

Lab 05: List ADT and its Applications

0. Credits

The lab content is the result of collective work of CSCI 204 instructors.

1. Objectives

In this lab, you will learn the following,

- Basic concepts of the List ADT;
- How to implement the List ADT using an array;
- How to implement exceptions for the List ADT.

Introduction

On the course Moodle, watch the Lab 5 introductory video for some advice to get started!

Partner Status: This lab must be done solo.

2. Preparation

To access the graphic Linux interface, follow the instructions posted on Moodle and in the previous lab (i.e. randomly pick one of <https://linuxremote1.bucknell.edu:3300/>, <https://linuxremote2.bucknell.edu:3300/>, or <https://linuxremote3.bucknell.edu:3300/>). For this lab, you must work in the linux environment, and are highly encouraged to use the Spyder IDE for us to be able to help debug.

Begin by creating a directory for lab 05 and copying lab files from the course Linux directory and examining them (if you do not remember how to copy the files, look back at previous labs). After creating a directory, you should use a command like:

```
cp ~csci204/student-labs/lab05/* .
```

You will not need to modify these files. Rather all your lab work should be contained in two files, `arraylist.py` that has all the required elements of the class `List`, and `ListException.py` that has the class `ListException`. You will create these two files. You must name these two files exactly as they are so the GroceryBuddy app will run as it is. Your task is to implement these two classes correctly in their respective files.

In the following sections, we will discuss some of the basic ideas of the list ADT and their use in our application, Grocery Buddy. Read these descriptions before implementing and testing your programs.

3. Grocery Buddy

You work for a company which creates computer apps to help people in the everyday world. You are currently working on the **GroceryBuddy** app which keeps a shopping list. It displays the name, number, and picture of desired grocery items in a list. Items can be added and removed from the list. The desired quantity of items can be updated and item positions can be swapped. You have been assigned the task of implementing a standard **List ADT** so that the app that uses a list will work properly.

4. List ADT

A list is a linear, ordered, indexed **ADT**. Like any ADT, a list has two sets of members, a collection of data and a set of operations defined on the data. The data held in a list can be of varying types. The set of operations (methods) supported by a list in general include the following:

- constructor: create an empty list;
- insert(item, index): insert a node with a given value **item** at the given location **index**;
- delete(index): remove a node at a given location **index**;
- len(): return the size of the list.

Some more elaborate lists may contain more operations. For this lab, you will also add the **peek** operation:

- peek(index): return the item at the given **index**.

A list ADT can be implemented using different data structures. In this lab you will implement a list ADT using an **array**.

5. List ADT with an Array

You must implement your list with an array in a file named **arraylist.py** (you will be asked to do so in the **Implementations** section.) Therefore your list will need data attributes for the **array** of items, the current **size**, and the current **capacity**.

- **array**: is an array implemented using the **Array** class inside the **array204.py** file. This array should initially have the capacity of 2 elements and can be created via `Array(2)`. Make sure to include the line

```
from array204 import *
```

in your Python implementation file for the List ADT.

- **capacity**: represents the current maximum size of your array underlying the List ADT. In the array above, the initial capacity is 2.
- **size**: represents the current number of elements in your List. When you first start out, the size is 0. Note that even though the underlying array has enough room to store 2 elements initially, the actual number of elements is zero to begin with. The size of your List will change with each **insert** and **delete** operation, while the capacity will only change when the **List** needs to be expanded or shrunk.

NOTE: When the size is equal to capacity, it means that you've run out of room in your array to store more elements. At this point, your array should automatically double in size, i.e., the capacity, to create room for more elements.

Because arrays cannot change size once created, you will need to create a new array that is double the capacity of the old one, and copy the elements from your old array into the new one.

Remember to name your methods and fields so it is clear which ones should be accessed from outside of the class. Names that only get inside access start with an underscore. For example, your **Array** will likely have an attribute called `self._array` since it should not be accessed from outside the list.

6. Implementations

Now that you have a basic understanding of the list ADT, let's implement it. We would like you to test the operations of the ADT as you make progress. Do not wait until the end for testing. For now, we will provide some testing programs for you. In future, you may be asked to come up these testing programs yourself. For this lab, we give you a test file. Among the Python programs you copied, one of them is **test_arraylist.py**. Open this file in your IDE (spyder or idle3), take a look at the program. The program has a

number of different functions that test various features of a list. You should comment out the sections in the function **test_arraylist()** that you have not implemented yet, then run the program to test the features that you have completed. Again, make sure you test often. Don't wait until all are implemented before testing.

6.1 Implement a ListException Class

First you are asked to implement a **ListException** class for your list using the knowledge you learned in the class. Your **ListException** class must inherit the **Exception** class--to be a python exception, your class must inherit the **BaseException** class, and **Exception** does that. See [the python documentation](#) for more details on the inheritance structure of python exceptions. You will have very little code in the class **ListException** itself. Put your **ListException** class in a file named **ListException.py**. You may find it helpful to review the exceptions you wrote for the counter class in Lab 02.

Test the ListException class: After completing the **ListException** class, run the program **test_exception.py** that you copied as one of the starting files in Spyder. You should see an output similar to the following (including the message printed by line 14 of the **test_exception.py** file), indicating your implementation of **ListException** is good. If your output is very different, you need to fix the errors in your exception class before moving forward.

```
--- Test exception ---
We will raise the ListException ...
Raise list exception.
We caught the exception raised.
--- Passed exception test ---
```

6.2 Implement the List Class

Implement your **List** class in a file named **arraylist.py**. Your **List** class must have the following standard methods:

- **__init__(self)**: creates an empty list. The starting capacity should be 2, the starting size should be zero. You will use the **Array** class in the

Array204, so you might have a line like ``self._array = Array(2),'` but will never need to directly call ctypes.

- `__len__(self)`: returns the size of the list. Remember that we name the method `__len__()` so that it can be used to overload the operation `len()`.
- `__str__(self)`: returns a string representation of the list. Again, this method will allow the programs that use the list to print the content of the list easily, as in `print(my_list)`, if **my_list** is an array based list as you are instructed to implement. For our test purpose, we ask you to construct a string representation similar to the following format. If the list contains a collection of words such as `['This', 'is', 'CSCI 204']`, we'd like your `__str__(self)` method return a similar string. (You may examine the program **test_arraylist.py** to see how this is tested.)
- `insert(self, item, index)`: inserts an **item** at the given **index**. If the index is less than zero, insert at index 0. If the index is larger than or equal to the size, insert at the last spot. If the index is occupied, shove the item at the index (and all the items after that) towards the end (right) to make room. If you need to expand the size of the underlying `:array`, do so.
- `delete(self, index)`: removes the item at the given **index**. If the index is illegal or unoccupied, raise a **ListException**. This method does not return anything.
- `peek(self, index)`: returns the item at the given **index**. If the index is illegal or unoccupied, raises a **ListException**.

We strongly recommend you test each method as you complete them. Take a look at the function **test_arraylist()** in the file **test_arraylist.py**. You can comment out the later segment of the code in the function **test_arraylist()** to test the ones you completed first. For example, you can comment out `test_insert()`, `test_peek()`, `test_delete()`, and `test_exception()` to test the constructor and `len` function. When the constructor passes the test, you can then test the `insert()` method, so on and so forth.

If any problems occur during the test, fix them before moving on, until all tests are executed successfully. Make sure that you verify the actual text printed by **test_arraylist()**; the "Passed BLAH test" lines verify only that the test ran without error, not that the output is correct. You may also want to directly test methods, like the constructor and string, that are used within other tests, and that your string method prints an empty list.

If everything is implemented properly, your final run of the **test_arraylist.py** should generate something similar to what is on the following.

```
--- Test the list constructor ---
A new list is created, its length should be 0 (zero), it is --> 0
--- Passed constructor test ---

--- Test the insert() method ---
Length of the list should be 6, it is --> 6
The list content should be ['hello', 'world', 'how', 'are', 'you', '?']
It is --> [hello, world, how, are, you, ?]
--- Passed insert() test ---

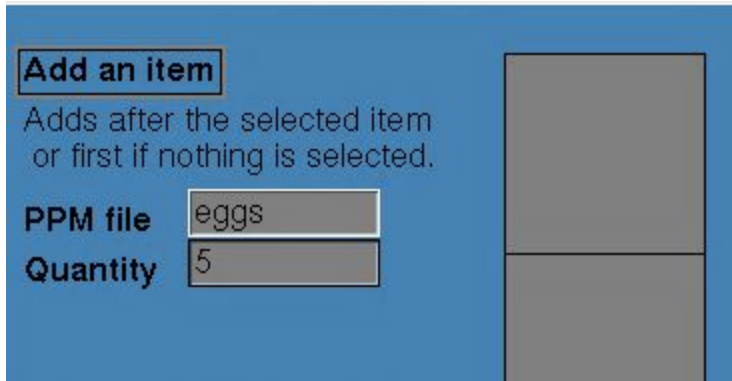
--- Test the peek() method ---
The items in the list should be ['hello', 'world', 'how', 'are', 'you', '?'],
it is --> [hello world how are you ? ]
--- Passed peek() test ---

--- Test the delete() method ---
The items in the list should be ['world', 'are', 'you'], it is --> [world
are you ]
--- Passed delete() test ---

--- Test the exceptions ---
Peek at a non-existing location should raise an exception
Index error
Caught peek error at a wrong index.
Deleting at index -1, should cause exception
Index error
Caught delete error at index -1
Deleting at index n, should cause exception
Index error
Caught delete error at index n
--- Passed exceptions test ---
```

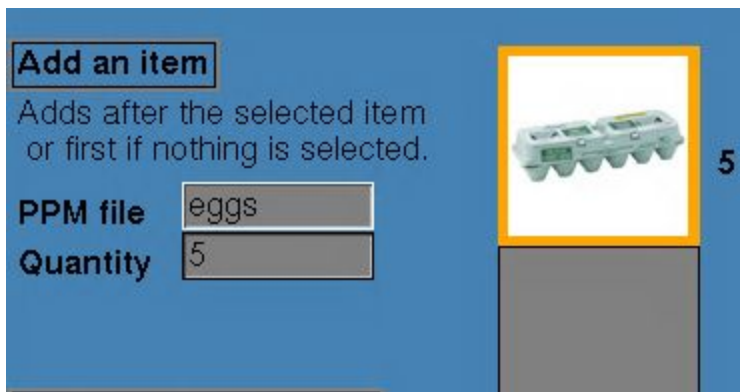
After completing the implementation, you should be able to load and run the Python program `grocery_app.py`. When the program is running, try your shopping experience (!!) by adding any item that is represented by a graphics file, for example, eggs, beans, or others, or removing things that you already put in your shopping cart.

To run this app, you can e.g. type `eggs` at the top:



The screenshot shows a blue background with a form titled "Add an item" in a red-bordered box. Below the title is the text "Adds after the selected item or first if nothing is selected." There are two input fields: "PPM file" containing the text "eggs" and "Quantity" containing the number "5". To the right of the form is a tall, empty gray rectangular box representing a list.

and then click "Add an item." Doing so should yield



This screenshot shows the same "Add an item" form, but the gray list box on the right now contains an image of a green egg carton. To the right of the image is the number "5". The "PPM file" field still contains "eggs" and the "Quantity" field still contains "5".

with 5 eggs at the top of your list! You can add any item corresponding to one of the ppm images you copied into your lab05 directory at the start of the lab! Play with the other features.

7. Submitting Your Work

Congratulations! You've completed this lab. Please make sure submit your **arraylist.py** and **ListException.py** files on Moodle. As always, you should submit to two places: to Moodle and repl.it; for both, you need only submit the two files you created.

Note: to be guaranteed at least an 80% on this lab, you must:

- Work on it for two contiguous hours by Friday at 6pm ET (either in your lab block, or a block you create).
- Submit partial progress to Moodle, under the Lab 5 - Friday Checkpoint assignment. Just submit files you edited for this checkpoint.