CS61A NOTE10 Linked Lists

Linked Lists

There are many different implementations of sequences in Python. Today, we'll explore the linked list implementation.链表的实现

A linked list is either an empty linked list, or a Link object containing a `first` value and the `rest` of the linked list.链表要么是一个空的链表，要么是一个包含 `first` 值和链表 `rest` 的 Link 对象。

To check if a linked list is an empty linked list, compare it against the class attribute `Link.empty`:要检查一个链表是否是一个空的链表，可以将它与类属性 `Link.empty` 比较

纯文本

```
if link is Link.empty:#是空,Link.empty的l是大写L
    print('This linked list is empty!')
else:
    print('This linked list is not empty!')
```

纯文本

```
#链接类的实现
class Link:
    """A linked list."""
    empty = ()

    def __init__(self, first, rest=empty):
        assert rest is Link.empty or isinstance(rest, Link) #为什么这一行
        self.first = first
        self.rest = rest

    def __repr__(self):
        if self.rest:
            rest_repr = ', ' + repr(self.rest)
        else:
            rest_repr = ''
        return 'Link(' + repr(self.first) + rest_repr + ')'

    def __str__(self):
        string = '<'
        while self.rest is not Link.empty:
            string += str(self.first) + ' '
            self = self.rest
        return string + str(self.first) + '>'
```

disc Q1: WWPD: Linked Lists

```
纯文本
>>> link = Link(1, Link(2, Link(3)))
>>> link.first
1
>>> link.rest.first
2
>>> link.rest.rest.rest is Link.empty
True
>>> link.rest = link.rest.rest 前者变为后者
>>> link.rest.first
3

>>> link = Link(1)
>>> link.rest = link
>>> link.rest.rest.rest.rest.first #无限loop
1

>>> link = Link(2, Link(3, Link(4)))
>>> link2 = Link(1, link)
>>> link2.first
1
>>> link2.rest.first
2

>>> link = Link(1000, 2000)
Error (second argument of Link constructor must be a Link instance)

>>> link = Link(1000, Link())
Error (first argument of Link constructor must be specified)必须指定链接构造函
```

```
>>> link = Link(Link("Hello"), Link(2))
>>> link.first
Link("Hello")
>>> link.first.rest is Link.Empty
True
>>> link.rest is Link.Empty
False
```

lab10 Q7: Insert

Implement a function `insert` that takes a `Link`, a `value`, and an `index`, and inserts the `value` into the `Link` at the given `index`. You can assume the linked list already has at least one element. Do not return anything -- `insert` should mutate the linked list.link list 的 index 处插入 value

> **Note**: If the index is out of bounds, you should raise an `IndexError` with: raise IndexError('Out of bounds!')

纯文本

```
def insert(link, value, index):
    """Insert a value into a Link at the given index.

    >>> link = Link(1, Link(2, Link(3)))
    >>> print(link)
    <1 2 3>
    >>> other_link = link
    >>> insert(link, 9001, 0)
    >>> print(link)
    <9001 1 2 3>
    >>> link is other_link # Make sure you are using mutation! Don't create
    True
    >>> insert(link, 100, 2)
    >>> print(link)
    <9001 1 100 2 3>
    >>> insert(link, 4, 5)
    Traceback (most recent call last):
        ...
    IndexError: Out of bounds!
    """
    "*** YOUR CODE HERE ***"
    if index==0:
        link.rest=Link(link.first,link.rest)
        link.first = value
        #c=link?
        #link=Link[value,c]？
    elif link.rest is Link.empty:
        raise IndexError('Out of bounds!')
    else:
        insert(link.rest, value, index-1)
```

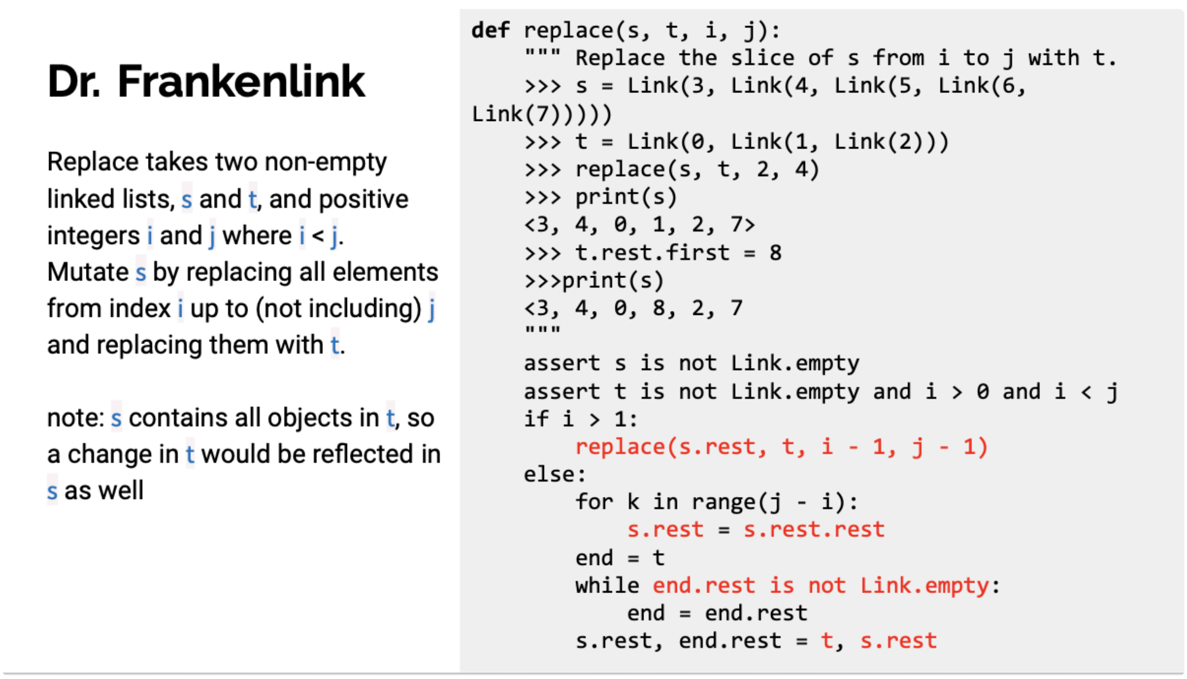
此题不 return，插一个值（不空），直接 mutable，最后直接操作

Insert At

Write a recursive function insert_at that takes as input three parameters, two linked lists, s and x, and an index index. insert_at should return a new Linked list with x insert at index of s . Assume that index will be a non-negative integer.

```
def insert_at(s, x, index):
    """
    >>> insert = Link(3, Link(4))
    >>> original = Link(1, Link(2, Link(5)))
    >>> insert_at(original, insert, 2)
    Link(1, Link(2, Link(3, Link(4, Link(5)))))
    >>> start = Link(1)
    >>> insert_at(original, start, 0)
    Link(1, Link(1, Link(2, Link(5))))
    """
    if s is Link.empty and x is Link.empty:
        return Link.empty
    if x is not Link.empty and index == 0:
        return Link(x.first, insert_at(s, x.rest, 0))
    return Link(s.first, insert_at(s.rest, x, index -
1))
```

此题 return，插一个 link list（可空），最后要写好 Link(,)，此外 Link(x.first,insert_at(s,x.rest,0))也很聪明



Dr. Frankenlink

Replace takes two non-empty linked lists, s and t, and positive integers i and j where i < j. Mutate s by replacing all elements from index i up to (not including) j and replacing them with t.

note: s contains all objects in t, so a change in t would be reflected in s as well

```
def replace(s, t, i, j):
    """ Replace the slice of s from i to j with t.
    >>> s = Link(3, Link(4, Link(5, Link(6,
Link(7)))))
    >>> t = Link(0, Link(1, Link(2)))
    >>> replace(s, t, 2, 4)
    >>> print(s)
    <3, 4, 0, 1, 2, 7>
    >>> t.rest.first = 8
    >>>print(s)
    <3, 4, 0, 8, 2, 7
    """
    assert s is not Link.empty
    assert t is not Link.empty and i > 0 and i < j
    if i > 1:
        replace(s.rest, t, i - 1, j - 1)
    else:
        for k in range(j - i):
            s.rest = s.rest.rest
        end = t
        while end.rest is not Link.empty:
            end = end.rest
        s.rest, end.rest = t, s.rest
```

Link list 的=是左变成右，此题 mutate，不 return,

lab10 Q8: Deep Linked List Length

A linked list that contains one or more linked lists as elements is called a *deep* linked list. Write a function `deep_len` that takes in a (possibly deep) linked list and returns the *deep length* of that linked list. The deep length of a linked list is the total number of non-link elements in the list, as well as the total number of elements contained in all contained lists. 所有 list 元素个数 See the function's doctests for examples of the deep length of linked lists.

Hint: Use `isinstance` to check if something is an instance of an object. 我记得 isinstance(s,link)

纯文本

```
def deep_len(lnk):
    """ Returns the deep length of a possibly deep linked list.

    >>> deep_len(Link(1, Link(2, Link(3))))
    3
    >>> deep_len(Link(Link(1, Link(2)), Link(3, Link(4))))
    4
    >>> levels = Link(Link(Link(1, Link(2)),Link(3)), Link(Link(4), Link(5)
    >>> print(levels)
    <<<1 2> 3> <4> 5>
    >>> deep_len(levels)
    5
    """
    if lnk is Link.empty:
        return 0
    elif not isinstance(lnk,Link): rest不能空？
        return 1
    else:
        return deep_len(lnk.rest)+deep_len(lnk.first)
```

lab10 Q9: Linked Lists as Strings

Kevin and Jerry like different ways of displaying the linked list structure in Python. While Kevin likes box and pointer diagrams, Jerry prefers a more futuristic way. Write a function `make_to_string` that returns a function（HOF，提示 helper） that converts the linked list to a string in their preferred style.

Hint: You can convert numbers to strings using the `str` function, and you can combine strings together using `+`.

纯文本

```
>>> str(4)
'4'
>>> 'cs ' + str(61) + 'a'
'cs 61a'
```

纯文本

```
def make_to_string(front, mid, back, empty_repr):
    """ Returns a function that turns linked lists to strings.

    >>> kevins_to_string = make_to_string("[", "|-]-->", "", "[]")
    >>> jerrys_to_string = make_to_string("(", " . ", ")", "()")
    >>> lst = Link(1, Link(2, Link(3, Link(4))))
    >>> kevins_to_string(lst)
    '[1|-]-->[2|-]-->[3|-]-->[4|-]-->[]'
    >>> kevins_to_string(Link.empty)
    '[]'
    >>> jerrys_to_string(lst)
    '(1 . (2 . (3 . (4 . ()))))'
    >>> jerrys_to_string(Link.empty)
    '()'
    """
    def printer(lnk): #前面没有lnk所以考虑helper(lnk)
        if lnk is Link.empty:
            return empty_repr
        else:
            return front+str(lnk.first)+mid+printer(lnk.rest)+back
    return printer
```

disc Q2: Convert Link 转换 Link

Write a function `convert_link` that takes in a linked list and returns the sequence as a Python list. You may assume that the input list is shallow; that is none of the elements is another linked list.编写函数 convert_link，它接收一个链表，并返回序列（as Python 列表）。可以假设输入的列表是浅层的，也就是说没有一个元素是另一个链接列表。

Try to find both an iterative and recursive solution for this problem!

纯文本

```
def convert_link(link): #递归思路最简单，不然就while循环
    """Takes a linked list and returns a Python list with the same elements

    >>> link = Link(1, Link(2, Link(3, Link(4))))
    >>> convert_link(link)
    [1, 2, 3, 4]
    >>> convert_link(Link.empty) #若是Link.empty返回[]
    []
    """
    "*** YOUR CODE HERE ***"
    #法一
    if link is Link.empty:
        return []
    return [link.first]+convert_link(link.rest) #记得link.first加[]

    #法二
    result=[]
    while link is not Link.empty:
        result.append(link.first)
        link=link.rest
    return result
```

```
#法三：挑战使用type,其实这是最准确的写法
if link is Link.empty:
    return []
if type(link.first) == Link:
    return [convert_link_challenge(link.first)] + convert_link_challeng
return [link.first] + convert_link_challenge(link.rest)
```

disc Q3: Duplicate Link

Write a function `duplicate_link` that takes in a linked list `link` and a `value`. `duplicate_link` will mutate `link` such that if there is a linked list node that has a `first` equal to `value`, that node will be duplicated 复制. **Note that** you should be mutating the original link list `link`; you will need to create new `Link`s, but you should not be returning a new linked list.编写函数 duplicate_link，它接收一个 linked list 和一个值。 duplicate_link 将改变链接，使得如果有一个链表节点的 first 等于值，该节点将被复制。注意，你应该突变原始 link list 的 link；创建新的 link，但不应返回一个新的 linked list。

纯文本

```
def duplicate_link(link, val):
    """Mutates `link` such that if there is a linked list
    node that has a first equal to value, that node will
    be duplicated. Note that you should be mutating the
    original link list.突变`link`，使得若一个链表节点的first等于值，
    该节点将被被复制。应该突变的是原始链接列表
    想法：递归，先if空return      (⚠️ 啥也没有啥也不写) elif first是val复制 else r

    >>> x = Link(5, Link(4, Link(3)))
    >>> duplicate_link(x, 5) #值是5，复制5
    >>> x
    Link(5, Link(5, Link(4, Link(3))))
    >>> y = Link(2, Link(4, Link(6, Link(8))))
    >>> duplicate_link(y, 10)
    >>> y
    Link(2, Link(4, Link(6, Link(8))))
    >>> z = Link(1, Link(2, (Link(2, Link(3)))))
    >>> duplicate_link(z, 2) #ensures that back to back links with val are
    >>> z
    Link(1, Link(2, Link(2, Link(2, Link(2, Link(3))))))#两个2各复制
    """
    "*** YOUR CODE HERE ***"
    if link is Link.empty: #注意大写
        return      #⚠
    elif link.first == val:
        c=link.rest      #注意设出c，不然可变性且infinite loop
        link.rest=Link[val,c]   #改变link.rest，Link新建，跳出循环，变长很危险，
        #注意是link不是list因此Link[val,c]而非[val]+c
        duplicate_link(link.rest.rest,val) #题目要求+可变=不用写reture，改变后
    else:
        duplicate_link(link.rest, val) #直接改变后面
```

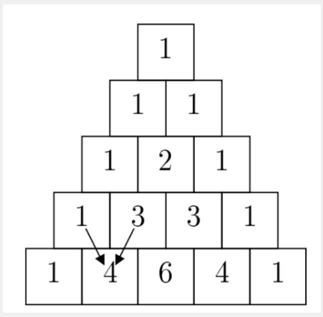
两道 mutate 考题

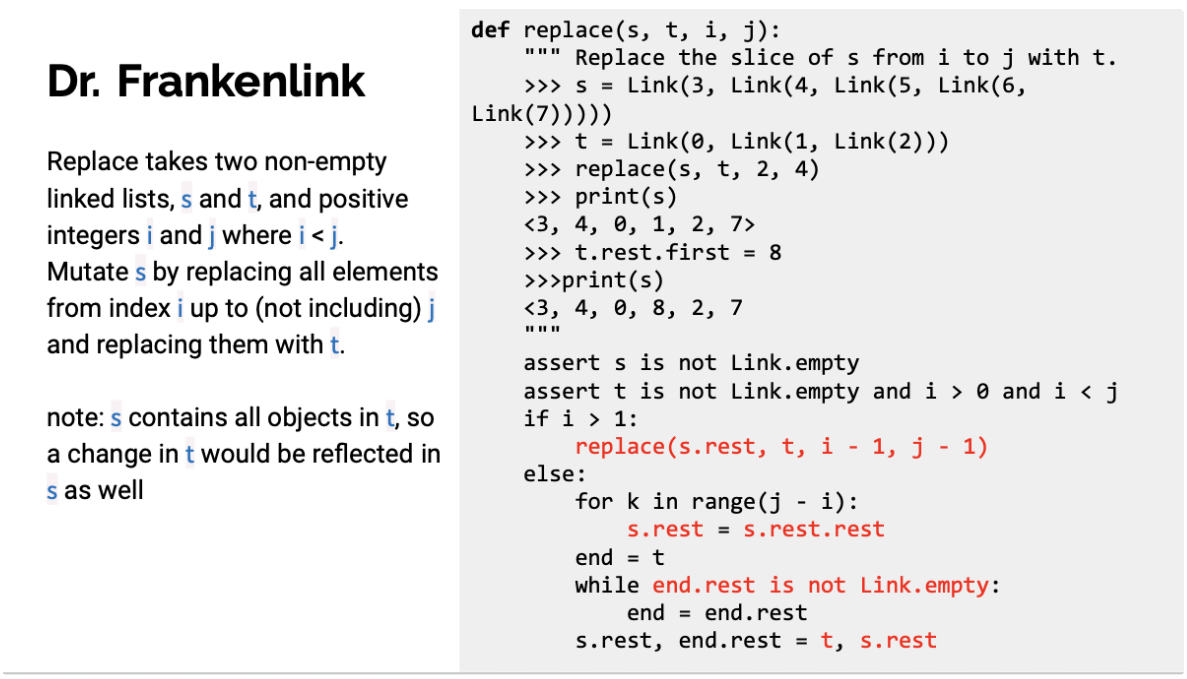
设出 start 和 s 会因为只想对 first/rest 操作，需要很短的，若每次 a=a.rest,就可以对 rest 操作，不过 start 跟着更新却不删之前的信息，最后可以返回 start

Pascal Row Sol

Given a linked list s that represents a row in Pascal's triangle, return a linked list that will represent the row below it.

```
def pascal_row(s):
    """
    >>> a = Link.empty
    >>> for _ in range(5):
    ...     a = pascal_row(a)
    ...     print(a)
    <1>
    <1 1>
    <1 2 1>
    <1 3 3 1>
    <1 4 6 4 1>
    """
    if s is Link.empty:
        return Link(1)
    start = Link(1)
    last, current = start, s
    while current.rest is not Link.empty:
        last.rest = Link(current.first +
current.rest.first)
        last, current = last.rest, current.rest
    last.rest = Link(1)
    return start
```



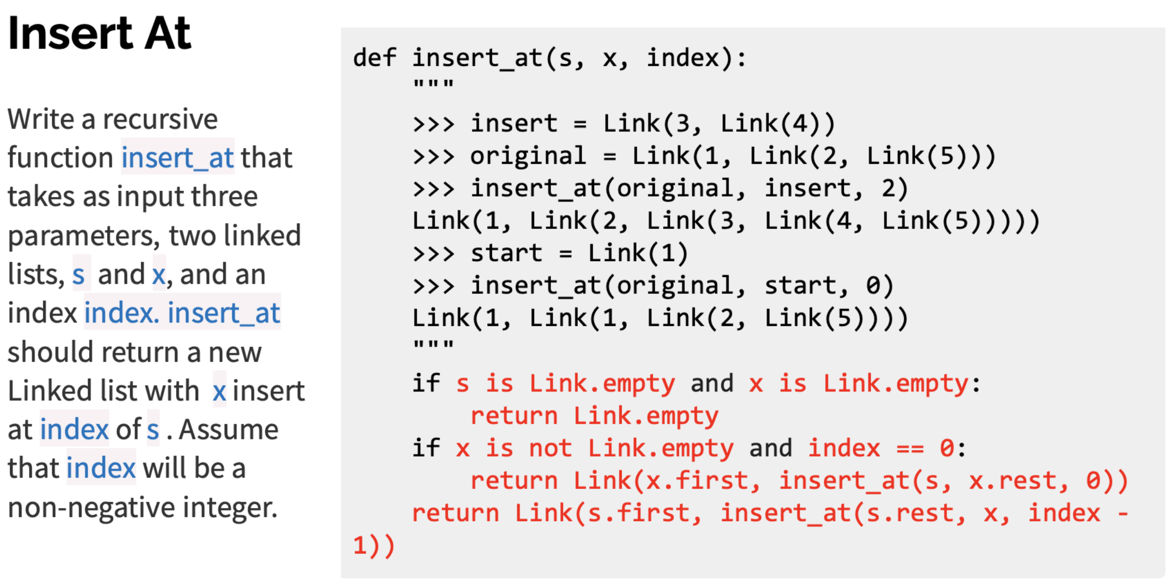

Dr. Frankenlink

Replace takes two non-empty linked lists, s and t, and positive integers i and j where i < j. Mutate s by replacing all elements from index i up to (not including) j and replacing them with t.

note: s contains all objects in t, so a change in t would be reflected in s as well

```
def replace(s, t, i, j):
    """ Replace the slice of s from i to j with t.
    >>> s = Link(3, Link(4, Link(5, Link(6,
Link(7)))))
    >>> t = Link(0, Link(1, Link(2)))
    >>> replace(s, t, 2, 4)
    >>> print(s)
    <3, 4, 0, 1, 2, 7>
    >>> t.rest.first = 8
    >>>print(s)
    <3, 4, 0, 8, 2, 7
    """
    assert s is not Link.empty
    assert t is not Link.empty and i > 0 and i < j
    if i > 1:
        replace(s.rest, t, i - 1, j - 1)
    else:
        for k in range(j - i):
            s.rest = s.rest.rest
        end = t
        while end.rest is not Link.empty:
            end = end.rest
        s.rest, end.rest = t, s.rest
```

Link list 的=是左变成右，此题 mutate，不 return,


Insert At

Write a recursive function insert_at that takes as input three parameters, two linked lists, s and x, and an index index. insert_at should return a new Linked list with x insert at index of s . Assume that index will be a non-negative integer.

```
def insert_at(s, x, index):
    """
    >>> insert = Link(3, Link(4))
    >>> original = Link(1, Link(2, Link(5)))
    >>> insert_at(original, insert, 2)
    Link(1, Link(2, Link(3, Link(4, Link(5)))))
    >>> start = Link(1)
    >>> insert_at(original, start, 0)
    Link(1, Link(1, Link(2, Link(5))))
    """
    if s is Link.empty and x is Link.empty:
        return Link.empty
    if x is not Link.empty and index == 0:
        return Link(x.first, insert_at(s, x.rest, 0))
    return Link(s.first, insert_at(s.rest, x, index -
1))
```

上题更简单版本，此题 return，要写好 Link(,)

disc Q4: Multiply Links

Write a function that takes in a Python list of linked lists and multiplies them element-wise. It should return a new linked list.编写一个函数传入一个 关于 link list 的 list，按元素相乘。它应该返回一个新的 link list。

If not all of the `Link` objects are of equal length, return a linked list whose length is that of the shortest linked list given. You may assume the `Link` objects are shallow linked lists, and that `lst_of_lnks` contains at least one linked list.
如果不是所有的 link 对象都是等长的，则返回一个长度为最短的 link list。假设 link 对象是浅层 link list，并且 lst_of_lnks 至少包含一个 link list。

纯文本

```
def multiply_lnks(lst_of_lnks):
    """
    >>> a = Link(2, Link(3, Link(5)))
    >>> b = Link(6, Link(4, Link(2)))
    >>> c = Link(4, Link(1, Link(0, Link(2))))
    >>> p = multiply_lnks([a, b, c])
    >>> p.first
    48
    >>> p.rest.first
    12
    >>> p.rest.rest.rest is Link.empty
    True             #不同位某个没了return Link.empty
    """
    # Implementation Note: you might not need all lines in this skeleton co
    #法一 递归较为直观
    mul = 1
    for i in lst_of_lnks: #对abc取
        if i is Link.empty:  #不用==/=，永is Link.empty:
            return Link.empty #不同位某个没了return Link.empty
        mul*=i.first
    lst_of_lnks_rest=[i.rest for i in lst_of_lnks] #外带[],用_rest不引起误解
    #注意[i.rest for i in lst_of_lnks]写法for i in一个一个取出，abc切少
    return Link(mul,multiply_lnks(lst_of_lnks_rest))

    #法二 改造为迭代
    mul = 1
    for i in lst_of_lnks:
        if i is Link.empty:
            ~~return Link.empty~~
            return     #mutate里返回用
        mul*=i.first
    lst_of_lnks_rest=[i.rest for i in lst_of_lnks]
    ~~return Link(mul,multiply_lnks(lst_of_lnks_rest))~~
    yield mul
    yield from multiply_lnks(lst_of_lnks_rest)
```

disc Q5: Flip Two

Write a recursive function `flip_two` that takes as input a linked list `s` and mutates `s` so that every pair is flipped.编写递归函数 flip_two，输入 linked list s，变 s 使每一对被翻转。

纯文本

```
def flip_two(s):
    """
    >>> 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)))))
    """
    # For an extra challenge, try writing out an iterative approach as well
    "*** YOUR CODE HERE ***"
    #法一：recursive递归
    if s.rest is Link.empty or s is Link.empty:
        return     #mutate里空用
    s.first, s.rest.first=s.rest.first, s.first
    flip_two(s.rest.rest)   #这里mutable改变自己即可

    #法二：Iterative迭代
    while s is not Link.empty and s.rest is not Link.empty:
        s.first, s.rest.first=s.rest.first, s.first
        s=s.rest.rest #变短是可以的
```

hw7 Q1: Store Digits

Write a function `store_digits` that takes in an integer `n` and returns a linked list where each element of the list is a digit of `n`.

纯文本

```
def store_digits(n):
    """Stores the digits of a positive number n in a linked list.

    >>> s = store_digits(1)
    >>> s
    Link(1)
    >>> store_digits(2345)
    Link(2, Link(3, Link(4, Link(5))))
    >>> store_digits(876)
    Link(8, Link(7, Link(6)))
    >>> store_digits(2450)
    Link(2, Link(4, Link(5, Link(0))))
    >>> # a check for restricted functions
    >>> import inspect, re
    >>> cleaned = re.sub(r"#.*\\n", '', re.sub(r'"{3}[\s\S]*?"{3}', '', ins
    >>> print("Do not use str or reversed!") if any([r in cleaned for r in
    >>> link1 = Link(3, Link(Link(4), Link(5, Link(6))))
    """
    "*** YOUR CODE HERE ***"
    result=Link.empty
    while n>0:
        result=Link(n%10,result) #从后往前常简单很多，直接嵌套即可
        n//=10
    return result
```

hw Q2: Mutable Mapping 一个数字并不是 link，有只对 link 操作的大函数，也有对数操作的 func

Implement `deep_map_mut(func, link)`, which applies a function `func` onto all elements in the given linked list `lnk`. If an element is itself a linked list, apply `func` to each of its elements, and so on. 若元素本身是 linked list，应用 func 对每个元素

Your implementation should mutate the original linked list. Do not create any new linked lists.

Hint: The built-in `isinstance`(s,格式) function may be useful.

纯文本

```
>>> s = Link(1, Link(2, Link(3, Link(4))))
>>> isinstance(s, Link)
```

```
True
>>> isinstance(s, int)
False
```

纯文本

```
def deep_map_mut(func, lnk):
    """Mutates a deep link lnk by replacing each item found with the
    result of calling func on the item.  Does NOT create new Links (so
    no use of Link's constructor).

    Does not return the modified Link object.

    >>> link1 = Link(3, Link(Link(4), Link(5, Link(6))))
    >>> # Disallow the use of making new Links before calling deep_map_mut
    >>> Link.__init__, hold = lambda *args: print("Do not create any new Li
    >>> try:
    ...     deep_map_mut(lambda x: x * x, link1)
    ... finally:
    ...     Link.__init__ = hold
    >>> print(link1)
    <9 <16> 25 36>
    """
    "*** YOUR CODE HERE ***"
    if lnk is Link.empty:
        return
    elif isinstance(lnk.first,Link): #是否是link
        deep_map_mut(func, lnk.first)
    else:
        lnk.first=func(lnk.first)
    deep_map_mut(func, lnk.rest)
```

hw Q3: Two List 每层不一样考虑 helper

Implement a function `two_list` that takes in two lists and returns a linked list. The first list contains the values that we want to put in the linked list 值, and the second list contains the number of each corresponding value 个数. Assume both lists are the same size and have a length of 1 or greater. Assume all elements in the second list are greater than 0.

纯文本

```
def two_list(vals, counts):
    """
    Returns a linked list according to the two lists that were passed in. A
    vals and counts are the same size. Elements in vals represent the value
    corresponding element in counts represents the number of this value des
    final linked list. Assume all elements in counts are greater than 0. As
    lists have at least one element.
    >>> a = [1, 3]
    >>> b = [1, 1]
    >>> c = two_list(a, b)
    >>> c
    Link(1, Link(3))
    >>> a = [1, 3, 2] #放进值
    >>> b = [2, 2, 1] #放进个数
    >>> c = two_list(a, b)
    >>> c
    Link(1, Link(1, Link(3, Link(3, Link(2)))))
    """
    "*** YOUR CODE HERE ***"
    #法一
    result=Link.empty
    for index in range(len(vals)-1,-1,-1):
        k=vals[index]
        for _ in range(counts[index]):
            result=Link(vals[index],result) #倒着从后往前好写Link list
    return result

    #法二
    def helper(count, index):
        if count == 0:
            if index + 1 == len(vals):
                return Link.empty
            return Link(vals[index + 1], helper(counts[index + 1] - 1, inde
        return Link(vals[index], helper(count - 1, index))
    return helper(counts[0], 0)
```