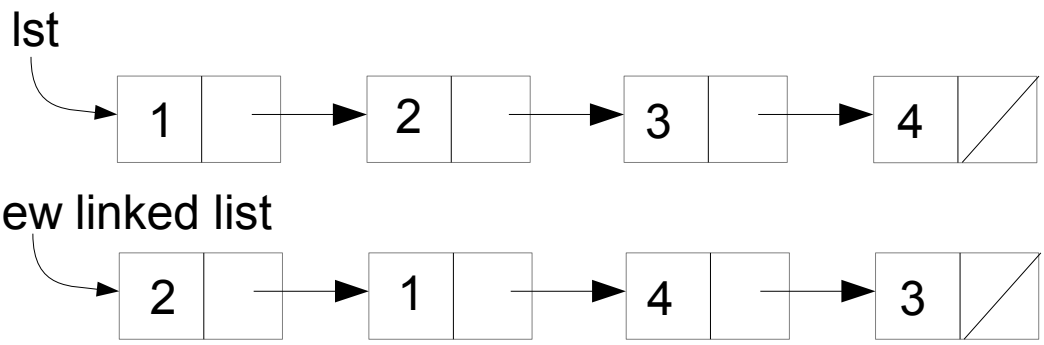
3a: Create a new linked list

Game plan:

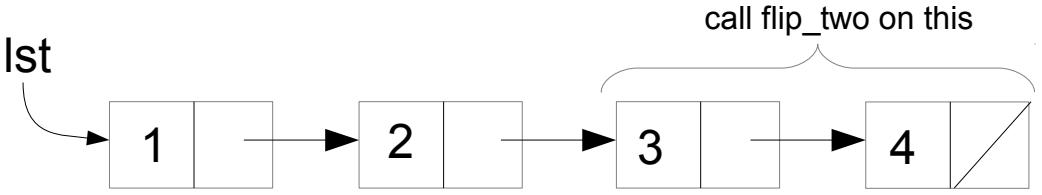
Note: lst must be unchanged



Since we know that linked list problems use recursion, we can imagine a general format of solution:

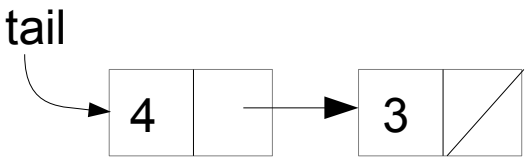
```
def flip_two(lst):
    if SOMETHING:
        return something
    RECURSIVE CALL
```

I generally start with the recursive call for these types of problems.
So assume I have a function flip_two that works (leap of faith) It would be great to just call it on the rest of the list, so then I can simply deal with the first two elements.



Here is the tricky part: flip_two returns a new linked list, it will not mutate lst. So we must somehow save the result of calling flip_two on the rest of the rest of lst.

```
tail = call_flip(lst.rest.rest)
```



Now that we have tail and lst, we can finish up and just flip 1 and 2 when we create our new list:

```
return Link(lst.rest.first, Link(lst.first, tail))
```

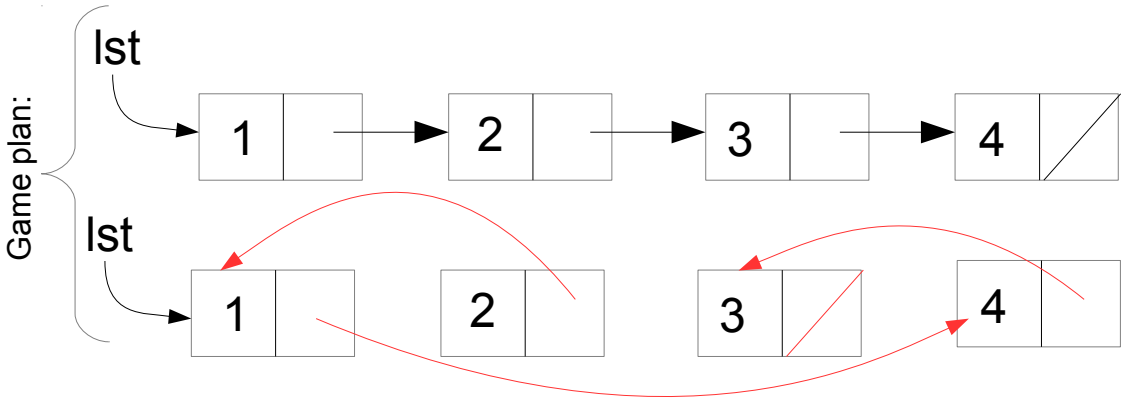
Base cases:
What do I need to ensure against to make sure that nothing causes my program to crash?
Every time I call an attribute of lst (first or rest) I need to make sure that lst exists, or I will be calling ().rest or ().first, which will make Python tell you that you have problems.

`lst.first` – lst must be not empty!

`lst.rest.rest` – lst.rest must be not empty!

Note: the order of the base cases matters!
I have to check that lst is not empty before I check that lst.rest is not empty, or my base case itself will cause the program to crash!

3b. Mutate the original list

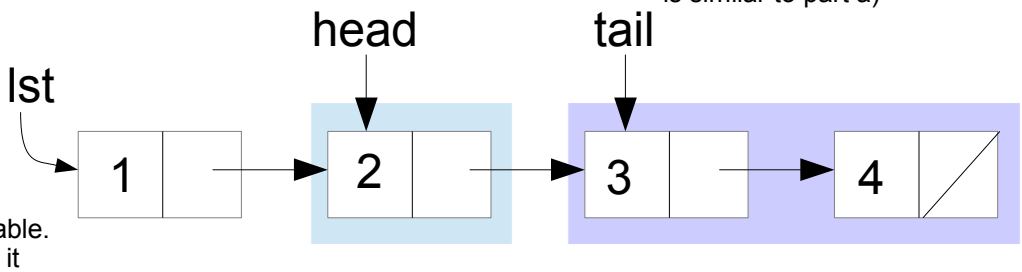


There is a lot going on in the picture to the left. Take a moment to understand how each pointer between the links changes.

Here are some questions that will facilitate the next few steps:

1. What do I want to return?
2. What should I do my recursive call on? (this step is similar to part a)

Before we start moving the pointers between the links, we should save what we want to return, and what we want the recursive call to be on.
Which link should be at the front of the list when I finish? (this is what I should return)



Once you identify the right link, save it in a variable. Here we save it in a variable called head, since it will be the new head of the list.
Now we know what we are going to return, head! Stop here and make sure you understand why we are doing this.

Look back at the game plan. We want the rest of the link with 2 to point to the 1.
What is the rest of the link with 2?

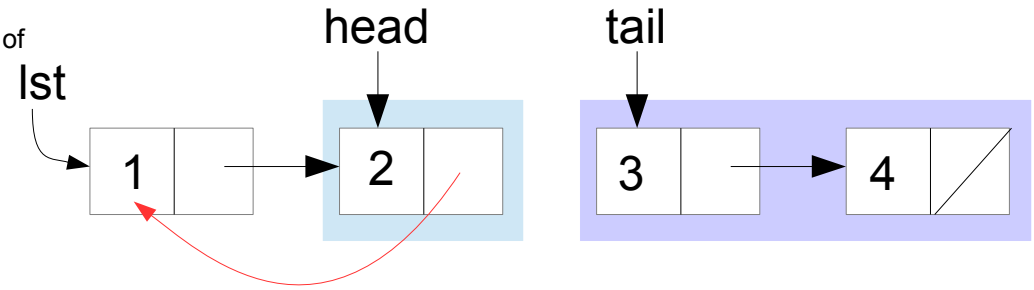
head.rest

What is the link with 1?

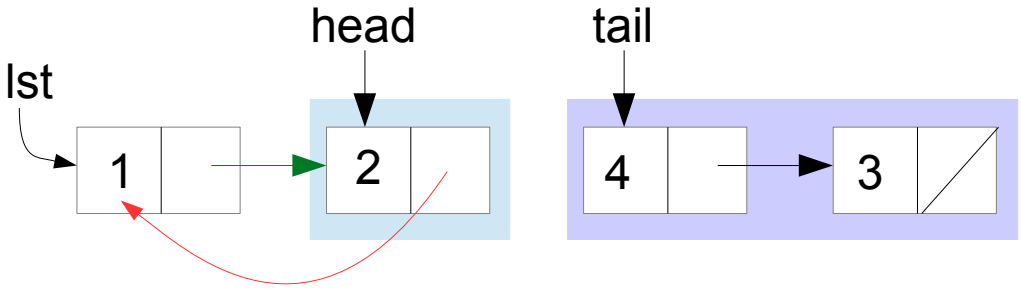
lst

So we just write:

head.rest = lst



Now we do the recursive call on the tail. Look at the image on the right (the result of calling flip_two on the tail) and compare it to the game plan.
What do we have to change to match the game plan?
The pointer out of the Link with 1 needs to go to the 4

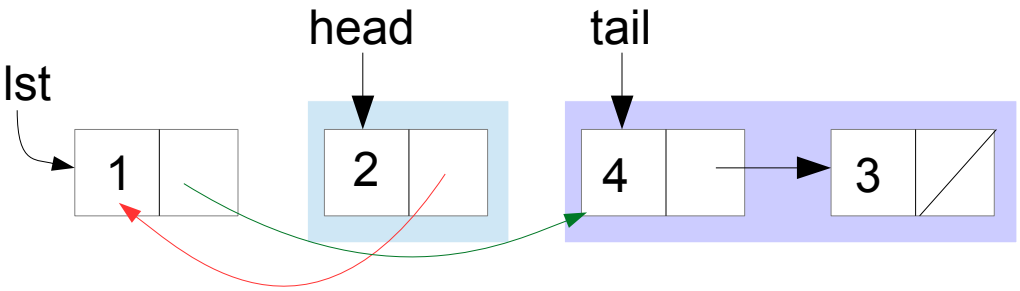


In other words, it points to the head of the flipped tail (the result of the recursive call)
So we write:

lst.rest = flip_two(tail)

Note this is the same as doing:

head.rest.rest = flip_two(tail)



Here is the final result of out pointer manipulations.
Just return head and you're done!