

sorting

Genome 559: Introduction to Statistical
and Computational Genomics

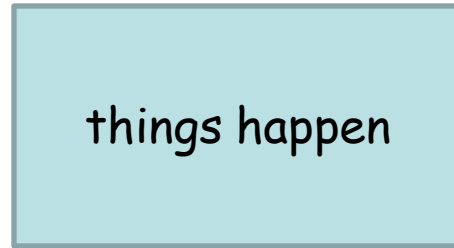
Prof. James H. Thomas

Review

- Functions - take arguments and (usually) return a value.
- Use to organize and clarify your code, reduce code duplication.

What a function does

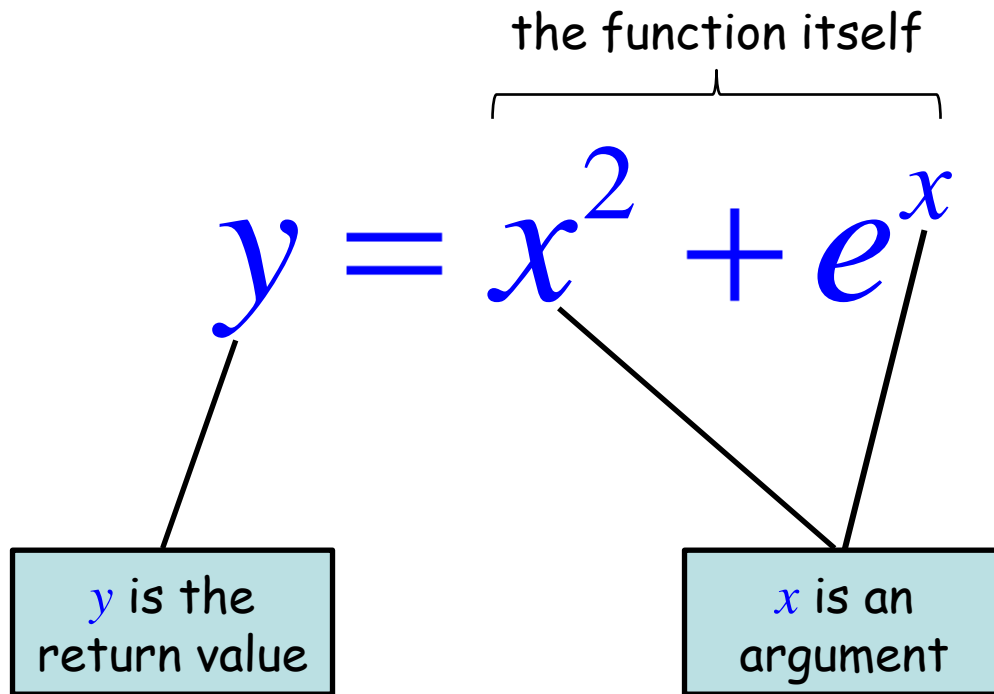
stuff goes in (arguments)



other stuff comes out (return)

Other than the arguments and the return, everything else inside the function is invisible outside the function (variables assigned, etc.).

A close analogy is the mathematical function



Functions can modify argument contents

- The function doesn't need to have a return (it just runs "off the end" of the code block).
- It might instead do something to one of the arguments.
- For example: if the argument is a list, the function could sort the list.

```
def listSort(argList):  
    argList.sort()
```

```
myList = [3,2,1]  
listSort(myList)  
print myList  
[1,2,3]
```

end of
function

the list was sorted
by the function

Sorting

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

```
>>> 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 things in the list are strings)

Sorting defaults

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

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

Number sorts - ascending order.

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

(FYI - under the hood, Python uses an algorithm called mergesort, which is fast, memory efficient, and stable. Stable means that the order of two equal elements in the source remain in the same order in the sorted output, which is very handy under some circumstances.)

What if we want to sort something else?

What if we want a different sort order?

We write a comparison function and provide it to the sort function:

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

The sorting algorithm is done for us - all we need to provide is a simple comparison rule in the form of a function.

Comparison function

- Always takes 2 arguments
- Returns -1 if first argument should appear earlier in sort
- Returns 1 if first argument should appear later in sort
- Returns 0 if they are tied

```
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?

```
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]
```

descending numeric sort

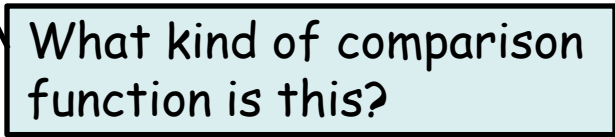
You can write a simple comparison function to sort ANYTHING in any way you want.

myListOfLists



```
>>> len(myLOL)
3
>>> print len(myLOL[0]), len(myLOL[1]), len(myLOL[2])
1 97 28
>>> myLOL.sort(myLOLComparison)
>>> print len(myLOL[0]), len(myLOL[1]), len(myLOL[2])
97 28 1
```

What kind of comparison function is this?



It specifies a descending sort based on the length of the elements:

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

Sample problem #1

- Write a function that compares two strings ignoring upper/lower case
- 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 "larry" should return -1

Solution #1

```
def caselessCompare(a, b):  
    a = a.lower()  
    b = b.lower() } or convert to uppercase  
    if a < b:  
        return -1  
    elif a > b:  
        return 1  
    else:  
        return 0
```

Save your function "caselessCompare" in a file called "util.py" (in your current working directory).

You can now import this file as a module:

```
import util    (note the absence of ".py")
```

... and use any functions in it as usual for modules, e.g.:

```
myList.sort(util.caselessCompare)
```

Sample problem #2

Write a program that:

- Reads the contents of a file
- Separates the contents into words
- Sorts the words using *YOUR* comparison function as saved and imported from your module
- Prints the sorted words

Try it out on the file "sonnet.txt", linked from the course web site.

Solution #2

```
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
import util # import my new module
wordlist.sort(util.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 their alphabetical order.
- 3) Modify the way that you split into words to account for the punctuation marks , . ' (I removed them from the sonnet to keep things simple)

Challenge solution 1

```
import sys
filename = sys.argv[1]
file = open(filename,"r")
filestring = file.read()
file.close()
wordlist = filestring.split()

import util
wordlist.sort(util.caselessCompare)

print wordlist[0]
for index in range(1,len(wordlist)):
    # if it's a new word, print it
    if wordlist[index] != wordlist[index-1]:
        print wordlist[index]
```

Alternative challenge solution 1

```
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() \

import util
uniquewords.sort(util.caselessCompare)
for word in uniquewords:
    print word
```

uses the fact that each key can appear only once (it doesn't matter what the value is - they aren't used)

(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)

Challenge solution 2

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

it may be slightly faster
to do these length
calculations once

or

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

Challenge solution 3

```
filestring = filestring.replace("'", "").replace(", ", "").replace(".", "")  
wordlist = filestring.split()  
etc.
```