100

ryan
Other

PDF generated at: 15 Jul 2024 23:48:28 UTC
View this report on HackerRank

Score

100% • 80 / 80
scored in CodePath TIP101: Unit 5 Assessment, Version A - Summer 2024 in 31 min 26 sec on 15 Jul 2024 16:15:08 PDT

Candidate Information

Email	concepting@protonmail.com
Test	CodePath TIP101: Unit 5 Assessment, Version A - Summer 2024
Candidate Packet	View
Taken on	15 Jul 2024 16:15:08 PDT
Time taken	31 min 26 sec/ 90 min
Work Experience	< 1 years
Invited by	CodePath

Suspicious Activity detected

Code similarity

Code similarity
1 question

Skill Distribution

There is no associated skills data that can be shown for this assessment

Tags Distribution

There is no associated tags data that can be shown for this assessment

Questions

Status	No.	Question	Time Taken	Skill	Score
✓	1	Bark Bark Bark Multiple Choice	56 sec	-	5/5
✓	2	Linked List Multiple Choice	2 min 4 sec	-	5/5

✓	3	Time Complexity Multiple Choice	31 sec	-	5/5
✓	4	Last Node Multiple Choice	16 min 13 sec	-	5/5
✓	5	Get Grandchildren Coding	5 min 47 sec	-	20/20
✓	6	Extend Linked List Coding	2 min 18 sec	-	20/20
✓	7	Copy First N Nodes (LL) Coding	3 min 25 sec	-	20/20

1. Bark Bark Bark

✓ Correct

Multiple Choice

Question description

What is the output of the following?

```
class Dog:
    def __init__(self, name, breed, owner):
        self.name = name
        self.breed = breed
        self.owner = owner

    def bark(self):
        print("Woof!")
```

```
dog1 = Dog("Fido", "Black lab", "Alan Turing")
dog2 = Dog("Spot", "Dalmatian", "Ada Lovelace")

dog1.bark()
dog2.bark()
```

Candidate's Solution

Options: (Expected answer indicated with a tick)

- [x] ** "Woof!"
 "Woof!" **
- [ ] "Woof!" "Woof!"<!-- notionvc: f19ee29f-4a1a-424a-86d5-940c5273298a -->
- [ ] "Fido"
 "Spot"

No comments.

2. Linked List

Correct

Multiple Choice

Question description

After running the following code, what will the output of `print_linked_list(node)`?

```
class Node:
    def __init__(self, value, next_node = None):
```

```
            self.value = value
            self.next = next_node

def print_linked_list(head):
    result = []
    current = head
    while current:
        result.append(current.value)
        current = current.next
    print(result)


node = Node('apple')
node.next = Node('banana')
node.next.next = Node('orange')
print_linked_list(node)
```

Candidate's Solution

Options: (Expected answer indicated with a tick)

- ○ 'apple' -> 'banana' -> 'orange'
- ● **['apple', 'banana', 'orange']** ✓
- ○ ['apple']
- ○ 'apple'

ⓘ **No comments.**

3. Time Complexity

Correct

Multiple Choice

Question description

What is the time complexity of the following?

```
def print_nums(n):
    for i in range(n):
        print(i)
```

Candidate's Solution

Options: (Expected answer indicated with a tick)

- ○ O(1)
- ● **O(n)** ✓
- ○ O(n^2)
- ○ O(n^3)

No comments.

4. Last Node

Correct

Multiple Choice

Question description

The following code wants to return the value of the last node in a list.
However, running the following code results in the following error:
AttributeError: 'NoneType' object has no attribute 'value'.

Which of the following options will fix this error?

```
class Node:
        def __init__(self, value, next_node = None):
                self.value = value
                self.next = next_node

def get_last(head):
    if not head:
        return None
    current = head
    while current:
        current = current.next
    return current.value

node1 = Node(1)
node2 = Node(2)
node3 = Node(3)
node1.next = node2
node2.next = node3
get_last(node1)
```

Candidate's Solution

Options: (Expected answer indicated with a tick)

- ( ) <pre> <code class="language-python">class Node: def __init__(self, value, next_node = None): self.value = value self.next = next_node def get_last(self, head): # Make get_last a method of the Node class if not head: return None current = head while current: current = current.next return current.value</code></pre>

<pre> <code class="language-python">class Node: def __init__(self, value, next_node = None): self.value = value self.next = next_node def get_last(head): if not head: return None current = head while current: current = current.next return current # Change to current instead of current.value</code></pre>

**<pre> <code class="language-python">class Node: def __init__(self, value, next_node = None): self.value = value self.next = next_node def get_last(head): if not head: return None current = head while current.next: # Change to current.next instead of current current = current.next return current.value</code></pre>   **

<pre> <code class="language-python">class Node: def __init__(self, value, next_node = None): self.value = value self.next = next_node def get_last(head): # if not head: # Remove if statement # return None current = head while current: current = current.next return current.value</code></pre>

No comments.

5. Get Grandchildren

Correct

Coding

Question description

The following `Person` class defines a person with a last name, first name, and list of children. Each child in the list of children is an instance of `Person`.
The method `add_child()` adds a child to a person's list of children.
Add a method to the `Person` class called `get_grandchildren()` that returns a list of the person's grandchildren.

Example 1:
johndoe = Person("John", "Doe")
janedoe = Person("Jane", "Doe")
jimmydoe = Person("Jimmy", "Doe")
johndoe.add_child(janedoe)
janedoe.add_child(jimmydoe)
johndoe.get_grandchildren()
Output: # [jimmydoe]

Example 2:
johndoe = Person("John", "Doe")
janedoe = Person("Jane", "Doe")
johndoe.add_child(janedoe)
johndoe.get_grandchildren()
Output: # []

Candidate's Solution Language used: **Python 3**

```python
#!/bin/python3

import math
import os
import random
import re
import sys


#
# Complete the 'Person' class below.
#
# The method 'get_grandchildren' is expected to return a LIST of the person's grandchildren.
#

class Person:
  def __init__(self, first, last):
    self.last_name = last
    self.first_name = first
    self.children = []
```

```
21
22    def add_child(self, child):
23      self.children.append(child)
24
25    # Write your code here
26    def get_grandchildren(self):
27          grandchildren = []
28          for child in self.children:
29              grandchildren.extend(child.children)
30          return grandchildren
31
32
33  if __name__ == '__main__':
34      fptr = open(os.environ['OUTPUT_PATH'], 'w')
35
36      inp = input()
37
38      if inp == '0':
39          johndoe = Person("John", "Doe")
40          janedoe = Person("Jane", "Doe")
41          jimmydoe = Person("Jimmy", "Doe")
42          johndoe.add_child(janedoe)
43          janedoe.add_child(jimmydoe)
44          assert johndoe.get_grandchildren() == [jimmydoe], "Test Case 0
    Failed"
45          result = "True"
46      elif inp == '1':
47          johndoe = Person("John", "Doe")
48          janedoe = Person("Jane", "Doe")
49          johndoe.add_child(janedoe)
50          assert johndoe.get_grandchildren() == [], "Test Case 1 Failed"
51          result = "True"
52      elif inp == '2':
53          splinter = Person("Master", "Splinter")
54          leo = Person("Leonardo", "Turtle")
55          raph = Person("Raphael", "Turtle")
56          don = Person("Donatello", "Turtle")
57          mich = Person("Michelangelo", "Turtle")
58          shredder = Person("The", "Shredder")
59          splinter.add_child(shredder)
60          shredder.add_child(leo)
61          shredder.add_child(raph)
62          shredder.add_child(don)
63          shredder.add_child(mich)
64          assert splinter.get_grandchildren() == [leo, raph, don, mich], "Test
    Case 2 Failed"
```

```
65          result = "True"
66      elif inp == '3':
67          johndoe = Person("John", "Doe")
68          janedoe = Person("Jane", "Doe")
69          jimmydoe = Person("Jimmy", "Doe")
70          livydoe = Person("Livy", "Doe")
71          babygronk = Person("Baby", "Gronk")
72          johndoe.add_child(janedoe)
73          janedoe.add_child(jimmydoe)
74          janedoe.add_child(livydoe)
75          livydoe.add_child(babygronk)
76          assert johndoe.get_grandchildren() == [jimmydoe, livydoe], "Test
    Case 3 Failed"
77          result = "True"
78      elif inp == '4':
79          johndoe = Person("John", "Doe")
80          janedoe = Person("Jane", "Doe")
81          jimmydoe = Person("Jimmy", "Doe")
82          johndoe.add_child(janedoe)
83          assert johndoe.get_grandchildren() == [], "Test Case 1 Failed"
84          janedoe.add_child(jimmydoe)
85          assert johndoe.get_grandchildren() == [jimmydoe], "Test Case 0
    Failed"
86          splinter = Person("Master", "Splinter")
87          leo = Person("Leonardo", "Turtle")
88          raph = Person("Raphael", "Turtle")
89          don = Person("Donatello", "Turtle")
90          mich = Person("Michelangelo", "Turtle")
91          shredder = Person("The", "Shredder")
92          splinter.add_child(shredder)
93          shredder.add_child(leo)
94          shredder.add_child(raph)
95          shredder.add_child(don)
96          shredder.add_child(mich)
97          assert splinter.get_grandchildren() == [leo, raph, don, mich], "Test
    Case 2 Failed"
98          livydoe = Person("Livy", "Doe")
99          babygronk = Person("Baby", "Gronk")
100         janedoe.add_child(livydoe)
101         livydoe.add_child(babygronk)
102         assert johndoe.get_grandchildren() == [jimmydoe, livydoe], "Test
    Case 3 Failed"
103         littledoe = Person("Little", "Doe")
104         jimmydoe.add_child(littledoe)
105         assert janedoe.get_grandchildren() == [littledoe, babygronk], "Test
    Case 4 Failed"
```

```
106             result = "True"
107         else:
108             # Ensure get_grandchildren() does not modify the people
109             gen_1 = Person("John", "Doe")
110             gen_2 = Person("Jane", "Doe")
111             gen_1.add_child(gen_2)
112             gen_3a = Person("Jimmy", "Doe")
113             gen_3b = Person("Livy", "Doe")
114             gen_2.add_child(gen_3a)
115             gen_2.add_child(gen_3b)
116             gen_4a = Person("Baby", "Gronk")
117             gen_3a.add_child(gen_4a)
118             gen_4b = Person("Little", "Doe")
119             gen_3b.add_child(gen_4b)
120             for person in [gen_1, gen_2, gen_3a, gen_3b, gen_4a, gen_4b]:
121                 person.get_grandchildren()
122             # Ensure no one was modified
123             assert gen_1.children == [gen_2], "Test Case 1 Failed"
124             assert gen_2.children == [gen_3a, gen_3b], "Test Case 2 Failed"
125             assert gen_3a.children == [gen_4a], "Test Case 3 Failed"
126             assert gen_3b.children == [gen_4b], "Test Case 4 Failed"
127             assert gen_4a.children == [], "Test Case 5 Failed"
128             assert gen_4b.children == [], "Test Case 6 Failed"
129             result = "True"
130 
131         fptr.write(result)
132         fptr.close()
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 0	Easy	Hidden	Success	0	0.0326 sec	10.6 KB
Testcase 1	Easy	Hidden	Success	0	0.0344 sec	10.5 KB
Testcase 2	Easy	Hidden	Success	0	0.0434 sec	10.4 KB

Testcase 3	Easy	Hidden	Success	0	0.0371 sec	10.6 KB
Testcase 4	Easy	Hidden	Success	20	0.0384 sec	10.5 KB
Testcase 5	Easy	Hidden	Success	0	0.0401 sec	10.6 KB

No comments.

6. Extend Linked List

Correct

Coding

Question description

Given the tail of a linked list and a list of values, write a function that adds each element in values as a node with value values[i] to the end of the linked list. The nodes added to the linked list should maintain the same order as the elements in value. The function does not need to return anything.

```
# linked list = a->b->c
# Example input: tail = c, values = ['d', 'e', 'f']
# expected output: a->b->c->d->e->f
```

Candidate's Solution

Language used: **Python 3**

```
1 #!/bin/python3
2
3 import math
4 import os
5 import random
```

```
import re
import sys


#
# Complete the 'extend_linked_list' function below.
#
# The function is not expected to return anything.
# The function accepts following parameters:
#  1. STRING tail
#  2. LIST(STRING) values
#

class Node:
 def __init__(self, value, next_node = None):
   self.value = value
   self.next = next_node

def extend_linked_list(tail, values):
    # Write your code here
    current = tail
    for value in values:
        new_node = Node(value)
        current.next = new_node
        current = new_node

if __name__ == '__main__':
    fptr = open(os.environ['OUTPUT_PATH'], 'w')

    instring = input()

    if len(instring) > 50:
        def parse_input(input_string):
            # Split the input string at commas to create major segments
            major_segments = input_string.split(',')
            # Trim spaces and organize into a clean list
            cleaned_segments = [segment.strip() for segment in
major_segments]
            return cleaned_segments
        parsed = parse_input(instring)

        def linked_list_to_string(head):
            values = []
            current = head
            while current:
```

```
                values.append(current.value)
                current = current.next
            return " ".join(values)

        def parse_and_extend_list(input_string):
            # Split the input string into initial elements and values to add
            parts = input_string.split(';')
            initial_elements = parts[0].strip().split()
            new_values = parts[1].strip().split()

            # Create the initial linked list from the first part
            if initial_elements:
                head = Node(initial_elements[0])
                current = head
                for element in initial_elements[1:]:
                    current.next = Node(element)
                    current = current.next
            tail = current  # The last node is the tail

            # Extend the linked list with new values
            extend_linked_list(tail, new_values)

            # Return the complete linked list as a string
            return linked_list_to_string(head)

        result = ""

        for part in parsed:
            result += parse_and_extend_list(part)
    else:
        def linked_list_to_string(head):
            values = []
            current = head
            while current:
                values.append(current.value)
                current = current.next
            return " ".join(values)

        def parse_and_extend_list(input_string):
            # Split the input string into initial elements and values to add
            parts = input_string.split(';')
            initial_elements = parts[0].strip().split()
            new_values = parts[1].strip().split()

            # Create the initial linked list from the first part
            if initial_elements:
```

```
97                  head = Node(initial_elements[0])
98                  current = head
99                  for element in initial_elements[1:]:
100                     current.next = Node(element)
101                     current = current.next
102             tail = current  # The last node is the tail
103
104             # Extend the linked list with new values
105             extend_linked_list(tail, new_values)
106
107             # Return the complete linked list as a string
108             return linked_list_to_string(head)
109
110         result = parse_and_extend_list(instring)
111
112     fptr.write(result)
113
114     fptr.close()
115
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 0	Easy	Sample	Success	0	0.0326 sec	10.5 KB
Testcase 1	Easy	Hidden	Success	0	0.0358 sec	10.4 KB
Testcase 2	Easy	Hidden	Success	0	0.0411 sec	10.4 KB
Testcase 3	Easy	Hidden	Success	20	0.0414 sec	10.4 KB

No comments.

7. Copy First N Nodes (LL)

Correct

Coding

Question description

Given the head of a linked list and a positive integer **n**, create a new linked list that copies the first **n** nodes of the original linked list. Return the head of the new list. If n is greater than or equal to the length of the original list, the function should copy the entire list. If the list contains no nodes, return `None`.
Example Input: **head=node1 , n=3**
Hint: Make sure you are creating new Nodes for the copy list -- if you re-use the same nodes, it's not a true copy!

Candidate's Solution

Language used: **Python 3**

```python
#!/bin/python3

import math
import os
import random
import re
import sys



#
# Complete the 'copy_first_n_nodes' function below.
#
# The function is expected to return a HEAD.
# The function accepts following parameters:
#  1. HEAD head
#  2. INTEGER n
#

class Node:
    def __init__(self, value, next_node = None):
        self.value = value
        self.next = next_node

def copy_first_n_nodes(head, n):
    # Write your code here
    if not head or n <= 0:
        return None
```

```
29
30      new_head = Node(head.value)
31      current_new = new_head
32      current_old = head.next
33      count = 1
34
35      while current_old and count < n:
36          new_node = Node(current_old.value)
37          current_new.next = new_node
38          current_new = new_node
39          current_old = current_old.next
40          count += 1
41      return new_head
42
43  if __name__ == '__main__':
44      fptr = open(os.environ['OUTPUT_PATH'], 'w')
45
46      # Helper function to convert str -> linked list
47      def str_to_linked_list(vals_str):
48          if vals_str == "None":
49              return None
50          vals = vals_str.split("->")
51          temp_head = Node("temp")
52          temp_curr = temp_head
53          for val in vals:
54              temp_curr.next = Node(val.strip())
55              temp_curr = temp_curr.next
56          return temp_head.next #Don't keep the temp head
57
58      # Helper function to convert linked list to str
59      def linked_list_to_str(head):
60          list_str = ""
61          curr = head
62          while curr:
63              list_str += curr.value
64              if curr.next:
65                  list_str += "->"
66              curr = curr.next
67          if len(list_str) == 0:
68              return "None"
69          return list_str
70
71      def verify_is_copy(orig_ll, new_ll):
72          orig_curr = orig_ll
73          new_curr = new_ll
74          while orig_curr and new_curr:
```

```
            if orig_curr == new_curr: # Compare NODES, not values
                return False
            orig_curr = orig_curr.next
            new_curr = new_curr.next
        return True

    # Read and convert test input
    n = int(input())
    head = str_to_linked_list(input())

    # Call the function
    answer = copy_first_n_nodes(head, n)

    # Turn the list into a string
    list_str = linked_list_to_str(head)
    answer_str = linked_list_to_str(answer)

    # Bundle result in format <result linked list>\n<original linked list>
    result = answer_str + "\n" + list_str

    # Verify the copy
    is_copy = verify_is_copy(head, answer)
    if not is_copy:
        result += "\nNot a copy - the same nodes were found in both input and output"

    fptr.write(result)
    fptr.close()
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 0	Easy	Sample	Success	0	0.0421 sec	10.4 KB
Testcase 1	Easy	Hidden	Success	5	0.0422 sec	10.4 KB
Testcase 2	Easy	Hidden	Success	5	0.0381 sec	10.4 KB

Testcase 3	Easy	Hidden	Success	5	0.0327 sec	10.4 KB
Testcase 4	Easy	Hidden	Success	5	0.0335 sec	10.4 KB

No comments.