Homework 5

● Graded

Student

Sangwon Ji

Total Points

2 / 2 pts

Autograder Score

2.0 / 2.0

Autograder Results

```
=====================================================================
Assignment: Homework 5
OK, version v1.18.1
=====================================================================

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Scoring tests

---------------------------------------------------------------------
Doctests for Mint

>>> from hw05 import *
>>> mint = Mint()
>>> mint.year
2023
>>> dime = mint.create(Dime)
>>> dime.year
2023
>>> Mint.present_year = 2103 # Time passes
>>> nickel = mint.create(Nickel)
>>> nickel.year # The mint has not updated its stamp yet
2023
>>> nickel.worth() # 5 cents + (80 - 50 years)
35
>>> mint.update() # The mint's year is updated to 2102
>>> Mint.present_year = 2178 # More time passes
>>> mint.create(Dime).worth() # 10 cents + (75 - 50 years)
35
>>> Mint().create(Dime).worth() # A new mint has the current year
10
>>> dime.worth() # 10 cents + (155 - 50 years)
115
>>> Dime.cents = 20 # Upgrade all dimes!
>>> dime.worth() # 20 cents + (155 - 50 years)
125
Score: 1.0/1
```

```
----------------------------------------------------------------------
Doctests for add_d_leaves

>>> from hw05 import *
>>> t_one_to_four = Tree(1, [Tree(2), Tree(3, [Tree(4)])])
>>> print(t_one_to_four)
1
2
3
4
>>> add_d_leaves(t_one_to_four, 5)
>>> print(t_one_to_four)
1
2
5
3
4
5
5
5
>>> t0 = Tree(9)
>>> add_d_leaves(t0, 4)
>>> t0
Tree(9)
>>> t1 = Tree(1, [Tree(3)])
>>> add_d_leaves(t1, 4)
>>> t1
Tree(1, [Tree(3, [Tree(4)])])
>>> t2 = Tree(2, [Tree(5), Tree(6)])
>>> t3 = Tree(3, [t1, Tree(0), t2])
>>> print(t3)
3
1
3
4
0
2
5
6
>>> add_d_leaves(t3, 10)
>>> print(t3)
3
1
3
4
10
10
10
10
10
10
0
```

```
10
2
5
10
10
6
10
10
10
Score: 1.0/1

---------------------------------------------------------------------
Doctests for store_digits

>>> from hw05 import *
>>> 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}', '', inspect.getsource(store_digits)))
>>> print("Do not use str or reversed!") if any([r in cleaned for r in ["str", "reversed"]]) else None
Score: 1.0/1

---------------------------------------------------------------------
Doctests for deep_map_mut

>>> from hw05 import *
>>> link1 = Link(3, Link(Link(4), Link(5, Link(6))))
>>> print(link1)
<3 <4> 5 6>
>>> # Disallow the use of making new Links before calling deep_map_mut
>>> Link.__init__, hold = lambda *args: print("Do not create any new Links."), Link.__init__
>>> try:
...     deep_map_mut(lambda x: x * x, link1)
... finally:
...     Link.__init__ = hold
>>> print(link1)
<9 <16> 25 36>
Score: 1.0/1

---------------------------------------------------------------------
Point breakdown
Mint: 1.0/1
add_d_leaves: 1.0/1
store_digits: 1.0/1
deep_map_mut: 1.0/1
```

Score:
Total: 4.0

Cannot backup when running ok with --local.

Final Score:2.0

Submitted Files

hw05.py Download

```
class Mint:
    """A mint creates coins by stamping on years.

    The update method sets the mint's stamp to Mint.present_year.

    >>> mint = Mint()
    >>> mint.year
    2023
    >>> dime = mint.create(Dime)
    >>> dime.year
    2023
    >>> Mint.present_year = 2103  # Time passes
    >>> nickel = mint.create(Nickel)
    >>> nickel.year     # The mint has not updated its stamp yet
    2023
    >>> nickel.worth()  # 5 cents + (80 - 50 years)
    35
    >>> mint.update()   # The mint's year is updated to 2102
    >>> Mint.present_year = 2178     # More time passes
    >>> mint.create(Dime).worth()    # 10 cents + (75 - 50 years)
    35
    >>> Mint().create(Dime).worth()  # A new mint has the current year
    10
    >>> dime.worth()     # 10 cents + (155 - 50 years)
    115
    >>> Dime.cents = 20  # Upgrade all dimes!
    >>> dime.worth()     # 20 cents + (155 - 50 years)
    125
    """
    present_year = 2023

    def __init__(self):
        self.update()

    def create(self, coin):
        return coin(self.year)

    def update(self):
        self.year = self.present_year


class Coin:
    cents = None  # will be provided by subclasses, but not by Coin itself

    def __init__(self, year):
        self.year = year

    def worth(self):
        age = Mint.present_year - self.year
```

```
        if age > 50:
            return self.cents + (age -50)
        else:
            return self.cents


class Nickel(Coin):
    cents = 5


class Dime(Coin):
    cents = 10


def add_d_leaves(t, v):
    """Add d leaves containing v to each node at every depth d.

    >>> t_one_to_four = Tree(1, [Tree(2), Tree(3, [Tree(4)])])
    >>> print(t_one_to_four)
    1
      2
      3
        4
    >>> add_d_leaves(t_one_to_four, 5)
    >>> print(t_one_to_four)
    1
      2
        5
      3
        4
          5
          5
        5

    >>> t0 = Tree(9)
    >>> add_d_leaves(t0, 4)
    >>> t0
    Tree(9)
    >>> t1 = Tree(1, [Tree(3)])
    >>> add_d_leaves(t1, 4)
    >>> t1
    Tree(1, [Tree(3, [Tree(4)])])
    >>> t2 = Tree(2, [Tree(5), Tree(6)])
    >>> t3 = Tree(3, [t1, Tree(0), t2])
    >>> print(t3)
    3
      1
        3
          4
      0
      2
        5
```

```
      6
    >>> add_d_leaves(t3, 10)
    >>> print(t3)
    3
      1
        3
          4
            10
            10
            10
          10
          10
        10
      0
        10
      2
        5
          10
          10
        6
          10
          10
        10
    """
    def leaves_helper(t,v, d=0):
        for b in t.branches:
            leaves_helper(b, v, d +1)
        for _ in range(d):
            t.branches.append(Tree(v))
    leaves_helper(t,v)



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}', '', inspect.getsource(store_digits)))
    >>> print("Do not use str or reversed!") if any([r in cleaned for r in ["str", "reversed"]]) else None
    """
    def reverse(link):
        if link == Link.empty:
```

```
            return link
        else:
            Links = Link (link.first, Link.empty)
            while link.rest != Link.empty:
                link = link.rest
                Links = Link(link.first, Links)
            return Links

    def store_digits_order(n):
        if n//10 == 0:
            return Link(n)
        else:
            return Link(n %10, store_digits_order(n //10))
    return reverse(store_digits_order(n))


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


def two_list(vals, counts):
    """
    Returns a linked list according to the two lists that were passed in. Assume
    vals and counts are the same size. Elements in vals represent the value, and the
    corresponding element in counts represents the number of this value desired in the
    final linked list. Assume all elements in counts are greater than 0. Assume both
    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]
    >>> b = [2, 2, 1]
    >>> c = two_list(a, b)
    >>> c
    Link(1, Link(1, Link(3, Link(3, Link(2)))))
    """
    "*** YOUR CODE HERE ***"


class Tree:
    """
    >>> t = Tree(3, [Tree(2, [Tree(5)]), Tree(4)])
    >>> t.label
    3
    >>> t.branches[0].label
    2
    >>> t.branches[1].is_leaf()
    True
    """

    def __init__(self, label, branches=[]):
        for b in branches:
            assert isinstance(b, Tree)
        self.label = label
        self.branches = list(branches)

    def is_leaf(self):
        return not self.branches

    def __repr__(self):
        if self.branches:
            branch_str = ', ' + repr(self.branches)
        else:
            branch_str = ''
        return 'Tree({0}{1})'.format(self.label, branch_str)

    def __str__(self):
        def print_tree(t, indent=0):
            tree_str = '  ' * indent + str(t.label) + "\n"
            for b in t.branches:
                tree_str += print_tree(b, indent + 1)
            return tree_str
        return print_tree(self).rstrip()


class Link:
    """A linked list.

    >>> s = Link(1)
```

```
    >>> s.first
    1
    >>> s.rest is Link.empty
    True
    >>> s = Link(2, Link(3, Link(4)))
    >>> s.first = 5
    >>> s.rest.first = 6
    >>> s.rest.rest = Link.empty
    >>> s                                  # Displays the contents of repr(s)
    Link(5, Link(6))
    >>> s.rest = Link(7, Link(Link(8, Link(9))))
    >>> s
    Link(5, Link(7, Link(Link(8, Link(9)))))
    >>> print(s)                           # Prints str(s)
    <5 7 <8 9>>
    """
    empty = ()

    def __init__(self, first, rest=empty):
        assert rest is Link.empty or isinstance(rest, Link)
        self.first = first
        self.rest = rest

    def __repr__(self):
        if self.rest is not Link.empty:
            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) + '>'

```