Lab 7: Recursive Objects lab07.zip (lab07.zip)

Due at 11:59pm on Friday, 03/15/2019.

Starter Files

Download lab07.zip (lab07.zip). Inside the archive, you will find starter files for the questions in this lab, along with a copy of the Ok (ok) autograder.

Submission

By the end of this lab, you should have submitted the lab with `python3 ok --submit`. You may submit more than once before the deadline; only the final submission will be graded. Check that you have successfully submitted your code on okpy.org (https://okpy.org/).

- To receive credit for this lab, you must complete Questions 2 through 6 in lab07.py (lab07.py) and submit through OK.
- Questions 7 through 8 are *extra practice*. They can be found in the lab07_extra.py (lab07_extra.py) file. It is recommended that you complete these problems on your own time.

Topics

Repr and Str

Linked Lists

Introduction to Linked Lists

Motivation for Linked Lists

Trees (Again)

Object Oriented Trees

Check-Off

Q1: Junk

What happens in the following code? Sign up for checkoffs in your lab if you'd like to get credit for this week's checkoff.

```
>>> class VendingMachine:
...     v = 0
...     def __init__(self):
...         self.soda = JunkDrink(self)
...
>>> class JunkDrink:
...     v = 0
...     def __init__(self, machine):
...         self.machine = machine
...         self.machine.v = self.machine.v + 1
...         machine.v = machine.v + 1
...         self.v = self.v + 1
...
```

```
>>> a = VendingMachine()
>>> a.v
------

>>> JunkDrink.v
------

>>> a.soda.v
------

>>> x = VendingMachine.__init__(a)
>>> x
------

>>> a.v
------

>>> JunkDrink.v
------

>>> a.soda.v
------
```

Required Questions

What Would Python Display?

Q2: WWPD: Linked Lists

Read over the `Link` class in `lab07.py`. Make sure you understand the doctests.

> Use Ok to test your knowledge with the following "What Would Python Display?" questions:
>
> ```
> python3 ok -q link -u
> ```
>
> Enter `Function` if you believe the answer is `<function ...>`, `Error` if it errors, and `Nothing` if nothing is displayed.
>
> If you get stuck, try drawing out the box-and-pointer diagram for the linked list on a piece of paper or loading the `Link` class into the interpreter with `python3 -i lab07.py`.

```
>>> from lab07 import *
>>> link = Link(1000)
>>> link.first
------

>>> link.rest is Link.empty
------

>>> link = Link(1000, 2000)
------

>>> link = Link(1000, Link())
------
```

```
>>> from lab07 import *
>>> link = Link(1, Link(2, Link(3)))
>>> link.first
------

>>> link.rest.first
------

>>> link.rest.rest.rest is Link.empty
------

>>> link.first = 9001
>>> link.first
------

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

>>> link = Link(1)
>>> link.rest = link
>>> link.rest.rest.rest.rest.first
------

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

>>> link2.rest.first
------
```

```
>>> from lab07 import *
>>> link = Link(5, Link(6, Link(7)))
>>> link                  # Look at the __repr__ method of Link
------

>>> print(link)          # Look at the __str__ method of Link
------
```

Q3: WWPD: repr and str

Use Ok to test your knowledge with the following "What Would Python Display?" questions:

```
python3 ok -q repr_str -u
```

If you get stuck, try running the examples on the interpreter.

```
>>> print("hi")
------

>>> "hi"
------

>>> print(repr("hi"))
------

>>> repr("hi")
------
```

```
>>> class A:
...     def __init__(self, x):
...         self.x = x
...     def __repr__(self):
...         return self.x
>>> class B(A):
...     def __str__(self):
...         return self.x + self.x
>>> A("hi")
______

>>> print(A("hi"))
______

>>> B("hi")
______

>>> print(B("hi"))
______
```

```
>>> class C:
...     def __str__(self):
...         print('hi')
...         return 'hihi'
...     def __repr__(self):
...         print('hihihi')
...         return 'hihihihi'
>>> C()
______

>>> print(C())
______

>>> q = str(C())
______

>>> q
______

>>> r = repr(C())
______

>>> r
______
```

Coding Practice

Q4: Link to List

Write a function `link_to_list` that takes in a linked list and returns the sequence as a Python list. You may assume that the input list is shallow; none of the elements is another linked list.

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

```
def link_to_list(link):
    """Takes a linked list and returns a Python list with the same elements.

    >>> link = Link(1, Link(2, Link(3, Link(4))))
    >>> link_to_list(link)
    [1, 2, 3, 4]
    >>> link_to_list(Link.empty)
    []
    """
    "*** YOUR CODE HERE ***"
```

Use Ok to test your code:

```
python3 ok -q link_to_list
```

Q5: 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`.

```
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)))
    """
    "*** YOUR CODE HERE ***"
```

Use Ok to test your code:

```
python3 ok -q store_digits
```

Q6: Cumulative Sum

Write a function `cumulative_sum` that mutates the Tree `t` so that each node's label becomes the sum of all labels in the subtree rooted at the node.

```
def cumulative_sum(t):
    """Mutates t so that each node's label becomes the sum of all labels in
    the corresponding subtree rooted at t.

    >>> t = Tree(1, [Tree(3, [Tree(5)]), Tree(7)])
    >>> cumulative_sum(t)
    >>> t
    Tree(16, [Tree(8, [Tree(5)]), Tree(7)])
    """
    "*** YOUR CODE HERE ***"
```

Use Ok to test your code:

```
python3 ok -q cumulative_sum
```

Optional Questions

> The following questions are for **extra practice** -- they can be found in the the lab07_extra.py (lab07_extra.py) file. It is recommended that you complete these problems on your own time.

Q7: Cycles

The `Link` class can represent lists with cycles. That is, a list may contain itself as a sublist.

```
>>> s = Link(1, Link(2, Link(3)))
>>> s.rest.rest.rest = s
>>> s.rest.rest.rest.rest.rest.first
3
```

Implement `has_cycle` ,that returns whether its argument, a `Link` instance, contains a cycle.

> *Hint*: Iterate through the linked list and try keeping track of which `Link` objects you've already seen.

```
def has_cycle(link):
    """Return whether link contains a cycle.

    >>> s = Link(1, Link(2, Link(3)))
    >>> s.rest.rest.rest = s
    >>> has_cycle(s)
    True
    >>> t = Link(1, Link(2, Link(3)))
    >>> has_cycle(t)
    False
    >>> u = Link(2, Link(2, Link(2)))
    >>> has_cycle(u)
    False
    """
    "*** YOUR CODE HERE ***"
```

Use Ok to test your code:

```
python3 ok -q has_cycle
```

As an extra challenge, implement `has_cycle_constant` with only constant space (http://composingprograms.com/pages/28-efficiency.html#growth-categories). (If you followed the hint above, you will use linear space.) The solution is short (less than 20 lines of code), but requires a clever idea. Try to discover the solution yourself before asking around:

```
def has_cycle_constant(link):
    """Return whether link contains a cycle.

    >>> s = Link(1, Link(2, Link(3)))
    >>> s.rest.rest.rest = s
    >>> has_cycle_constant(s)
    True
    >>> t = Link(1, Link(2, Link(3)))
    >>> has_cycle_constant(t)
    False
    """
    "*** YOUR CODE HERE ***"
```

Use Ok to test your code:

```
python3 ok -q has_cycle_constant
```

Q8: Reverse Other

Write a function `reverse_other` that mutates the tree such that **labels** on *every other* (odd-depth) level are reversed. For example, `Tree(1,[Tree(2, [Tree(4)]), Tree(3)])` becomes `Tree(1,[Tree(3, [Tree(4)]), Tree(2)])`. Notice that the nodes themselves are *not* reversed; only the labels are.

```
def reverse_other(t):
    """Mutates the tree such that nodes on every other (odd-depth) level
    have the labels of their branches all reversed.

    >>> t = Tree(1, [Tree(2), Tree(3), Tree(4)])
    >>> reverse_other(t)
    >>> t
    Tree(1, [Tree(4), Tree(3), Tree(2)])
    >>> t = Tree(1, [Tree(2, [Tree(3, [Tree(4), Tree(5)]), Tree(6, [Tree(7)])]), Tree(8)]
    >>> reverse_other(t)
    >>> t
    Tree(1, [Tree(8, [Tree(3, [Tree(5), Tree(4)]), Tree(6, [Tree(7)])]), Tree(2)])
    """
    "*** YOUR CODE HERE ***"
```

Use Ok to test your code:

```
python3 ok -q reverse_other
```

CS 61A (/)

Weekly Schedule (/weekly.html)

Office Hours (/office-hours.html)

Staff (/staff.html)

Resources (/resources.html)

Studying Guide (/articles/studying.html)

Debugging Guide (/articles/debugging.html)

Composition Guide (/articles/composition.html)

Policies (/articles/about.html)

Assignments (/articles/about.html#assignments)

Exams (/articles/about.html#exams)

Grading (/articles/about.html#grading)