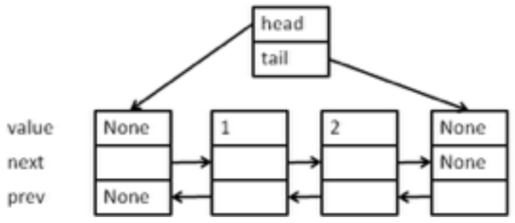
CS417 Programming Assignment 7

*Due: Friday, November 6th.

Late penalty: Sat/Sun/Mon 5%, Tue 10%, Wed 20%, Thu 50%, Fri 100%.

Linked lists

This assignment focuses on linked lists. Your task is to implement a doubly-connected linked list with two dummy sentinel nodes. Each node in the list contains a value, and two pointers (`_prev`, `_next`) which refer to the nodes before and after it in the list. In addition, there are sentinel nodes (the head and tail nodes) at the very start and end of the list; each of these dummy nodes holds the value `None`, and has one `None` pointer: the head node has `head._prev == None` and the tail node has `tail._next == None`. All other pointers refer to actual nodes. Here is a sketch of the list `[1, 2]`:



Starting Code

Begin by creating a directory for your work, and download the starting file into it: `single_list.py` . This is an implementation of an ordinary singly-linked list.

Then, copy this file into another called `double_list.py`. You will edit this file, and turn it when you are done.

The implementation includes two methods that you will probably find useful:

- `has_back_links()` returns `True`/`False` if your list has/hasn't next pointers that refer to the current, or previous nodes in the list.
- `__repr__` shows the list in gory detail, including the addresses (`id`s) of all its nodes.

What you should do

1. In your file `double_list.py`, rename the class. It should now be `class Dou-`

ble_List:.

2. In the `main()` function, mentions of `Linked_List` should be replaced with `Double_List`.

3. The `Double_List` class should implement the following methods:

Method	Description	Work needed?
`__init__`	Constructor. Note that the constructor has an optional argument `orig` , which defaults to `None` if omitted. If `orig` is omitted, create an empty list. If `orig` is another `Double_List`, make a copy of it (like a copy constructor in C++). If it is a list, insert that list's elements.	Missing tail field and dummy head, tail nodes
`copy()`	Make a copy of this list	Should return a `Double_List`
`__iter__()`	Generator which visits all the values	Should start at second node, not the head (which is now a sentinel)
`__reversed__()`	Generator which visits all the values, in *reverse* order	YES. Not implemented
`List_Node`	needs a `_prev` field in `__init__` and `__repr__` methods	Missing `_prev` field
`is_empty()`	return `True`/`False` if the list `is`/`isn't` empty.	You must implement this from scratch
`insert(self,value,`	inserts a node with the given `value` at a given `index`. The method should `raise IndexError` if the index is not valid (eg, if you're	Doesn't account for sentinel nodes. You must also set

`index)`	inserting at a negative index, or at an index that lies beyond the list's tail).	the `_prev` fields in several nodes.
`__add__(self,other_list)`	Returns a new list, which is the concatenation of `self` and `other_list`.	No work needed
`add_front(self,value)`	equivalent to `self.insert(value, 0)`. Call that method, or implement this method separately. Use your judgement.	Doesn't account for sentinel nodes, or for `_prev` fields
`add_tail(self,value)`	equivalent to `self.insert(value, len(self))`	Doesn't account for sentinel nodes, or for `_prev` fields
`__len__(self)`	number of non-`None` nodes in the list. This is called by python when you invoke the `len()` function. IMPORTANT: the sentinel nodes should not counted in your calculation.	Doesn't account for sentinel nodes
`__setitem__(self,index, value)`	replace the value at the given `index`. May `raise IndexError` if the `index` is out of bounds.	Doesn't account for sentinel nodes
`__getitem__(self,index, value)`	return the value at the given `index`. May `raise IndexError` if the index is out of bounds.	Doesn't account for sentinel nodes
`__delitem__(self,index)`	remove the node at the given `index`. Return that node's old value.	Doesn't account for sentinel nodes
	returns a simple string representation of the list, showing the values in the nodes, separated by spaces and commas, and	

`__str__(self)`	delimited by square brackets. For example, a list in which node 0 holds 3, node 1 holds `'.'`, and node 2 holds 1 should return `'[3, ., 1]'`. Python will call this method in code like `str(my_list)`. IMPORTANT: the sentinel nodes should not be listed in the result.	Doesn't account for sentinel nodes
`__repr__(self)`	returns a string, which shows each of the list's nodes, showing their three fields (value, next, previous). Each node in the string should be separated from the previous one with a `'\n'` newline char. IMPORTANT: the sentinel nodes should *definitely* be listed in the result.	Doesn't account for sentinel nodes, or for `_prev` field

Turning in your work

When you are done, go to `mycourses.unh.edu`, find CS417 and assignment 7, and upload `double_list.py` .

IMPORTANT: You MUST create a new file called `double_list.py`, which defines a class called `Double_List`.