CS 61A Fall 2020

Structure and Interpretation of Computer Programs

Quiz 6 Solutions

INSTRUCTIONS

- Please review this worksheet before the exam prep session. Coming prepared will help greatly, as the TA will be live solving without allocating much time for individual work.
- Either Sean or Derek will be on video live solving these questions. The other TA will be answering questions in the chat. It is in your best interest to come prepared with **specific** questions.
- This is not graded, and you do not need to turn this in to anyone.
- Fall 2020 students: the boxes below are an artifact from more typical semesters to simulate exam environments. Obviously this doesn't apply to this semester's exams, but we just kept the fields to keep our materials looking professional :) Feel free to ignore them.
- For multiple choice questions, fill in each option or choice completely.
  - ☐ means mark **all options** that apply
  - ◯ means mark a **single choice**

Last name	
First name	
Student ID number	
CalCentral email (_@berkeley.edu)	
Discussion Section	___ ___ ___
All the work on this exam is my own. **(please sign)**	

1. **Quarantine Dieting**

For each of the expressions in the table below, write the output displayed by the interactive Python interpreter when the expression is evaluated. The output may have multiple lines. If an error occurs, write "Error", but include all output displayed before the error. If a function value is displayed, write "Function".

Assume that you have started `python3` and executed the following statements. Also assume that **effects from a previous subpart persist to future subparts**.

```
class VendingMachine:
    k = 0
    def __init__(self, k, v):
        self.soda = JunkDrink(self)
        self.k = k
        if v:
            print(isinstance(self.soda.machine, VendingMachine))

class JunkDrink:
    def __init__(self, machine):
        self.machine = machine

a = VendingMachine(1, False)
b = VendingMachine.__init__(a, 2, False)
VendingMachine.__init__(VendingMachine, 10, False)
```


Expression	Interactive Output
`a.k`	`2`
`b.k`	`Error`
`VendingMachine.k`	`10`
`isinstance(b, VendingMachine)`	`False`
`a is a.soda.machine`	`True`
`VendingMachine is a.soda.machine`	`False`
`c = VendingMachine` `c.__init__(c, 11, True)`	`False`
`c.soda.machine is VendingMachine`	`True`
`a.k == c.k`	`False`
`c.soda.machine.k`	`11`

2. **COVID Party**

Fill in each of the blanks in the code such that the expressions and outputs in the table below are consistent. Assume that **effects from a previous subpart persist to future subparts**. The `Link` class is provided on the next page. **Do not** use the `Link` constructor unless it is already provided in the skeleton

```
class Party:
    guests = Link.empty

    def __init__(self, time):
        Party.guests = Link(time + 1, Party.guests)

    def attend(self):
        self.guests.rest = Link(self.guests.rest)
        return (self.guests is Party.guests) and Party.guests

class Costume(Party):
    def __init__(self, bow, tie):
        Party.guests.rest, self.ie = Link(bow), Link(self)

    def attend(self):
        print(repr(self.ie)) # A: Costume(5, 6).attend() would have been "whacky" with no quotes
        Party.attend = lambda self: Party(9).guests
    def __repr__(self):
        print("Nice")
        return "Costume"
```


Expression	**Interactive Output**
`Link(1, Link.empty)`	Link(1)
`Link(1, Link(2))`	Link(1, Link(2))
`Party(1).guests`	`Link(2)`
`Party(3).attend()`	`Link(4, Link(Link(2)))`
`Costume(5, 6).attend()`	`Nice` `Link(Costume)`
`Party(7).attend()`	`Link(10, Link(8, Link(4, Link(5))))`

3. **Link class**

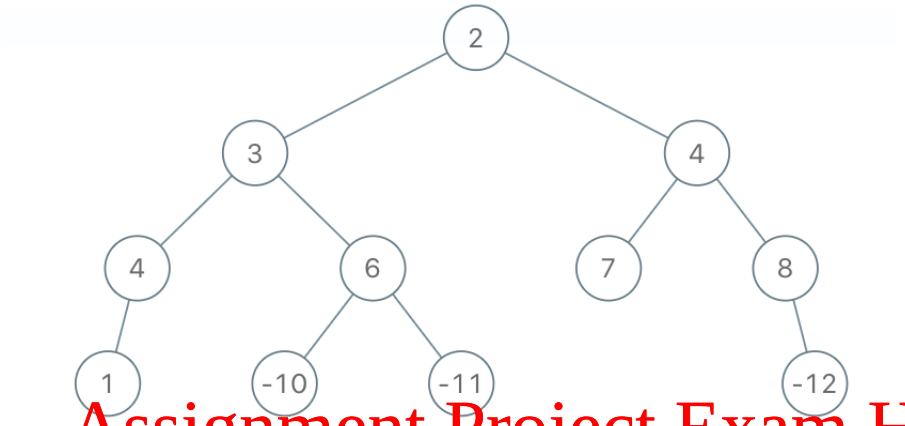
```
class Link:
    empty = ()
    def __init__(self, first, rest=empty):
        assert rest is Link.empty or isinstance(rest, Link)
        self.first = first
        self.rest = rest

    def __repr__(self):
        if self.rest:
            rest = ", " + repr(self.rest)
        else:
            rest = ""
        return "Link(" + repr(self.first) + rest + ")"

    def __str__(self):
        return "whacky"
```

4. **Max Path**

(a) Given a tree, find the maximum path sum. A path sum is the sum of a sequence of connected nodes. The path can start anywhere (not necessarily the root or leaf), end anywhere, and move from parent to child or child to parent. The only constraint is that the `i`th node in the sequence has to be either the parent or direct child of the `(i - 1)`th node in the sequence. As an example, the function should return 23 for the following tree. The path is 6 => 3 => 2 => 4 => 8. Assume each node has at most 2 branches for simplicity.




```
def maxPathSum(tree):
    # Every node is either
    # a) starting point of a path
    # b) an intermediate node from a left child going upward
    # c) an intermediate node from a right child going upward
    # d) the connector node for a path between left and right
    # e) not in the max path at all
    tree.options = [tree.label, tree.label, tree.label] # encodes options b through d
    if tree.is_leaf():
        return tree.label

    b = tree.branches # this just makes it easier to fit solution on the page
    x = maxPathSum(b[0]) # (e) for `tree`
    p = max(max(b[0].options[:-1]), b[0].label) # checks (a)-(c) in left subtree
    tree.options[0] = tree.label + p # (b) for `tree`

    if len(b) == 2:
        x = max(maxPathSum(b[1]), x) # checks (e) for `tree`
        q = max(max(b[1].options[:-1]), b[1].label) # checks (a)-(c) in right subtree
        tree.options[1] = tree.label + q # (c) for `tree`
        tree.options[2] = tree.label + p + q # (d) for `tree`
    return max(x, tree.label, max(tree.options)) # checks (a)-(e) for `tree`
```


An alternate and easier to read solution is posted here: https://piazza.com/class/kdz4wzqnb6052o?cid=2558