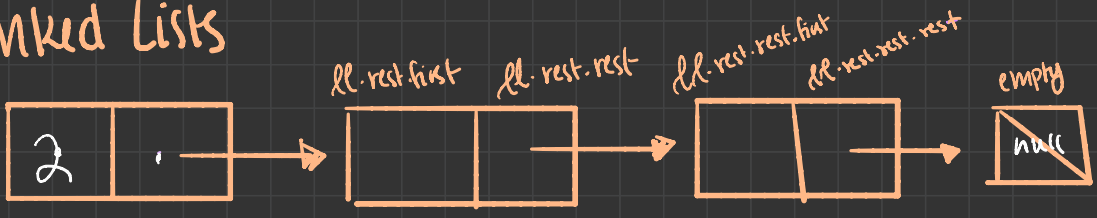
# Linked Lists




Link
- .first : value of linked list
- .rest : pointer to next linked list

can recursively go down and call function by reassigning current

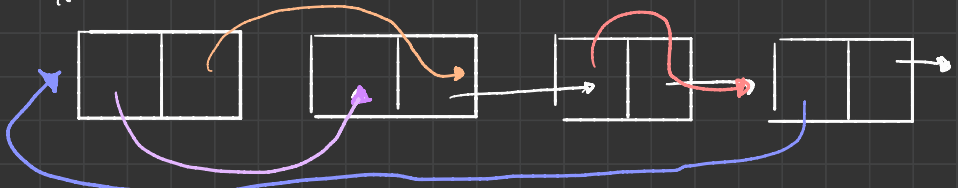
**Recursive**

```
def traversing (ll):
    if ll is Link.empty or ll.rest is Link.empty:
        return
    traversing (ll.rest)
```

**Iterative**

```
def traversing (ll):
    while (ll is not Link.empty and ll.rest is not Link.empty):
        ll = ll.rest
```

link:



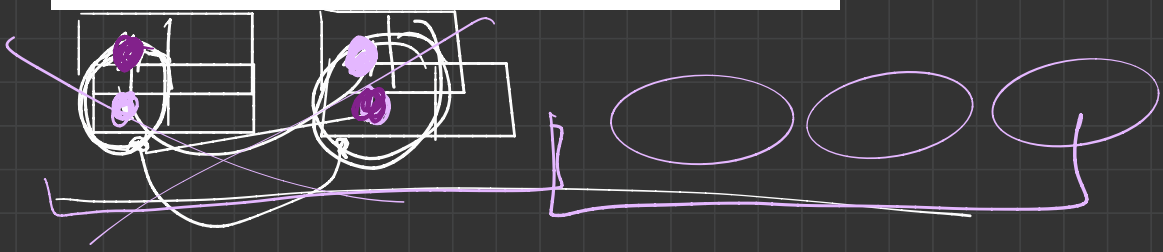
- link.first = link.rest.first
- link.rest = link.rest.rest
- link.rest.first = link.rest.rest
- link.rest.first.first = link

Hilfinger

# Question 1:

2.3 **Tutorial:** Write a recursive function `flip_two` that takes as input a linked list `lnk` and mutates `lnk` so that every pair is flipped.

```
def flip_two(lnk):
    """
    >>> one_lnk = Link(1)
    >>> flip_two(one_lnk)
    >>> one_lnk
    Link(1)
    >>> lnk = Link(1, Link(2, Link(3, Link(4, Link(5)))))
    >>> flip_two(lnk)
    >>> lnk
    Link(2, Link(1, Link(4, Link(3, Link(5)))))
    """
```



Recursive:

```
def flip_two (lnk):
    if lnk is Link.Empty or lnk.rest is Link.Empty:
        return
    lnk.first, lnk.rest.first = lnk.rest.first, lnk.first
    return flip_two(lnk.rest.rest)
```

Iterative:

```
def flip_two (lnk):
    while( lnk is not Link.Empty or lnk.rest is not Link.Empty):
        lnk.first, lnk.rest.first = lnk.rest first  link.first
        lnk : lnk.rest.rest
```

Question 2:

2.4 **Tutorial:** Implement `filter_link`, which takes in a linked list `link` and a function `f` and returns a generator which yields the values of `link` for which `f` returns `True`.

yield!

Try to implement this both using a while loop and without using any form of iteration.

```
def filter_link(link, f):
    """
    >>> link = Link(1, Link(2, Link(3)))
    >>> g = filter_link(link, lambda x: x % 2 == 0)
    >>> next(g)
    2
    >>> next(g)
    StopIteration
    >>> list(filter_link(link, lambda x: x % 2 != 0))
    [1, 3]
    """

    while link is not Link.empty:
        if f(link.first):
            yield link.first
        link = link.rest
```