0. (0 points) **Determination**  What’s been fun? What are you grateful for?
1. (10 points) Talking to Ducks

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”.

```python
class Duck:
    family = []
    ducks = 0

def __init__(self, name):
    self.name = name
    Duck.family.append(self)
    Duck.ducks += 1

def __iter__(self):
    while True:
        yield Duck.family[0]
        first = Duck.family.pop(0)
        Duck.family.append(first)

def __str__(self):
    return 'A Duck'

def __repr__(self):
    return 'Duck("{}")'.format(self.name)

def clean(self):
    self.family = []

class Duckling:
    mother_duck = Duck

    def __init__(self, name):
        Duck.__init__(self, name)
        self.mother_duck = self.mother_duck

    def __repr__(self):
        return 'Duckling("{}")'.format(self.name)

class Swan(Duckling):
    def __init__(self, name='Autumn'):
        Duckling.__init__(self, name)
        self.mother_duck = self.mother_duck('Swan')

    def __iter__(self):
        Duckling.next = Duck.__iter__(self)
        while True:
            yield next(Duckling.next)

drake = Duckling('Drake')
helen = drake.mother_duck('Helen')
drakerator = iter(helen)

<table>
<thead>
<tr>
<th>Expression</th>
<th>Interactive Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pow(2, 3)</code></td>
<td>8</td>
</tr>
<tr>
<td><code>print(4, 5) + 1</code></td>
<td>4 5 Error</td>
</tr>
<tr>
<td><code>print(print(next(drakerator)), next(drakerator))</code></td>
<td>Duckling(&quot;Drake&quot;)</td>
</tr>
<tr>
<td></td>
<td>None A Duck</td>
</tr>
<tr>
<td>Duck.ducks</td>
<td>2</td>
</tr>
<tr>
<td>different = Swan()</td>
<td>4</td>
</tr>
<tr>
<td>different.mother_duck.ducks</td>
<td>Duck(&quot;Helen&quot;)</td>
</tr>
<tr>
<td>next(iter(different))</td>
<td>Duck(&quot;Helen&quot;)</td>
</tr>
<tr>
<td>Duck.family[-2:]</td>
<td>[Duckling(&quot;Autumn&quot;), Duck(&quot;Swan&quot;)]</td>
</tr>
<tr>
<td>clean(different.mother_duck)</td>
<td>Duck(&quot;Swan&quot;)</td>
</tr>
<tr>
<td>Duck.family[-1]</td>
<td>Duck(&quot;Swan&quot;)</td>
</tr>
</tbody>
</table>
2. (8 points) A Link to the Past

Implement slice_reverse which takes a linked list s and mutatively reverses the elements on the interval, \([i, j)\) (including \(i\) but excluding \(j\)). Assume \(s\) is zero-indexed, \(0 < i < j\), and that \(s\) has at least \(j\) elements.

You must use mutation; solutions which call the Link constructor will not receive credit. The Link class reference is provided below.

class Link:
    
    empty = ()

    def __init__(self, first, rest=empty):
        assert rest is Link.empty or isinstance(rest, Link)
        self.first = first
        self.rest = rest

    def slice_reverse(s, i, j):
        
        >>> s = Link(1, Link(2, Link(3)))
        >>> slice_reverse(s, 1, 2)
        >>> s
        Link(1, Link(2, Link(3)))
        >>> s = Link(1, Link(2, Link(3, Link(4, Link(5)))))
        >>> slice_reverse(s, 2, 4)
        >>> s
        Link(1, Link(2, Link(4, Link(3, Link(5)))))

        start = s
        for _ in range(i - 1):
            start = start.rest
        reverse = Link.empty
        current = start.rest
        for _ in range(j - i):
            rest = current.rest
            current.rest = reverse
            reverse = current
            current = rest
        start.rest.rest = current
        start.rest = reverse
3. (6 points) Lost Woods

(a) (4 pt) A Binary Search Tree is a tree where each node contains either 0, 1, or 2 nodes and where the left branch (if present) contains values strictly less than (<) the root value, and the right branch (if present) contains values strictly greater than (>) the root value. The definition is recursive: both the left and right branches must themselves also be BST for the entire tree to be a BST.

Implement is_binary which takes in a Tree t, and returns True if t is a Binary Search Tree and False otherwise. Trees can contain any number of branches, but if a tree contains only one branch, interpret it as a left branch.

```python
class Tree:
    def __init__(self, root, branches=[]):
        self.root = root
        self.branches = list(branches)
    
    def is_leaf(self):
        return not self.branches
    
    def is_binary(self):
        def binary(t, lo, hi):
            if lo < t.root < hi:
                if t.is_leaf():
                    return True
                elif len(t.branches) == 1 and t.branches[0].root < t.root:
                    return binary(t.branches[0], lo, t.root)
                elif len(t.branches) == 2 and t.branches[0].root < t.root < t.branches[1].root:
                    return binary(t.branches[0], lo, t.root) and binary(t.branches[1], t.root, hi)
            return False
        return binary(t, float('-inf'), float('inf'))
```

(b) (1 pt) Choose the \( \Theta(\cdot) \) expression that best describes the runtime of is_binary on a well-formed binary search tree with \( n \) nodes. Assume the implementation of is_binary is optimal.

\[ \Theta(1) \quad \Theta(\log n) \quad \Theta(n) \quad \Theta(n \log n) \quad \Theta(n^2) \quad \Theta(n^3) \quad \Theta(2^n) \quad \Theta(3^n) \]

(c) (1 pt) Choose the \( \Theta(\cdot) \) expression that best describes the runtime of is_binary on a tree where each node contains 3 branches and the overall height of the tree is \( n \). Assume the implementation of is_binary is optimal.

\[ \Theta(1) \quad \Theta(\log n) \quad \Theta(n) \quad \Theta(n \log n) \quad \Theta(n^2) \quad \Theta(n^3) \quad \Theta(2^n) \quad \Theta(3^n) \]

4. (0 points) Designated Exam Chillout Zone
5. (8 points) ▲

(a) (4 pt) Implement merge which takes in two sorted lists of numbers and returns a new sorted list containing all the values. Ties can be broken in either direction.

(define (merge lst1 lst2)
  (cond ((null? lst1) lst2)
        ((null? lst2) lst1)
        ((< (car lst1) (car lst2)) (cons (car lst1) (merge (cdr lst1) lst2)))
        (else (cons (car lst2) (merge lst1 (cdr lst2))))))

scm> (merge '(1 2 3 4 5) '(6 7 8 9 10))
(1 2 3 4 5 6 7 8 9 10)
scm> (merge '(1 3 5 7 9) '(2 4 6 8 10))
(1 2 3 4 5 6 7 8 9 10)
scm> (merge '(3 4 7 9 10) '(1 2 5 6 8))
(1 2 3 4 5 6 7 8 9 10)
scm> (merge () ())
()

(b) (1 pt) Choose the Θ(·) expression that best describes the runtime of merge where the length of each list is n. Assume the implementation of merge is correct, optimized, but not tail-recursive.

○ Θ(1) ○ Θ(log n) ● Θ(n) ○ Θ(n log n) ○ Θ(n^2) ○ Θ(n^3) ○ Θ(2^n) ○ Θ(3^n)

(c) (1 pt) Choose the Θ(·) expression that best describes the number of frames opened during the execution of merge where the length of each list is n. Assume the implementation of merge is correct, optimized, but not tail-recursive.

○ Θ(1) ○ Θ(log n) ● Θ(n) ○ Θ(n log n) ○ Θ(n^2) ○ Θ(n^3) ○ Θ(2^n) ○ Θ(3^n)

(d) (1 pt) Implement trimerge which takes in three sorted lists of numbers and returns a new sorted list containing all the values. You may use merge; assume it is implemented correctly.

(define (trimerge lst1 lst2 lst3)
  (merge lst1 (merge lst2 lst3)))

scm> (trimerge '(1 2 3) '(4 5 6) '(7 8 9))
(1 2 3 4 5 6 7 8 9)
scm> (trimerge '(3 7 8) '(1 4 5) '(2 6 9))
(1 2 3 4 5 6 7 8 9)
scm> (trimerge () () ())
()

(e) (1 pt) Choose the Θ(·) expression that best describes the runtime of trimerge where the length of each list is n. Assume the implementation of trimerge is optimal.

○ Θ(1) ○ Θ(log n) ● Θ(n) ○ Θ(n log n) ○ Θ(n^2) ○ Θ(n^3) ○ Θ(2^n) ○ Θ(3^n)
6. (8 points) Telephony Booth

Suppose you're analyzing data collected from a regional telephone operator. Their database consists of the following schemas which define the columns of each table.

```sql
create table location(lname, areacode);
create table people(pid, pname, areacode);
create table calls(cid, date, from, to, duration);
```

- The `areacode` data in the `people` table references the area codes defined in `location`.
- The `from` and `to` columns of `calls` each reference one person's `pid`.

(a) (2 pt) Select the names of all pairs of people who called each other on the date 2017-07-04.

```sql
select p1.pname, p2.pname
from people as p1, people as p2, calls as c
where c.from = p1.pid and c.to = p2.pid and c.date = '2017-07-04';
```

(b) (3 pt) Select the name of the location and the number of calls to that location for the location which has had the most phone calls made to it.

```sql
select l.lname as name, count(*) as numcalls
from location as l, people as p, calls as c
where p.areacode = l.areacode and p.pid = c.to
group by l.areacode
order by numcalls desc limit 1;
```

(c) (3 pt) Select the name of the location of the person who made the longest call.

```sql
with durations as (select p.areacode as areacode, c.duration as duration
from calls as c, people as p
where c.from = p.pid)
select l.lname
from durations as d, location as l
where l.areacode = d.areacode
order by d.duration desc limit 1;
```