1. What does the following code block output?
   ```python
def foo():
a = 0
    if a < 10:
        print("Hello")
yield a
        print("World")
    for i in foo():
        print(i)
```
3. Define `hailstone_sequence`, a generator that yields the hailstone sequence. Remember, for the hailstone sequence, if \( n \) is even, we need to divide by two. Otherwise, we multiply by 3 and add by 1.

```python
def hailstone_sequence(n):
    """
    >>> hs_gen = hailstone_sequence(10)
    >>> next(hs_gen)
    10
    >>> next(hs_gen)
    5
    >>> for i in hs_gen:
    ...     print(i)
    16
    8
    4
    2
    1
    """
```

4. Define `tree_sequence`, a generator that iterates through a tree by first yielding the root value and then yielding the values from each branch.

```python
def tree_sequence(t):
    """
    >>> t = Tree(1, [Tree(2, [Tree(5)]), Tree(3, [Tree(4)])])
    >>> print(list(tree_sequence(t)))
    [1, 2, 5, 3, 4]
    """
```
1. What are the differences between streams and scheme lists? What’s the advantage of using a stream over a linked list?

2. What’s the maximum size of a stream?

3. When is the next element actually calculated?
4. What Would Scheme Display?

(a) `scm> (define x 1)

(b) `scm> (define p (delay (+ x 1)))

(c) `scm> p

(d) `scm> (force p)

(e) `scm> (define (foo x) (+ x 10))

(f) `scm> (define bar (cons-stream (foo 1) (cons-stream (foo 2) bar)))

(g) `scm> (car bar)

(h) `scm> (cdr bar)

(i) `scm> (define (foo x) (+ x 1))

(j) `scm> (cdr-stream bar)

(k) `scm> (define (foo x) (+ x 5))

(l) `scm> (car bar)

(m) `scm> (cdr-stream bar)
3 Code Writing for Streams

1. Implement double_naturals, which is a stream that evaluates to the sequence 1, 1, 2, 2, 3, 3, etc.
   (define (double-naturals)
      (double-naturals-helper 1 #f)
   )
   (define (double-naturals-helper first go-next)

2. Implement interleave, which returns a stream that alternates between the values in stream1 and stream2. Assume that the streams are infinitely long.
   (define (interleave stream1 stream2)
4 Challenge Question

1. (Optional) Write a generator that takes in a tree and yields each possible path from root to leaf, represented as a list of the values in that path. Use the object-oriented representation of trees in your solution.

```python
def all_paths(t):
    ""
    >>> t = Tree(1, [Tree(2, [Tree(5)]), Tree(3, [Tree(4)])])
    >>> print(list(all_paths(t)))
    [[1, 2, 5], [1, 3, 4]]
    ""
    if ____________:
        yield ____________
    for ________________:
        for ________________:
            __________________________
```