WEEK 11 STUDIO

AGENDA

- Comments from Mission
- Reading assessment
- Streams - What the *heck* is going on?
- Studio
- Relook and reorganize teaching materials

COMMENTS FROM RECENT MISSION

- Don't ditch your recursion just yet! Recursion is a powerful concept and is still relevant now.
- It is a way of thinking and helps to solve certain problems in a much straightforward manner than using iteration loops
    - `d_merge`'s recursive solution is arguably more elegant than its iterative counterpart

READING ASSESSMENT 2

It's okay to make mistakes, better now than during finals.

But if they're careless ones,
kill urself.
jk.

STREAMS

Hell no

STREAMS

In order to generate an infinite list of `{1, 1, 1, ...}`, you are given the two approaches as follows:

```
// 1st approach
const ones = pair(1 , ones);
// 2nd approach
const ones = pair(1 , () => ones);
```

What happens here?

STREAMS

- Main idea is **delayed evaluation** - you are delaying your evaluation till the time when you need it.
- Definition of streams - just like definition of lists
    - A stream of a certain type is either `null`, or a pair whose head is of that type and whose tail is a nullary function that returns a stream of that type. (from lecture notes)
- Stream discipline, just like list discipline
- Stream functions are not that different from list functions
    - Just add the nullary function at the tail of the list, and the `tail` calls are changed to `stream_tail` instead

STREAMS

- How to think with streams?
    - Use abstraction, wishful thinking and HOF (higher order functions) thinking. Always try to look at the big picture and visualize what the stream should give you.
    - You can look at the specifics and details, but only use that to convince yourself that it works. Getting too sucked into the details might confuse you about what's happening, so take a step back, and observe what each stream represents on a higher level.

STREAMS

A function that creates an infinite stream. Note the recursive call: why is a base case not needed?

```
function integers_from(n) {
    return pair(n, () => integers_from(n + 1));
}
```

STREAMS

More and more. Very fun stream!

```
function more(a, b) {
    if (a > b) {
        return more(1, b + 1);
    } else {
        return pair(a, () => more(a + 1, b));
    }
}
```

STREAMS

Fibonacci streams

```
function fib_list(n) {
    function helper(a, b, count, result) {
        if (count === n) {
            return result;
        } else {
            return helper(b, a + b, count + 1, pair(a, result));
        }
    }
    return reverse(helper(0, 1));
}

function fib_stream(n) {
    function fibgen(a, b) {
        return pair(a, () => fibgen(b, a + b));
    }
    return eval_stream(fibgen(0, 1), n);
}
```

STREAMS

More Fibonacci streams!

```
const fibs =
    pair(0,
         () => pair(1,
                    () => add_streams(stream_tail(fibs),
                                      fibs)));
```

ATTENDANCE

STUDIO

QUESTION 2

What is happening here?

```
function stream_pairs(s) {
    return is_null(s)
        ? null
        : stream_append(
            stream_map(
                sn => pair(head(s), sn),
                stream_tail(s)),
            stream_pairs(stream_tail(s)));
}
```

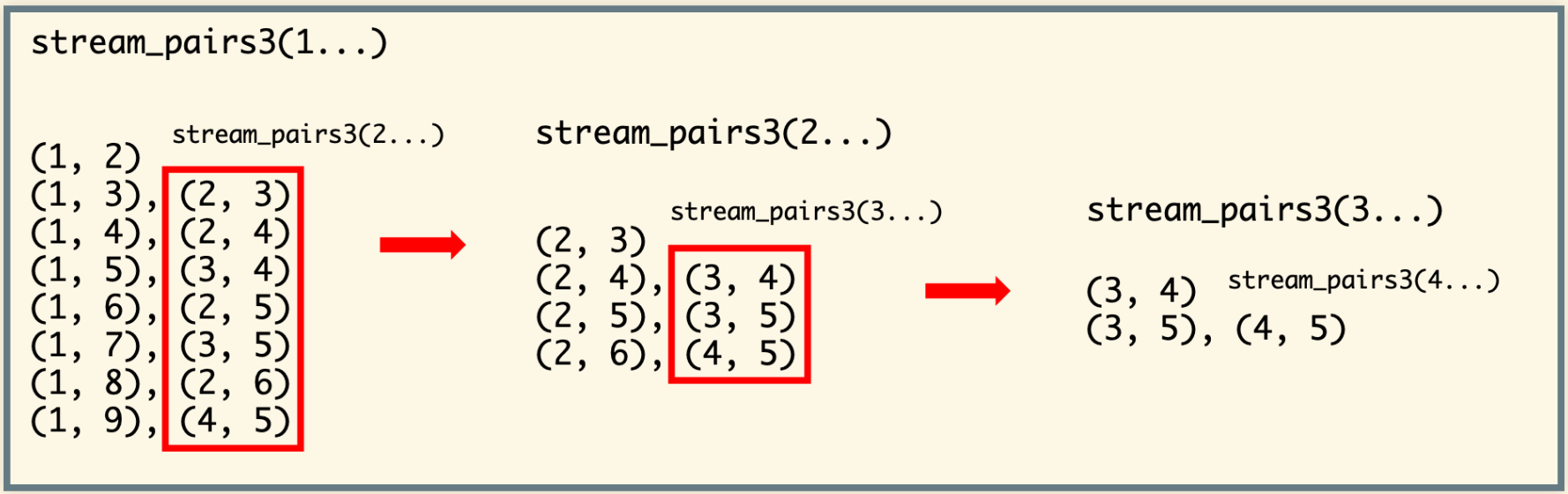
QUESTION 2 - INTERLEAVING STREAMS

```
function interleave_stream_append(s1, s2) {
    return is_null(s1)
        ? s2
        : pair(head(s1),
               () => interleave_stream_append(s2, stream_tail(s1)));
}

function stream_pairs3(s) {
    return (is_null(s) || is_null(stream_tail(s)))
        ? null
        : pair(pair(head(s), head(stream_tail(s))),
            () => interleave_stream_append(
                    stream_map(x => pair(head(s), x),
                               stream_tail(stream_tail(s))),
                    stream_pairs3(stream_tail(s))));
}
```

QUESTION 2 - INTERLEAVING STREAMS

How does interleaving work? Look at the results:



QUESTION 3

Sample solutions

QUESTION 4

- Look at how the problem is defined. Can you spot a recursive pattern?
- How can you make use of this pattern and the given prompt to produce the answer that you want?

QUESTION 4

Multiplying streams

```
function mul_series(s1, s2) {
    return pair(head(s1) * head(s2),
        () => add_series(
            scale_stream(head(s2), stream_tail(s1)),
            mul_series(s1, stream_tail(s2))));
}
```

- Look at the definition of how you can multiply two large numbers provided in the studio sheet. Does it look recursive?
- Same thing as before, stay on the outside, look at the big picture and don't get too sucked into the details if you are not confident.