

# TAIL RECURSION AND STREAMS

---

COMPUTER SCIENCE MENTORS 61A

April 10 to April 14, 2017

---

## 1 Tail Recursion

---

1. What is a tail context/tail call? What is a tail recursive function?
2. Why are tail calls useful for recursive functions?

Answer the following questions with respect to the following function:

```
(define (sum-list lst)
  (if (null? lst)
    0
    (+ (car lst) (sum-list (cdr lst)))))
```

3. Why is sum-list not a tail call? Optional: draw out the environment diagram of this sum-list with list: (1 2 3). When do you add 2 and 3?
4. Rewrite sum-list in a tail recursive context.

## 2 Streams

---

A Stream is a linked list where the first element is calculated, but the rest isn't until it is needed. Here is the definition of Stream:

```
class Stream:
    class empty:
        def __repr__(self):
            return 'Stream.empty'

    empty = empty()

    def __init__(self, first, compute_rest=lambda: Stream.empty):
        assert callable(compute_rest), 'compute_rest must be callable.'
        self.first = first
        self._compute_rest = compute_rest

    @property
    def rest(self):
        """Return the rest of the stream, computing it if necessary."""
        if self._compute_rest is not None:
            self._rest = self._compute_rest()
            self._compute_rest = None
        return self._rest

    def __repr__(self):
        return 'Stream({0}, <...>'.format(repr(self.first))

empty_stream = Stream.empty
```

Here is an example of how to construct a Stream of integers:

```
def make_integer_stream(first=1):
    def compute_rest():
        print( computing rest )
        return make_integer_stream(first+1)
    return Stream(first, compute_rest)
```

---

## 2.1 General Streams

---

1. Whats the advantage of using a stream over a linked list?
2. Whats the maximum size of a stream?
3. Whats stored in first and rest? What are their types?
4. When is the next element actually calculated?

---

## 2.2 What Would Python Print?

---

1. For each of the following lines of code, write what Python would output.

```
>>> a = make_integer_stream()  
>>> a
```

```
>>> a.first
```

```
>>> a.rest
```

```
>>> a.rest
```

```
>>> a.rest.rest.rest
```

```
>>> a.rest.rest
```

```
>>> a.rest.rest.rest.rest.first
```

## 2.3 Code Writing for Streams

---

1. Write out `double_naturals`, which is a stream that evaluates to the sequence 1, 1, 2, 2, 3, 3, etc.

```
def double_natural(first=1, flag=0):
```

```
    """
```

```
        >>> a = double_natural()
```

```
        >>> a.first
```

```
        1
```

```
        >>> a.rest.rest.first
```

```
        computing rest
```

```
        computing rest
```

```
        2
```

```
    """
```

```
    def compute_rest():
```

```
        print('computing rest')
```

```
        #Your code here
```

```
    return Stream(first, compute_rest)
```

2. Write out `interleave`, which returns a stream that alternates between the values in `stream1` and `stream2`. Assume that the streams are infinitely long.

```
def interleave(stream1, stream2):
    """
    Note: ignore "compute rest" prints from make_integer_stream.
    >>> s1, s2 = make_integer_stream(1), make_integer_stream(10)
    >>> mixed = interleave(s1, s2)
    >>> mixed.first
    1
    >>> mixed.rest.first
    10
    >>> very_mixed = interleave(mixed, mixed)
    >>> very_mixed.first
    1
    >>> very_mixed.rest.first
    1
    >>> very_mixed.rest.rest.first
    10
    >>> very_mixed.rest.rest.rest.first
    10
    """
```