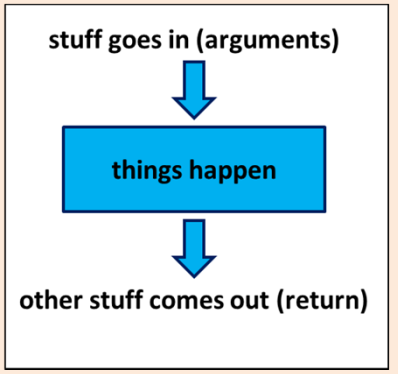
Functions as Arguments, Sorting

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:

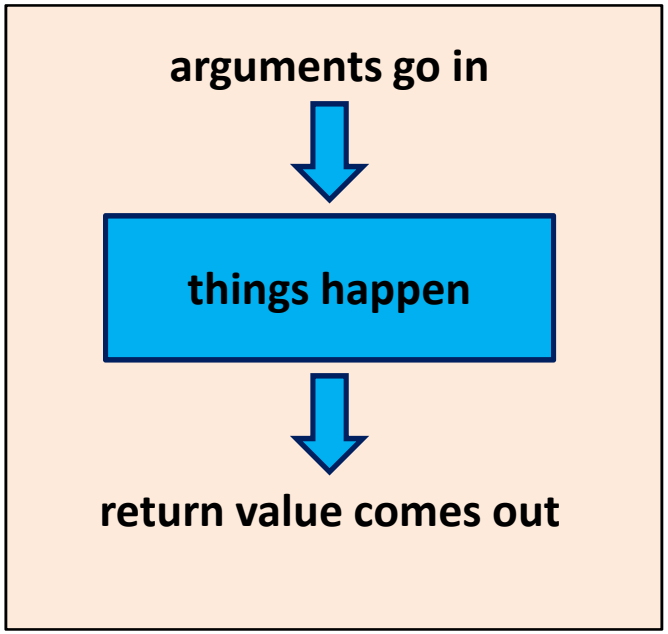
```
def <function_name>(<arguments>):
      <function code block>
      <usually return something>
```

- Using (calling) a function:

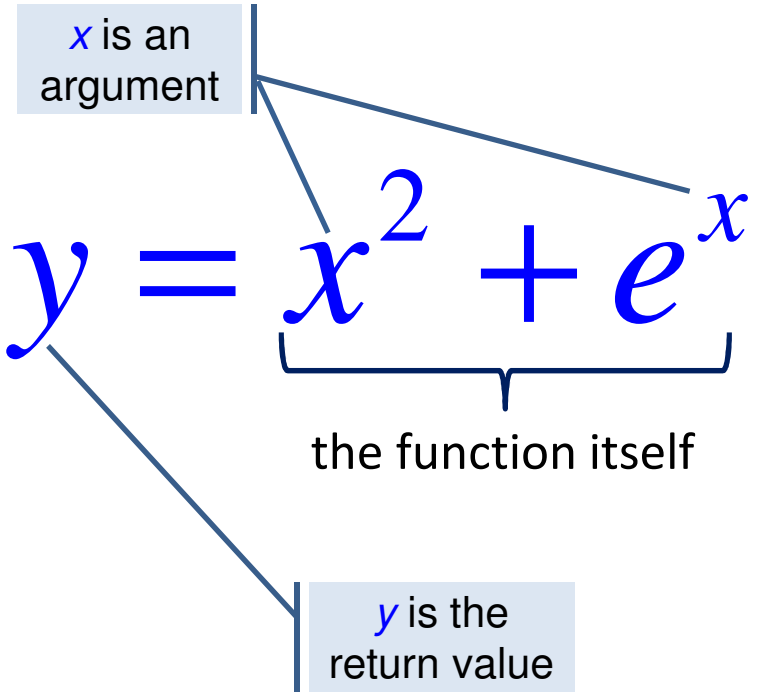
```
<function defined here>

<my_variable> = function_name(<my_arguments>)
```

A close analogy is the mathematical function

A Python Function

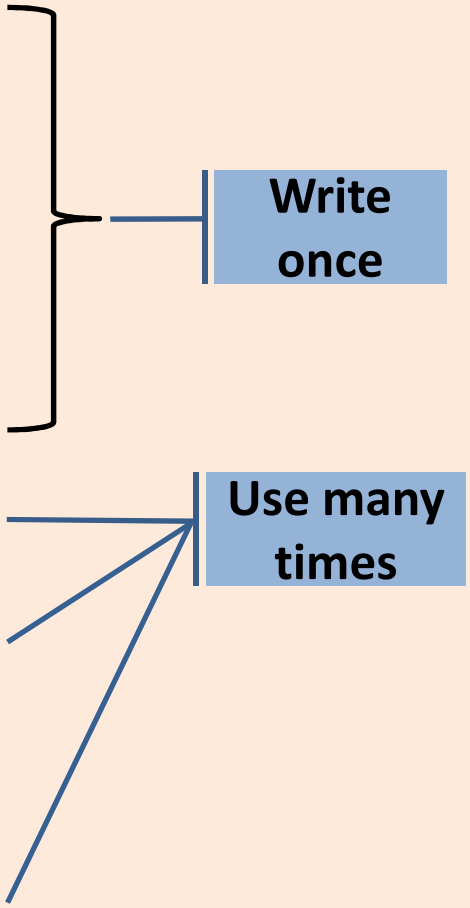


A mathematical Function



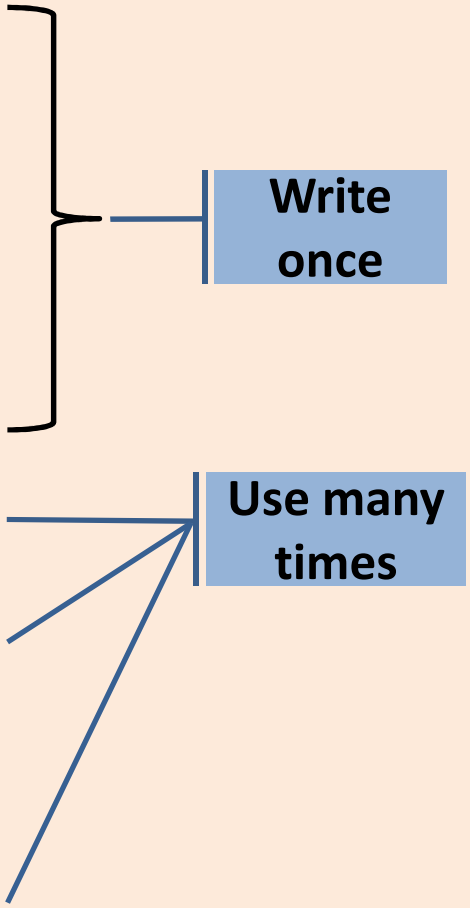
A quick example

```
import sys

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

FirstFileName = sys.argv[1]
FirstDict = makeDict(FirstFileName)

SecondFileName = sys.argv[2]
SecondDict = makeDict(SecondFileName)

…

FlyGenesDict = makeDict("FlyGeneAtlas.txt")
```



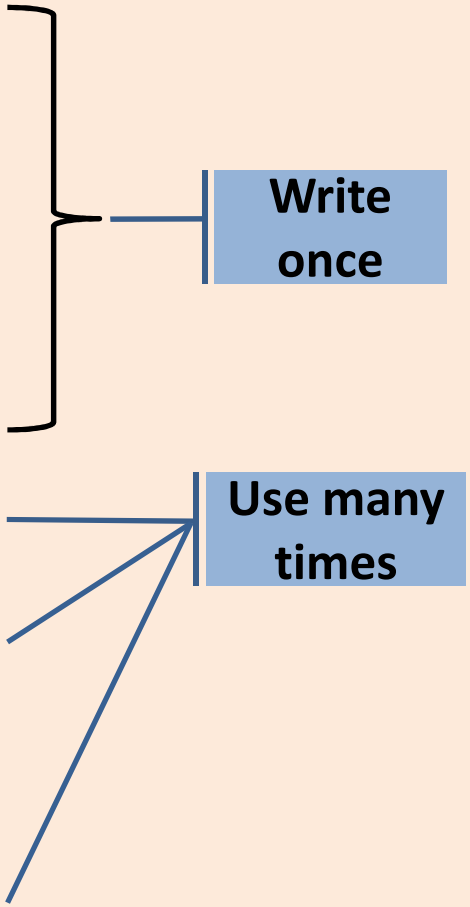
A note about namespace

```
import sys

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

FirstFileName = sys.argv[1]
FirstDict = makeDict(FirstFileName)

SecondFileName = sys.argv[2]
SecondDict = makeDict(SecondFileName)

…

FlyGenesDict = makeDict("FlyGeneAtlas.txt")
```



A note about namespace

```
import sys

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

FirstFileName = sys.argv[1]
FirstDict = makeDict(FirstFileName)

SecondFileName = sys.argv[2]
SecondDict = makeDict(SecondFileName)

…

FlyGenesDict = makeDict("FlyGeneAtlas.txt")
```



Arguments can be of any type!

- Arguments do not have to be strings (as in most of the examples we have seen so far).
- You can use **ANY** data type!
  (but make sure the code in the function is appropriate for the data type you use)

```
# This function increment every element in
# the input list by 1
def incrementEachElement(a_list):
    new_list = []
    for item in a_list:
        new_list.append(item+1)
    return new_list

# Now, create a list and use the function
my_list = [1, 20, 34, 8]
inc_list = incrementEachElement(my_list)
Print inc_list
```

```
[2, 21, 35, 9]
```

Here's a cool feature:

In Python, you can in fact use a function as an argument to another function!

Why is this useful?

Applying a function to a list

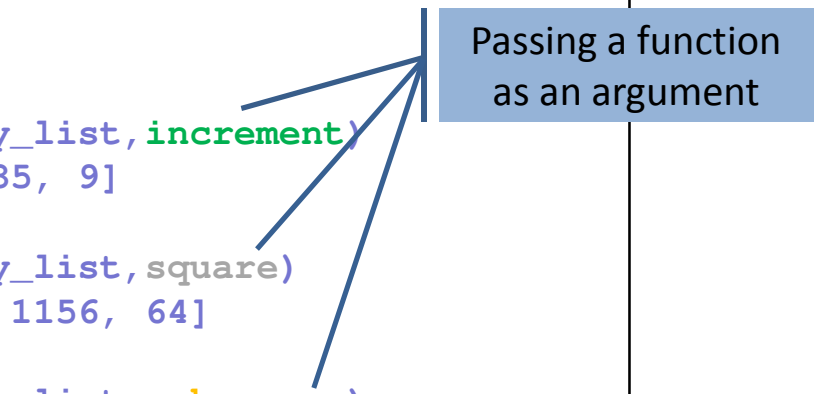
- Recall the increment function:

```
# This function increment every element in
# the input list by 1
def incrementEachElement(a_list):
    new_list = []
    for item in a_list:
        new_list.append(item+1)
    return new_list
```

- Let's make it more general:
  - Include an additional argument that will point to some function
  - Instead of incrementing each element, we will apply the specified function to each element

Applying an arbitrary function to a list

```
# This function does something to every element in the input list
def DoSomethingToEachElement(a_list, my_func):
    new_list = []
    for item in a_list:
        new_list.append(my_func(item))
    return new_list

def increment(i):
    return i+1

def square(i):
    return i*i

def make_zero(i):
    return 0

my_list = [1, 20, 34, 8]

inc_list  = DoSomethingToEachElement(my_list,increment)
print inc_list   # will print [2, 21, 35, 9]

sqr_list  = DoSomethingToEachElement(my_list,square)
print sqr_list   # will print [1, 400, 1156, 64]

zero_list = DoSomethingToEachElement(my_list,make_zero)
print zero_list  # will print [0, 0, 0, 0]
```



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 elements 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]
```

But …

What if we want to sort something else?

What if we want a different sort order?

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 readlines() work?
    (remember – you in fact implemented your own version of readlines in class last time)
  - How does split() work?
  - ***How does sort() work?***

(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.)

But …

What if we want to sort something else?

What if we want a different sort order?

But …

What if we want to sort something else?

What if we want a different sort order?

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!)

Comparison function

- Always takes 2 arguments
- Returns:
  - -1 if first argument should appear earlier in sort
  - 1 if first argument should appear later in sort
  - 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?

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]
```

descending numeric sort

You can write a comparison function to sort anything in any way you want!!

```
>>> 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!!

```
>>> 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]]
```

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

```
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
- 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

Solution #1

```
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.)

Solution #2

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

The function you wrote for problem #1

```
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)

Challenge solution 1

```
<your caselessCompare function here>

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]
```

Alternative challenge solution 1

```
<your caselessCompare function here>

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
```

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.
```

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: