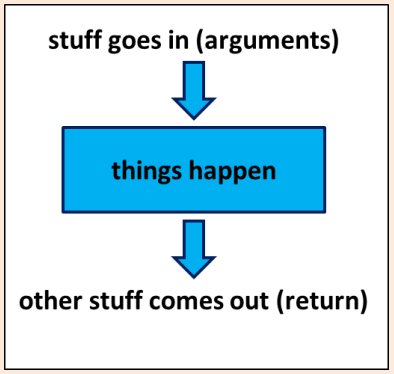
Sorting, Functions as Arguments

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:

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

- Using (calling) a function:

```
<function defined here>

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

A quick review

- Returning multiple values from a function

```
return [sum, prod]
```

- Pass-by-reference vs. pass-by-value
- Default and keyword Arguments

```
def printMulti(text, n=3):
```

- **Modules:**
  - Easy to create and use your own modules
  - To use a module, first import it:

```
import utils
```

  - Use the dot notation:

```
utils.makeDict()
```

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):
    < ... >
```

my_prog.py

```
import utils
import sys

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

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

...
```

Sorting

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]
```

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

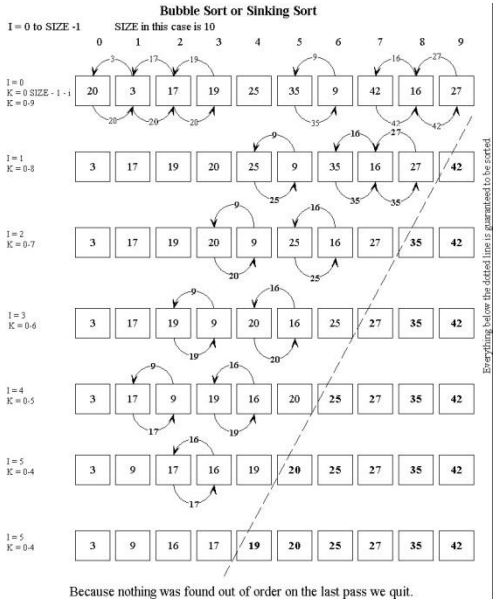
- A sorting algorithm takes a list of elements in an arbitrary order, and sort these elements in an ascending order.

- **… but how does a sorting algorithm work?**

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!)

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