Week 5 - list operations & functions

Programming tips .

As your programs become larger, it is even more important to develop good habits:

* Use a comment at the top of the file to describe the purpose of the program. Include your name and the date, too.
* Use meaningful variable names -- i.e. names that describe the contents of the variables.
* Use comments for each function that include:
  + a description of what the function does (“purpose”)
  + a description of the input parameter value(s)
  + a description of the return value
* Use def main() to combine the steps of your program into a function.

The above are mandatory and you will be graded on them.

I strongly advise you to **write your programs incrementally and test them as you go.** No one writes interesting programs all at once. As you write, test that your program does what you expect (even if it doesn't yet do what you want). If it surprises you, make sure you understand what's happening before writing more code.

I have provided some test input and output; make sure you test your program! You should also test your program using some inputs and outputs that were not shown here; after all, I will. Come up with your own test cases and verify that the program is producing the right output on them.

**Topics for this Assignment**: *~~stack diagrams~~, functions, lists*

**~~Note~~** ~~that there is a written portion to this lab that is~~ **~~due in class~~** ~~on Thursday, November 1.~~

~~1. Stack Diagram~~

~~The first part of the lab involves you tracing a program with functions and showing the resulting stack that the function calls generate. This is excellent practice for what will almost certainly be a quiz question!~~ [~~Download the PDF~~](https://docs.google.com/document/d/1RCXudFYazAFMdd2OYZR1kH6C-yiOkkBtLF2zHWIctXI/edit?usp=sharing)~~, print it out, and turn it in at the start of class on Thursday.~~

2. List Operations and Functions

For this week's lab, you will write one program called listFunctions.pythat implements a number of list operations. In order to complete this program you must implement each of the following functions. Note that normally it is fine to use built-­in Python functions such as the list.countand list.index methods, the **max** function, and the list.sortfunction, you should avoid using them in this lab because I would like you to learn how to write these functions yourself.

All of your functions should include a block comment describing the behavior of the function, the parameters and the return value.

1. mainshould test each of the functions you are required to write below. You should test your program incrementally. First, write the inputIntegerfunction and be sure that it works before continuing to the getListFromUserfunction. Keep this practice up until you have written all of the functions. An example of the output of this program is included at the end of the writeup, but you should be sure to test each of the functions fully in your main.
2. inputIntegertakes a prompt string as its only parameter and asks the user to enter an integer using that prompt. The function repeats the prompt until the user enters a valid integer, and returns the first valid integer entered by the user. Your function should use try/exceptto validate that the user only enters integers.
3. getListFromUsertakes no parameters. The function asks the user to enter a series of positive integers (using the inputIntegerfunction). Each time the user enters a value that is added to the list, display the current contents of the list. When the user enters zero or a negative integer, return a list of all the positive integers entered, in the order that the user entered them.
4. largestIndextakes a list as its only parameter and returns the indexof the largest element in the list. If there are no elements in the list, this function returns **­**-1. Examples:

* largestIndex([60,29,82,45])returns 2, the index of 82.
* largestIndex([45])returns 0, the index of 45.
* largestIndex([2, 63, 2, 63])returns 1, the lowest index of 63.
* largestIndex([ ])returns ­-1, since the list is empty.

1. smallestIndextakes a list as its only parameter and returns the indexof the smallest element in the list. If there are no elements in the list, this function returns **­**-1. Examples:

* smallestIndex([60, 29, 82, 45])returns 1, the index of 29.
* smallestIndex([45])returns 0, the index of 45.
* smallestIndex([2, 63, 2, 63])returns 0, the lowest index of 2
* smallestIndex([ ])returns ­-1, since the list is empty.

1. findElementtakes a list and an element (in that order) and returns the indexof the first occurrence of the element in the list. If the element does not occur in the list or there are no elements in the list, this function returns **­**-1. Examples:

* findElement([60, 29, 82, 45], 60)returns 0, the index of 60.
* findElement([60, 29, 82, 45], 82)returns 2, the index of 82.
* findElement([60, 29, 29, 29], 29)returns 1, the first index where 29 occurs.
* findElement([60, 29, 82, 45], 20)returns ­-1, since 20 is not in the list.
* findElement([ ], 20)returns -­1, since the list is empty.

1. averagetakes a list as its only parameter and returns the average of the elements in the list. The returned average should be a float. If there are no elements in the list, return 0.0. Examples:

* average([60, 29, 82, 45]) returns 54.0
* average([45])returns 45.0
* average([2, 63, 2, 63]) returns 32.5
* average([ ])returns 0.0, since the list is empty

1. insertAtIndextakes a list, an element, and an index (in that order) and returns a new list with the element inserted at the specified index. The index can be between 0 (inserted as the first element in the list) and the length of the list (inserted as the last element in the list). If position index is not valid, return the original list unchanged. Examples:

* insertAtIndex([60, 29, 82, 45], 18, 0)returns [18, 60, 29, 82, 45]
* insertAtIndex([60, 29, 82, 45], 18, 1)returns [60, 18, 29, 82, 45]
* insertAtIndex([60, 29, 82, 45], 18, 4)returns [60, 29, 82, 45, 18]
* insertAtIndex([60, 29, 82, 45], 18, 5)returns [60, 29, 82, 45] because 5 is an invalid index to insert at.
* insertAtIndex([60, 29, 82, 45], 18, -1)returns [60, 29, 82, 45] because -­1 is an invalid index to insert at.

1. removeAtIndextakes a list and an index (in that order) and returns a new list with the element at the specified index removed. If the index is not valid, return the original list unchanged. Examples:

* removeAtIndex([60, 29, 82, 45], 0)returns [29, 82, 45]
* removeAtIndex([60, 29, 82, 45], 1)returns [60, 82, 45]
* removeAtIndex([60, 29, 82, 45], 2) returns [60, 29, 45]
* removeAtIndex([60, 29, 82, 45], 5)returns [60, 29, 82, 45] because 5 is an invalid index.
* removeAtIndex([60, 29, 82, 45], -1)returns [60, 29, 82, 45] because ­-1 is an invalid index.
* removeAtIndex([ ], 0) returns [] because 0 is an invalid index.

1. doubleListtakes a list as its only parameter and doubles every element in the list. This function does not create a new list, rather it modifies each element of the list. For example:

* >>> **lst = [3, -2, 4, 22]** >>> **lst2 = [ ]**  
  >>> **double\_list(lst)** >>> **double\_list(lst2)**  
  >>> **print(lst)** >>> **print(lst2)**  
  [6, -4, 8, 44] [ ]

Here is an example of the program running. Your main does not have to follow this format exactly but it should test each of the functions you were required to write.

Enter positive integers that will be stored in a list.

Enter a number less than or equal to 0 to stop.

Enter a positive integer: **60**

The list is now [60]

Enter a positive integer: **29**

The list is now [60, 29]

Enter a positive integer: **hello**

That is not a valid integer.

Enter a positive integer: **82**

The list is now [60, 29, 82]

Enter a positive integer: **45**

The list is now [60, 29, 82, 45]

Enter a positive integer: **-1**

The original list is:

[60, 29, 82, 45]

The largest item in the list is 82

The position of this item is 2

The smallest item in the list is 29

The position of this item is 1

Enter an integer to search for: **29**

This element was found at index 1

The average of the items is 54.0

The list with all the elements doubled is: [120, 58, 164, 90]

Inserting element 97 at index 3 yields the list: [120, 58, 164, 97, 90]

The list with the largest and smallest elements removed is: [120, 97, 90]

Extra Challenges .

Below are some ideas for some optional extensions to your List Operations implementation. **Note: extra challenges are just for fun (i.e., no bonus points). Please only attempt them after completing your regular assignment.**

* Throw TypeError and prevent the program from crashing if the user provides an index with a decimal (*i.e.* a floating point number).

>>> **L = ["dog","cat","bird","snake"]**

>>> **L.insert(1,"pony")**

>>> **L**

[ 'dog', 'pony', 'cat', 'bird', 'snake' ]

>>> **L .insert(1.0, "horse")**

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: integer argument expected, got float