

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

```
1 class Mint:
2     """A mint creates coins by stamping on years.
3
4     The update method sets the mint's stamp to Mint.present_year.
5
6     >>> mint = Mint()
7     >>> mint.year
8     2023
9     >>> dime = mint.create(Dime)
10    >>> dime.year
11    2023
12    >>> Mint.present_year = 2103 # Time passes
13    >>> nickel = mint.create(Nickel)
14    >>> nickel.year # The mint has not updated its stamp yet
15    2023
16    >>> nickel.worth() # 5 cents + (80 - 50 years)
17    35
18    >>> mint.update() # The mint's year is updated to 2102
19    >>> Mint.present_year = 2178 # More time passes
20    >>> mint.create(Dime).worth() # 10 cents + (75 - 50 years)
21    35
22    >>> Mint().create(Dime).worth() # A new mint has the current year
23    10
24    >>> dime.worth() # 10 cents + (155 - 50 years)
25    115
26    >>> Dime.cents = 20 # Upgrade all dimes!
27    >>> dime.worth() # 20 cents + (155 - 50 years)
28    125
29    """
30    present_year = 2023
31
32    def __init__(self):
33        self.update()
34
35    def create(self, coin):
36        return coin(self.year)
37
38    def update(self):
39        self.year = self.present_year
40
41
42 class Coin:
43     cents = None # will be provided by subclasses, but not by Coin itself
44
45     def __init__(self, year):
46         self.year = year
47
48     def worth(self):
49         age = Mint.present_year - self.year
```

```
50     if age > 50:
51         return self.cents + (age - 50)
52     else:
53         return self.cents
54
55
56 class Nickel(Coin):
57     cents = 5
58
59
60 class Dime(Coin):
61     cents = 10
62
63
64 def add_d_leaves(t, v):
65     """Add d leaves containing v to each node at every depth d.
66
67     >>> t_one_to_four = Tree(1, [Tree(2), Tree(3, [Tree(4)])])
68     >>> print(t_one_to_four)
69     1
70     2
71     3
72     4
73     >>> add_d_leaves(t_one_to_four, 5)
74     >>> print(t_one_to_four)
75     1
76     2
77     5
78     3
79     4
80     5
81     5
82     5
83
84     >>> t0 = Tree(9)
85     >>> add_d_leaves(t0, 4)
86     >>> t0
87     Tree(9)
88     >>> t1 = Tree(1, [Tree(3)])
89     >>> add_d_leaves(t1, 4)
90     >>> t1
91     Tree(1, [Tree(3, [Tree(4)])])
92     >>> t2 = Tree(2, [Tree(5), Tree(6)])
93     >>> t3 = Tree(3, [t1, Tree(0), t2])
94     >>> print(t3)
95     3
96     1
97     3
98     4
99     0
100    2
101    5
```

```

102     6
103     >>> add_d_leaves(t3, 10)
104     >>> print(t3)
105     3
106     1
107     3
108     4
109     10
110     10
111     10
112     10
113     10
114     10
115     0
116     10
117     2
118     5
119     10
120     10
121     6
122     10
123     10
124     10
125     """
126     def leaves_helper(t,v, d=0):
127         for b in t.branches:
128             leaves_helper(b, v, d + 1)
129         for _ in range(d):
130             t.branches.append(Tree(v))
131     leaves_helper(t,v)
132
133
134
135     def store_digits(n):
136         """Stores the digits of a positive number n in a linked list.
137
138         >>> s = store_digits(1)
139         >>> s
140         Link(1)
141         >>> store_digits(2345)
142         Link(2, Link(3, Link(4, Link(5))))
143         >>> store_digits(876)
144         Link(8, Link(7, Link(6)))
145         >>> store_digits(2450)
146         Link(2, Link(4, Link(5, Link(0))))
147         >>> # a check for restricted functions
148         >>> import inspect, re
149         >>> cleaned = re.sub(r"#.*\\n", "", re.sub(r'"{3}[\s\S]*?"{3}', "", inspect.getsource(store_digits)))
150         >>> print("Do not use str or reversed!") if any([r in cleaned for r in ["str", "reversed"]]) else None
151         """
152         def reverse(link):
153             if link == Link.empty:

```

```

154         return link
155     else:
156         Links = Link(link.first, Link.empty)
157         while link.rest != Link.empty:
158             link = link.rest
159             Links = Link(link.first, Links)
160         return Links
161
162 def store_digits_order(n):
163     if n//10 == 0:
164         return Link(n)
165     else:
166         return Link(n % 10, store_digits_order(n // 10))
167     return reverse(store_digits_order(n))
168
169
170 def deep_map_mut(func, lnk):
171     """Mutates a deep link lnk by replacing each item found with the
172     result of calling func on the item. Does NOT create new Links (so
173     no use of Link's constructor).
174
175     Does not return the modified Link object.
176
177     >>> link1 = Link(3, Link(Link(4), Link(5, Link(6))))
178     >>> print(link1)
179     <3 <4> 5 6>
180     >>> # Disallow the use of making new Links before calling deep_map_mut
181     >>> Link.__init__, hold = lambda *args: print("Do not create any new Links."), Link.__init__
182     >>> try:
183     ...     deep_map_mut(lambda x: x * x, link1)
184     ... finally:
185     ...     Link.__init__ = hold
186     >>> print(link1)
187     <9 <16> 25 36>
188     """
189     if not isinstance(lnk.first, Link):
190         lnk.first = func(lnk.first)
191     else:
192         deep_map_mut(func, lnk.first)
193     if lnk.rest != Link.empty:
194         deep_map_mut(func, lnk.rest)
195
196
197 def two_list(vals, counts):
198     """
199     Returns a linked list according to the two lists that were passed in. Assume
200     vals and counts are the same size. Elements in vals represent the value, and the
201     corresponding element in counts represents the number of this value desired in the
202     final linked list. Assume all elements in counts are greater than 0. Assume both
203     lists have at least one element.
204     >>> a = [1, 3]
205     >>> b = [1, 1]

```



```

206 >>> c = two_list(a, b)
207 >>> c
208 Link(1, Link(3))
209 >>> a = [1, 3, 2]
210 >>> b = [2, 2, 1]
211 >>> c = two_list(a, b)
212 >>> c
213 Link(1, Link(1, Link(3, Link(3, Link(2)))))
214 """
215 """ YOUR CODE HERE """
216
217
218 class Tree:
219     """
220     >>> t = Tree(3, [Tree(2, [Tree(5)]), Tree(4)])
221     >>> t.label
222     3
223     >>> t.branches[0].label
224     2
225     >>> t.branches[1].is_leaf()
226     True
227     """
228
229     def __init__(self, label, branches=[]):
230         for b in branches:
231             assert isinstance(b, Tree)
232         self.label = label
233         self.branches = list(branches)
234
235     def is_leaf(self):
236         return not self.branches
237
238     def __repr__(self):
239         if self.branches:
240             branch_str = ', ' + repr(self.branches)
241         else:
242             branch_str = ''
243         return 'Tree({0}{1})'.format(self.label, branch_str)
244
245     def __str__(self):
246         def print_tree(t, indent=0):
247             tree_str = ' ' * indent + str(t.label) + "\n"
248             for b in t.branches:
249                 tree_str += print_tree(b, indent + 1)
250             return tree_str
251         return print_tree(self).rstrip()
252
253
254 class Link:
255     """A linked list.
256
257     >>> s = Link(1)

```

```

258 >>> s.first
259 1
260 >>> s.rest is Link.empty
261 True
262 >>> s = Link(2, Link(3, Link(4)))
263 >>> s.first = 5
264 >>> s.rest.first = 6
265 >>> s.rest.rest = Link.empty
266 >>> s                                # Displays the contents of repr(s)
267 Link(5, Link(6))
268 >>> s.rest = Link(7, Link(Link(8, Link(9))))
269 >>> s
270 Link(5, Link(7, Link(Link(8, Link(9)))))
271 >>> print(s)                        # Prints str(s)
272 <5 7 <8 9>>
273 ""
274 empty = ()
275
276 def __init__(self, first, rest=empty):
277     assert rest is Link.empty or isinstance(rest, Link)
278     self.first = first
279     self.rest = rest
280
281 def __repr__(self):
282     if self.rest is not Link.empty:
283         rest_repr = ', ' + repr(self.rest)
284     else:
285         rest_repr = ''
286     return 'Link(' + repr(self.first) + rest_repr + ')'
287
288 def __str__(self):
289     string = '<'
290     while self.rest is not Link.empty:
291         string += str(self.first) + ' '
292         self = self.rest
293     return string + str(self.first) + '>'
294

```