1. (12 points) Class Hierarchy

For each row below, write the output displayed by the interactive Python interpreter when the expression is evaluated. Expressions are evaluated in order, and expressions may affect later expressions.

Whenever the interpreter would report an error, write ERROR. You should include any lines displayed before an error. Reminder: The interactive interpreter displays the repr string of the value of a successfully evaluated expression, unless it is None. Assume that you have started Python 3 and executed the following:

class Worker:
greeting = 'Sir'
def __init__(self):
    self.elf = Worker
def work(self):
    return self.greeting + ', I work'
def __repr__(self):
    return Bourgeoisie.greeting

class Bourgeoisie(Worker):
greeting = 'Peon'
def work(self):
    print(Worker.work(self))
    return 'My job is to gather wealth'

class Proletariat(Worker):
greeting = 'Comrade'
def work(self, other):
    other.greeting = self.greeting + ' ', '+ other.greeting
    other.work() # for revolution
    return other

jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'

<table>
<thead>
<tr>
<th>Expression</th>
<th>Interactive Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>5*5</td>
<td>25</td>
</tr>
<tr>
<td>1/0</td>
<td>ERROR</td>
</tr>
</tbody>
</table>

Worker().work()

Expression | Interactive Output
---|------------------
john.work()[:10:]

Proletariat().work(john)

Expression | Interactive Output
---|------------------
john.elf.work(john)

jack

jack.work()
4. (8 points) Tree Time

(a) (4 pt) A GrootTree $g$ is a binary tree that has an attribute parent. Its parent is the GrootTree in which $g$ is a branch. If a GrootTree instance is not a branch of any other GrootTree instance, then its parent is BinaryTree.empty. BinaryTree.empty should not have a parent attribute. Assume that every GrootTree instance is a branch of at most one other GrootTree instance and not a branch of any other kind of tree. Fill in the blanks below so that the parent attribute is set correctly. You may not need to use all of the lines. Indentation is allowed. You should not include any assert statements. Using your solution, the doctests for fib_groot should pass. The BinaryTree class appears on your study guide.

Hint: A picture of fib_groot(3) appears on the next page.

class GrootTree(BinaryTree):
    """A binary tree with a parent."""

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

    def fib_groot(n):
        """Return a Fibonacci GrootTree.
        
>>> t = fib_groot(3)
>>> t.entry
2
>>> t.parent.is_empty
True
>>> t.left.parent.entry
2
>>> t.right.left.parent.right.parent.entry
1
"""
        if n == 0 or n == 1:
            return GrootTree(n)
        else:
            left, right = fib_groot(n-2), fib_groot(n-1)
            return GrootTree(left.entry + right.entry, left, right)
(b) (4 pt) Fill in the blanks of the implementation of paths, a function that takes two arguments: a GrootTree instance g and a list s. It returns the number of paths through g whose entries are the elements of s. A path through a GrootTree can extend either to a branch or its parent.

You may assume that the GrootTree class is implemented correctly and that the list s is non-empty.

The two paths that have entries \([2, 1, 2, 1, 0]\) in \texttt{fib_groot(3)} are shown below (left). The one path that has entries \([2, 1, 0, 1, 0]\) is shown below (right).

```
def paths(g, s):
    """The number of paths through g with entries s."

    >>> t = fib_groot(3)
    >>> paths(t, [1])
    0
    >>> paths(t, [2])
    1
    >>> paths(t, [2, 1, 2, 1, 0])
    2
    >>> paths(t, [2, 1, 0, 1, 0])
    1
    >>> paths(t, [2, 1, 2, 1, 2, 1])
    8
    """

    if g is BinaryTree.empty:
        return 0

    elif:
        return 1

    else:
        xs = [______________]

        return sum([_____________ for x in xs])
```
2. Interpretation (Fa14 Mock Final Q5e)

```python
def g(n):
    if n % 2 == 0 and g(n + 1) == 0:
        return 0
    return 5
```

Circle the correct order of growth for a call to $g(n)$:

- $\Theta(1)$
- $\Theta(\log n)$
- $\Theta(n)$
- $\Theta(n^2)$
- $\Theta(n^3)$
- $\Theta(b^n)$

3. Not with a fizzle, but with a bang (Su13 Midterm 2 Q2b) Consider the following linked list functions:

```python
def boom(n):
    if n == 0:
        return "BOOM!"
    return boom(n - 1)

def explode(n):
    if n == 0:
        return boom(n)
    i = 0
    while i < n:
        boom(n)
        i += 1
    return boom(n)
```

Circle the correct order of growth for a call to $\text{explode}(n)$:

- $\Theta(1)$
- $\Theta(\log n)$
- $\Theta(n)$
- $\Theta(n^2)$
- $\Theta(n^3)$
- $\Theta(2^n)$

4. Not with a fizzle, but with a bang (Su13 Midterm 2 Q2c) Consider the following linked list functions:

```python
def dreams(n):
    if n <= 0:
        return n
    if n > 0:
        return n + dreams(n // 2)
```

Circle the correct order of growth for a call to $\text{dreams}(n)$:

- $\Theta(1)$
- $\Theta(\log n)$
- $\Theta(n)$
- $\Theta(n^2)$
- $\Theta(n^3)$
- $\Theta(2^n)$
5. Various Programs (Sp14 Final Q5c) Give worst-case asymptotic bounds, in terms of \(m\) and \(n\), for the running time of the following functions.

```python
def a(m, n):
    for i in range(m):
        for j in range(n // 100):
            print("hi")

Bound:

def b(m, n):
    for i in range(m // 3):
        print("hi")
    for j in range(n * 5):
        print(bye)

Bound:

def d(m, n):
    for i in range(m):
        j = 0
        while j < i:
            print("hi")
            j = j + 100

Bound:
```

6. OOG Potpourri What is the order of growth of each of the following functions?

a. Weighted

```python
def weighted_random_choice(lst):
    temp = []
    for i in range(len(lst)):
        temp.extend([lst[i] * (i + 1)])
    return random.choice(temp)
```

Order of Growth:

b. Iceskate

```python
def ice(n):
    skate = n
    def rink(n):
        nonlocal skate
        print(n)
        if skate > 0:
            skate = 1
            rink(skate)
        return skate
    return rink(n//2)
```

Order of Growth:
(d) (2 pt) Consider the following linked list functions:

```python
def append(link, value):
    """Mutates link by adding value to the end of link."""
    if link.rest is Link.empty:
        link.rest = Link(value)
    else:
        append(link.rest, value)

def extend(link1, link2):
    """Mutates link1 so that all elements of link2 are added to the end of link1.
    """
    while link2 is not Link.empty:
        append(link1, link2.first)
        link2 = link2.rest
```

Circle the order of growth that best describes the runtime of calling `append`, where \( n \) is the number of elements in the input link.

\[
O(1) \quad O(\log n) \quad O(n) \quad O(n^2) \quad O(2^n)
\]

Assuming the two input linked lists to `extend` both contain \( n \) elements, circle the order of growth that best describes the runtime of calling `extend`.

\[
O(1) \quad O(\log n) \quad O(n) \quad O(n^2) \quad O(2^n)
\]

6. (0 points) A second chance

In each of the two boxes below, write a positive integer. If one of the numbers you pick is the lowest unique integer in the class, you get one extra credit point. In other words, you get two chances to write the smallest positive integer that you think no one else will write.
(d) (6 pt) Implement `double_up`, which mutates a linked list by inserting elements so that each element is adjacent to an equal element. The `double_up` function inserts as few elements as possible and returns the number of insertions. The Link class appears on the midterm 2 study guide.

```python
def double_up(s):
    """Muate s by inserting elements so that each element is next to an equal."

    >>> s = Link(3, Link(4))
    >>> double_up(s) # Inserts 3 and 4
    2
    >>> s
    Link(3, Link(3, Link(4, Link(4))))
    >>> t = Link(3, Link(4, Link(4, Link(5))))
    >>> double_up(t) # Inserts 3 and 5
    2
    >>> t
    Link(3, Link(3, Link(4, Link(4, Link(5, Link(5))))))
    >>> u = Link(3, Link(4, Link(3)))
    >>> double_up(u) # Inserts 3, 4, and 3
    3
    >>> u
    Link(3, Link(3, Link(4, Link(4, Link(3, Link(3))))))
    ""
    if s is Link.empty:
        return 0

    elif s.rest is Link.empty:
        #
        return

    elif
        #
        return double_up(

    else:
        #
        return
```

```python

```