CS 61A
Fall 2016

Structure and Interpretation of Computer Programs

INSTRUCTIONS

- You have 2 hours to complete the exam.

- The exam is closed book, closed notes, closed computer, closed calculator, except one hand-written 8.5" × 11" crib sheet of your own creation and the official CS 61A study guides.

- Mark your answers **on the exam itself**. We will *not* grade answers written on scratch paper.

Last name	
First name	
Student ID number	
CalCentral email (`_@berkeley.edu`)	
TA	
Name of the person to your left	
Name of the person to your right	
All the work on this exam is my own. **(please sign)**	

1. (8 points) **Halloween** *(All are in Scope: WWPD, Object-Oriented Programming, Linked Lists)*

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. The Link class appears on page 2 of the midterm 2 study guide.

The first two rows have been provided as examples.

Recall: The interactive interpreter displays the value of a successfully evaluated expression, unless it is None.

Assume that you have started python3 and executed the following statements:

```python
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

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

    def attend(self):
        print(repr(self.ie))
        Party.attend = lambda self: Party(9).guests

    def __repr__(self):
        print('Nice')
        return 'Costume'
```
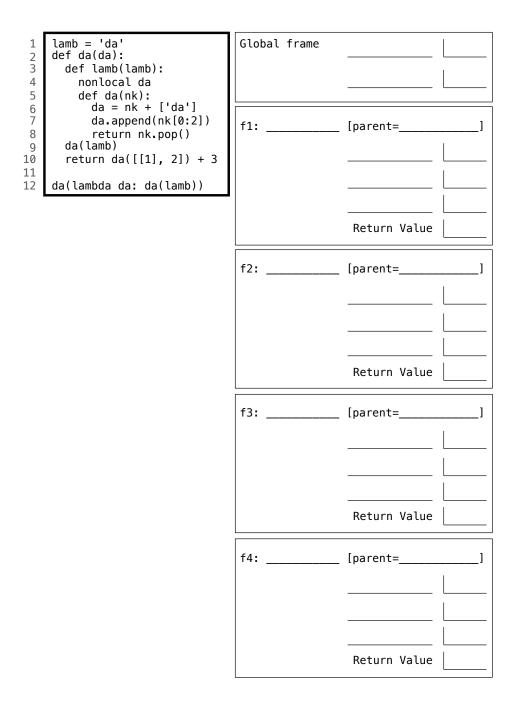
Expression	Interactive Output
Link(1, Link.empty)	Link(1)
Link(1, Link(2))	Link(1, Link(2))
Party(1).guests	Link(2)
Party(3).attend()	Link(4, Link(Link(2)))
Costume(5, 6).attend()	Nice Link(Costume)
Party(7).attend()	Link(10, Link(8, Link(4, Link(5))))

2. (8 points)   A List with a Twist *(At least one of these is out of Scope: Environment Diagrams, Nonlocal, Python Lists)*

Fill in the environment diagram that results from executing the code below until the entire program is finished, an error occurs, or all frames are filled. *You may not need to use all of the spaces or frames.*

A complete answer will:

- Add all missing names and parent annotations to frames.
- Add all missing values created or referenced during execution.
- Show the return value for each local frame.
- Use box-and-pointer notation for list values. You do not need to write index numbers or the word "list".

```
1   lamb = 'da'
2   def da(da):
3     def lamb(lamb):
4       nonlocal da
5       def da(nk):
6         da = nk + ['da']
7         da.append(nk[0:2])
8         return nk.pop()
9       da(lamb)
10    return da([[1], 2]) + 3
11
12  da(lambda da: da(lamb))
```

Global frame

_____ |____|

_____ |____|

f1: _____   [parent=_____]

_____ |____|

_____ |____|

_____ |____|

Return Value |_____|

f2: _____   [parent=_____]

_____ |____|

_____ |____|

_____ |____|

Return Value |_____|

f3: _____   [parent=_____]

_____ |____|

_____ |____|

_____ |____|

Return Value |_____|

f4: _____   [parent=_____]

_____ |____|

_____ |____|

_____ |____|

Return Value |_____|

3. (6 points) Return Policy *(At least one of these is out of Scope: Nonlocal, Python Lists, Mutability)*

Implement `quota`, which takes a one-argument function `f` and a non-negative integer `limit`. The function it returns has the same behavior as `f`, except that each value is only returned up to `limit` times. After that, the function returns an "Over quota" message instead, and the `limit` is decreased by 1 for future calls.

```python
def quota(f, limit):
    """A decorator that limits the number of times a value can be returned.

    >>> square = lambda x: x * x
    >>> square = quota(square, 3)
    >>> square(6)                    # 1st call with return value 36
    36
    >>> [square(5) for x in range(3)] # 3 calls when the limit is 3
    [25, 25, 25]
    >>> square(5)                    # 4th call with return value 25
    'Over quota! Limit is now 2'
    >>> square(-6)                   # 2nd call with return value 36
    36
    >>> square(-6)                   # 3rd call when the limit is 2
    'Over quota! Limit is now 1'
    >>> square(7)                    # 1st call when the limit is 1
    49
    >>> square(5)                    # 5th call with return value 25
    'Over quota! Limit is now 0'
    """

    values = []

    def limited(n):

        _____

        y = _____

        count = len(_____)

        values.append(y)

        if _____:

            return _____

        limit = _____

        return 'Over quota! Limit is now ' _____

    return limited
```

4. (6 points)   A Classy Election *(All are in Scope: Object-Oriented Programming)*

Implement the `VotingMachine` and `Ballot` classes based on the doctest below. The voting machine must determine which choice has the most votes (the `winner`) and detect if a ballot is used more than once. In case of a tie, the `winner` among choices with maximal votes is the one that most recently received a vote. `Ballot.vote` takes a string, and a `VotingMachine` must handle an arbitrary number of choices.

```python
class VotingMachine:
    """A machine that creates and records ballots.

    >>> machine = VotingMachine(4)
    >>> a, b, c, d = machine.ballots
    >>> d.vote('Bruin')
    'Bruin is winning'
    >>> b.vote('Bruin')
    'Bruin is winning'
    >>> c.vote('Bear')
    'Bear is losing'
    >>> a.vote('Bear')
    'Bear is winning'
    >>> c.vote('Tree')
    'Fraud: multiple votes from the same ballot!'
    >>> machine.winner
    'Bear'
    """
    def __init__(self, k):
        self.ballots =   [_____ for i in range(k)]
        self.votes = {}

    def record(self, ballot, choice):
        if ballot.used:
            return 'Fraud: multiple votes from the same ballot!'


        _____


        self.votes[choice] = _____ + 1


        if _____:
            return choice + ' is losing'
        else:


            _____
            return choice + ' is winning'
class Ballot:

    _____

    def __init__(self, machine):
        self.machine = machine
    def vote(self, x):

        return _____
```

5. (6 points) Trick or Tree *(All are in Scope: Trees, Linked Lists)*

Implement `path`, which takes a linked list `s` and a `Tree` instance `t`. It returns whether `s` is a path from the root of `t` to some leaf. The `Tree` and `Link` classes are on page 2 of the midterm 2 study guide.

Restrictions:

- You may not call the built-in `len` function on a linked list or invoke its `__len__` method.
- You may not apply element selection (e.g., `s[2]`) on a linked list or invoke its `__getitem__` method.

```python
def path(s, t):
    """Return whether Link S is a path from the root to a leaf in Tree T.

    >>> t = Tree(1, [Tree(2), Tree(3, [Tree(4), Tree(5)]), Tree(6)])
    >>> a = Link(1, Link(3, Link(4)))  # A full path
    >>> path(a, t)
    True
    >>> b = Link(1, Link(3))           # A partial path
    >>> path(b, t)
    False
    >>> c = Link(1, Link(2, Link(7)))  # A path and an extra value
    >>> path(c, t)
    False
    >>> d = Link(3, Link(4))           # A path of a branch
    >>> path(d, t)
    False
    """

    if _____:

        return False


    if _____:

        return True


    return _____([_____ for b in t.branches])
```

6. (6 points)   Left it Right There

(a) **(4 pt)** *(All are in Scope: Trees)* Implement `binary`, which takes a list `s` of unique integers. It returns a *binary search tree* containing all of those integers, represented as a `BTree` instance or `BTree.empty`. The values in any path of this tree must appear in the same order as they did in `s`. The `BTree` class is on page 2 of the midterm 2 study guide.

```python
def binary(s):
    """Construct a binary search tree from S for which all paths are in order.

    >>> binary([3, 5, 1])
    BTree(3, BTree(1), BTree(5))
    >>> binary([4, 3, 7, 6, 2, 9, 8])
    BTree(4, BTree(3, BTree(2)), BTree(7, BTree(6), BTree(9, BTree(8))))
    """

    assert len(s) == len(set(s)), 'All elements of s should be unique'


    if _____:


        return _____


    root = _____


    left = _____


    right = _____


    return BTree(root, binary(left), binary(right))
```

(b) **(1 pt)** *(At least one of these is out of Scope: Asymptotic Notation)* Circle the Θ expression that describes the **smallest** possible height of the tree returned by `binary(s)` for a list `s` of length n. The height of a tree is the length of the longest path from its root to a leaf.

$\Theta(1)$ $\qquad$ $\Theta(\log n)$ $\qquad$ $\Theta(n)$ $\qquad$ $\Theta(n^2)$ $\qquad$ $\Theta(2^n)$ $\qquad$ None of these

(c) **(1 pt)** *(At least one of these is out of Scope: Asymptotic Notation)* Circle the Θ expression that describes the **largest** possible height of the tree returned by `binary(s)` for a list `s` of length n. The height of a tree is the length of the longest path from its root to a leaf.

$\Theta(1)$ $\qquad$ $\Theta(\log n)$ $\qquad$ $\Theta(n)$ $\qquad$ $\Theta(n^2)$ $\qquad$ $\Theta(2^n)$ $\qquad$ None of these

7. (10 points)   Summer Camp

(a) (6 pt) *(All are in Scope: Tree Recursion, Python Lists)* Implement `sums`, which takes two positive integers `n` and `k`. It returns a list of lists containing all the ways that a list of `k` positive integers can sum to `n`. Results can appear in any order.

```python
def sums(n, k):
    """Return the ways in which K positive integers can sum to N.

    >>> sums(2, 2)
    [[1, 1]]
    >>> sums(2, 3)
    []
    >>> sums(4, 2)
    [[3, 1], [2, 2], [1, 3]]
    >>> sums(5, 3)
    [[3, 1, 1], [2, 2, 1], [1, 3, 1], [2, 1, 2], [1, 2, 2], [1, 1, 3]]
    """


    if _____:


        return _____


    y = []


    for x in _____:


        y.extend([_____ for s in sums(_____)])


    return y
```

(b) (4 pt) *(All are in Scope: Python Lists, Lambda Expressions)* *Why so many lines?* Implement `f` and `g` for this alternative version of the `sums` function.

```python
f = lambda x, y: (x and [_____ for z in y] + f(_____, _____)) or []


def sums(n, k):
    """Return the ways in which K positive integers can sum to N."""


    g = lambda w: (w and f(_____)) or [[]]


    return [v for v in g(k) if sum(v) == n]
```