CS 61A Fall 2012 — Structure and Interpretation of Computer Programs

ALTERNATE MIDTERM 2 SOLUTIONS

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 two official 61A midterm study guides attached to the back of this exam.
- Mark your answers ON THE EXAM ITSELF. If you are not sure of your answer you may wish to provide a *brief* explanation.

Last name	
First name	
SID	
Login	
TA & section time	
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)**	

For staff use only

Q. 1	Q. 2	Q. 3	Q. 4	Total
/16	/12	/14	/8	/50

THIS PAGE INTENTIONALLY LEFT BLANK

1. (16 points) Expressionism

(a) (8 pt) For each of the following expressions, write the `repr` string of the value to which the expression evaluates. Special cases: If an expression evaluates to a function, write FUNCTION. If evaluation would never complete, write FOREVER. None of these expressions cause an error.

Assume that the expressions are evaluated in order. Evaluating the first may affect the value of the second, etc.

Assume that you have started Python 3 and executed the following statements:

```
def fruit(y):
    def ninja(angry):
        nonlocal y
        if y < len(angry):
            return angry
        y = y - 2
        return pig(angry)

    def pig(bird):
        bird.append(y)
        return ninja(bird)

    return pig

slingshot = fruit(5)
green = lambda x: fruit(x)([x])
```

Expression	Evaluates to
`5*5`	25
`slingshot([1, 2, 3, 4])`	[1, 2, 3, 4, 5, 3]
`green(5)`	[5, 5, 3, 1]
`slingshot([6, 7])`	[6, 7, 3, 1]
`green(8)`	[8, 8, 6, 4, 2]

(b) (8 pt) For each of the following expressions, write the `repr` string of the value to which the expression evaluates. Special cases: If an expression evaluates to a function, write FUNCTION. If evaluation would never complete, write FOREVER. None of these expressions cause an error.

Assume that the expressions are evaluated in order. Evaluating the first may affect the value of the second, etc.

Assume that you have started Python 3 and executed the following statements:

```
class Student(object):
    def __init__(self, s):
        if len(s) < 2:
            self.s = s
        else:
            self.s = Student(s[1:])

    def __repr__(self):
        return 'Student(' + repr(self.s) + ')'

    def learn(self):
        if hasattr(self, 'teach'):
            return self.teach()
        while type(self.s) == Student:
            self.s = self.s.s
        return self.s

class Teacher(Student):
    def teach(self):
        return 'Good Job'

luke = Student([1, 3])
yoda = Teacher([5, luke])
```

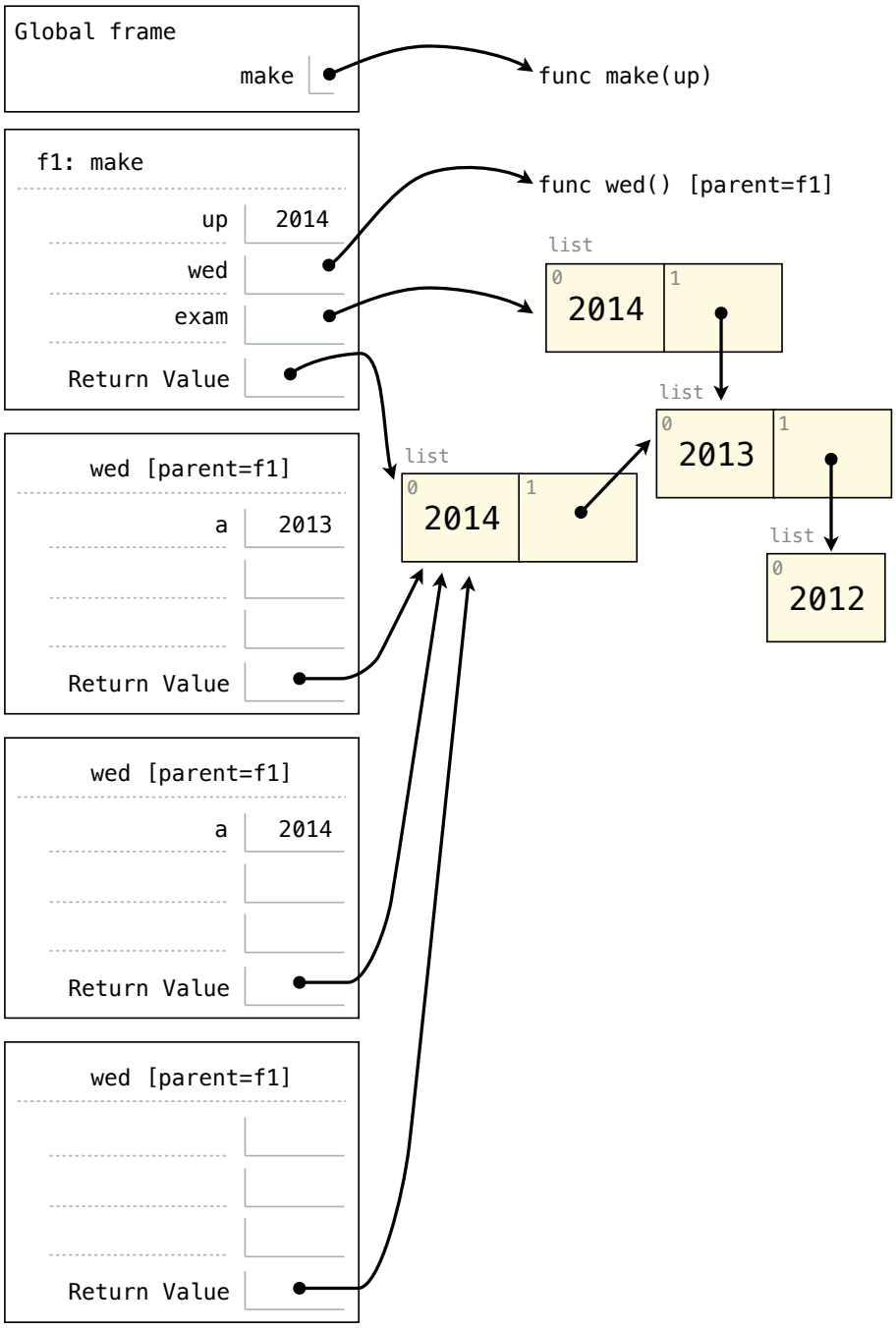
Expression	Evaluates to
`5*5`	25
`luke.learn()`	[3]
`luke`	Student([3])
`yoda`	Student(Student([Student([3])]))
`Student.learn(yoda)`	'Good Job'

2. (12 points) Picture Frame

(a) (5 pt) 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, labels, and parent annotations to all local frames.
- Add all missing values created during execution.
- Show the return value for each local frame.

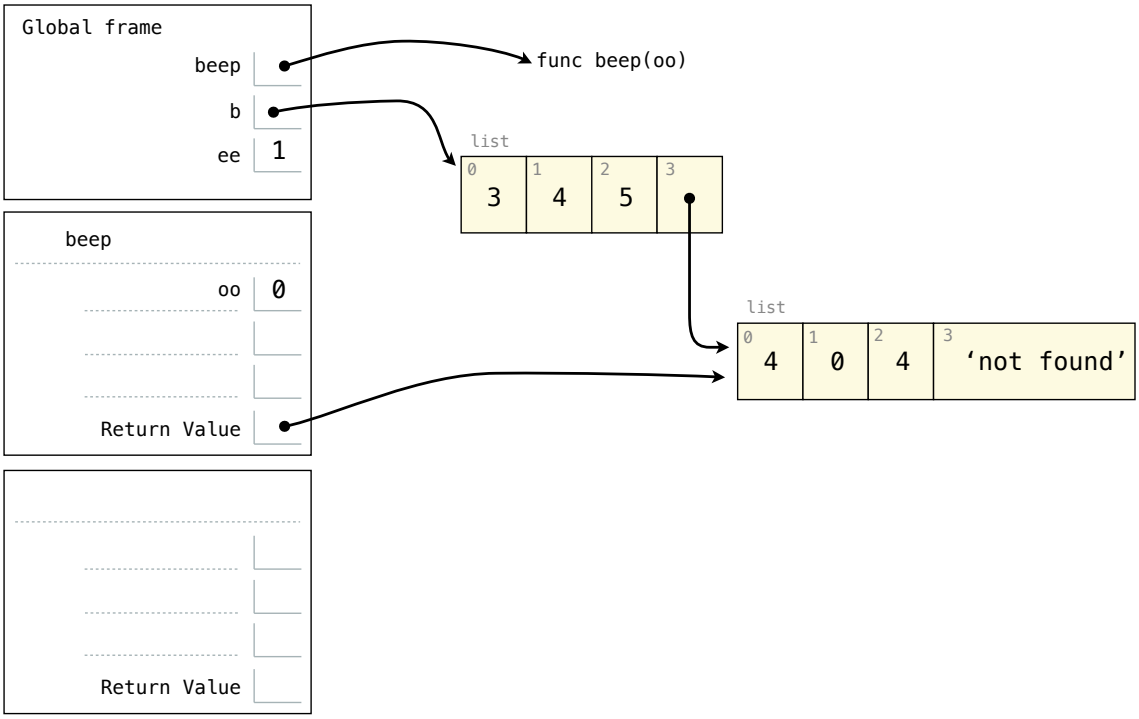
```
def make(up):
    exam = [2012]
    def wed():
        nonlocal exam
        if exam[0] >= up:
            return list(exam)
        a = exam[0] + 1
        exam = [a, exam]
        return wed()
    return wed()

make(2014)
```



(b) (1 pt) Circle **True** or **False**: This diagram would change if the `nonlocal` statement were removed.

(c) (5 pt) 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, labels, and parent annotations to all local frames.
- Add all missing values created during execution.
- Show the return value for each local frame.




```
def beep(oo):
    b[b[oo]] = [b[ee], oo, b[ee]]
    return b[len(b)-1]

b = list(range(3, 7))
ee = 1
beep(0).append('not found')
```

(d) (1 pt) What will `print(b)` output after executing this code?

`[3, 4, 5, [4, 0, 4, 'not found']]`

3. (14 points) Objets d'Art

(a) (6 pt) Cross out whole lines in the implementation below so that the doctests for `Vehicle` pass. In addition, cross out all lines that **have no effect**. Don't cross out docstrings, doctests, or decorators.

```
class Vehicle(object):
    """
    >>> c = Car('John', 'CS61A')
    >>> c.drive('John')
    John is driving
    >>> c.drive('Jack')
    Car stolen: John CS61A
    >>> c.pop_tire()
    3
    >>> c.pop_tire()
    2
    >>> c.fix()
    >>> c.pop_tire()
    3
    """
    def __init__(self, owner):
        self.owner = owner
    def move(self):
        print(self.owner + ' is driving')

class Car(Vehicle):
    tires = 4
    def __init__(self, owner, license_plate):
        Vehicle.__init__(self, owner)
        self.plate = license_plate
        self.tires = Car.tires  # This line is optional
    def drive(self, person):
        if person != self.owner:
            print('Car stolen: ' + self.identification)
        else:
            Car.move(self)
    @property
    def identification(self):
        return self.owner + ' ' + self.plate
    def pop_tire(self):
        self.tires -= 1
        return self.tires
    def fix(self):
        setattr(self, 'tires', type(self).tires)
```

(b) (4 pt) The `max_path` function takes an instance of the `Tree` class from Study Guide 2. It is meant to return the maximal sum of internal `entry` values on a path from the *root* to a *leaf* of the tree.

```
def max_path(tree):
    """Return the sum of entries in a maximal path from the root to a leaf.

    >>> max_path(Tree(3, Tree(4), Tree(-2, Tree(8), Tree(3))))
    9
    >>> max_path(Tree(9, None, Tree(1, Tree(-2, Tree(5), Tree(2)), None)))
    13
    """
    paths = [0]
    if tree.right is not None:
        paths.append(max_path(tree.right))
    if tree.left is not None:
        paths.append(max_path(tree.left))
    if min(paths) < 0:
        return tree.entry + max([p for p in paths if p < 0])
    return tree.entry + max(paths)
```

Circle **True** or **False** to indicate whether each of the following statements about `max_path` is true.

i. (**True**/**False**) It returns the correct result for all doctests shown.

ii. (**True**/**False**) It returns the correct result for all valid trees with integer entries.

(c) (4 pt) Define a function `coerce` that takes a `Tree` instance and returns a dispatch dictionary representing the same tree using messages `left`, `right`, and `entry`. The base case has been provided for you.

```
def coerce(tree):
    """Return a dispatch dictionary representing the tree.

    >>> t = Tree(4, None, Tree(5, None, Tree(1, Tree(3, Tree(5), Tree(2)))))
    >>> d = coerce(t)
    >>> d['entry']
    4
    >>> d['left'] is None
    True
    >>> d['right']['right']['left']['entry']
    3
    """
    if tree is None:
        return None

    return {'entry': tree.entry,
            'left': coerce(tree.left),
            'right': coerce(tree.right)}
```

4. (8 points) Form and Function

(a) (4 pt) You have been hired to work on AI at UnitedPusherElectric, the leading manufacturer of Pusher Bots. The latest model, PusherBot 5, keeps pushing people down stairs when it gets lost. Fix it!

Assume that you have an abstract data type `position` that combines `x` and `y` coordinates (in meters).

```
>>> pos = position(3, 4)
>>> x(pos)
3
>>> y(pos)
4
```

`pathfinder` should return a `visit` function that takes a `position` argument. `visit` returns `True` unless:

i. Its argument position is more than 6 meters from `position(0, 0)`, or

ii. Its argument position has been visited before.

The implementation below is incorrect. Cross out each line (or part of a line) that must change and write a revised version next to it, so that `pathfinder` is correct **and** does not depend on the implementation of `position`. Assume your corrections have the same indentation as the lines they replace. You may not add or remove lines. Make as few changes as necessary.

```
from math import sqrt
def equal(position, other):
    return x(position) == x(other) and y(position) == y(other)

def pathfinder():
    """Return a visit function to help with path-finding.
    >>> visit1, visit2 = pathfinder(), pathfinder()
    >>> visit1(position(3, 4))
    True
    >>> visit1(position(5, 12)) # Too far away
    False
    >>> visit1(position(3, 4))  # Already visited
    False
    >>> visit2(position(3, 4))
    True
    """
    visited = [] # was ()

    def visit(pos):

        if sqrt( x(pos)*x(pos) + y(pos)*y(pos) ) > 6:

            return False

        for p in visited: # was visit:

            if equal(p, pos): # was p == pos:

                return False # was True

        visited.append(pos)

        return True

    return visit # was visited
```

(b) (4 pt) Fill in missing expressions in the implementation for `subwords`, which lists all of the `n` letter subwords of a given word `w`. A subword consists of some subset of the letters in a word, in their original order. You may assume that the word has no repeated letters. Some hints about string slicing appear in the doctest.

```
def subwords(w, n):
    """List all subwords of word w that have length n.

    >>> 'conc' + 'atenate'
    'concatenate'
    >>> w = 'spot'
    >>> w[len(w):]
    ''
    >>> subwords(w, 2)
    ['sp', 'so', 'st', 'po', 'pt', 'ot']
    >>> subwords(w, 3)
    ['spo', 'spt', 'sot', 'pot']
    """
    if n == 0:



        return ['']

    if w == '':


        return []



    r = [w[0] + x for x in subwords(w[1:], n-1)]



    return r + subwords(w[1:], n)
```

(c) (0 pt) Draw a picture of PusherBot 5.