CS 61A Structure and Interpretation of Computer Programs

Spring 2017

Mock Final

INSTRUCTIONS

- You have 1 hour to complete the exam.
- The exam is closed book, closed notes, closed computer, closed calculator, except one 8.5" × 11" cheat sheet of your own creation.
- Mark your answers **on the exam itself**. We will *not* grade answers written on scratch paper.

Last name	
First name	
Student ID number	
Instructional account (`cs61a-__`)	
BearFacts 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. (10 points) On My Way to San Jose

For each of the expressions in the table below, write the output displayed by the interactive Python interpreter when the expression is evaluated. If an error occurs, write "Error". The first box has been filled in for you. Assume that the `Link` class has been defined. Assume that you have started `python3` and executed the following: statements:

```
class City:
  num = 0
  def __init__(self, name, \
      pop, people=[]):
    self.name = name
    self.pop = pop
    self.people = list(people)
    self.num += 1

class Place(City):
  lnk = Link.empty
  def __init__(self, name, city=None):
    self.name = name
    self.city = city
    lnk = self.lnk
    while lnk != Link.empty:
      lnk = lnk.rest
    lnk = self
```

```
class People:
  def __init__(self,place,name,first=0):
    self.place = place
    self.name = name
    if not first:
      self.friend=People(self.place, \
          "Friend", 1)
    print(self.place.city)
    self.place.city.people.append(self)

  def goto(self, place):
    self.place = place
    print(self.name+" is at "+place.name)
```



```
san_jose = City("San Jose", 1)
tech_museum = Place("Tech Museum",san_jose)
steve, bob = People(tech_museum, "Steve"), People(Place("Library", san_jose), "bob")
```

`len(Place.lnk)`	`san_jose.goto = People.goto` `san_jose.goto(tech_museum)`
`bob.goto(tech_museum)`	`san_jose.goto = steve.goto` `san_jose.goto(tech_museum)`

`print(bob.goto(san_jose))`	`berkeley = City("Berkeley", 2, \`
	` [steve, bob])`
	`City.num`

`People.__init__(san_jose, \`
`  san_jose, "Yali's")`
 	
`san_jose.name`
	`berkeley.people[0] == \`
	`  san_jose.people[1]`

`san_jose.city = san_jose`
`People.__init__(san_jose, \`
`  san_jose, "Yali's")`
 	
`san_jose.name`
	`[i.name for i in berkeley.people]`

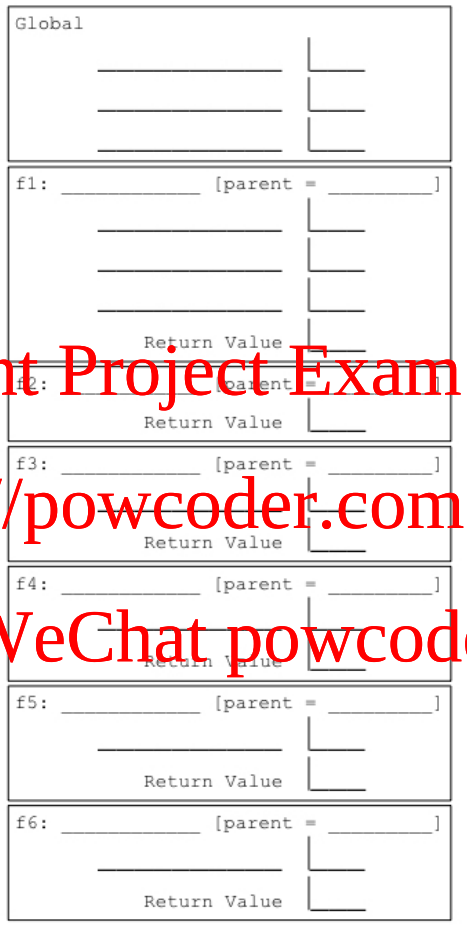
2. (10 points) Aaaaaaaaaaa

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

A complete answer will:

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

```
1  w, x = lambda a: x, lambda: x
2  def a(x):
3      def a():
4          nonlocal a
5          b()
6          a = w
7
8      def b():
9          nonlocal a
10         a = [x]
11         for i in range(2):
12             a.append(a[-1]())
13     a()
14     return a
15 x = a(x)
16 w = x(a)
```



3. (10 points) Scheme-ing Merge Given two sorted lists, lst1 and lst2, return a list that sorts both in ascending order. Break ties in any way you wish.

```
(define (merge lst1 lst2)

  (cond ((____________________) ____________________________________)

        ((____________________) ____________________________________)

        ((____________________) ____________________________________)

        (else (_____________________________________________________))))
```

4. (10 points) Scheme-ing to Find a Path
Here is the `BinTree` class provided for your reference:

```
class BinTree:
    empty = ()
    def __init__(self, label, left=empty, right=empty):
        self.label = label
        self.left = left
        self.right = right
```


Given a binary search tree and an entry, return the path in order to reach the entry from the root in the form of a list.


```
def pathfinder(bst, entry):
    """
    >>> bintree = BinTree(4, BinTree(2, BinTree(1), BinTree(5))
    >>> pathfinder(bst, 2)
    [4, 2]
    >>> pathfinder(bst, 1)
    [4, 2, 1]
    """
    if ________________________________________:

        ________________________________________

    elif ________________________________________:

        ________________________________________

    elif ________________________________________:

        return ____________________________________

    else:

        return ____________________________________________________
```

5. (10 points) Homework Party: The SQL

You are a veteran at RuneSQL, a popular RPG (role-playing game) where you hone your skills to become the best player in the database! However, you are a little short on SUPER DUPER EPIC RARE 61A homework party hats. Other players (a.k.a. n00bs) are fortunately predictable. Through your many years of being a crafty RuneSQL economist, you have taken note of the trends in hat_prices. The following chart shows the price per unit (in millions of RuneSQL coins) and quantity for a batch offer of party hats at a certain time (in minutes).

hat_prices

time	price	quantity
0	0.5	20
30	.3	10
60	0.75	40
90	0.7	25
120	1.3	25
150	1.25	30
180	0.4	5
210	0.45	10

Theres a catch! You will have to wait 1 hour after buying a single batch of hats or n00bs will get suspicious and market prices will change. Write a SQL select statement to show you the path to the maximum number of hats you can buy for 50 million coins, your current budget.


```
-- Expected result:
-- 0, 60, 210|70
WITH paths(path, prev_time, units, money) as (

    SELECT ------------------------------------------------

        FROM hat_prices UNION

    SELECT ------------------------------------------------------

           ------------------------------------------------------

        FROM hat_prices, paths

        WHERE money >= 0 and time - prev_time > 30

)
SELECT ------------------------- FROM -------------------------;
```