argv[2])

Mtrx = utils.makeMatrix("blsm.txt")
...
```

```python
utils.py

# This function makes a dictionary
def makeDict(fileName):
    myFile = open(fileName, "r")
    myDict = {}
    for line in myFile:
        fields = line.strip().split("\t")
        myDict[fields[0]] = float(fields[1])
    myFile.close()
    return myDict

# This function reads a 2D matrix
def makeMatrix(fileName):
    < ... >
```
The dot notation

- Why did we use `utils.makeDict()` instead of just `makeDict()`?

- Dot notation allows the Python interpreter to organize and divide the namespace.
Sorting
Sorting

- Typically applied to lists of things
- Input order of things can be anything
- Output order is determined by the type of sort

```python
>>> myList = ['Curly', 'Moe', 'Larry']
>>> print myList
['Curly', 'Moe', 'Larry']
>>> myList.sort()
>>> print myList
['Curly', 'Larry', 'Moe']
```

(by default this is a lexicographical sort because the elements in the list are strings)
Sorting defaults

- **String sorts** - ascending order, with all capital letters before all small letters:

  ```python
  myList = ['a', 'A', 'c', 'C', 'b', 'B']
  myList.sort()
  print myList
  ['A', 'B', 'C', 'a', 'b', 'c']
  ```

- **Number sorts** - ascending order:

  ```python
  myList = [3.2, 1.2, 7.1, -12.3]
  myList.sort()
  print myList
  [-12.3, 1.2, 3.2, 7.1]
  ```
Code like a pro ...

- When you’re using a function that you did not write, try to guess what’s under the hood!
  *(hint: no magics or divine forces are involved)*

  - How does split() work?
  - How does readlines() work?
  - *How does sort() work?*
Sorting algorithms
Sorting algorithms

- A sorting algorithm takes a list of elements in an arbitrary order, and sort these elements in an ascending order.

- Commonly used algorithms:
  - Naïve sorting (a.k.a. selection sort)
    Find the smallest element and move it to the beginning of the list
  - Bubble sort
    Swap two adjacent elements whenever they are not in the right order
  - Merge sort
    ???
But ...

What if we want a different sort order?

What if we want to sort something else?
But ...

What if we want a different sort order?

What if we want to sort something else?
But ...

What if we want a different sort order?

What if we want to sort something else?
But ...

What if we want a different sort order?
What if we want to sort something else?

The sort() function allows us to define how comparisons are performed! We just write a comparison function and provide it as an argument to the sort function:

```
myList.sort(myComparisonFunction)
```

(The sorting algorithm is done for us. All we need to provide is a comparison rule in the form of a function!)
def myComparison(a, b):
    if a > b:
        return -1
    elif a < b:
        return 1
    else:
        return 0

assuming a and b are numbers, what kind of sort would this give?
Using the comparison function

def myComparison(a, b):
    if a > b:
        return -1
    elif a < b:
        return 1
    else:
        return 0

myList = [3.2, 1.2, 7.1, -12.3]
myList.sort(myComparison)
print myList

[7.1, 3.2, 1.2, -12.3]
You can write a comparison function to sort anything in any way you want!!

```python
>>> print myListOfLists
[[1, 2, 4, 3], ['a', 'b'], [17, 2, 21], [0.5]]

>>> myListOfLists.sort(myLOLComparison)

>>> print myListOfLists
[[1, 2, 4, 3], [17, 2, 21], ['a', 'b'], [0.5]]
```

What kind of comparison function is this?
You can write a comparison function to sort anything in any way you want!!

```python
def myLOLComparison(a, b):
    if len(a) > len(b):
        return -1
    elif len(a) < len(b):
        return 1
    else:
        return 0
```

It specifies a descending sort based on the **length** of the elements in the list:
Sample problem #1

- Write a function that compares two strings ignoring upper/lower case

- Remember, your comparison function should:
  - Return -1 if the first string should come earlier
  - Return 1 if the first string should come later
  - Return 0 if they are tied

(e.g. comparing "JIM" and "jIm" should return 0, comparing "Jim" and "elhanan" should return 1)

- Use your function to compare the above 2 examples and make sure you get the right return value
def caselessCompare(a, b):
    a = a.lower()
    b = b.lower()
    if a < b:
        return -1
    elif a > b:
        return 1
    else:
        return 0

alternatively convert to uppercase
Sample problem #2

- Write a program that:
  - Reads the contents of a file
  - Separates the contents into words
  - Sorts the words using the default sort function
  - Prints the sorted words
- Try it out on the file “crispian.txt”, linked from the course web site.

- Now, sorts the words using YOUR comparison function

(Remember: For now, your function will have to be defined within your program and before you use it. Next week you'll learn how to save a function in a separate file (module) and load it whenever you need it without having to include it in your program.)
def caselessCompare(a, b):
    a = a.lower()
    b = b.lower()
    if a < b:
        return -1
    elif a > b:
        return 1
    else:
        return 0

import sys
filename = sys.argv[1]
file = open(filename, "r")
filestring = file.read()  # whole file into one string
file.close()

wordlist = filestring.split()  # split into words
wordlist.sort(caselessCompare)  # sort

for word in wordlist:
    print word
Challenge problems

1. Modify the previous program so that each word is printed only once (hint - don't try to modify the word list in place).

2. Modify your comparison function so that it sorts on the length of words, rather than on their alphabetical order.

3. Modify the way that you split into words to account for the punctuation marks ,.' (I removed most of them from the text to keep things simple)
import sys
filename = sys.argv[1]
file = open(filename,"r")
filestring = file.read()
file.close()

wordlist = filestring.split()
wordlist.sort(caselessCompare)

print wordlist[0]
for index in range(1,len(wordlist)):
    # if it's a new word, print it
    if wordlist[index].lower() != wordlist[index-1].lower():
        print wordlist[index]
import sys
filename = sys.argv[1]
file = open(filename,"r")
filestring = file.read()
file.close()

wordlist = filestring.split()

tempDict = {}
for word in wordlist:
    tempDict[word] = "foo"
uniquewords = tempDict.keys()
uniquewords.sort(caselessCompare)
for word in uniquewords:
    print word

(it would be slightly better to have the values in your dictionary be an empty string or None in order to save memory; recall that None is Pythonese for null or nothing)
def lengthCompare(a, b):
    lenA = len(a)
    lenB = len(b)
    if lenA < lenB:
        return -1
    elif lenA > lenB:
        return 1
    else:
        return 0

or

def lengthCompare(a, b):
    if len(a) < len(b):
        return -1
    elif len(a) > len(b):
        return 1
    else:
        return 0

it may be slightly faster to do these length calculations once
filestring = filestring.replace("\", "").replace(",", "").replace(".", "")
wordlist = filestring.split()
etc.
Comments on sorting in Python (FYI)

- The sorting algorithm used in Python is called "merge sort".
- It is a recursive divide-and-conquer algorithm.
- It is among the fastest known sorting algorithms and it is "stable", which means that elements with the same value (i.e., two elements for which your comparison function returns 0) stay in their original order in the output.
- Being stable is extremely useful when multiple sorts are performed in series:

```
A  7
B  5
B  1
A  3
B  2
A  1

sort on field 2
```

```
B  1
A  1
B  2
A  3
B  5
A  7

sort on field 1
```

```
A  1
A  3
A  7
B  1
B  2
B  5
```