

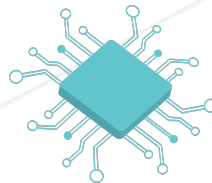


Object Oriented Programming, Part 2 - Linked Lists

The Archnemesis of Arrays

tingo - tingo@student.42.us.org
jchung - jchung@student.42.us.org

*Summary: This is an introduction to **Linked Lists**.*



**HACK
HIGH
SCHOOL**



*This work is licensed under a [Creative Commons
Attribution-NonCommercial-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-nc-sa/4.0/).*

Contents

I	Foreword	2
II	Introduction	4
III	Goals	5
IV	General instructions	6
IV.1	Node Class	7
IV.2	SinglyList Class	7
V	Exercise 00 : Print All Nodes in a List	8
VI	Exercise 01 : Add an Item to the End of a List	9
VII	Exercise 02 : Remove an Item From a List	10
VIII	Exercise 03 : Cycle Detection	11
IX	Exercise 04 : Merge Two Lists	12
X	Exercise 05 : Sort a Linked List	13
XI	Bonus part	14
XII	Exercise 06 : Trainyard Revisited	15
XIII	Turn-in and peer-evaluation	16

Chapter I

Foreword

Rick Astley - Never Gonna Give You Up

We're no strangers to love
You know the rules and so do I
A full commitment's what I'm thinking of
You wouldn't get this from any other guy
I just wanna tell you how I'm feeling
Gotta make you understand

Never gonna give you up
Never gonna let you down
Never gonna run around and desert you
Never gonna make you cry
Never gonna say goodbye
Never gonna tell a lie and hurt you

We've known each other for so long
Your heart's been aching but you're too shy to say it
Inside we both know what's been going on
We know the game and we're gonna play it
And if you ask me how I'm feeling
Don't tell me you're too blind to see

Never gonna give you up
Never gonna let you down
Never gonna run around and desert you
Never gonna make you cry
Never gonna say goodbye
Never gonna tell a lie and hurt you
Never gonna give you up
Never gonna let you down
Never gonna run around and desert you
Never gonna make you cry
Never gonna say goodbye
Never gonna tell a lie and hurt you
Never gonna give, never gonna give

(Give you up)
(Ooh) Never gonna give, never gonna give
(Give you up)

We've known each other for so long
Your heart's been aching but you're too shy to say it
Inside we both know what's been going on
We know the game and we're gonna play it
I just wanna tell you how I'm feeling
Gotta make you understand

Never gonna give you up
Never gonna let you down
Never gonna run around and desert you
Never gonna make you cry
Never gonna say goodbye
Never gonna tell a lie and hurt you
Never gonna give you up
Never gonna let you down
Never gonna run around and desert you
Never gonna make you cry
Never gonna say goodbye
Never gonna tell a lie and hurt you
Never gonna give you up
Never gonna let you down
Never gonna run around and desert you
Never gonna make you cry

Chapter II

Introduction

Linked lists are the foundation for more advanced data structures and are commonly used in place of arrays for data storage due to their ability to store information in non-contiguous blocks of memory. This makes linked lists a very powerful tool for parallel computation.

There are many uses for the different variations of linked lists. For example, many games will store cycling character animations in a circular linked list or a circular array. Many operating systems line up processes and jobs in a queue. A simulation of genetic mutations like removing and swapping base pairs in DNA strands can best be organized by a doubly linked list.

Within the 42 curriculum, the projects in graphics branch frequently utilize linked lists to store coordinates of points for an object in a map or rendering. In many Unix projects, making a process and job queue for operating systems is essential to keep the computer running without a crash.

Chapter III

Goals

This project lays groundwork for all of the data structures you will learn in the following days. Today, you will learn how to implement your very own data structure. You will be responsible for creating the basic functions that allow a data structure to access values, add values, remove values, and modify itself in fun and interesting ways.

This project is a simple gateway into the larger world of Data Structures.

Chapter IV

General instructions

The following exercises are designed such that written functions are to be turned in as standalone functions; they will not be included in the class for `LinkedLists` but will still perform standardized operations. All exercises should be written in Python and no external function calls should be made except for the first question; only the functions supplied in the classes provided to you should be needed.

Both a `Node` and `SinglyList` class have been given to you for use. The `SinglyList` class has an in-class iterator defined to traverse a list, but you are free to iterate through the list yourself :)

If a function requires the use of the `Node` or `SinglyList` class, do not include that code in your file submission. If you use a previous solution please make sure to include that function in the file submission.

IV.1 Node Class


IV.2 SinglyList Class



What does the @ symbol do in Python? - It creates a Decorator:
[Stack Overflow](#)

Chapter V

Exercise 00 : Print All Nodes in a List

	Print All Nodes in a List
Topics to study :	
Files to turn in : <code>print_list.py</code>	
Notes : n/a	

Understanding how to traverse a linked list is important to mastering its concept. Given the head node of a singly linked list, print out all the items in the list.

Your function will be defined as such:

```
print_list(list_head)
```

Input Format


Your function will take the head node of a list.

Output Format

Nothing special. Just print the items of the list in the order they appear.

Chapter VI

Exercise 01 : Add an Item to the End of a List

	Add an Item to the End of a List
Topics to study :	
Files to turn in : <code>add_tail.py</code>	
Notes : There is a similar function in the <code>SinglyList</code> class to reference :)	

You're given the pointer to the head node of a singly linked list and the value of a node to add to the list. Create a new node with the given value. Insert this node at the tail of the linked list and return the head node after the insertion.

Your function will be defined as such:

```
add_tail(list_head, val)
```

Input Format


Your function will take the head node of a list and a value.

Output Format

There will be no return, but the list must now contain a new node with the specified value at the end of it.

Chapter VII

Exercise 02 : Remove an Item From a List

	Remove an Item From a List
Topics to study :	
Files to turn in : <code>remove.py</code>	
Notes : If the list is empty, head will be null. The list will not contain duplicates.	

You're given the pointer to the head node of a singly linked list and the value of a node to delete from the list. Delete the node with the given value if it exists.

Your function will be defined as such:

```
remove(list_head, val)
```

Input Format


Your function will take the head node of a list and a value.

Output Format

There will be no return, but the list must no longer contain the node with that value.

Chapter VIII

Exercise 03 : Cycle Detection

	Cycle Detection
Topics to study :	
Files to turn in : <code>has_cycle.py</code>	
Notes : If the list is empty, head will be null.	

In a turn-based multiplayer game, a linked list can be used to cyclically repeat player order by having the last player's `next` reference the first player. Check that a given linked list cycles or not.

Your function will be defined as such:

`has_cycle(list_head)`

Input Format


Your function will take the head node of a list.

Output Format

It will return `True` if there is a cycle and `False` if there is no cycle.

Chapter IX

Exercise 04 : Merge Two Lists

	Merge Two Lists
Topics to study :	
Files to turn in : <code>merge.py</code>	
Notes : All trains have at least one car, and no two cars have the same weight.	

Two cargo trains arrive in the trainyard and their cargo needs to be consolidated into a single train set before it can depart. Both trains were organized with the heaviest car in the front of the train descending to the lightest car in the back.

Your function will be defined as such:

```
merge(train1, train2)
```

Input Format


Your function will take two list heads, *train1* and *train2*.

Output Format

Return the head of the new merged train.

Chapter X

Exercise 05 : Sort a Linked List

	Sort a Linked List
Topics to study :	
Files to turn in : <code>sort_asc.py</code>	
Notes : There is no limitation on which sort to implement for this exercise. You will find some sorts are easier to work into a linked list than others...	

Understanding how to organize information in a list is important. Implement a sorting algorithm that organizes a linked list with numbers in **ascending** order.

Your function will be defined as such:

```
sort_asc(unsorted_list)
```

Input Format

Your function will take the head node of an unsorted list.

Output Format

Sort the list in function, but return nothing.

Chapter XI

Bonus part


Now that you've had a chance to get your feet wet with the linked list, let's revisit a previous problem. The trainyard has run into some more trouble. See if you can help them.



This will only count if and only if you have completed the previous exercises correctly. Go back now to make sure.

Chapter XII

Exercise 06 : Trainyard Revisited

	Trainyard Revisited
Topics to study :	
Files to turn in : <code>trainyard.py</code>	
Notes : All trains have at least one cart, and two carts may have the same weight.	

Two more cargo trains arrive in the trainyard and need to be merged before departure. These two trains are no longer organized by weight from heaviest to lightest prior to the merge. The two lightest cars must also stay behind at the trainyard for inspection and leave with the next incoming train. Take into account cars with equal weights.

Your function will be defined as in exercise 04.

Input Format

Same as 04

Output Format

Same as 04, but missing the lightest 2 and sorted.

Chapter XIII

Turn-in and peer-evaluation

This project will be evaluated by your peers. Submit to the **GIT** repository as usual. Only work in your repository will be graded. Make sure you have the *exact* function names as shown in this document. Only the specified functions will be looked at and graded. Remember, the bonus will only be evaluated if *all* the previous excercises are correct.