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

- Default and keyword Arguments

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

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

    ```python
    import utils
    ```
  - Use the dot notation:

    ```python
    utils.makeDict()
    ```
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]
  ```
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.k.a. “how would you sort a list of numbers?”)
Common sorting algorithms

- **Naïve sorting**

  - **Selection sort**: At each iteration, find the smallest element and move it to the beginning of the list

  - **Insertion sort**: At each iteration, removes one element and insert it to the correct location in the sorted sub-list

- **Bubble sort**
  
  Swap two adjacent elements whenever they are not in the right order

- **Merge sort**
  
  Split your list into two halves, sort each half, merge the two sorted halves, maintaining a sorted order
But ...

What if we want a different sort order?

What if we want to sort something else?

What if I don’t know any sorting algorithm?
But ...

What if we want a different sort order?
What if we want to sort something else?
What if I don’t know any sorting algorithm?
But ...

What if we want a different sort order?
What if we want to sort something else?
What if I don’t know any sorting algorithm?
But ...  

What if we want a different sort order?  
What if we want to sort something else?  
What if I don’t know any sorting algorithm?
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
>>> 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:

```python
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
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