WEEK 5 STUDIO

AGENDA

- Reading Assessment
- Updates
- Data Structures - Lists & Pairs
- Identity vs Equality
- Studio
- Extra Questions

READING ASSESSMENT

Trick question!

```
function g(x) {
    function g(x) {
        function g(x) {
            return (x <= 3) ? 34 : g(x - 3);
        }
        return (x <= 2) ? 23 : g(x - 2);
    }
    return (x <= 1) ? 12 : g(x - 1);
}

g(100);
```

- Any other questions you want me to go through?

UPDATES

- Source 2 documentation (**Highly recommend reading**)
- No unsubmit option for missions anymore. **Check** before you submit!

DATA STRUCTURES

- Data structures can be seen as containers that store information.
- Functions as data structures!
- Different data structures have different ways of interacting, processing and manipulating the data stored inside them.
- Definitions of data structures are important, because they **define precisely how you can interact with the data within them**.

LISTS

Definition: A list is either `null` or a pair whose tail is a list

Definition of a list is recursive!

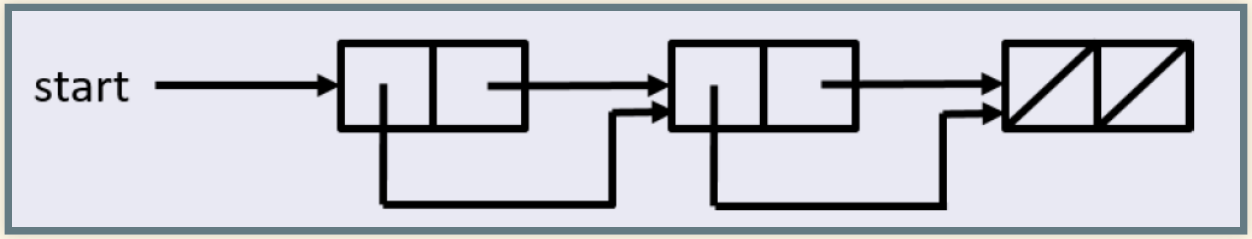
```
/*
list(1, 2, 3, 4) -> pair(1,
                         pair(2,
                              pair(3,
                                   pair(4, null))))
*/
```

BOX AND POINTER DIAGRAMS

- Head
  - Always contains current item if the current item is **primitive**
  - Otherwise, contains a pointer to the current item
- Tail
  - Points to the next item in the sequence
  - Contains `null` if it is the last item of a list.
  - Note that `null` can also be considered as an item!

BOX AND POINTER DIAGRAMS

Write out a program which gives this box and pointer diagram:



Is this a list?

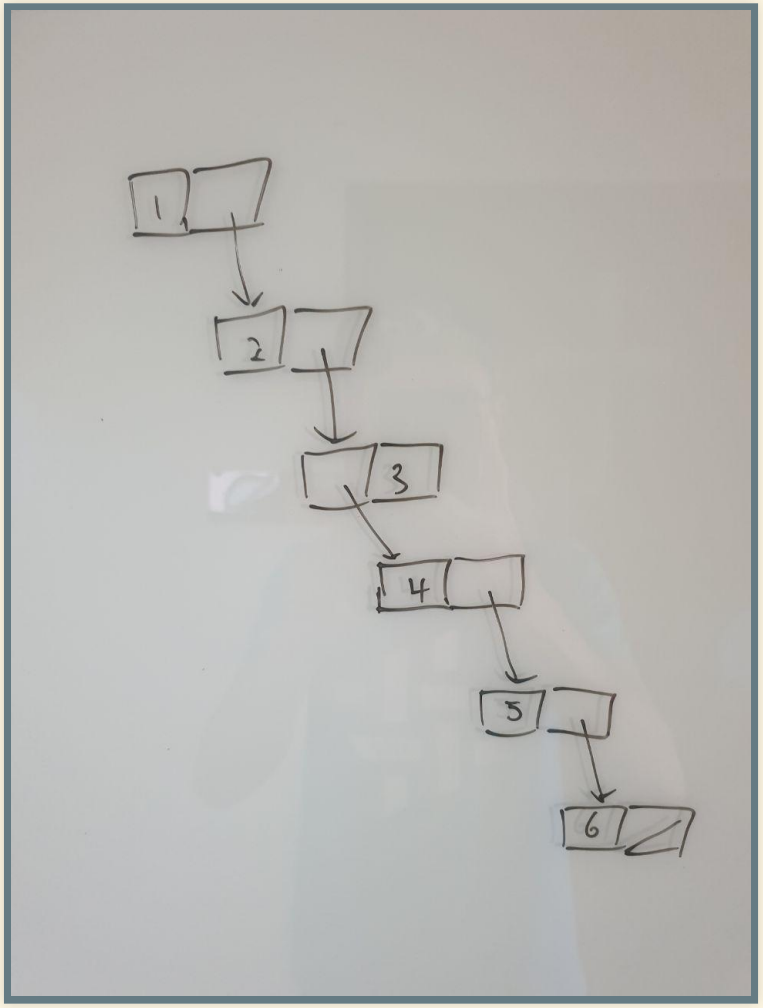
SOME PRACTICE WITH LISTS

```
// Is this a list?

pair(null, null);
pair(pair(pair("fwe", "hello"), null),
     pair(pair(1, 2), pair(null, null)));
```

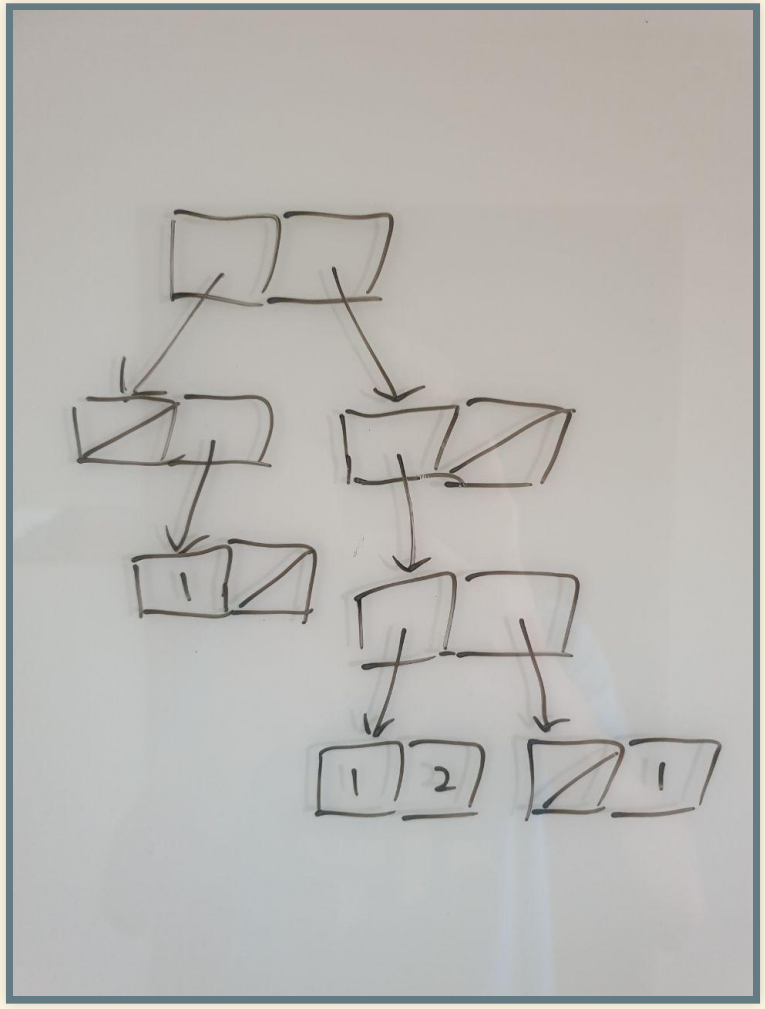
SOME PRACTICE WITH LISTS

Is this a list?



SOME PRACTICE WITH LISTS

Is this a list?



SOME PRACTICE WITH LISTS

- `is_pair()` checks if input is a pair
- Can we do the same for lists as well?

> *Create a recursive function `is_list()` that returns `true` if its input is a list, and `false` otherwise.*

SOME PRACTICE WITH LISTS

How to check for equality of lists?

Create a recursive function `equal()` *that takes in two list inputs, x and y, and returns* `true` *if list x is equal to list y, and* `false` *otherwise*

What kind of structure will result in the worst possible performance for this function?

EQUALITY VS IDENTITY

- Equality and Identity are not the same
- **Identity** - Two things are actually the same object, just with different names.
- **Equality** - Two things hold the same value (or have the same struture), but are different objects.

EQUALITY VS IDENTITY

- **Booleans**: straightforward
- **Strings**: straightforward
- **Numbers**: straightforward for small integers, but not the case for non-intergers and large numbers
- **Functions**: Two separately defined functions are always not identical, even if they have the exact same behaviour
- **Lists**: Two separately defined lists are always not identical, even if they have the same value and structure

EQUALITY VS IDENTITY

```
const p1 = pair;
const p2 = pair;
p1 === p2;

const p1 = pair(1, 2);
const p2 = pair(1, 2);
p1 === p2;

const l1 = list;
const l2 = list;
l1 === l2;

const l1 = list();
const l2 = list();
l1 === l2;
```

EQUALITY VS IDENTITY

```
// Given the following definitions:
const random1 = () => null;
const random2 = () => null;
const random3 = null;
const random4 = null;
const a = random1;
const b = random1;

// Why do these two lines give different results?
random1 === random2; // false
random3 === random4; // true

// What does this evaluate to?
random1() === random3;
a === b;
```

ATTENDANCE

STUDIO 5

REVERSING A LIST

```
function reverse(list) {
    function reverse(lst, newlst) {
        return is_null(lst)
            ? newlst
            : reverse(tail(lst), pair(head(lst), newlst));
    }
    return reverse(list, null);
}
```

STUDIO Q3

STUDIO Q5

Click on the link and look at the first sum function defined. Is the implementation correct? Why or why not?

EXTRA QUESTIONS

Given two lists of the same length `xs` and `ys`, try to construct a 3rd list of the same length in which each element is a pair composed of the elements in the same position from `xs` and `ys`. Your function name should be called `make_pairs`.

```
// For example, the following returns:
// list(pair(1, 11), pair(2, 12), pair(3, 13)).

make_pairs(list(1, 2, 3), list(11, 12, 13));
```

EXTRA QUESTIONS

Now, generalize this concept by defining a new function. Given two lists of the same length xs and ys, try to construct a 3rd list of the same length in which each element is the result of applying a certain zip function to the two elements on the same position from xs and ys. Your function name should be zip.

```
// For example, the following will return list(11, 24, 39)

zip((x, y) => x * y,
    list (1, 2, 3),
    list (11, 12, 13));
```