PACKAGE 2:

Exercise 1

Do the following using loops and then again using slicing

Insert

Write a function called **insert** that takes 3 parameters, **listA**, **listB** and an index, then returns a copy of **listA** with the elements of **listB** inserted at the index. Your code should work with either strings or lists e.g.,

```
>>> insert([1, 2, 3], ['a', 'b', 'c'], 2)
 [1, 2, 'a', 'b', 'c', 3]
>>> insert("123","abc",2)
 '12abc3'
```

Up_to_first

Write a function called **up_to_first** that takes two parameters, a list (or string) and an object, and returns a copy of the list up to (but not including) the first occurrence of that object, or all of the elements if that object is not in the list. e.g.,

```
>>> up_to_first([1, 2, 3, 4], 3)
 [1, 2]
>>> up_to_first([1, 2, 3, 4], 9)
 [1, 2, 3, 4]
>>> up_to_first('abcdef', 'd')
 'abc'
```

cut list

Write a function called **cut_list** that "cuts" a list. That is, given a list and an index, returns a copy of the list, but with the items before and after the index swapped. As always, your code should also work with strings. e.g.,

```
>>> cut_list([0,1,2,3,4,5,6,7,8,9], 3)
 [4, 5, 6, 7, 8, 9, 3, 0, 1, 2]
>>> cut_list("ABCDEFGX1234",7)
 '1234XABCDEFG'
```

Exercise 2

Finding Function Names

Write a function called **function_names** that takes as a parameter a file handle open for reading, and returns a list of all of the function names in that file. Remember that function definitions have a very specific format: def function name(parameters). You can assume that all

functions are exactly correctly formatted according to PEP-8 standards. e.g., 1 space after the def, no space before the (.

Calling function names on the (Previous Exercise) would (hopefully) return the result:

['insert', 'up_to_first', 'cut_list']

Section Average

Write a function called **section_average** that takes two parameters: The first parameter is an open file of midterm marks (no, they're not real marks). Each line represents a single student and consists of a student number, a name, a section code and a midterm grade, all separated by whitespace. An example file has been uploaded as **grade_file.txt**. The second parameter is a section code. Return the average midterm mark for all students in that section, or return None if the section code does not appear in the marks file for any students. As before, this will be run through the auto-marker, but the marks won't count. It's just for your own enjoyment.

Hint: the split method might be particularly useful here.

Hint: Notice that not everyone has the same number of names, so we can't just assume that the mark and section code will be at a particular index... at least not reading from the left.

Exercise 3

Changing a Copy of a List

Write a function called **copy_me** that takes as input a list, and returns a copy of the list with the following changes:

- Strings have all their letters converted to upper-case
- Integers and floats have their value increased by 1
- booleans are negated (False becomes True, True becomes False)
- Lists are replaced with the word "List"

The function should leave the original input list unchanged (i.e., do not mutate the list)

Mutating a List

Write a function called **mutate_me** that takes as input a list, returns None, and changes the input list in the following ways:

- Strings have all their letters converted to upper-case
- Integers and floats have their value increased by 1
- booleans are negated (False becomes True, True becomes False)
- Lists are replaced with the word "List"

So we're performing the same task, but this time we're changing the list itself instead of making a copy to return.

Exercise 4

Building the Dictionary

Write a function named **create_dict**, that takes one parameter, an open file handle (remember ex5, this is an open file handle, not the name of a file), and returns a dictionary that maps a string to a list of strings and ints. In particular, the type contract of this function will be:

(io.TextIOWrapper) -> dict of {str: [str, str, str, int, str]}

This function will read a file formatted like the provided **data.txt**. That is, each line will have a username, first name, last name, age, gender (either M, F or X) and an e-mail address, all separated by spaces. The function will insert each person's information into a dictionary with their username as the key, and the value being a list of **[last name, first name, e-mail, age, gender]**. When the entire file has been processed, it will then return the dictionary.

Changing the Dictionary

Write a function called **update_field**, that takes 4 parameters: A dictionary in the format created by the previous function, a username, the name of a field (One of: **'LAST'**, **'FIRST'**, **'E-MAIL'**, **'AGE'** or **'GENDER'**), and a new value to replace the current value of the specified field. The function should not return anything, but instead mutate the dictionary as appropriate. An example call of the function is below:

```
>>> my_dict = {'sclause':['Clause', 'Santa', 'santa@christmas.np', 450, 'M']}
>>> update_field(my_dict, 'sclause', 'AGE', 999)
>>> my_dict == {'sclause': ['Clause', 'Santa', 'santa@christmas.np', 999, 'M']}
 True
```