CS 61A Fall 2014

Structure and Interpretation of Computer Programs

Practice Final Solutions

INSTRUCTIONS

- You have 3 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 three official 61A midterm study guides attached to the back of this exam (not included for the practice 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	Q. 5	Total
/20	/20	/14	/20	/6	/80

1. (20 points) Say What?

(a) (6 pt) For each of the following call expressions, write would be output by the interactive Python interpreter. The first two rows have been provided as examples.

In the column labeled **Interactive Output**, write all output that would be displayed during an interactive session, after entering each call expression. This output may have multiple lines. Whenever the interpreter would report an error, write Error. You *should* include any lines displayed before an error.

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

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

```
cookies = [ "cookies", "monster"]
monster = ["cookies", cookies]


def more(monster, cookies):
    more = monster + cookies
    if monster[0][0] == cookies[0][0]:
        monster[1] = cookies[1:]
    return more
```

Expression	Interactive Output
`square(5)`	25
`1 / 0`	Error
`[x % 3 for x in [1, 5, 6] if x % 2 != 0]`	[1, 2]
`[cookies[1], monster[2]]`	Error
`more(cookies, monster)`	['cookies', 'monster', 'cookies', ['cookies', [[...]]]]
`monster`	['cookies', ['cookies', [[...]]]]
`cookies[1][0]`	['cookies', [[...]]]
`cookies is monster[1]`	True

(b) (8 pt) (Similar to Fall 2013 Final) Each of the following expressions evaluates to a `Stream` instance. For each one, write the values of the three elements in the stream. The first value of the first stream is filled in for you.

Assume that you have started Python 3 and executed the following statements, in addition to the `Stream` class statement on your final study guide.

```
def q(t):
    def compute_rest():
        return q(t.rest.rest)
    return Stream(t.rest.first + t.first, compute_rest)

s = lambda t: Stream(t, lambda: s(t-1))
t = s(5)
u = q(t)
```

Stream	Has the first three elements
`t`	5, _____4_____, _____3_____
`u`	_____9_____, _____5_____, _____1_____
`q(u)`	_____14_____, _____-2_____, _____-18_____

(c) (6 pt) (Similar to Fall 2013 Final) For each of the following Scheme expressions, write the Scheme value to which it evaluates. The first three rows are completed for you. If evaluation causes an error, write ERROR. If evaluation never completes, write FOREVER. **Hint**: No dot should appear in a well-formed list.

Assume that you have started the Project 4 Scheme interpreter and evaluated the following definitions.

```
(define b (lambda () (b)))
```

Expression	Evaluates to
`(* 5 5)`	25
`'(1 2 3)`	(1 2 3)
`(/ 1 0)`	ERROR
`'((1 . (2 3 . (4))) . 5)`	((1 2 3 4) . 5)
`(cons 1 (list 2 3 '(car (4 5))))`	(1 2 3 (car (4 5)))
`((lambda () (b)))`	FOREVER
`((lambda (x) '(x 2)) 4)`	(x 2)

2. (20 points) Such Enviroment Diagram

(a) (10 pt) (From CS61A Practice Problems - http://cs61a.org/problems) Draw 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.

A complete answer will:

- Add all names, labels, and parent annotations to all local frames.
- Add all values created during execution.
- Show the return value for each local frame.

```
wow = 6
def much(wow):
    if much == wow:
        such = lambda wow: 5
        def wow():
            return such
        return wow
    such = lambda wow: 4
    return wow()
wow = much(much(much))(wow)


# See Python Tutor
```

(b) (10 pt) (Based off Lecture 15) Draw the environment diagram that results from executing the code below until the entire program is finished, an error occurs, or all frames are filled.

- Add all names, labels, and parent annotations to all local frames.
- Add all values created during execution.
- Show the return value for each local frame.

```
def f(x):
    x = 4
    w = [x, 5]
    def g(y):
        def h(z):
            nonlocal x
            x = x + y
            w[0] = x
            return x + y + z
        return h
    return g
a = f(1)
b = a(2)
c = b(3) + b(4)

# See Python Tutor
```

3. (14 points) Dressing Classy

(a) (9 pt) For the following class definition, cross out any incorrect or unnecessary **lines** in the following code so that the doctests pass. **Do not cross out class declarations or doctests.** You can cross out any other line of code, including method declarations, and your final code should be as short as possible.

```
class Wardrobe:
    """
    >>> shirts = [Shirt("green"), Shirt("blue")]
    >>> winter = Wardrobe(shirts)
    >>> winter.count
    2
    >>> winter.wear()
    'green'
    >>> shirts[0].stains
    1
    >>> winter.count
    1
    """

    def __init__(self, shirtsList):
        self.shirts = shirtsList

    @property
    def count(self):
        sum = 0
        for i in self.shirts:
            if i.washed:
                sum += 1
        return sum

    def wear(self):
        for s in self.shirts:
            if s.washed:
                s.washed = False
                s.stains += 1
                return s.color
class Cloth:
    def __init__(self, color):
        self.color = color
        self.washed = True

class Shirt(Cloth):
    def __init__(self, color):
        self.stains = 0
        Cloth.__init__(self, color)
```

(b) (5 pt) Implement an iterator for the wardrobe that only iterates over the shirts that have been washed. Fill in the blanks in the implementation of the **__iter__** method for the **Wardrobe** class.

For this part, you may assume that the **Wardrobe** class stores the list of shirts in an instance attribute called **shirts** and that the **wear** method of the **Wardrobe** class has been correctly implemented. Each instance of a shirt has an boolean attribute **washed** that indicates whether it has been washed.

```
Class Wardrobe:
    """
    >>> shirts = [Shirt("yellow"), Shirt("green"), Shirt("blue")]
    >>> winter = Wardrobe(shirts)
    >>> winter.wear()
    'yellow'
    >>> for option in winter:
    ...   print(option)
    ...
    'green'
    'blue'
    >>> len(winter.shirts)
    3
    """
    """ Assume the methods from Part A are implemented """
    def __iter__(self):

        for shirt in self.shirts

           if shirt.washed:

               yield shirt.color
```

4. (20 points) Where's Groot?

(a) (7 pt) Groot has hidden himself in a `Tree` object. Define a function `wheres_groot`, which takes in a tree object (that may or may not contain the string `'Groot'` as a node), and returns `'Groot'` if it exists somewhere in the tree, and `'Nowhere'` if it does not. Here's the implementation of the `Tree` and `Link` classes.

```
nil = "Troog"
class Link:
    def __init__(self, first, rest=nil):
        self.first = first
        self.rest = rest

class Tree:
    def __init__(self, entry, branches=nil):
        self.entry = entry
        self.branches = branches

def wheres_groot(gtree):
    """
    >>> evil = Tree("Ronan", Link(Tree("Nebula"), Link(Tree("Korath"))))
    >>> good = Tree("Gamora", Link(Tree("Rocket", Link(Tree("Groot")))))
    >>> grooted = tree("Star-Lord", Link(evil, Link(good)))
    >>> wheres_groot(evil)
    'Nowhere'
    >>> wheres_waldo(grooted)
    'Groot'
    """

    if gtree.entry == "Groot":

        return "Groot"

    children = gtree.branches

    while children != nil:

        if wheres_groot(children.first) == 'Groot'

            return 'Groot'

        children = children.rest

    return "Nowhere"
```

(b) **(8 pt)** Implement **deep-reverse**, which takes in a Scheme list and reverses the entire list, all sublists, all sublists within that, etc. **Hint**: You can use the **list?** operator to determine whether something is a list.

```
STk> (deep-reverse '(foo bar baz))
(baz bar foo)
STk> (deep-reverse '(1 (2 3) (4 (5 6) 7)))
((7 (6 5) 4) (3 2) 1)
```

```
(define (deep-reverse lst)
  (cond ((null? lst) '())
        ((list? (car lst))
          (append (deep-reverse (cdr lst))
                  (list (deep-reverse (car lst))) ))
        (else
          (append (deep-reverse (cdr lst))
                  (list (car lst)) )) ))
```

(c) **(5 pt)** Terminal SQL

You've decided to use the SQL Travel Agency to plan a vacation. Your travel agent hands you the following table called `airport`.

```
create table airport as
  select "SFO" as orig, "NYC" as dest, 500 as price union
  select "SFO" , "MAD", 3000 union
  select "NYC" , "LHR", 600 union
  select "LHR" , "FRA", 600 union
  select "LHR" , "MAD", 400;
```

Write a SQL query that finds combination of flights (that have at least two flights) that are under a set budget of $1750 ordered by the total trip cost. You may not need to use all of the provided lines. Here is the expected output:

```
start | end | cost | hops
--------------------------
NYC     MAD     1000    2
SFO     LHR     1100    2
NYC     FRA     1200    2
SFO     MAD     1500    3
SFO     FRA     1700    3
```

```
with
  trips(src, end, cost, hops) as (
    select orig, orig, 0, 0 from airport union
        select src,dest,cost+price,hops+1 from airport,trips where orig = end
  )
select * from trips where cost < 1750 and hops > 1 order by cost;
```

5. (6 points) Interpretation

(From Summer 2014 MT2) Select which function(s) you would have to modify in order to add the new syntax features in Calculator. **For full credit, you must justify your answers with at most two sentences.**

(a) (1 pt) `=` (equality checker) – e.g. `(= 3 1)` returns False

`calc_eval` `calc_apply` Both Neither

Justification: No change to control flow. We just have to check for the operator now.

(b) (1 pt) `or` – e.g. `(or (= 5 2) (= 2 2) (\ 1 0))` returns True

`calc_eval` `calc_apply` Both Neither

Justification: From discussion. Must change calc_eval due to or's short-circuiting properties.

(c) (1 pt) Creating and calling lambdas (Assume `define` has been implemented.) – e.g.

```
(define square (lambda (x) (* x x)))
(square 4)
```

`calc_eval` `calc_apply` Both Neither

Justification: calc_eval because 'lambda' is a special form. calc_apply because you need to create a new frame as a result of applying the user-defined function to the arguments once the function is called.

(d) (1 pt) Which of the following is not a benefit of Client-Server Architecture

Creates a Abstraction Barrier Server Reuses Computation Both Neither

Justification:

Both are benefits of having a Client Server Architecture

(e) (2 pt) Consider the following function definition.

```
def g(n):
    if n % 2 == 0 and g(n + 1) == 0:
        return 0
    return 5
```

Circle the correct order of growth for a call to `g(n)`:

$\Theta(1)$ $\Theta(\log n)$ $\Theta(n)$ $\Theta(n^2)$ $\Theta(b^n)$