INSTRUCTIONS

• You have 3 hours 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 sheets of your own creation.

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

<table>
<thead>
<tr>
<th>Last (Family) Name</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>First (Given) Name</td>
<td></td>
</tr>
<tr>
<td>Student ID Number</td>
<td></td>
</tr>
<tr>
<td>Berkeley Email</td>
<td></td>
</tr>
</tbody>
</table>
| Teaching Assistant | ○ Alex Stennet ○ Christina Zhang ○ Jennifer Tsui
  ○ Alex Wang        ○ Derek Wan     ○ Jenny Wang
  ○ Cameron Malloy   ○ Erica Kong    ○ Kevin Li
  ○ Chae Park        ○ Griffin Prechter ○ Nancy Shaw
  ○ Chris Allsman    ○ Jemin Desai  |
| Exam Room and Seat |   |
| 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)

POLICIES & CLARIFICATIONS

• You may use built-in Python functions that do not require import, such as \texttt{min}, \texttt{max}, \texttt{pow}, and \texttt{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.
1. (11 points) Jennifer and Chae's Cat Cafe

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**.

Assume that you have started **python3** (not **ipython** or other variants) and executed the code shown on the left first, then you evaluate each expression on the right in the order shown.

```python
class Animal:
    def __init__(self, parent):
        self.parent = parent
        is_alive = False
    def __repr__(self):
        return 'cookie'
    def __str__(self):
        return 'I am an animal'

class Cat(Animal):
    is_alive = True
    def meow(self):
        if self.is_alive:
            return 'meower'
        return 'I am a ghost!'
    def meower(self):
        return self.is_alive
    def curiosity(self, cat):
        print('adventure')
        cat.is_alive = not self.is_alive
        return cat

class CalicoCat(Cat):
    def __repr__(self):
        return 'brownie'
    def meow(self):
        print('purr')
        return Cat.meow(self)

rachel = Cat(None)
aaron = CalicoCat(rachel)
amy = Cat(None)
```

<table>
<thead>
<tr>
<th>Expression</th>
<th>Interactive Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>print(4, 5) + 1</td>
<td>4 5 ERROR</td>
</tr>
<tr>
<td>print(aaron)</td>
<td>CalicoCat.meow(rachel)</td>
</tr>
<tr>
<td>Animal.meow(aaron)</td>
<td>aaron.parent.curiosity(amy)</td>
</tr>
<tr>
<td>Cat.meow = rachel.meow()</td>
<td>amy.meow()</td>
</tr>
</tbody>
</table>
2. (8 points) Jemin Watches Some Soccer

Fill in the environment diagram that results from executing the code to the right 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 all local frames.
- Add all missing values created or referenced during execution.
- Show the return value for each local frame.

```python
fa = 0

def fi(fa):
    def world(cup):
        nonlocal fa
        fa = lambda fi: world or fa or fi
        world = 0
        if (not cup) or fa:
            fa(2022)
            fa, cup = world + 4, fa
        return fa(cup)
    return world

won = lambda opponent, x: opponent(x)
france = won(fi(fa), 2018)
```

Global frame

<table>
<thead>
<tr>
<th></th>
<th>fa</th>
<th>fi</th>
<th>won</th>
<th>france</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1: [parent: ]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f2: [parent: ]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f3: [parent: ]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f4: [parent: ]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f5: [parent: ]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Value

func fi(fa) [parent=Global]
3. (11 points) While You Evaluate These Function Calls—man, Don’t Make An Error—ca

(a) (8 pt) 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.
- To display a procedure value, write PROCEDURE.
- If an expression would take forever to evaluate, write FOREVER.

Assume that you have executed the code shown on the left first, then you evaluate each expression on the right in the order shown.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Interactive Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>(+ (length '(3 4)) 1)</td>
<td>3</td>
</tr>
<tr>
<td>((shide 0 b) 'amy)</td>
<td></td>
</tr>
<tr>
<td>(jericho 5)</td>
<td></td>
</tr>
<tr>
<td>(jacob hermish)</td>
<td>asli</td>
</tr>
</tbody>
</table>

(b) (3 pt) Draw a box-and-pointer diagram for the state of the Scheme pairs after executing the block of code below. Please erase or cross out any boxes or pointers that are not part of a final diagram. This code does not error. We’ve provided the diagram for ex as an example. The built-in procedure length returns the length of a Scheme list.

```scheme
(define ex '(1 2 3))
(define (f x)
  (if (= x 0)
    5
    (list x (f (- x 1)))))
(define kate (f 3))
(define (g x)
  (if (list? x) (length x) x))
(define ajan (map g '(1 (2 (3)))))
```
4. (3 points) BSTs
The following questions reference the BST class and diagrams provided below.

class BST:
    # Other methods not shown
    def __contains__(self, e):
        if self.label == e:
            return True
        elif e > self.label and self.right is not BST.empty:
            return e in self.right
        elif self.left is not BST.empty:
            return e in self.left
        return False

![Tree Diagrams]

(a) (1 pt) Of tree1 and tree2, which are valid BSTs?
○ Both ○ tree1 only ○ tree2 only ○ Neither

(b) (1 pt) What is the runtime of the __contains__ method of the BST class with respect to \( n \), the number of nodes in tree3?
○ \( \Theta(1) \) ○ \( \Theta(\log n) \) ○ \( \Theta(\sqrt{n}) \) ○ \( \Theta(n) \) ○ \( \Theta(n^2) \) ○ \( \Theta(2^n) \)

(c) (1 pt) What is the runtime of the __contains__ method of the BST class with respect to \( h \), the height of tree3?
○ \( \Theta(1) \) ○ \( \Theta(\log h) \) ○ \( \Theta(\sqrt{h}) \) ○ \( \Theta(h) \) ○ \( \Theta(h^2) \) ○ \( \Theta(2^h) \)
5. (9 points) Kevin and Griffin’s Lunch Order

Kevin and Griffin are getting lunch before their sections. They each want to buy a main item, a snack, and a soft drink while staying within their budget.

(a) (2 pt) We represent the various lunch items with the `Food` class and its subclasses. There’s a sale on snacks right now, so all snacks cost 40% less than their listed price. Berkeley charges a 5% tax on soft drinks, so those cost more than their base cost. Fill in the `cost` method for the `Snack` and `SoftDrink` classes.

For full credit, you must not hard-code the snack discount or the soda tax, in case they change in the future.

class Food:
    def __init__(self, name, base_cost):
        self.name = name
        self.base_cost = base_cost

    def cost(self):
        return self.base_cost

class Main(Food):
    type = "main"

class Snack(Food):
    type = "snack"
    discount = 0.4

    def cost(self):
        """
        >>> chips = Snack("chips", 1)
        >>> chips.cost()  
        0.6
        """
        return  ____________________________________________________________

class SoftDrink(Food):
    type = "softdrink"
    sugar_tax = 0.05

    def cost(self):
        """
        >>> cola = SoftDrink("cola", 2)
        >>> cola.cost() 
        2.1
        """
        return  ____________________________________________________________
(b) (5 pt) Write three_sum_budget, which takes in three nonempty lists of positive numbers and a number n. This function should return the maximum sum less than or equal to n of one element from each of lst1, lst2, and lst3. If staying less than or equal to n is not possible, return 0.

```python
def three_sum_budget(lst1, lst2, lst3, n):
    """Find the maximum sum <= n of one element from each of lst1, lst2, and lst3.
    >>> three_sum_budget([1, 2, 3], [6, 8, 10], [4], 100)
    17
    >>> three_sum_budget([1, 2, 4], [6, 8, 10], [2], 15)
    14
    >>> three_sum_budget([1, 2, 3], [4, 5, 6], [1, 2, 4], 6)
    6
    """
    def helper(lst1, lst2, lst3, total):
        if __________________________________________________________________________________:
            return 0
        elif lst1 == "done":
            return __________________________________________________________________________
        options = []
        for item in lst1:
            options.append(__________________________________________________________________)  # These should be replaced with the correct maximum sum less than or equal to total
        return ______________________________________________________________________________

    return helper(lst1, lst2, lst3, 0)
```

(c) (2 pt) Now let's put it all together. Implement lunch_cost, which takes in a list of foods (each one is either a Main, a Snack, or a SoftDrink) and a budget. Return the maximum you'll spend if you buy one of each item and your total cost does not exceed your budget. Use the functions and classes you wrote in parts (a) and (b).

```python
def lunch_cost(foods, budget):
    """
    >>> lobster = Main('Lobster Tail', 25)
    >>> hotdog = Main('Hotdog', 5)
    >>> cider = SoftDrink('Sparkling Cider', 10)
    >>> cola = SoftDrink('Cola', 3)
    >>> fries = Snack('French Fries', 3)
    >>> lunch_cost([lobster, hotdog, cider, cola, fries], 100)
    37.3
    >>> lunch_cost([lobster, hotdog, cider, cola, fries], 25)
    17.3
    """
    def costs_by_type(type):
        return ______________________________________________________________________________

    mains = costs_by_type("main")
    snacks = costs_by_type("snack")
    drinks = costs_by_type("softdrink")

    return ______________________________________________________________________________
```
6. (8 points) Christreena Finds Longer Paths

On HW 5, you wrote long_paths, which found all paths of a certain length that extend from the root to a leaf. Now, write longer_paths, which removes the restrictions that paths must begin at the root and end at a leaf.

The length of a path is the number of edges in the path (i.e. one less than the number of nodes in the path). A path may begin and end at any node. Paths must always go from one node to one of its branches; they may not go upwards. You do not need to worry about the order of the different paths.

The Tree class is provided below.

class Tree:
    def __init__(self, label, branches=[]):
        self.label = label
        self.branches = list(branches)

    def is_leaf(self):
        return not self.branches

def longer_paths(t, n):
    """Return a list of all paths in T with length at least N."
    def helper(t, n, can_start_path):
        paths = []
        if ________________:
            paths.append(____________________)
        for b in t.branches:
            for subpath in ________________:
                paths.append(____________________)
            if ________________:
                paths.extend(____________________)
        return ________________
    return helper(t, n, True)
7. (7 points) Streams and Jennyrators

(a) (1 pt) Write `generate_constant`, a generator function that repeatedly yields the same value forever.

```python
def generate_constant(x):
    """A generator function that repeats the same value X forever.
    >>> two = generate_constant(2)
    >>> next(two)
    2
    >>> next(two)
    2
    >>> sum([next(two) for _ in range(100)])
    200
    ""
```

(b) (3 pt) Now implement `black_hole`, a generator that yields items in `seq` until one of them matches `trap`, in which case that value should be repeatedly yielded forever. You may assume that `generate_constant` works. You **may not** index into or slice `seq`.

```python
def black_hole(seq, trap):
    """A generator that yields items in SEQ until one of them matches TRAP, in which case that value should be repeatedly yielded forever.
    >>> trapped = black_hole([1, 2, 3], 2)
    >>> [next(trapped) for _ in range(6)]
    [1, 2, 2, 2, 2, 2]
    >>> list(black_hole(range(5), 7))
    [0, 1, 2, 3, 4]
    ""
```
(c) (3 pt) Now let's implement this in Scheme using streams. black-hole takes in an infinite stream of numbers and a value trap. It should return a stream that contains the items of stream until one of its elements matches trap, in which case the stream should repeat that value forever.

```
scm> (define (prefix s k) (if (= k 0) nil (cons (car s) (prefix (cdr-stream s) (- k 1)))))
prefix
scm> (define (naturals start) (cons-stream start (naturals (+ start 1))))
naturals
scm> (prefix (black-hole (naturals 1) 3) 8)
(1 2 3 3 3 3 3 3)
scm> (prefix (black-hole (naturals 5) 3) 5)
(5 6 7 8 9)
```

(define (black-hole stream trap)
  (cons-stream
    __________________________________________________________
    (if ______________________________________________________
    ________________________________________________________
    ________________________________________________________)

```
8. (10 points) Nan-scheme Writes Cam-acros

In Python, we can do arithmetic using infix notation, where the operator goes between two operands, e.g. 3 + 4. In Scheme, we have to use prefix notation for all call expressions, e.g. (+ 3 4).

Let's add support for infix notation in Scheme!

(a) (2 pt) First, write the helper function skip, which skips the first n items in a list, returning the rest. For full credit, your solution must be tail recursive. You may assume that n is non-negative.

```
scm> (skip 2 '(1 2 3 4))
(3 4)
scm> (skip 10 '(1 2 3 4))
()  
```

(define (skip n lst)
  (if ____________________________________________________
  ______________________________________________________
  ______________________________________________________)

```
(b) (6 pt) Now let’s write infix, which takes in a list containing some arithmetic in infix notation and evaluates it. You only need to support addition and multiplication, but you do need to take the order of operations and parentheses into account. You may use skip, as well as cadr and caddr.

```
scm> (infix '(5)) 5
scm> (infix '(2 + 3)) 5
scm> (infix '(2 * (3 + 6))) 18
scm> (infix '(2 * 3)) 12
scm> (infix '(2 * 3 + 6)) 20
```

```
(define (infix expr)
  (cond
    ((not (pair? expr)) expr) ; a single value
    ((or (equal? (car expr) '+) (equal? (car expr) '*)) (eval expr)) ; already in prefix form
    ((null? (cdr expr)) ___________________________________________________________________)
    (else
      (define left (infix _________________________________________________________________))
      (define right (infix _________________________________________________________________))
      (define operator ____________________________________________________________________)
      (cond
        ((equal? operator '+) (+ left _____________________________________________________))
        ((equal? operator '*) (infix (cons ________________________________________________
                                      __________________________________________________))))))))
```

(c) (2 pt) infix is great, but it only works on number literals. If we try to reference names, it errors.

```
scm> (define x 4)
x
scm> (infix '(x + 3))
Error: x is not a number
```

We can fix this by making a macro instead. Let’s say we define infix-macro as:

```
(define-macro (infix-macro . expr) (infix expr))
```

Unfortunately, this doesn’t quite work. What changes would need to be made to the code in part (b) so that infix-macro works like the tests below?

```
scm> (infix-macro x + 3) 7
scm> (infix-macro 4 + (x + 3) * 5) 39
```

Please describe the specific changes you’d make and why you’d make them, mentioning line numbers.
9. (6 points) Birthday Query Language

Tiffany’s birthday is coming up and the CS 61A staff wants to throw her a party! She’s put the times she’s available in a SQL table called `party_times`. The party will last 2 hours. Unfortunately, all of the times Tiffany is available conflict with some staff member’s section. Each staff member’s section is listed in `sections`.

The `time` column of both tables refers to the number of hours after noon that a section or party starts at. The `length` column refers to the length of a section in hours. The tables below are not complete.

<table>
<thead>
<tr>
<th>party_times</th>
<th>sections</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>staff_member</td>
</tr>
<tr>
<td>2</td>
<td>&quot;Daniel&quot;</td>
</tr>
<tr>
<td>4</td>
<td>&quot;Jemmy&quot;</td>
</tr>
<tr>
<td>5.5</td>
<td>&quot;Lauren&quot;</td>
</tr>
<tr>
<td>7</td>
<td>&quot;Wenyuan&quot;</td>
</tr>
</tbody>
</table>

(a) (3 pt) First, let’s make a table called `available` which has a row for every combination of a staff member and a time slot for which that staff member is available. Note that a staff member can be available for multiple times and there can be multiple staff members available for a given time.

A staff member is available for a timeslot if their section does not conflict with any part of it. If their section ends at the same time the party starts (or vice versa), the staff member can still attend the party. You may assume that each staff member only has 1 section on the day of the party.

For example, Lauren could attend a party at 4, since her section ends at 3.5, but could not attend a party at 2, since her section would overlap with it.

```
CREATE TABLE available AS

SELECT staff_member AS staff_member, __________________________ AS time
FROM ____________________________________________
WHERE ____________________________________________ OR

__________________________________________________;
```

(b) (1 pt) Uh, oh! Tiffany can no longer make the party time starting at 2. Write a single SQL statement that will mutate the `available` table to remove any availabilities listed for this time.

```
________________________________________________________________________________________;
```

(c) (2 pt) Now let’s find out what time the most staff members can make. Create a table called `best_times` that lists each party time and the number of staff members that can make it in descending order.

```
CREATE TABLE best_times

SELECT _______________________________________________ FROM _____________________________
GROUP BY __________________________________ ORDER BY __________________________________;
```
10. (2 points) Alex-tra Lectures
There are three problems here: one for each of the extra lectures. Each problem is worth 1 points, but you can only earn a maximum of 2 points on this problem, so you only need to know two answers.

(a) (1 pt) Logic Programming
Why are logic programming languages (like Logic or Prolog) less efficient than SQL? Limit your response to 15 words or less.

(b) (1 pt) Dynamic Programming
What is the main goal of memoization and dynamic programming? Limit your response to 10 words or less.

(c) (1 pt) Natural Language Processing
What does a leaf represent in a natural language syntax tree?

- a single word
- a noun phrase
- a verb phrase
- a subordinate clause
- a sentence

11. (0 points) Perfectly Balanced, As All Things Should Be
In this extra credit problem, you may choose one of the four instructors for the course. Your goal as a class is to evenly distribute your selections across the four options. If each instructor receives at least 20% of the votes, then everyone who properly marked an instructor for this problem will receive one (1) extra credit point.
You will not receive extra credit if you leave this problem blank, mark more than one bubble, or your selection is not clear.

- James Uejio
- Jen Thakar
- Mitas Ray
- Tammy Nguyen

12. (0 points) Wan More Thing
Thank you all for a fantastic summer!
We’ve hidden the names of all the instructors, TAs, and tutors somewhere within the exam. Can you find them all? This isn’t worth extra credit or anything. Obviously don’t do this until you’ve finished the exam.
No more questions.