CS 61A Structure and Interpretation of Computer Programs
Summer 2018

MOCK FINAL

INSTRUCTIONS

- You have 80 minutes to complete the exam individually.
- The exam is closed book, closed notes, closed computer, and closed calculator, except for two hand-written 8.5" × 11" crib sheet of your own creation.
- Mark your answers **on the exam itself**. We will *not* grade answers written on scratch paper.

Last (Family) Name	
First (Given) Name	
Student ID Number	
Berkeley Email	
Teaching Assistant	◯ Alex Stennet ◯ Alex Wang ◯ Cameron Malloy ◯ Chae Park ◯ Chris Allsman ◯ Christina Zhang ◯ Derek Wan ◯ Erica Kong ◯ Griffin Prechter ◯ Jemin Desai ◯ Jennifer Tsui ◯ Jenny Wang ◯ Kevin Li ◯ Nancy Shaw
All the work on this exam is my own. **(please sign)**	

POLICIES & CLARIFICATIONS

- You may use built-in Python functions that do not require import, such as `min`, `max`, `pow`, and `abs`.
- For fill-in-the blank coding problems, we will only grade work written in the provided blanks. You may only write one Python statement per blank line, and it must be indented to the level that the blank is indented.
- Unless otherwise specified, you are allowed to reference functions defined in previous parts of the same question.
- The topics covered in this mock exam are not comprehensively representative of the topics that will appear on the actual final exam.

1. (6 points) Retrieve the output

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. Each expression has at least one line of output.

- If an error occurs, write **Error**, but include all output displayed before the error.
- To display a function value, write **Function**.
- If an expression would take forever to evaluate, write **Forever**.

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

Assume that you have started `python3` and executed the code shown on the left first, and then you evaluate each expression on the right in the order shown. Expressions evaluated by the interpreter have a cumulative effect.

```
class Retriever:
    height = 5
    def __init__(self, boy):
        self.good = boy
    def bark(self):
        print('Woof!')
        return self.height - 2
    def fetch(self, other):
        return 'Fetch!'
    def __str__(self):
        return 'I retrieve things'

class Golden(Retriever):
    name = 'Golden'
    height = 6
    def __init__(self, fluffy, boy):
        self.fluffy = fluffy
        self.boy = boy
    def bark(self, dog):
        print(dog.name + ' says hi!')
        return Retriever.bark(self)
    def sniff(self, friend):
        for _ in range(self.boy):
            print(friend)

class Labrador(Retriever):
    name = 'Labrador'
    height = 4
    def __init__(self, name):
        self.name = name
        self.d = Golden.bark(self, self)
    def fluffy(self, fluff):
        return fluff
    def fetch(self, other):
        return self.fluffy(other)
    def __repr__(self):
        return 'A pup!'

goldie = Golden(lambda x: 10, 5)
```

Expression	Interactive Output
`print(4, 5) + 1`	4 5 **Error**
`goldie.fluffy(goldie)`	
`goldie.fetch(Labrador)`	
`lucy = Labrador('Lucy')`	
`lucy.fetch(Labrador('Olly'))`	

`goldie.boy = lucy.d`
 	
`goldie.sniff(lucy.fetch(goldie))`	

`Retriever.bark(goldie)`	

2. (6 points) Schemin'

For each of the expressions in the table below, write the output displayed by the interactive Scheme interpreter when the expression is evaluated. The output may have multiple lines. Each expression has at least one line of output.

- If an error occurs, write **Error**, but include all output displayed before the error.

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

Assume that you have started our implementation of the Scheme interpreter with `python3 scheme` (or, equivalently, the interpreter at scheme.cs61a.org) and executed the code shown on the left first, and then you evaluate each expression on the right in the order shown. Expressions evaluated by the interpreter have a cumulative effect.

```
(define (mischief x f g . args)
  (if (even? x)
      (apply f args)
      (apply g args)
  )
)

(define (mystery a b)
  (if (= a b)
    'okay
    (begin (print 'hmmm)
           (mystery (- a 1) (+ b 1)))
  )
)

(define s (cons-stream 1
            (cons-stream (mystery 5 1)
                         nil)))
```

Expression	Interactive Output
`(+ (print 4) 1)`	4 **Error**
`(mystery 3 1)`	
`(mischief (+ 1 3) cons list 1 2)`	
`(cons (cons 6 (cons 1 nil)) '(a))`	
`` `(list ,(cons 1 2) (list 3 4)) ``	
`(define rest (cdr-stream s))`	
`(car (cdr-stream s))`	

3. (4 points) Zero

Write a macro called `zero-cond` that takes in a list of clauses, where each clause is a two-element list containing two expressions, a predicate and a corresponding result expression. All predicates evaluate to a number. The macro should evaluate each predicate and return the value of the expression corresponding to the first true predicate, *treating 0 as a false value.*

```
scm> (zero-cond
              ((0 'result1)
               ((- 1 1) 'result2)
               ((* 1 1) 'result3)
               (2 'result4)))
result3

(define-macro (zero-cond clauses)
    (cons 'cond

          (map ____________________________________________________________________

            ____________________________________________________________________)))
```

4. (6 points) DoubleTree

The following questions use this implementation of the `Tree` class:

```
class Tree:
    def __init__(self, label, branches=[]):
        for b in branches:
            assert isinstance(b, Tree)
        self.label = label
        self.branches = list(branches)
    def is_leaf(self):
        return not self.branches
```

```
def __str__(self):
    """ Represents trees in a readable format.
    >>> print(Tree(1, [Tree(2)]))
    1
      2
    """
    ...
```

(a) (2 pt) Fill in the definition of `copy_tree` below, which takes in a `Tree` instance `t` and returns a new tree object that contains the same items as `t`.

```
def copy_tree(t):
    """
    >>> t1 = Tree(1, [Tree(2)])
    >>> t2 = copy_tree(t1)
    >>> print(t2)
    1
      2
    >>> t1 is t2
    False
    >>> t1.branches is t2.branches
    False
    """
    return ____________________________________________________________________________
```

(b) (4 pt) Now, use `copy_tree` to write the function `double_tree`. This function takes in a tree and mutates it by duplicating every branch at every level in the tree. Assume that the `copy_tree` function works as expected, regardless of what you wrote above.

```
def double_tree(t):
    """
    >>> t = Tree(3, [Tree(4, [Tree(5)])])
    >>> double_tree(t)
    >>> print(t)
    3
      4
        5
        5
      4
        5
        5
    >>> t.branches[0].label = 6 # Make sure to copy branches instead of repeating them!
    >>> t.branches[1].label     # Changing original branch label doesn't affect new branch
    4
    """


    [___________________________ for ____________ in _________________________________]


    t.branches.____________________________________________________________________
```

5. (6 points) Don't repeat yourself

(a) (2 pt) Implement `repeater`, which takes in a list of positive numbers and returns a list where every number in the original list except for the first number appears a number of times equivalent to the previous number.

```
scm> (repeater nil)
()
scm> (repeater '(1 2 3))
(2 3 3)
scm> (repeater '(4 1 2 5))
(1 1 1 1 2 5 5)
```

```
(define (repeater nums)
    (define (repeat nums n)

        (cond (_________________________________  ___________________________________)

              ((= n 0) ____________________________________________________________)

              (else ______________________________________________________________)))

    (repeat  _________________________________  ___________________________________))
```

(b) (4 pt) Implement `zip-tail`, which is a tail recursive procedure that takes in two lists `a` and `b` and returns a single list containing two-element lists of co-indexed elements from `a` and `b`. If one list is shorter than the other, the zipped list's length is that of the shorter list. Your solution should be tail recursive.

```
scm> (zip-tail '(1 2 3) '(4 5 6))
((1 4) (2 5) (3 6))
scm> (zip-tail '(c 6 a) '(s 1 ! hello world))
((c s) (6 1) (a !))
```

Hint: Use the built-in `append` procedure, which you can assume is tail recursive, to concatenate two lists together. For example:

```
scm> (append '(1 2 3) '(4 5 6))
(1 2 3 4 5 6)
```

```
(define (zip-tail a b)

    (define (zipper _____________________________________________)

        (if ____________________________________________________________________

            ____________________________________________________________________

            ____________________________________________________________________))

    ____________________________________________________________________________)
```

6. (7 points) Teaqual

Chae and Jennifer decide to open a tea house, called ADTeas, where customers can build their own custom drinks. A *drink* is defined as some combination of a `tea` from the `teas` table as the base, a `syrup` from the `syrups` table, and a `topping` from the `toppings` table.

Each tea variety belongs to some tea type, either green, black, oolong, or white. Each syrup has a popularity level from 1-5 (5 being the most popular). Each topping has a particular tea type that it complements the most as well as a popularity level from 1-5.

teas	
tea	tea_type
jasmine	green
high mountain	oolong
silver needle	white
assam	black
osmanthus	oolong
gong fu	black

syrups	
syrup	popularity
honey	3
mango	4
peach	5
passionfruit	4
grapefruit	2

toppings		
topping	tea_type	popularity
lychee jelly	green	4
tapioca pearl	black	5
milk pudding	oolong	4
red bean	green	2
grass jelly	white	3

(a) (2 pt) Tammy needs help deciding what drink to get at ADTeas. She has no preference for the tea base or syrup, but only wants drinks with toppings that complement the type of the tea base and with combinations of toppings and syrups that have a combined average popularity of at least 4.5. Create a table called `tammys_drinks`, which contains all drinks that Tammy would like given by some `tea`, a `syrup`, and a `topping`.

```
CREATE TABLE tammys_drinks AS

    SELECT _____ AS tea, _____ AS syrup, _____ AS topping

        FROM ____________________________________________________________

        WHERE ___________________________________________________________

            _____________________________________________________________;
```

(b) (4 pt) Jennifer creates the table `special_drinks` to represent a special menu of popular drink combinations. It has 4 columns, each representing a `tea` base for the drink, a `syrup`, a `topping`, and a `popularity` value, which is the average of the popularity values of the topping and syrup in the drink.

```
CREATE TABLE special_drinks(tea, syrup, topping, popularity);
```

Tammy decides to have office hours at ADTeas and shares her drink choices with the students, which they love! Insert one drink of each tea type from `tammys_drinks` into `special_drinks`. Specifically, for each tea type, insert the drink that has the highest topping and syrup popularity average value out of all of the drinks of that type in `tammys_drinks`. Assume there is at most one drink of each tea type that fits this description.

```
INSERT INTO special_drinks
    SELECT d.tea, d.syrup, d.topping, ____________________________________

        FROM tammys_drinks AS d, ____________________________________________

        WHERE _______________________________________________________________

        GROUP BY ____________________________________________________________;
```

(c) (1 pt) Chae notices that not many people are purchasing drinks with grass jelly and that red bean is becoming more popular. She wants to remove grass jelly as an topping option to cut costs and raise red bean's popularity level to 3. Fill in the statements below to reflect this.

```
DELETE FROM _____________________ WHERE ____________________________;

UPDATE toppings SET _____________________ WHERE _____________________;
```