CS 61A Structure and Interpretation of Computer Programs

Spring 2016

FINAL EXAMINATION (WITH CORRECTIONS)

INSTRUCTIONS

- This exam should have 18 pages. You have 3 hours to complete the exam.
- The exam is open book, open notes, closed computer, closed calculator. The official CS 61A midterm 1, 2, and final study guides will be provided.
- Mark your answers **on the exam itself**. We will *not* grade answers written on scratch paper.

Last name	
First name	
Student ID number	
BearFacts email (`@berkeley.edu`)	
If you took the HKN survey, your code.	
Room in which you are taking this exam	
TA	
Name of the person to your left	
Name of the person to your right	
I pledge my honor that during this examination I have neither given nor received assistance. **(please sign)**	

Reference. Some questions make use of the following class definitions from labs and homework. There is one difference: we have changed the definition of `Link.empty` so that it is a kind of `Link` instead of the empty tuple.

```
class Link:
    def __init__(self, first, rest=None):
        # When called as Link(x), resets rest to empty
        if rest is None:
            rest = Link.empty
        self.first = first
        self.rest = rest
    def __repr__(self): ... # (Not shown)

class EmptyLink(Link):
    def __init__(self):
        pass
Link.empty = EmptyLink()   # This makes Link.empty a special kind of Link

class Tree:
    def __init__(self, label, children=()):
        self.label = label
        self.children = list(children)

    def is_leaf(self):
        return not self.children

    def __repr__(self): ... # (Not shown)

class BinTree:
    empty = ()

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

class Stream:
    class empty:
        pass
    empty = empty()

    def __init__(self, first, compute_rest=lambda: Stream.empty):
        self.first, self._compute_rest = first, compute_rest

    @property
    def rest(self):
        if self._compute_rest is not None:
            self._rest, self._compute_rest = self._compute_rest(), None
        return self._rest
```

1. (8 points) Silence of the Lambdas

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`". If an expression yields (or prints) a function, write "`<Function>`". The first two rows have been provided as examples.

Important: The statements in the table are cumulative—assume that all preceding statements in the table have been executed before each entry.

Assume that `python3` has executed the statements on the left initially:

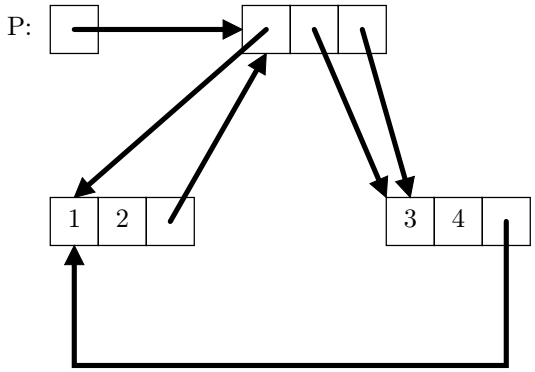
```
foster = 1
def f(foster):
   hopkins = foster+1
   def g(glenn):
       nonlocal foster
       foster = glenn
       hopkins = 2*glenn
   return (g, lambda: [foster, hopkins])
```

Expression	Interactive Output
`pow(2, 3)`	`8`
`print(4, 5) + 1`	`4 5` `Error`
`levine, demme = f(5)` `foster`	
`demme`	
`tally = demme()` `tally[0]`	
`tally[1]`	
`print(levine(9))`	
`foster`	
`tally = demme()` `tally[0]`	
`tally[1]`	

2. (8 points) Point(er) of Order

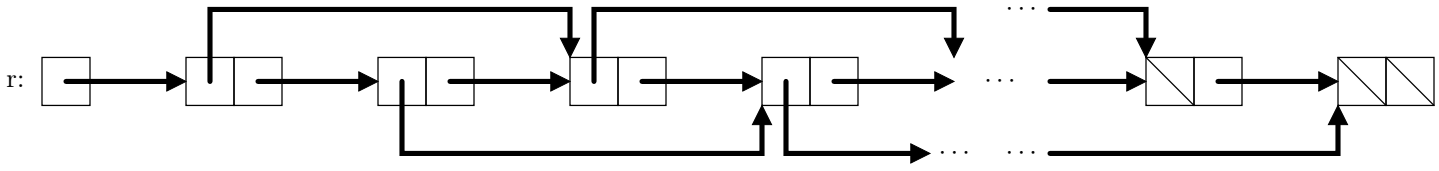
(a) (3 pt) Fill in code on the left that, when executed, yields the situation on the right. Assume that the objects are Python lists. Single boxes with labels to their left denote variables, not list objects.

```
P = ______________________________________

P____________________ = _________________

P____________________ = _________________

P____________________ = _________________
```



(b) (1 pt) Why can't the structure depicted in the diagram for part (a) be built from Python tuples, no matter what program is used on the left? Give a brief answer in a few sentences.

(c) (4 pt) In the diagram below, the two-slot objects are `Links` (see page 2). There are `N` of these `Link` objects altogether, with each `.first` field pointing to the second `Link` following it (except for the last two, whose `.first` fields are `Link.empty`.) Fill in the blanks to produce such a list (for any non-negative value of `N`.)

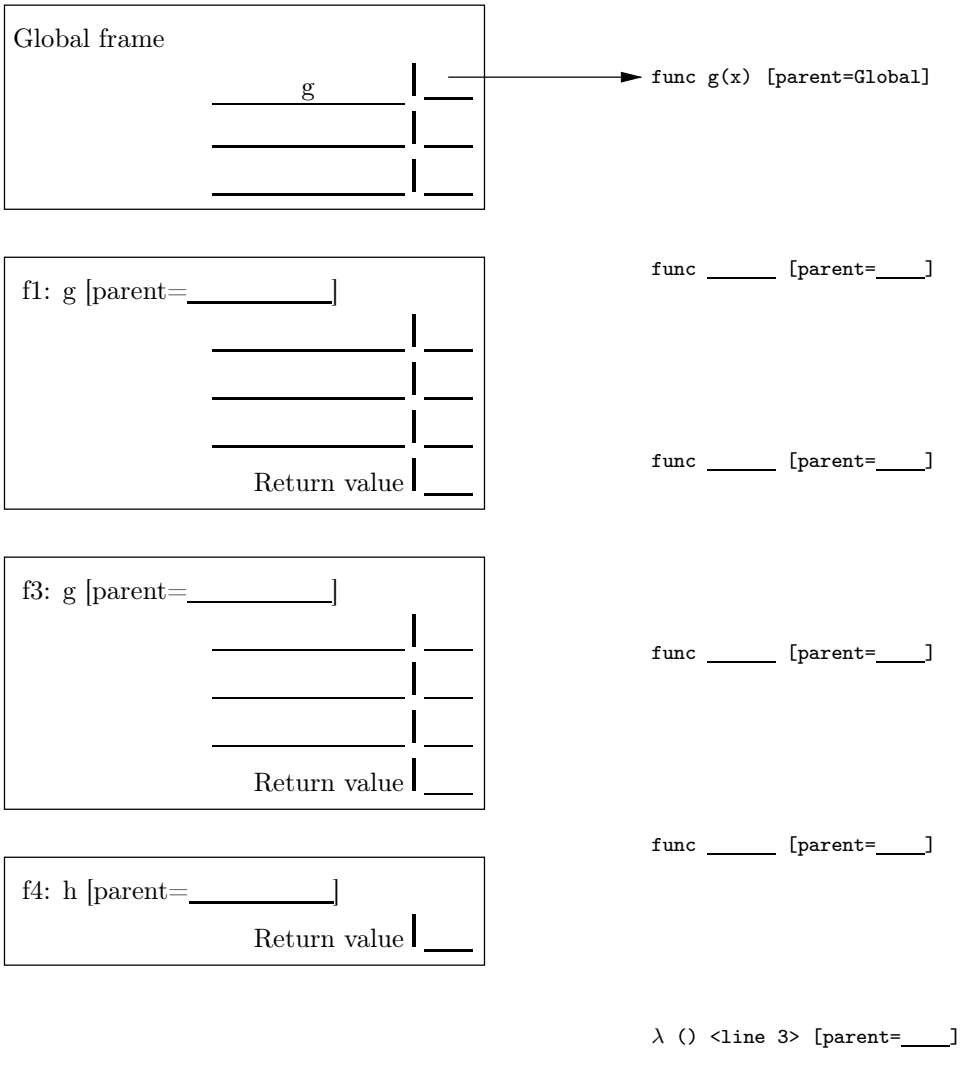



```
r = Link.empty
p = Link.empty

while ____________________________:

    t = Link_____________________________________

    p, r, N = _________, __________, ___________
```

3. (8 points) Environmentally Sound

(a) (6 pt) Fill in the environment diagram that results from executing the code below until the entire program is finished. Fill in the function values (`func ...[parent=...]`) with all function values created during execution. As you can see from the numbering, some environment frames are not shown. **Include only the environment frames for the functions indicated in the frames shown.**

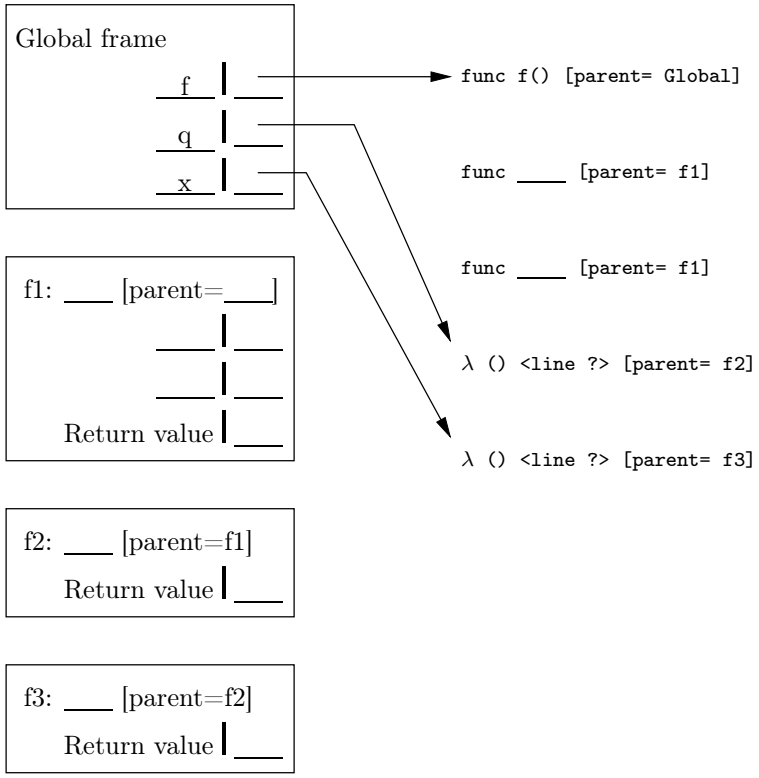
```
1  def g(x):
2      def h():
3          return lambda: x
4      def f(x):
5          return g(h)
6      if x == 1:
7          return f(5)
8      else:
9          return x
10 p = g(1)()
11 x = p()
```



(b) (3 pt)

The diagram on the right below shows *part* of an environment diagram; various items are left blank, and various arrows are not shown. Fill in the blanks in the program on the left so as to create a situation consistent with the diagram on the right (you do not have to fill in any of the blanks in the environment diagram.) Assume that any variable or function is assigned or defined exactly once. Assume also that function values that start with `func` in the diagram are defined with **def**.

```
1 def f():
2   ------------------------------
3   ------------------------------
4   ------------------------------
5   ------------------------------
6   return p()
7 q = f()
8 x = q()
```



4. (10 points) Tiptoe through the Links

During the semester, we defined a `__getitem__` method for the `Link` class. With a little revision, it allows us to use Python iteration:

```
def __getitem__(self, k):
    if self is Link.empty:
        raise IndexError
    elif k == 0:
        return self.first
    else:
        return self.rest[k-1]
```

(Here, we are making use of the definition of `Link.empty` on page 2, so that it works on empty lists.) If this definition is added to `Link`, `Link`s become iterable: the Python `iter` function (used by **for**, and other constructs that deal with iterables) is able to create an iterator whose `__next__` method calls `__getitem__` with `k` set to 0, 1, 2, ..., until `IndexError` is raised.

(a) (1 pt) Assuming that `Link.__getitem__` is defined as shown, what is the execution time of

```
for i in L:
   print(i)
```

if `L` is a `Link` that heads a linked list of length N?

A. $\Theta(1)$

B. $\Theta(\log N)$

C. $\Theta(\sqrt{N})$

D. $\Theta(N)$

E. $\Theta(N^2)$

F. $\Theta(2^N)$

Problem continues on next page.

(b) (5 pt) Suppose that instead of defining `Link.__getitem__`, we instead want to define `Link.__iter__` to return some kind of iterator whose `__next__` method runs in constant time (so it won't work to count from the beginning of the list each time). Fill in the definition of `ListIter` (whose instances are intended to be iterators over linked lists) and `Link.__iter__` to make `Link` an iterable class whose iterators take constant time to perform a `__next__` operation. **Warning:** The `Link` class in this exam does *not* include `__len__`, so the `len` function does not work.

```
class ListIter:
    def __init__(self, lst):


        ____________________________________________________

    def __next__(self):


        ____________________________________________________


        ____________________________________________________


        ____________________________________________________


        ____________________________________________________


        return ____________________________________________

class Link:
    ...
    def __iter__(self):


        return ____________________________________________
```

(c) (1 pt) For the revised `Link` class from part (b), what is the execution time of

```
for i in L:
   print(i)
```

if `L` is a `Link` that heads a linked list of length N? (Assume your solution in (b) works as specified, regardless of what you wrote).

A. $\Theta(1)$

B. $\Theta(\log N)$

C. $\Theta(\sqrt{N})$

D. $\Theta(N)$

E. $\Theta(N^2)$

F. $\Theta(2^N)$

(d) (3 pt) Now re-implement `__iter__` from part (b) so that instead of using a separate iterator class, it creates a generator (so that from the user's point of view, execution of `for i in L` does not change). (Not all lines need be used.)

```
class Link:
    ...
    def __iter__(self):

        ____________________________________________________

        while ______________________________________________

            ________________________________________________

            ________________________________________________
```

5. (12 points) Triangular!

In a (lower) triangular array (which we represent as a Python list of lists), row r has length $r + 1$, like this:

```
tri1 = [                        # Boxed items give maximum sum
  [ [1]],
  [ [2],  1],
  [-2, [-1],  1],
  [ 3,  [3],  1,  1]
]
```

Suppose that we choose one item from each row of this triangle in such a way that if we choose item $\#k$ in row $\#r$, we choose either item $\#k$ or $\#k + 1$ from row $\#r + 1$. Assuming our arrays contain integers, you are to write a program to find the maximum sum of a proper choice of selected items from all rows.

In the example above, we can maximize our sum by choosing the boxed items: item 0 from row 0, item 0 from row 1, item 1 from row 2, and item 1 from row 3 (summing to 5). Choosing instead to take the last item (1) of the first three rows would only give us a sum of 4, since we could not then choose either of the 3's from the last row.

(a) (5 pt) Fill in the blanks in the following recursive program to find the maximum sum:

```
def triangle_sum(tri):
    """Given that tri is a triangular array, return the maximum
    sum attainable by selecting one item from each row, where if
    item #k is selected from row #r, either item #k or item #k+1 is
    selected from row #r+1."""

    rows = len(tri)

    def partial_sum(r, k):
        """The maximum partial sum of items from rows #R, R+1, ...
        starting from selecting item #K in row #R."""

        if _______________________________:

            return ________________________________________________
        else:
            return ________________________________________________ \

                   ________________________________________________
             # (Use above line to Continue if needed)

    return partial_sum(_____________________________________)
```

(b) (1 pt) As a function of the number of rows, R, of the input triangle, what is the running time of this program?

A. $\Theta(R)$

B. $\Theta(R^2)$

C. $\Theta(\sqrt{R})$

D. $\Theta(\log R)$

E. $\Theta(2^R)$

(c) (5 pt) The program may be sped up by noticing that

- When the inner function gets called with the same arguments multiple times, it always returns the same value.
- The values of the inner function when called on a given row depend only on the values of the function on the row below it.

Use these observations to create a faster, iterative version of the program.

```
def triangle_sum(tri):
    """Given that tri is a triangular array, return the maximum
    sum attainable by selecting one item from each row, where if
    item k is selected from row r, either item k or item k+1 is
    selected from row r+1."""

    # Create a deep copy of tri to avoid destroying the input data.
    tri = [ list(row) for row in tri ]

    rows = len(tri)
    r = rows - 2

    while ____________________________:


        for k in ________________________________________________


            _________________ += _________________________________


         ____________________________________________________________


    return ____________________________________________________
```

(d) (1 pt) As a function of the number of rows, R, of the input triangle, what is the running time of this program?

A. $\Theta(R)$

B. $\Theta(R^2)$

C. $\Theta(\sqrt{R})$

D. $\Theta(\log R)$

E. $\Theta(2^R)$

6. (8 points) Six Degrees of Separation

You've probably heard that we are all within "six degrees of separation." That is, either we are friends (one degree), friends of friends (two degrees), friends of friends of friends (three degrees), etc. up to six degrees. We may, of course, be separated by several different distances, as when our friend is also a friend of a friend. Although there are obviously many different paths leading from you back to yourself, however, we won't consider you as being connected with yourself.

Suppose that `friends` is an SQL table with two columns, `F1` and `F2`, where in each row, `F1` and `F2` are the names of two friends—i.e., two people with one degree of separation between them. To make life easier, we'll assume that if (Peter, Paul) is in the table, then so is (Paul, Peter). We would like an SQL query that produces a two-column table named `linked` of people separated (by some chain of friends) by N or fewer degrees of separation, where N is some integer. In your solution, use '`N`' as if it is an integer literal, like 6. (The idea of using '`N`' instead of a specific number is to force your solution to be general.) Each pair in the resulting table should appear exactly once, with the name in the first column being first in alphabetical order.

For example, suppose that $N = 2$, then given the `friends` table on the left, we should get the `linked` table on the right, in some order. (The column names don't matter for `linked`, and so are not shown.)

friends

F1	**F2**
Peter	Paul
Jack	Paul
Rose	Jack
Paul	Sam
Cindy	Rose
Paul	Peter
Paul	Jack
Jack	Rose
Sam	Paul
Rose	Cindy

linked

Cindy	Rose
Cindy	Jack
Jack	Paul
Jack	Rose
Jack	Peter
Jack	Sam
Paul	Peter
Paul	Sam
Paul	Rose
Peter	Sam

```
create table linked as

    with sep(S1, S2, degrees) as (

       select ________________________________________________ union

       select ____________________________________ from friends, sep

           where ____________________________________________________________
    )

select _________________ from ___________ where ______________________________;
```

7. (4 points) Tail Recursing the Dog

Fill in the Scheme `iota` function so that `(iota n)` produces a list of the numbers from 1 to n (empty if `n` is not positive). The resulting `add-iota` function must be tail-recursive.

```
scm> (iota 5)
(1 2 3 4 5)
scm> (iota 0)
()
scm> (iota -5)
()
```

```
(define (iota n)
  (define (add-iota lst m)

    (if ____________________________________________________________

        ____________________________________________________________

        ____________________________________________________________))

  (add-iota '() ________________________________________))
```

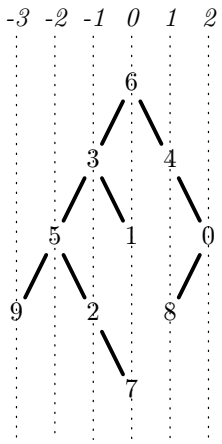
8. (1 points) Sum of Human Knowledge What may be recognized by the following characteristics?

- A meager and hollow, but crisp, taste.
- A habit of getting up late.
- A slowness in taking a jest.
- A fondness for bathing machines.
- Ambition.

Answer: _____

9. (8 points) In the Trees

If you are careful about how you draw a binary tree, its nodes will line up in columns. Here, for example, is a tree with six columns of nodes (indicated by dotted lines; the numbers at top are column numbers):



Let's designate the column containing the root as column 0. For any node in column k, we'll says its left child is in column $k-1$ and its right in column $k+1$. In general, nodes will overlap when you do this, but we'll ignore that possibility.

Write a program to print all the nodes in a given column of a tree, given the tree and the column number. The type `BinTree` is as defined in the Reference section on page 2 of this test. (Don't forget that the empty tree is a valid input to `print_column`.) You may not need all the lines.

```
def print_column(tree, col):
    """Print the labels of the nodes in column COL of BinTree TREE,
    in any order, one per line.
    >>> e = BinTree.empty
    >>> tree = BinTree(6,
    ...                BinTree(3,
    ...                        BinTree(5, BinTree(9), BinTree(2, e, BinTree(7))),
    ...                        BinTree(1)),
    ...                BinTree(4, e, BinTree(0, BinTree(8))))
    >>> print_column(tree, -1)
    3
    2
    >>> print_column(tree, 2)
    0"""

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

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

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

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

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

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

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

    ------------------------------------------------------
```

10. (10 points) Treebeard's Revenge

This problem combines `Tree`s and `Link`s (see the reference section on page 2). (Be careful not to confuse type `Tree` with type `BinTree` used earlier.)

(a) (5 pt) Fill in the `tree_search` function so that `tree_search(tr, pred)` returns a linked list of all labels in `tr` (of type `Tree`) that satisfy `pred` (that is, for which the one-argument function `pred` returns a true value). The order of the list is irrelevant.

```
def tree_search(tr, pred):
    """Returns a linked list (type Link) of labels in Tree tr that
    satisfy PRED.
    >>> t = Tree(4, [Tree(5, [Tree(6, [Tree(5)])]), Tree(1)])
    >>> tree_search(t, lambda x: x%2 == 1)
    Link(5, Link(5, Link(1)))
    """
    L = ____________________________________________________

    def subtree_search(subtr):

        ____________________________________________________

        if ____________________________________________

            L = ____________________________________________

        for ____________________________________________

            ____________________________________________

    ____________________________________________________

    return L
```

(b) (5 pt) An `Organizer` is a kind of object that divides the labels in a tree into a sequence of disjoint lists, one for each of a given set of categories. To create an `Organizer`, one provides a sequence of predicates (one-argument functions) that define the categories. When this object is subsequently handed a `Tree`, it will return a sequence of linked lists of tree labels, one per category in the same order as the original sequence of category predicates. Once a tree label is included in one list, it must not appear again in that list, nor in any subsequent list. Fill in the class below to fulfill this specification. You may assume that labels are numbers or strings.

```
class Organizer:

    def __init__(self, categories):
        """Create a new Organizer whose categories are defined by the
        predicates in CATEGORIES (an iterable)."""
        self._categories = categories

    def categorize(self, tr):
        """Return a Python sequence of linked lists, where the kth
        list contains tree labels from TR that satisfy my kth
        category.  Each tree label appears exactly once in the entire
        set of lists returned, regardless of how often it occurs in TR.
        NOTE: The order of values in the linked lists in the example
        below is just one possible result.
        >>> tr = Tree(6, [Tree(3, [ Tree(5, [Tree(9), Tree(2, [Tree(9)])]),
        ...                         Tree(1) ])
        ...                Tree(4, [Tree(0, [Tree(4)])])])
        >>> or = Organizer([lambda x: x > 4, lambda x: x%2 == 0])
        >>> or.categorize(tr)
        [Link(6, Link(5, Link(9))), Link(4, Link(0, Link(2)))]
        """
        result = []

        labels_seen = _____________________________________

        def take_it(x):  # Hint: you can see pred from inside take_it

            if ________________________________________:

                ________________________________________

                return True
            return False

        for pred in _____________________________________:
            result.append(tree_search(tr, take_it))

        return result
```

11. (4 points) Exstream!

You may have seen the power series for computing the exponential function:

$$e^x = \sum_{k \geq 0} \frac{x^k}{k!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots.$$

By computing and adding one term at a time, one can get as close as desired to the value of e^x, giving an infinite sequence of approximations:

$$1,\ 1 + x,\ 1 + x + \frac{x^2}{2!},\ 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!}, \ldots.$$

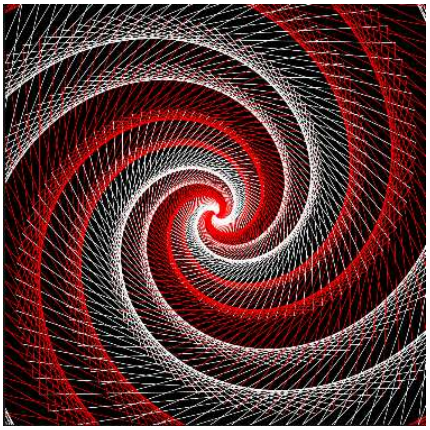
Fill in the blanks below so that `e2x(x)` creates a Python `Stream` (see page 2) consisting of this sequence of approximations. Do not introduce any additional lambda expressions or **def** statements than are already there.

```
# The following all deal with infinite streams only (don't work with finite
# ones).

def const_stream(x):
    r = Stream(x, lambda: r)
    return r

def add_streams(s0, s1):
    return Stream(s0.first + s1.first,
                  lambda: add_streams(s0.rest, s1.rest))
def mul_streams(s0, s1):
    return Stream(s0.first * s1.first,
                  lambda: mul_streams(s0.rest, s1.rest))
def div_streams(s0, s1):
    return Stream(s0.first / s1.first,
                  lambda: div_streams(s0.rest, s1.rest))

positives = Stream(1, lambda: add_streams(positives, const_stream(1)))

def e2x(x):
    powers_of_x = Stream(x, lambda: mul_streams(const_stream(x), powers_of_x))
    factorials = Stream(1, lambda: mul_streams(positives.rest, factorials))

    r = Stream(1, lambda: ____________________________________________________

                          ____________________________________________________)
    return r
```

Scheme Art Contest Winners

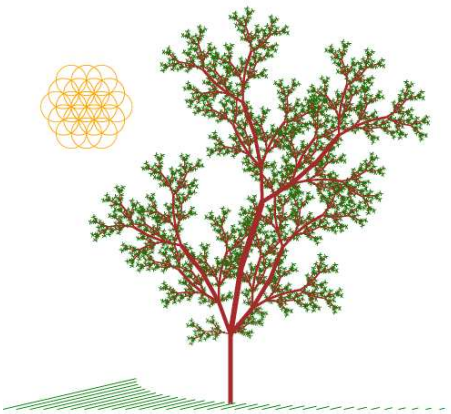
Featherweight Division

First Place
Recursive Bleeding
Carson Trinh



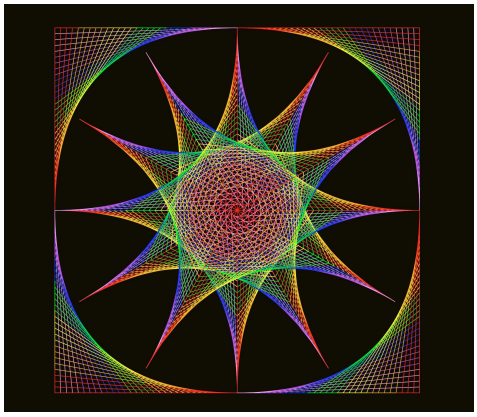
Hilfinger in spring
and then Hilfinger in fall
Will it ever end?

Second Place (tie)
tree in early spring
Matej Sebo



tree in early spring
branches weaving up, sprouting
into myriad leaves

Second Place (tie)
T.L.O.P
Abraham Chen, Bill Cai



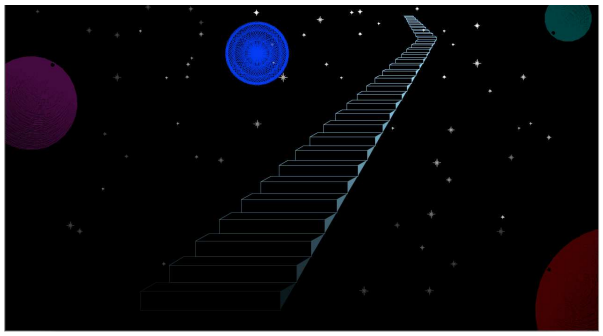
The Life of Pablo
Vote yes and we'll love you like
Kanye loves Kanye

Heavyweight Division

First Place
Twisted Logic
Kyla Woyshner

What if you were lost,
An endless kaleidoscope,
No hope of square one

Second Place
Stairway to Heaven
Sharabesh Ramesh, Jared Gutierrez



The one last project
Alone it stands defeated
61A no more