Homework 6: Object Oriented Programming, Linked Lists

hw06.zip (hw06.zip)

Due by 11:59pm on Thursday, October 20

Instructions

Download hw06.zip (hw06.zip). Inside the archive, you will find a file called hw06.py (hw06.py), along with a copy of the `ok` autograder.

Submission: When you are done, submit with `python3 ok --submit`. You may submit more than once before the deadline; only the final submission will be scored. Check that you have successfully submitted your code on okpy.org (https://okpy.org/). See Lab 0 (/lab/lab00#submitting-the-assignment) for more instructions on submitting assignments.

Using Ok: If you have any questions about using Ok, please refer to this guide. (/articles/using-ok)

Readings: You might find the following references useful:

- Section 2.5 (http://composingprograms.com/pages/25-object-oriented-programming.html)
- Section 2.9 (http://composingprograms.com/pages/29-recursive-objects.html)

Grading: Homework is graded based on correctness. Each incorrect problem will decrease the total score by one point. There is a homework recovery policy as stated in the syllabus. **This homework is out of 2 points.**

Required Questions

Getting Started Videos

OOP

Q1: Mint

A mint is a place where coins are made. In this question, you'll implement a `Mint` class that can output a `Coin` with the correct year and worth.

- Each `Mint` *instance* has a `year` stamp. The `update` method sets the `year` stamp of the instance to the `present_year` *class attribute* of the `Mint` *class*.
- The `create` method takes a subclass of `Coin` (*not* an instance!), then creates and returns an *instance* of that class stamped with the `mint` 's year (which may be different from `Mint.present_year` if it has not been updated.)
- A `Coin` 's `worth` method returns the `cents` value of the coin plus one extra cent for each year of age *beyond 50*. A coin's age can be determined by subtracting the coin's year from the `present_year` class attribute of the `Mint` class.

```
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
    2022
    >>> dime = mint.create(Dime)
    >>> dime.year
    2022
    >>> Mint.present_year = 2102  # Time passes
    >>> nickel = mint.create(Nickel)
    >>> nickel.year     # The mint has not updated its stamp yet
    2022
    >>> nickel.worth()  # 5 cents + (80 - 50 years)
    35
    >>> mint.update()   # The mint's year is updated to 2102
    >>> Mint.present_year = 2177     # 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 = 2022

    def __init__(self):
        self.update()

    def create(self, coin):
        "*** YOUR CODE HERE ***"

    def update(self):
        "*** YOUR CODE HERE ***"

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

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

    def worth(self):
        "*** YOUR CODE HERE ***"
```

```
class Nickel(Coin):
    cents = 5

class Dime(Coin):
    cents = 10
```

Use Ok to test your code:

```
python3 ok -q Mint
```

Linked Lists

Q2: 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`.

> **Important**: Do not use any string manipulation functions like `str` and `reversed`.

```
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)))
    >>> # a check for restricted functions
    >>> import inspect, re
    >>> cleaned = re.sub(r"#.*\\n", '', re.sub(r'"{3}[\s\S]*?"{3}', '', inspect.getsou
    >>> print("Do not use str or reversed!") if any([r in cleaned for r in ["str", "re
    >>> link1 = Link(3, Link(Link(4), Link(5, Link(6))))
    """
    "*** YOUR CODE HERE ***"
```

Use Ok to test your code:

```
python3 ok -q store_digits
```

Q3: Mutable Mapping

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.

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

Hint: The built-in `isinstance` function may be useful.

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

Construct Check: The last doctest of this question ensures that you do not create new linked lists. If you are failing this doctest, ensure that you are not creating link lists by calling the constructor, i.e.

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

```
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 Links."), Linl
    >>> try:
    ...     deep_map_mut(lambda x: x * x, link1)
    ... finally:
    ...     Link.__init__ = hold
    >>> print(link1)
    <9 <16> 25 36>
    """
    "*** YOUR CODE HERE ***"
```

Use Ok to test your code:

```
python3 ok -q deep_map_mut
```

Q4: Two List

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, and the second list contains the number of each corresponding value. 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.

```
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, 2]
    >>> b = [1, 1, 1]
    >>> c = two_list(a, b)
    >>> c
    Link(1, Link(3, Link(2)))
    >>> 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 ***"
```

Use Ok to test your code:

```
python3 ok -q two_list
```

Submit

Make sure to submit this assignment by running:

```
python3 ok --submit
```

Optional Questions

Q5: Next Virahanka Fibonacci Object

Implement the `next` method of the `VirFib` class. For this class, the `value` attribute is a Fibonacci number. The `next` method returns a `VirFib` instance whose `value` is the next Fibonacci number. The `next` method should take only constant time.

Note that in the doctests, nothing is being printed out. Rather, each call to `.next()` returns a `VirFib` instance. The way each `VirFib` instance is displayed is determined by the return value of its `__repr__` method.

> *Hint*: Keep track of the previous number by setting a new instance attribute inside `next`. You can create new instance attributes for objects at any point, even outside the `__init__` method.

```
class VirFib():
    """A Virahanka Fibonacci number.

    >>> start = VirFib()
    >>> start
    VirFib object, value 0
    >>> start.next()
    VirFib object, value 1
    >>> start.next().next()
    VirFib object, value 1
    >>> start.next().next().next()
    VirFib object, value 2
    >>> start.next().next().next().next()
    VirFib object, value 3
    >>> start.next().next().next().next().next()
    VirFib object, value 5
    >>> start.next().next().next().next().next().next()
    VirFib object, value 8
    >>> start.next().next().next().next().next().next() # Ensure start isn't changed
    VirFib object, value 8
    """

    def __init__(self, value=0):
        self.value = value

    def next(self):
        "*** YOUR CODE HERE ***"

    def __repr__(self):
        return "VirFib object, value " + str(self.value)
```

Use Ok to test your code:

```
python3 ok -q VirFib
```

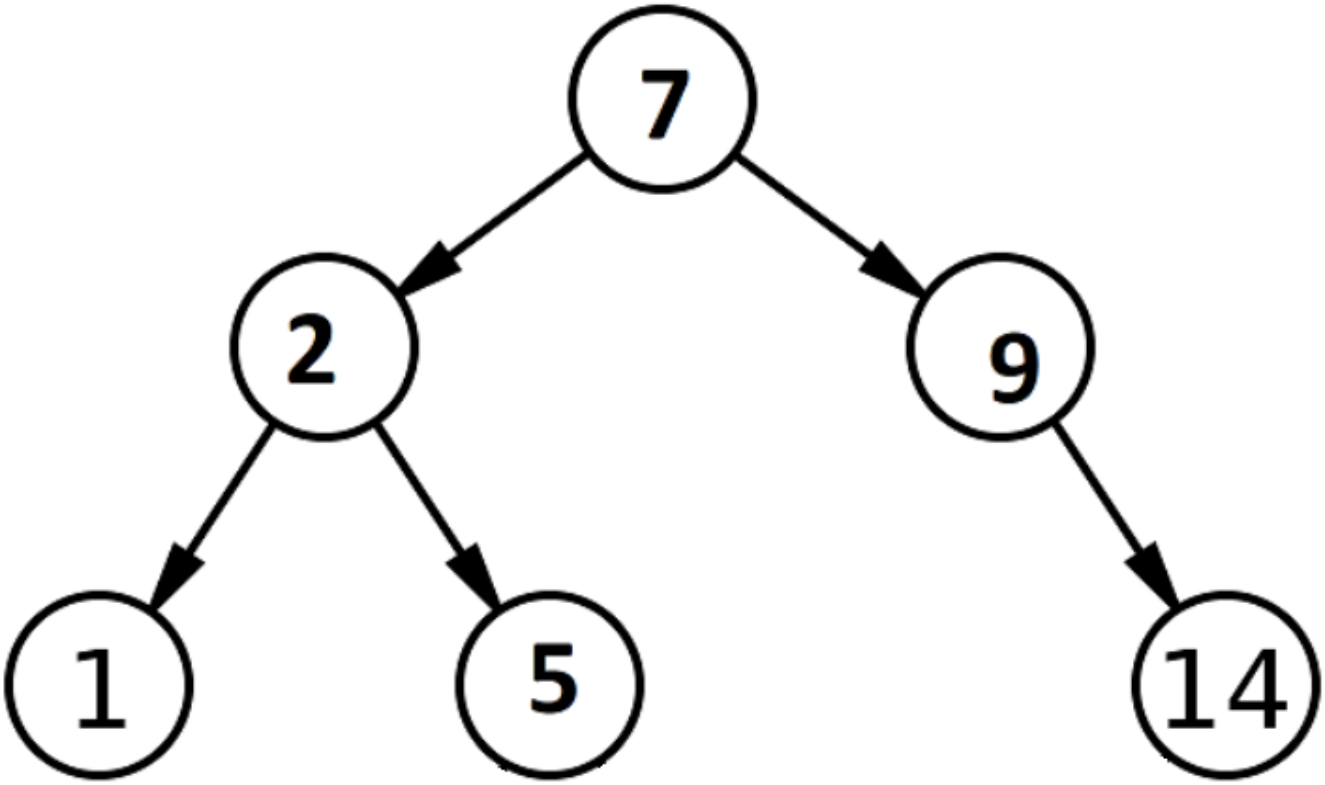
Q6: Is BST

Write a function `is_bst`, which takes a Tree `t` and returns `True` if, and only if, `t` is a valid binary search tree, which means that:

- Each node has at most two children (a leaf is automatically a valid binary search tree)
- The children are valid binary search trees
- For every node, the entries in that node's left child are less than or equal to the label of the node

- For every node, the entries in that node's right child are greater than the label of the node

An example of a BST is:



Note that, if a node has only one child, that child could be considered either the left or right child. You should take this into consideration.

Hint: It may be helpful to write helper functions `bst_min` and `bst_max` that return the minimum and maximum, respectively, of a Tree if it is a valid binary search tree.

```
def is_bst(t):
    """Returns True if the Tree t has the structure of a valid BST.

    >>> t1 = Tree(6, [Tree(2, [Tree(1), Tree(4)]), Tree(7, [Tree(7), Tree(8)])])
    >>> is_bst(t1)
    True
    >>> t2 = Tree(8, [Tree(2, [Tree(9), Tree(1)]), Tree(3, [Tree(6)]), Tree(5)])
    >>> is_bst(t2)
    False
    >>> t3 = Tree(6, [Tree(2, [Tree(4), Tree(1)]), Tree(7, [Tree(7), Tree(8)])])
    >>> is_bst(t3)
    False
    >>> t4 = Tree(1, [Tree(2, [Tree(3, [Tree(4)])])])
    >>> is_bst(t4)
    True
    >>> t5 = Tree(1, [Tree(0, [Tree(-1, [Tree(-2)])])])
    >>> is_bst(t5)
    True
    >>> t6 = Tree(1, [Tree(4, [Tree(2, [Tree(3)])])])
    >>> is_bst(t6)
    True
    >>> t7 = Tree(2, [Tree(1, [Tree(5)]), Tree(4)])
    >>> is_bst(t7)
    False
    """
    "*** YOUR CODE HERE ***"
```

Use Ok to test your code:

```
python3 ok -q is_bst
```

Exam Practice

Homework assignments will also contain prior exam questions for you to try. These questions have no submission component; feel free to attempt them if you'd like some practice!

Object-Oriented Programming

1. Spring 2022 MT2 Q8: CS61A Presents The Game of Hoop. (https://cs61a.org/exam/sp22/mt2/61a-sp22-mt2.pdf#page=17)
2. Fall 2020 MT2 Q3: Sparse Lists (https://cs61a.org/exam/fa20/mt2/61a-fa20-mt2.pdf#page=9)
3. Fall 2019 MT2 Q7: Version 2.0 (https://cs61a.org/exam/fa19/mt2/61a-fa19-mt2.pdf#page=8)

Linked Lists

1. Fall 2020 Final Q3: College Party (https://cs61a.org/exam/fa20/final/61a-fa20-final.pdf#page=9)
2. Fall 2018 MT2 Q6: Dr. Frankenlink (https://cs61a.org/exam/fa18/mt2/61a-fa18-mt2.pdf#page=6)
3. Spring 2017 MT1 Q5: Insert (https://cs61a.org/exam/sp17/mt1/61a-sp17-mt1.pdf#page=7)