Fall 2020 Final Exam Solutions

CS 88

WWPD

Question 1: Create a linked list called `foo` such that the following statement returns True.

```
>>> len(foo.first) == len(foo.rest.first) and foo.first[0] == "t" and
foo.rest.rest.rest == Link.empty
True
```

```
One example:
foo = Link("turtle", Link("lizard", Link("frog")))

Explanation:
The data type of the value in lnk.first and lnk.rest.first should be
equal in length.

The first element of foo.first should be equal to "t".

Also, the linked list should only be 3 elements long, since
foo.rest.rest.rest is Link.empty
```

Question 2: Fill in the values for `arg1` and `arg2`

```
class Candy:
      def __init__(self, color, price):
            self.color = color
            self.price = price

def shop(candies, colors):
      total = 0
      for candy in candies:
            if candy.color in colors:
                  print("I like " + candy.color)
                  total += candy.price
            else:
                  print("I don't like " + candy.color)
       return total

>>> total = shop(arg1, arg2)
I don't like blue
I like red
```

```
I like green
I like red
>>> print(total)
42
```

One example:
arg1 = [Candy("blue", 10), Candy("red", 20), Candy("green", 2), Candy("red", 20)]
arg2 = ["red", "green"]

Explanation:
arg1 needs to be a list of 4 Candy instances with the value of color for each one, respectively, being blue, red, green, and red in that order. Additionally, the last 3 candies' prices should sum up to 42.

arg2 should be an iterable (list, tuple, etc.) containing the strings "red" and "green", and should not contain "blue" because one of the print statements says "I don't like blue", which means blue is not in the iterable.

Question 3: Explain in one sentence what the following function does if given an instance `t` of the Tree class. The Tree class is also included for reference.

```
class Tree:
    def __init__(self, entry, branches=[]):
        self.entry = entry
        for b in branches:
            assert isinstance(b, Tree)
        self.branches = branches

    def is_leaf(self):
        return not self.branches

def mystery(t):
    def strange(i, z):
        m  = 0
        if i % 2 == 0:
            m += z.entry
        return m + sum([strange(i+1, b) for b in z.branches])
    return strange(0, t)
```

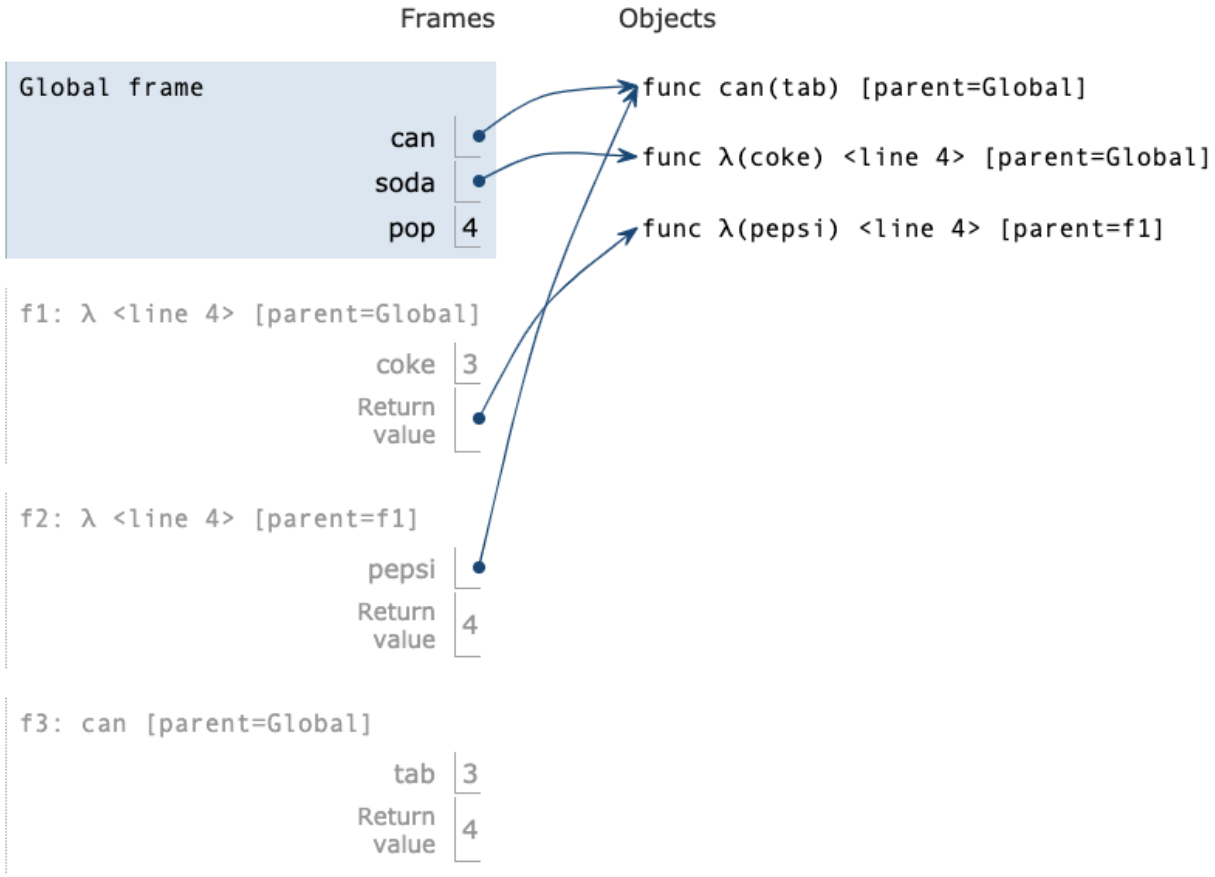
Adds up the values of entries of Tree nodes at even depths in the tree t.

Reverse Env. Diagram

1.

```
def can(tab):
    return _________

soda = lambda coke: lambda pepsi: ___________
pop = soda(____)(_____)
```




```
def can(tab):
    return tab + 1

soda = lambda coke: lambda pepsi: pepsi(coke)
pop = soda(3)(can)
```

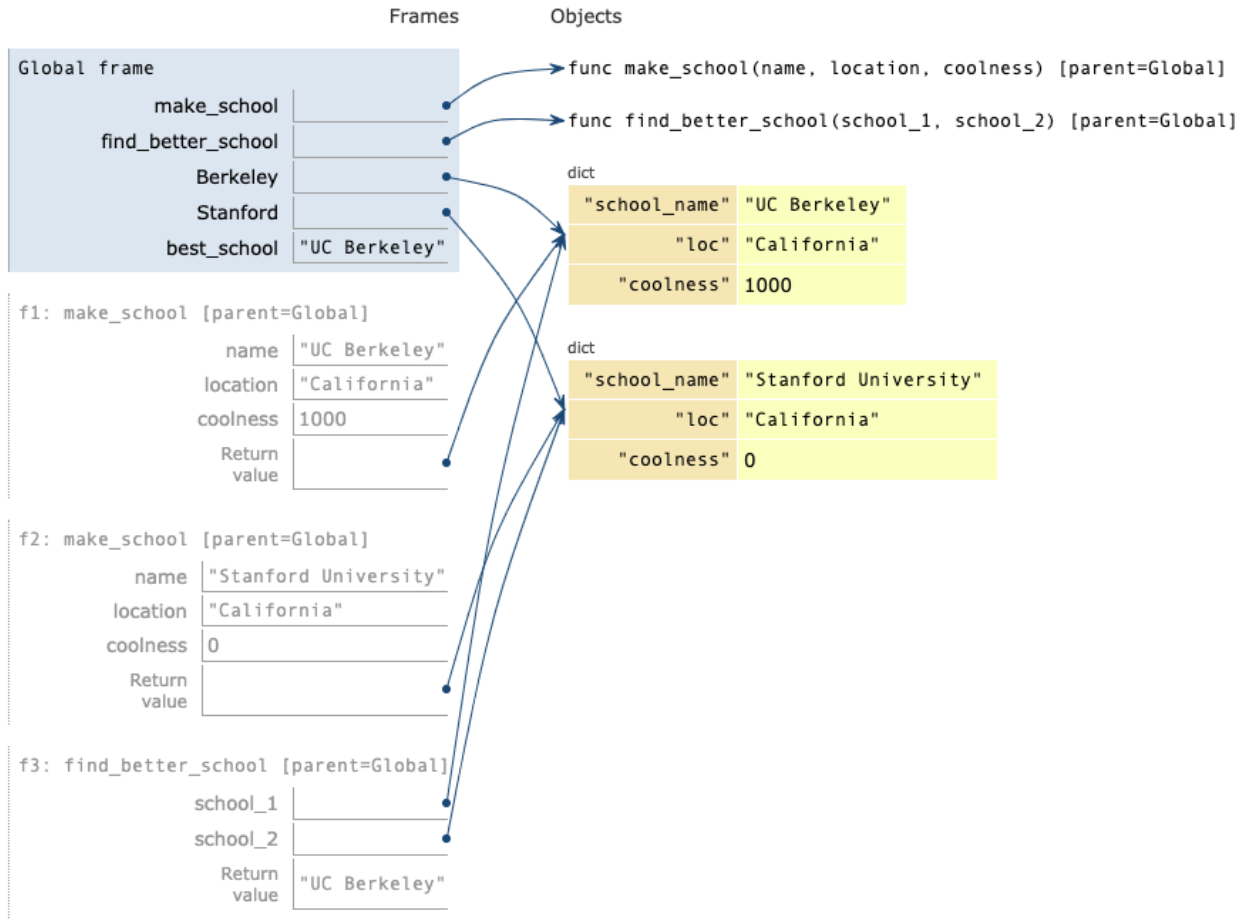
2.

```
def make_school(name, location, coolness):
    return ____________________________________

def find_better_school(school_1, school_2):
    if school_1["coolness"] > school_2["coolness"]:
        return ____________________
    else:
        return ____________________

Berkeley = make_school("UC Berkeley", ___________, ________)
Stanford = make_school("Stanford University", __________, ________)

best_school = find_better_school(Berkeley, Stanford)
```




```
def make_school(name, location, coolness):
    return {"school_name": name, "loc": location, "coolness":
coolness}
```

```
def find_better_school(school_1, school_2):
    if school_1["coolness"] > school_2["coolness"]:
        return school_1["school_name"]
    else:
        return school_2["school_name"]

Berkeley = make_school("UC Berkeley", "California", 1000)
Stanford = make_school("Stanford University", "California", 0)

best_school = find_better_school(Berkeley, Stanford)
```

The Disney LINKup

```
class Character:
    def __init__(self, name, charm):
        self.name = name
        self.charm = charm

    def __repr__(self):
        return 'Character({0}, {1})'.format(self.name, self.charm)
```

Your favorite Disney characters are preparing for an epic face-off, but you need to pair up characters into teams before the games begin! For this problem, you can imagine that the characters are standing in a line (represented as a linked list) .

Question: Given a linked list of characters, return a list of pairs, where each pair is a two element list containing both characters on that team. Pairs should be created from left to right. However, if a Character is part of a team where their teammate has a greater charm factor than them, this Character's charm factor increases to meet their teammate's!

Before you return the final teams, you should also print whether or not all the teams are full, i.e. if every team has two members. See the doctests on the message to print in either case!

```
def make_pairs(lnk):
    """
    >>> belle = Character("Belle", 35)
    >>> simba = Character("Simba", 45)
    >>> flynn = Character("Flynn", 30)
    >>> teams1 = make_pairs(Link(belle, Link(simba, Link(flynn))))
    Not all teams are full!
    >>> teams1
    [[Character(Belle, 45), Character(Simba, 45)], [Character(Flynn, 30)]]
    >>> mulan = Character("Mulan", 40)
    >>> aladdin = Character("Aladdin", 30)
    >>> olaf = Character("Olaf", 50)
    >>> jafar = Character("Jafar", 5)
    >>> teams2 = make_pairs(Link(mulan, Link(aladdin, Link(olaf, Link(jafar)))))
    All teams full!
    >>> teams2
    [[Character(Mulan, 40), Character(Aladdin, 40)], [Character(Olaf, 50),
Character(Jafar, 50)]]
    """

    if lnk is Link.empty:
        print("All teams full!")
        return []
    elif lnk.rest is Link.empty:
        print("Not all teams are full!")
        return [[lnk.first]]
    else:
        lnk.first.charm = max(lnk.first.charm, lnk.rest.first.charm)
        lnk.rest.first.charm = lnk.first.charm

        return [[lnk.first, lnk.rest.first]] + make_pairs(lnk.rest.rest)
```

HOFfinetly a safe passcode

Fill out the blanks below to encode a passcode by adding numbers to each digit. The input of the encoder function is a list of single digit numbers to use to encode a certain passcode. The encoder function returns a function that will take in the passcode to encode.

The way a passcode will be encoded is by incrementing each digit in the passcode by a number from the variable *lst* passed into the encoder function. It uses a number from *lst* one at a time to increment each digit in passcode, cycling around "lst" if necessary. See the doctests for some examples

```
def encoder(lst):
      """
      Returns a function that will use the input list to encode an
      input passcode
      >>>increment_password = encoder([3])
      >>>increment_password("111")
      "444"
      >>>encoder([1,2,3])("234")
      "357"
      >>>encoder([5,2])("1111")
      "6363"
      """
    def increment(passcode):
        new_code = "";
        for i in __________________:
            curr = ___(____________) + lst[__________]
            new_code = ____________________
        return ______________________
    return _________________
```

Choose the runtime of the **encoder** function:
   a) O(1) b) O(n) c) O(n^2) d) O(2^n)

Choose the runtime of the **increment** function:
   a) O(1) b) O(n) c) O(n^2) d) O(2^n)

```
def encoder(lst):
    def increment(passcode):
        new_code = "";
        for i in range(len(passcode)):
            curr = int(passcode[i]) + lst[i%len(lst)]
            new_code = new_code + str(curr)
        return new_code
    return increment
```

Select the runtime of the encoder function where n is the length of *lst*: a) O(1)

Select the runtime of the increment function where n is the length of *passcode*: b) O(n)

The Silent TREEtment

In this problem, every node in a given Tree has a string as its entry. Your job is to write a function that, given a Tree T, will return its "message" which is defined as the messages of all of T's branches (in order) followed by T.entry. If T is a leaf node, then T's message is T.entry.

However, some trees are not being cooperative and giving you the silent treatment - how rude! Any subtree that has an empty string as a message is giving you the silent treatment. If any of the subtrees in a tree T give you the silent treatment, then T's **entire message** is an empty string.

Assume that you will be passed a valid tree.

```
>>> t = Tree("abc", [Tree("def", [Tree("ghi")]), Tree("jkl")])
>>> getTreeMsg(t)
"ghidefjklabc"
>>> t = Tree("abc", [Tree("def", [Tree("")]), Tree("jkl")])
>>> getTreeMsg(t)
""
```

Skeleton:

```
def getTreeMsg(tree):
    if ____________________:
        return ____________________
    msg = ______________________________
    for ___________________________________:
        ___________________________________
        if ___________________________________:
            ___________________________________
        msg += ___________________________________
    return ___________________________________
```

Answer:

```
def getTreeMsg(tree):
    if tree.entry == "":
        return ""
    msg = ""
    for b in tree.branches:
        branchMsg = getTreeMsg(b)
        if branchMsg == "":
            return ""
        msg += branchMsg
    return msg + tree.entry
```

SQL for Sequels

```
CREATE TABLE books AS
     SELECT “Sorcerer’s Stone” as name, 1 as number, “Harry Potter”
as series UNION
     SELECT “Chamber of Secrets”, 2, “Harry Potter” UNION
     SELECT “Catching Fire”, 2, “Hunger Games” UNION
     SELECT “Mockingjay”, 3, “Hunger Games” UNION
     SELECT “Deathly Hallows”, 7, “Harry Potter” UNION
     SELECT “Cat in the Hat”, 1, “N/A” UNION
     SELECT “Tale of Two Cities”, 1, “N/A” UNION
     SELECT “Divergent”, 1, “Divergent” UNION
     SELECT “Insurgent”, 2, “Divergent” UNION
     SELECT “Mark of Athena”, 3, “Heroes of Olympus” UNION
     SELECT “House of Hades”, 4, “Heroes of Olympus” UNION
     SELECT “Son of Neptune”, 2, “Heoes of Olympus”;

CREATE TABLE series_authors AS
     SELECT “Harry Potter” as series, “JK Rowling” as author, “UK”
as country, “Yate” as city UNION
     SELECT “Hunger Games”, “Suzanne Collins”, “USA”, “Hartford”
UNION
     SELECT “Divergent”, “Veronica Roth”, “USA”, “New York” UNION
     SELECT “Heroes of Olympus”, “Rick Riordan”, “USA”, “San
Antonio”;
```

Use the above tables to write queries below. **Keep in mind that not all blanks need to be filled in for each query to get the query correct.**

1. Write a SELECT statement which would output series with more than with more than two books in the books table

```
Output:
Harry Potter
Heroes of Olympus

SELECT _____________________________________
FROM _____________________________________
WHERE _____________________________________
GROUP BY _____________________________________
HAVING _____________________________________
ORDER BY _____________________________________
```

Solution:

```
SELECT books.series FROM books GROUP BY series HAVING count(*) > 2
```

2. Given the tables above, write a SELECT statement which has each book name with the name of its **direct** sequel, meaning the book that comes right after it (ex. Book 5 is not Book 2's direct sequel but Book 3 is Book 2's direct sequel). The prequel should be listed before the sequel.

Output:

```
Sorcerer's Stone | Chamber of Secrets
Catching Fire    | Mockingjay
Divergent        | Insurgent
Mark of Athena   | House of Hades
Son of Neptune   | Mark of Athena
```

```
SELECT _____________________________________
FROM _____________________________________
WHERE _____________________________________
GROUP BY _____________________________________
HAVING _____________________________________
ORDER BY _____________________________________
```

Solution:

```
SELECT prequel.name, sequel.name
FROM books AS prequel, books AS sequel
WHERE prequel.series = sequel.series AND prequel.number + 1 =
sequel.number;
```

3.Write a SELECT statement which would output the book name, author, and author's city of birth for books in the books table that are the second book in a series of books and have authors from the USA. Order the output by the author's name alphabetically.

Output:

```
Son of Neptune  | Rick Riordan       | San Antonio
Catching Fire   | Suzanne Collins    | Hartford
Insurgent       | Veronica Roth      | New York
```

```
SELECT _____________________________________
FROM _____________________________________
WHERE _____________________________________
GROUP BY _____________________________________
```

HAVING _____
ORDER BY _____

Solution:

```
SELECT name, author, city
FROM books, series_authors
WHERE books.series = series_authors.series AND number = 2 AND country
= “USA”
ORDER BY author
```

AlTeRnAtInG Generator

Create a generator that when given 3 generators, cycles through yielding from each generator.

```
def alternating_generator(gen1, gen2, gen3):
    """
    >>> def evens():
            "a generator that generates even numbers starting at 2"
    >>> def odds():
            "a generator that generates odd numbers starting at 1"
    >>> def nines():
            "a generator that generates multiples of 9 starting at 9"

    >>> gen = alternating_generator(evens, odds, nines)
    >>> next(gen) # return a value from evens
    2
    >>> next(gen) # return a value from odds
    1
    >>> next(gen) # return a value from nines
    9
    >>> next(gen) # return a value from evens
    4
    >>> next(gen) # return a value from odds
    3
    """
    iter_lst = [gen1, gen2, gen3]
    ____________________________________
    ____________________________________
    ____________________________________
    ____________________________________
```

```
def alternating_generator(gen1, gen2, gen3):
    iter_lst = [gen1(), gen2(), gen3()]
    while True:
        for i in iter_lst:
            yield next(i)
```

Stop ORDERing me around

You work for a shipping company and are making a program that creates an order. Each order contains Packages. Depending on the weight of the item, it either goes into a LargePackage or a SmallPackage (a package with greater than 5 units of weight is a LargePackage, and any other package is a SmallPackage)

The cost for shipping for LargePackages is 5 and the cost of SmallPackages is 2. Each order has a `calculateShippingCost` method that calculates the total shipping cost of the order. It also has an `allItems` method which prints out the names of all the items in the order.

```
>>>newOrder = Order()
>>>newOrder.addItem("Keyboard", 2)
>>>newOrder.addItem("Mouse", 1)
>>>newOrder.addItem("Monitor", 6)

>>>newOrder.calculateShippingCost()#2(Keyboard) + 2(Mouse) +
5(Monitor) = 9
9

>>>newOrder.allItems()
Keyboard
Mouse
Monitor

>>>SmallPackage.shippingCost = 3
>>>newOrder.calculateShippingCost()#3(Keyboard) + 3(Mouse) +
5(Monitor) = 11
11
```

```
class Order:

    def __init__(self):
        self.packages = []

    def addItem(self, item, weight):
        if weight > 5:
            self.packages.append(LargePackage(item, weight))
        else:
            self.packages.append(SmallPackage(item, weight))
```

```
    def calculateShippingCost(self):
        totalCost = 0
        for package in self.packages:
            totalCost += package.shippingCost
        return totalCost

    def allItems(self):
        for package in self.packages:
            print(package.item)

class Package:

    def __init__(self, item, weight):
        self.item = item
        self.weight = weight

class LargePackage(Package):

    shippingCost = 5

class SmallPackage(Package):

    shippingCost = 2
```

Debugging

You are writing a function that takes in a list of pairs and a key. The first element of each pair is an integer which represents a key and the second element is a string which represents the value. The second argument to the function is a key. The function removes each of the pairs that has a key that matches the inputted key. Finally, it returns the values that were removed from the list, order of the returned values does not matter.

```
def test_func(pairs, key):
      """
      >>>lst = [[1, "A"], [2, "B"], [1,"C"]]
      >>>test_func(lst, 1)
      ['A', 'C']                    # ['C', 'A'] is also acceptable
      >>>lst
      [[2, 'B']]                    # Mutate the original list

      >>>lst2 = [[8, "pop?"], [8, "pop!"], [8, "pop."]]
      >>>test_func(lst2, 8)
      ['pop.', 'pop!', 'pop?']    # Any order of the three pops is
fine
      >>>lst2
      []                            # All of the pairs were removed
      """
1.     output = []
2.     for i in range(len(pairs)):
3.         if pairs[i] == key:
4.             output.append(pairs.pop(i))
5.     return output
```

1. For the following doctest what should the function return
   ```
   >>>lst = [[1, "one"], [2, "two"], [2, "Two!"], [1, "One!"]]
   >>>test_func(lst, 1)
   ```
2. What are the contents of lst after the doctest above has been run?
3. For the input specified in part 1, what **should** (assume the function is working as intended) the function return?
4. Identify 3 bugs in the function. For each bug, include the line number, the code that is incorrect and describe the fix (a few words is enough). There are 3 distinction *types* of bugs in the question, but some of the bugs have multiple ways of fixing them. After fixing the bugs, the function should work correctly.
   a. …
   b. …
   c. …

5. Include a fully working definition of `test_func`:

Answers

1. Empty List, [], or Error
2. Unchanged from before, [[1, "one"], [2, "two"], [2, "Two!"], [1, "One!"]], or Error
3. [“one”, “One!”]
4. 1st Bug: len(pairs) index pairs in order, while we need to index in the reverse order. The reason we need to go in reverse is because we are deleting items while in a for loop. If we do not do this, then our indexing will be off. Another option is to make a copy and then remove from the original list after. A final option would be to use a while loop.
   2nd Bug: The key is the first item of a pair, so to access the key, we must use `[0]`.
   3rd Bug: Our output should include only the first item of the pair.

```
def test_func(pairs, key):
    """
    >>> lst = [[1, "A"], [2, "B"], [1,"C"]]
    >>> test_func(lst, 1)
    >>> lst
    """
    output = []
    for i in range(len(pairs) - 1, -1, -1):
        if pairs[i][0] == key:
            output.append(pairs.pop(i)[1])
    return output
```

Note: There were 3 acceptable ways to solve this problem that I encountered while grading, the first is looping in reverse order which is shown above. Another way was using a copy of the list and then removing from the original list sometime after. The last way was using a while loop or some sort of special indexing.

One Giant Leap Forward, One Small Step back

Jeff likes to traverse his LinkedLists weirdly. He likes to skip around a bit, specifically, jumping forward two and then jumping back one. As an example, if we had a linked list of:

1 -> 2 -> 3 -> 4 -> 5

Jeff wants 1 then 3 then 2 then 4 then 3 then 5, ending when we get to the last value in the linked list. Jeff would like you to create an iterator for him to iterate through any linked list like this, beginning with the first value in the linked list and ending with the last value.

```
class Link:
    """
    >>> s = Link(1, Link(2, Link(3)))
    >>> s
    Link(1, Link(2, Link(3)))
    """
    empty = ()
    def __init__(self, first, rest=empty):
        assert rest is Link.empty or isinstance(rest, Link)
        self.first = first
        self.rest = rest

class LinkedListIterator:
    """
     >>> ll = Link(1, Link(2, Link(3, Link(4, Link(5)))))

     >>> lst = []
     >>> for i in LinkedListIterator(ll):
     ...    lst.append(i)
     >>> print(lst)
     [1, 3, 2, 4, 3, 5]
    """
    def __init__(self, lnk):
        self.pointer1 = _________________
        self.pointer2 = _________________
        self.counter = 0

    def __iter__(self):
        return self
```

```
def __next__(self):
    if self.pointer2 is Link.empty:
        ____________________________

    if self.counter % 2 == 0:
        return_value = _________________
        ____________________________
    else:
        return_value = _________________
        ____________________________

    ________________________

    return return_value
```

```
class Link:
    """
    >>> s = Link(1, Link(2, Link(3)))
    >>> s
    Link(1, Link(2, Link(3)))
    """
    empty = ()
    def __init__(self, first, rest=empty):
        assert rest is Link.empty or isinstance(rest, Link)
        self.first = first
        self.rest = rest

class LinkedListIterator:
    """
    Two steps forward, one step back
    """
    def __init__(self, lnk):
        self.pointer1 = lnk
        self.pointer2 = lnk.rest.rest
        self.counter = 0

    def __iter__(self):
        return self

    def __next__(self):
        if self.pointer2 is Link.empty:
            raise StopIteration

        if self.counter % 2 == 0:
            value = self.pointer1.first
            self.pointer1 = self.pointer1.rest
        else:
            value = self.pointer2.first
            self.pointer2 = self.pointer2.rest

        self.counter += 1

        return value
```