CS 61A Structure and Interpretation of Computer Programs
Summer 2017

FINAL

- You have 2 hours and 50 minutes to complete this exam.
- This exam is closed book, closed notes, closed computer, closed calculator, except *four* 8.5" × 11" cheat sheets.
- Mark your answers **on the exam itself**. We will *not* grade answers written on scratch paper.
- For multiple choice questions, **fill in each option or choice completely.**
  - ☐ means mark **all options** that apply
  - ◯ means mark a **single choice**

Last name	
First name	
Student ID number	
CalCentral email (`_@berkeley.edu`)	
Teaching Assistant	◯ Alex Stennet ◯ Kelly Chen ◯ Angela Kwon ◯ Michael Gibbes ◯ Ashley Chien ◯ Michelle Hwang ◯ Joyce Luong ◯ Mitas Ray ◯ Karthik Bharathala ◯ Rocky Duan ◯ Kavi Gupta ◯ Samantha Wong
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)**	

0. (1 points) Determination What makes you strong?

1. (12 points) We're all the same to them ...

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. If an error occurs, write "Error", but include all output displayed before the error. If a function value is displayed, write "Function". If an iterator or generator value would be displayed, write "Iterator" (instead of something like `<iterator object at 0x...>`).

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

Assume that you have started `python3` and executed the following statements:

```
def tee(iterable):
    it = iter(iterable)
    queues = [[], []]
    def gen(lst):
        while True:
            if not lst:
                try:
                    value = next(it)
                except StopIteration:
                    return
                for q in queues:
                    q.append(value)
            yield lst.pop(0)
    return [gen(queues[0]), gen(queues[1])]

yum = ['avocado', 'quinoa', 'cream cheese']
```

```
>>> print(yum[0] + ' ' + yum[-1]) * 5
avocado cream cheese
Error
```

```
>>> print(print(next(iter(yum))), next(yum))
```

```
>>> next(iter(next(iter(yum))))
```

```
>>> eat = iter(tee(yum))
>>> neat = next(eat)
>>> next(neat)
```

```
>>> yum[0] = yum.pop()
>>> yum.append(next(neat))
>>> yum[:2]
```

```
>>> next(neat)
```

```
>>> munch = next(iter(tee(yum)))
>>> next(munch) + ' ' + next(next(eat))
```

2. (8 points) Winter is coming.

```
class Link:
    """A linked list."""

    empty = ()

    def __init__(self, first, rest=empty):
        self.first = first
        self.rest = rest
```

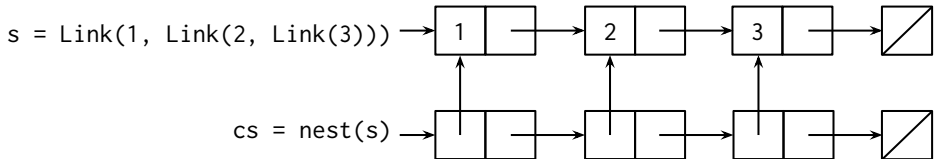
(a) (2 pt) Implement `nest`, which takes in a linked list `s` and returns a new linked list as shown below.

```
def nest(s):

    if ________________________________________:

        ____________________________________

    return ____________________________________________________
```



(b) (6 pt) *Clearly* draw the final box-and-pointer diagram for each of the two calls to `mystery`.

```
def mystery(a, b):
    if isinstance(a.first, Link):
        mystery(a.first, b)
    if a.rest is Link.empty:
        a.rest = b
    else:
        mystery(a.rest, b)
```

```
x = Link(1, Link(2, Link(3)))
mystery(x, x)
```

x ⟶

```
y = Link(Link(1, Link(2, Link(3))), Link(4, Link(Link(5, Link(Link(6, Link(7)), Link(8))))))
mystery(y, y)
```

y ⟶

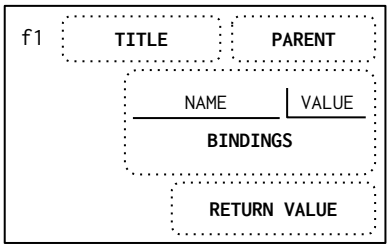
3. (10 points) You know nothing, Jon Snow.

(a) On the next page, 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.

(b) Then, for each FIELD below, fill in the corresponding bubble or *fig.* if referring to a drawn figure such as a list. Leave a row blank if the space in the environment diagram should be left blank.

To receive credit, you must list your bindings in the order in which they are ***first bound in the frame****.*



	FIELD	NAMES	VALUES
f1	Binding 1	f	◯ 0 ◯ 1 ◯ True ◯ False ◯ α ◯ β ◯ γ ◯ δ ◯ *fig.*
	Binding 2	hits	◯ 0 ◯ 1 ◯ True ◯ False ◯ α ◯ β ◯ γ ◯ δ ◯ *fig.*
	Binding 3	cache	◯ 0 ◯ 1 ◯ True ◯ False ◯ α ◯ β ◯ γ ◯ δ ◯ *fig.*
	Binding 4	run	◯ 0 ◯ 1 ◯ True ◯ False ◯ α ◯ β ◯ γ ◯ δ ◯ *fig.*
	Return		◯ 0 ◯ 1 ◯ True ◯ False ◯ α ◯ β ◯ γ ◯ δ ◯ *fig.*
f2	Title	◯ cache ◯ run ◯ hits ◯ lambda	
	Binding 1	◯ f ◯ hits ◯ cache ◯ run ◯ x ◯ y	◯ 0 ◯ 1 ◯ True ◯ False ◯ α ◯ β ◯ γ ◯ δ ◯ *fig.*
	Binding 2	◯ f ◯ hits ◯ cache ◯ run ◯ x ◯ y	◯ 0 ◯ 1 ◯ True ◯ False ◯ α ◯ β ◯ γ ◯ δ ◯ *fig.*
	Binding 3	◯ f ◯ hits ◯ cache ◯ run ◯ x ◯ y	◯ 0 ◯ 1 ◯ True ◯ False ◯ α ◯ β ◯ γ ◯ δ ◯ *fig.*
	Binding 4	◯ f ◯ hits ◯ cache ◯ run ◯ x ◯ y	◯ 0 ◯ 1 ◯ True ◯ False ◯ α ◯ β ◯ γ ◯ δ ◯ *fig.*
	Return		◯ 0 ◯ 1 ◯ True ◯ False ◯ α ◯ β ◯ γ ◯ δ ◯ *fig.*
f3	Title	◯ cache ◯ run ◯ hits ◯ lambda	
	Binding 1	◯ f ◯ hits ◯ cache ◯ run ◯ x ◯ y	◯ 0 ◯ 1 ◯ True ◯ False ◯ α ◯ β ◯ γ ◯ δ ◯ *fig.*
	Binding 2	◯ f ◯ hits ◯ cache ◯ run ◯ x ◯ y	◯ 0 ◯ 1 ◯ True ◯ False ◯ α ◯ β ◯ γ ◯ δ ◯ *fig.*
	Binding 3	◯ f ◯ hits ◯ cache ◯ run ◯ x ◯ y	◯ 0 ◯ 1 ◯ True ◯ False ◯ α ◯ β ◯ γ ◯ δ ◯ *fig.*
	Binding 4	◯ f ◯ hits ◯ cache ◯ run ◯ x ◯ y	◯ 0 ◯ 1 ◯ True ◯ False ◯ α ◯ β ◯ γ ◯ δ ◯ *fig.*
	Return		◯ 0 ◯ 1 ◯ True ◯ False ◯ α ◯ β ◯ γ ◯ δ ◯ *fig.*
f4	Title	◯ cache ◯ run ◯ hits ◯ lambda	
	Binding 1	◯ f ◯ hits ◯ cache ◯ run ◯ x ◯ y	◯ 0 ◯ 1 ◯ True ◯ False ◯ α ◯ β ◯ γ ◯ δ ◯ *fig.*
	Binding 2	◯ f ◯ hits ◯ cache ◯ run ◯ x ◯ y	◯ 0 ◯ 1 ◯ True ◯ False ◯ α ◯ β ◯ γ ◯ δ ◯ *fig.*
	Binding 3	◯ f ◯ hits ◯ cache ◯ run ◯ x ◯ y	◯ 0 ◯ 1 ◯ True ◯ False ◯ α ◯ β ◯ γ ◯ δ ◯ *fig.*
	Binding 4	◯ f ◯ hits ◯ cache ◯ run ◯ x ◯ y	◯ 0 ◯ 1 ◯ True ◯ False ◯ α ◯ β ◯ γ ◯ δ ◯ *fig.*
	Return		◯ 0 ◯ 1 ◯ True ◯ False ◯ α ◯ β ◯ γ ◯ δ ◯ *fig.*

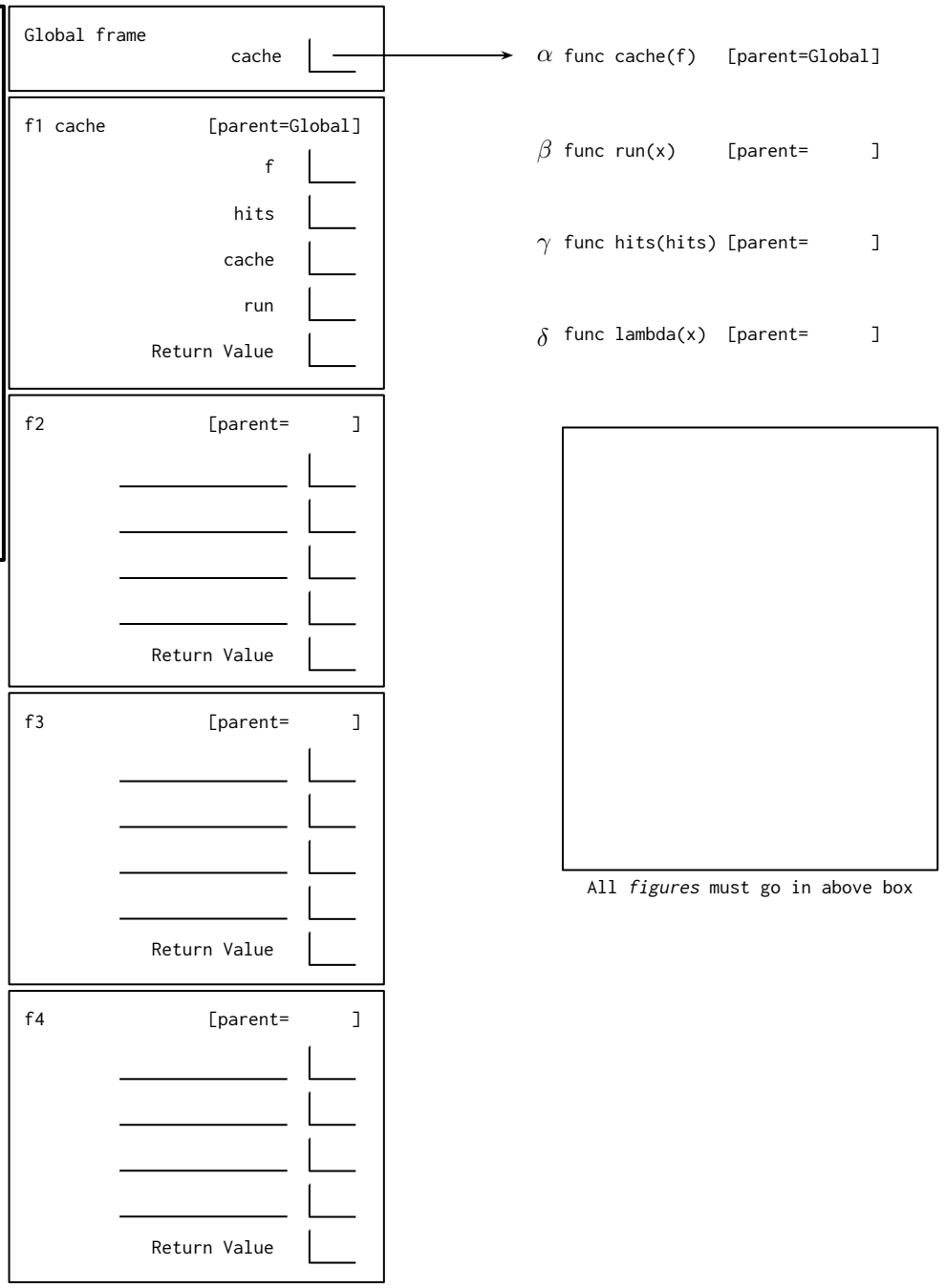
Remember to draw figures in the designated box and fill out the choices to receive credit.

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.
- Use box-and-pointer notation for lists. You don't need to write index numbers or the word "list".
- Include all *figures* or diagrams of objects (such as lists) in the **designated box**.

```
def cache(f):
    hits = 0
    cache = [hits]
    def run(x):
        nonlocal hits, cache
        def hits(hits):
            if hits:
                cache = []
            else:
                cache = [not hits]
            return cache
        y = f(x)
        hits(hits).append([x, y])
        return x or y
    return run

cache(lambda x: cache)(1)
```



All *figures* must go in above box

4. (6 points) The things I do for love.

(a) (4 pt) What are the first 7 elements of `my_stream()`? **Fill in the blanks below.**

```
from operator import add, mul

class Stream:
    """A lazily computed linked list."""
    empty = 'empty'

    def __init__(self, first, compute_rest=lambda: Stream.empty):
        """A stream with a first element and a rest that is a stream-returning parameterless
        function compute_rest."""

    @property
    def rest(self):
        """Return the rest, computing it if necessary."""

def make_integer_stream(first):
    """Return an infinite stream of integers counting up from first."""

def map_stream(f, s):
    """Return a new stream that is the result of applying f on every element in s."""

def filter_stream(f, s):
    """Return a new stream containing only the elements of s where f applied to the element
    returns True."""

def combine_streams(f, a, b):
    """Return a new stream that is the result of applying f on the values in a and b such
    that the output is f(a1, b1), f(a2, b2), ..."""

def my_stream():
    return filter_stream(lambda x: x % 2 == 1,
            Stream(1, lambda: Stream(2, lambda: Stream(3, lambda: Stream(4, lambda: Stream(5,
                    lambda: combine_streams(add, my_stream().rest,
                                combine_streams(mul, my_stream(),
                                    map_stream(lambda x: 2 * x, make_integer_stream(1))))))))))
```

_____ , _____ , _____ , _____ , _____ , _____ , _____

(b) (2 pt) Implement `pairwise`, a generator function which takes in an infinite stream `s` and yields the next sliding pair of elements from `s`. Evaluate as lazily as possible; do not compute the rest until it is needed.

```
def pairwise(s):

    pair = [None, s.first]

    while ______________________________________:

        ______________________________________

        pair = ___________________________________

        yield ____________________________________
```

```
>>> s = make_integer_stream(1)
>>> pairs = pairwise(s)
>>> for _ in range(3):
...     print(next(pairs))
[1, 2]
[2, 3]
[3, 4]
>>> s.rest.rest
Stream(3, Stream(4, <...>))
```

5. (3 points) When you play the game of thrones, you win or you die.

Implement `prune_tree` which takes in a Tree `t` and an integer `total` and mutates `t` so that the sum of each root-to-leaf path is at most `total`. Assume values are positive numbers and `t.root` $\leq$ `total`.

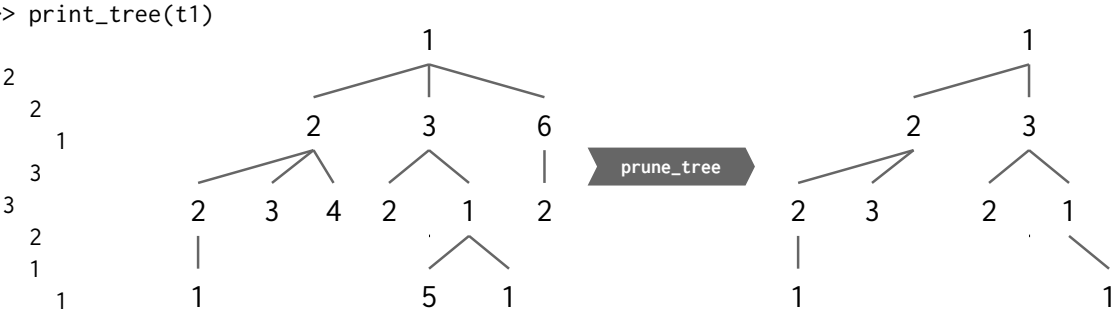
```
class Tree:
    """A mutable tree data type containing a root value and a list of branches."""

    def __init__(self, root, branches=[]):
        self.root = root
        self.branches = list(branches)

    def is_leaf(self):
        return not self.branches


def prune_tree(t, total):
    """Destructively prune the tree t so that the sum of each path from root-to-leaf is less
    than or equal to total. All values are positive numbers and t.root <= total.

    >>> t1 = Tree(1, [Tree(2, [Tree(2, [Tree(1)]),
                               Tree(3),
                               Tree(4)]),
                      Tree(3, [Tree(2), Tree(1, [Tree(5), Tree(1)])]),
                      Tree(6, [Tree(2)])])
    >>> prune_tree(t1, 6)
    >>> print_tree(t1)
    1
      2
        2
          1
        3
      3
        2
        1
          1
    """
```




```
    t.branches = ____________________________________________________________________________

    ____________________________________________________________________________________________

    ____________________________________________________________________________________________
```

6. (0 points) Designated Exam Fun Zone

Draw something. Leave a scent on the paper. It is up to you.

7. (6 points) There is no middle ground.

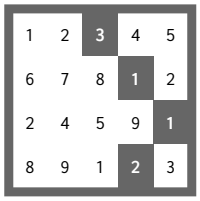
Implement `smallest_path` which takes in a *rectangular* `board` and returns the top-to-bottom path with the smallest total sum. A path can start from any position at the top of the board and move in one of three directions: 1 place `down`, 1 place down and 1 place `left`, or 1 place down and 1 place `right`.

The board is a deep list of integers. `board[0]` returns the first row while `board[-1]` returns the last row.

```
def smallest_path(board):
    """Given a rectangular board represented as a deep list of integers, return the
    top-to-bottom path with the smallest total sum. From a starting position, a path can move
    1 place down; 1 place down and 1 place left; or 1 place down and 1 place right.

    >>> small_board = [[1],
    ...                [2],
    ...                [3],
    ...                [4]]
    >>> smallest_path(small_board)
    [1, 2, 3, 4]
```


```
    >>> medium_board = [[1, 2, 3, 4, 5],
    ...                 [6, 7, 8, 1, 2],
    ...                 [2, 4, 5, 9, 1],
    ...                 [8, 9, 1, 2, 3]]
    >>> smallest_path(medium_board)
    [3, 1, 1, 2]
```




```
    >>> large_board = [[-3, 3, -1, 3, 8], [4, 8, 7, 6, -5], [2, 3, -2,  7, 2], [-1, 8, 2, 4, 3],
    ...                [ 0, 2, -1, 3, 5], [3, 6, 5, 7, -7], [1, 3, -8, -9, 1], [3, 0, -2, 7, 8]]
    >>> smallest_path(large_board)
    [3, -5, 2, 3, 3, -7, -9, -2]
    """

    left_bound, right_bound = 0, len(board[0])

    def path(board, location):

        if ____________________________________________________________________:

            return ______________________________

        elif ____________________________________________________________________:

            return [float('inf')]  # a board with the value representing infinity

        place = board[0][location]

        left  = ____________________________________________________________________________________

        down  = ____________________________________________________________________________________

        right = ____________________________________________________________________________________

        return _________________ + min(_____________________________________________, key=________)

    return min(_____________________________________________________________________, key=________)
```

8. (12 points) The Lannisters send their regards.

(a) (6 pt) Implement `split` which takes in a linked list `s` and a one-argument function `pred` and destructively splits `s` in two, returning one linked list with all the elements that satisfy `pred` and another with the rest. **Do not call the `Link` constructor!** The order of the elements does *not* need to be preserved. You may not need all the lines.

```
class Link:
    """A linked list."""

    empty = ()

    def __init__(self, first, rest=empty):
        self.first = first
        self.rest = rest

def split(s, pred):
    """Mutatively split s, returning one with elements that satisfy pred and one without.

    >>> link = Link(1, Link(2, Link(3, Link(4, Link(5)))))
    >>> evens, odds = split(link, lambda x: x % 2 == 0)
    >>> evens
    Link(4, Link(2))
    >>> odds
    Link(5, Link(3, Link(1)))
    """

    satisfy, not_satisfy = ____________________________________________________________

    while ____________________________________:

        rest = ___________________________________________

        if ___________________________________________________:

            ____________________________________________________________________________

            ____________________________________________________________________________

            ____________________________________________________________________________

        else:

            ____________________________________________________________________________

            ____________________________________________________________________________

            ____________________________________________________________________________

        ___________________________________________________

        ___________________________________________________

    return satisfy, not_satisfy
```

(b) (6 pt) Implement `split` *tail-recursively* in Scheme. Unlike the previous problem in Python, your solution may construct new pairs by calling `cons` or `list`, or by using the `quote` special form.

```
(define (split s pred)

  (define (split-tail ____________________________________________)

    (cond ((null? _______________) ____________________________________________________)

          ((pred _________________________________)

            (split-tail ____________________________________________________________))

          (else

            (split-tail ____________________________________________________________))))

  (split-tail ____________________________________))

(define (even? x) (= (remainder x 2) 0))
(split '() even?)
; expect (() ())
(split '(1 2 3 4 5) even?)
; expect ((4 2) (5 3 1))
```

9. (4 points) One voice may speak you false, but in many there is always truth to be found.

For each of the following Scheme expressions, choose the correct number of calls that would be made to `scheme_eval` and `scheme_apply` when evaluating the expression in our Scheme interpreter from the Scheme Project. Assume the normal `scheme_eval` is in use, *not* tail-call-optimized `scheme_optimized_eval`.

Suppose we have already evaluated the following definition in the current environment.

```
(define lst (list 1 2 3))
```

(a) (2 pt)

```
(+ (car lst) (- 5 (car (cdr lst))))
```

Number of calls to `scheme_eval` ◯ 3 ◯ 5 ◯ 8 ◯ 10 ◯ 11 ◯ 13 ◯ 15 ◯ 17 ◯ 19

Number of calls to `scheme_apply` ◯ 1 ◯ 2 ◯ 3 ◯ 4 ◯ 5 ◯ 6 ◯ 7 ◯ 8 ◯ 9 ◯ 10

(b) (2 pt)

```
((if (or (null? lst) (null? (cdr lst)))

    (lambda (s) 0)

    (lambda (s) (car (cdr s))))

 lst)
```

Number of calls to `scheme_eval` ◯ 8 ◯ 9 ◯ 11 ◯ 13 ◯ 16 ◯ 17 ◯ 18 ◯ 21 ◯ 22

Number of calls to `scheme_apply` ◯ 1 ◯ 2 ◯ 3 ◯ 4 ◯ 5 ◯ 6 ◯ 7 ◯ 8 ◯ 9 ◯ 10

10. (18 points) I've brought ice and fire together.

Consider the following schema that represents `users`, `products`, and `sales` in a database management system.

```
create table users(uid, uname, date_created);
create table products(pid, pname, description, rating, price);
create table sales(time, pid, uid);
```

- `uid` (user ID), `pid` (product ID), `rating`, `price` are numbers while all other columns are strings.
- The `uid` uniquely identifies one user because there may be users with the same `uname` and `date_created`.
- The `pid` uniquely identifies one product because there may be products with the same column values.
- The `uid` and `pid` in each row of `sales` references a `uid` in `users` and a `pid` in `products`.

Express the following queries in SQL using only features we've covered in this course.

Recall: Rows can be ordered in either `asc` ending (increasing) or `desc` ending (decreasing) order.

(a) (2 pt) Select the `uname` and product `rating` of any one user who purchased a highest-rated product (a product such that there is no other product rated higher). If there is more than one such product, return any one product.

```
select ____________________________________________________

    from ____________________________________________________________________________

    where ____________________________________________________________________________

    order by _____________________________________ desc limit 1;
```

(b) (3 pt) Select the `uid`, `uname`, and the number of products purchased for each user that has *purchased at least one product*.

```
with s as (select ____________________________________________________________

           from _____________________________ group by _____________________________)

select ____________________________________________________________________________

    from ____________________________________________________

    where ____________________________________________________;
```

(c) (3 pt) Select the `pid` and the `diff` of the product with a price that is closest to the average price of all products. That is, select the product with the *smallest absolute difference* between the product's price and the average price of all products. If there is more than one such product, return any one product.
Hint: The SQL function for absolute value is `abs`.

```
with a as (select ________________________________ as ____________________________

           from _________________________________)

select ____________________________________________________________________________ as diff

    from ____________________________________________________

    order by _____________________________________ asc limit 1;
```

(d) (6 pt) Let's design a SQL-like query engine in Python. Implement `LoopJoin`, an iterator which takes in two iterables and yields all possible combinations of rows from the two iterables and returns a joined row on each call to `__next__`. Order matters! The first row of `left_iterable` should match with all the rows of `right_iterable` *before* moving on to the second row of `left_iterable`. Each row is a tuple (an immutable list) so two rows can be joined with the `+` operator. You may not need all the lines.

```
class LoopJoin:
    """A database join iterator that takes in two iterables and joins their rows.

    >>> users = [(1, 'Kevin', '2017-05-19'), (2, 'Stan', '2017-06-20')]
    >>> sales = [('2017-07-20', 9580, 2), ('2017-07-24', 8483, 2)]
    >>> for row in LoopJoin(users, sales):
    ...     print(row)
    (1, 'Kevin', '2017-05-19', '2017-07-20', 9580, 2)
    (1, 'Kevin', '2017-05-19', '2017-07-24', 8483, 2)
    (2, 'Stan', '2017-06-20', '2017-07-20', 9580, 2)
    (2, 'Stan', '2017-06-20', '2017-07-24', 8483, 2)
    """

    def __init__(self, left_iterable, right_iterable):

        self.left_iterator = iter(left_iterable)

        self.right_iterator = iter(right_iterable)

        ____________________________________________________________________________________

        ____________________________________________________________________________________

    def __iter__(self):

        return self

    def __next__(self):

        try:

            ____________________________________________________________________________________

            ____________________________________________________________________________________

            return ____________________________________________________________________________

        except ____________________________________________________:

            ____________________________________________________________________________________

            ____________________________________________________________________________________

        ____________________________________________________________________________________

        ____________________________________________________________________________________

        return next(self)
```

(e) (2 pt) Implement `select` and `where`, two generators that each take in two parameters: a one-argument function and an iterator. `select` yields the result of applying `columns` to each row from the iterator. `where` yields each row from the iterator if the row meets the given `pred`. You may not need all the lines.

```
def select(columns, iterator):                  def where(pred, iterator):

    ________________________________________        ________________________________________

    ________________________________________        ________________________________________

    ________________________________________        ________________________________________
```

(f) (2 pt) Now that we have assembled all the parts of the query engine, let's compute! Complete the assignment statement for `query` such that it passes the test below. Assume that the implementations for `select`, `where`, and `LoopJoin` are correct.

```
>>> users = [(1, 'Kevin', '2017-05-19'), (2, 'Stan', '2017-06-20')]
>>> sales = [('2017-07-20', 9580, 2), ('2017-07-24', 8483, 2)]

>>> products = {8483: 'flowers', 9580: 'perfume'}

>>> query = select(____________________________________________________________________________,
...
...                where(_____________________________________________________________________,
...
...                      LoopJoin(users, sales)))

>>> for row in query:
...     print(row)
Stan purchased perfume
Stan purchased flowers
```

11. (0 points) **It seems your journey is finally over. You're filled with DETERMINATION.**

In this extra credit problem, you may **write the first and last name of one other CS 61A student** in the blank below. If the other student also chose to write your name, then a *friendship* has formed.

The goal is to, as a class, form as many friendships as possible.

For each friendship, everyone receives *two one-hundredths* (0.02) extra credit points, representing each student in the friendship. This means that if, as a class, 100 friendships form, then *everyone* receives two points.

You reach out and call their name:

_____

Leave a message below.

This page intentionally left blank.