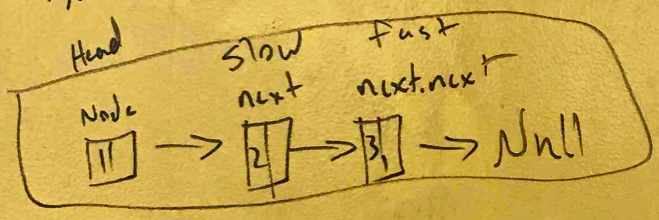
Jacob
E.

1) Create Singly list with object literal
2) Traverse list <- fast & slow
3) Note BigO

```
node ={"node: 1",
       "node.next: 2",
       "node.next.next: 3".
};
```




```
Const traverseNodes=(node) =>{

  let NextNode = node.next

  while (Node.next !==null){
    return node.value
```

}
}

Part 1

Matthew LeBlanc

Array [numbers]

{ biggest: num,
  2nd biggest: num2, }

fun [ ]

if array length is null
    return { };

if array length is 1
    return { biggest: array[0]
             2nd biggest: null }

let biggest, 2nd bigger

fun (each in array)
  if each = array[0] => set values;
  if # > biggest
    ① biggest = each 10
    ② 2nd biggest = biggest
  else if # > 2nd biggest
    2nd biggest = each 8;

[1, 7, 8, 3, 2, 10]

return { biggest, 2nd biggest }

Part 2

```
Find Biggest And Second Biggest (arr) {
  if (arr.length === 0)
      return {};
  if (arr.length === 1)
      return { biggest: arr[0],
               secondBiggest: null };

  let biggest, secondBiggest;
  for (let i; i < arr.length; i++) {
    if(i === 0)
      return biggest = arr[i];
    if(arr[i] > biggest) {
      secondBiggest = biggest;
      biggest = arr[i]
    }
  else if (arr[i] > secondBiggest)
      secondBiggest = arr[i]
    }
  return { biggest, secondBiggest }
}
```

Part I:

```
let LinkedListRepresentation = {
    value: 10,
    next: {
        value: 20,
        next: {
            value: 30,
            next: null
        }
    }
};
```

LinkedLists have a memory complexity of O(n) where n is the # of nodes in the list

Part II:

```
const traverseLinkedList = head => {
    if (!head instanceof LinkedListClass) {
        throw new TypeError(<head> must be an instance of LL);
    }
    if (!head) {
        return null;
    }
    let current = head;
    while (current) {
        console.log(current.value);
        current = current.next;
    }
}
```

Cameron

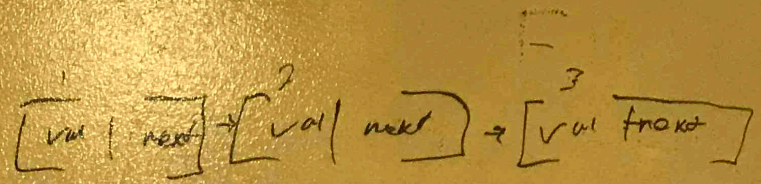
The traverseLinkedList fn has a linear runtime complexity of O(n) where n is the # of elements in the list. It also has a constant memory complexity of O(1).

head = null,

head = 10 → 20 → 30 → X

Matthew LeBlanc

12/4/18




```
class List {
  constructor(value) {
    this.value = value,
    this.next = null,
  }
  append(node) {
    if(!node instanceof List)
      return 'not instance';
    If (this.next === null)
    return this.next = node;
    else
      return this.next.append(node);
  }
  evaluate() {
      console.log(this.value);
      if (!this.next === null)
          this.next.evaluate();
      return 'evaluate over';
  }
}
```

evaluate each value
+(console.log)

```
let Class { ✓
  constructor(value)  {
    this.value = value,
    this.next = null,
  }                                  ✓
  append(node) {   if not instance
    if (this.next === null)
        this.next = node
    else
      this.next.append(node)
                                     ✓
  evaluate (  ) {   if not instance
    console.log(this.value)
    this.next.evaluate();
```

```
Let first = new List('one')
Let second = new List('two')
Let third = new List('three')

first.append(second)
first.append(third)

first.evaluate();
    => log: 'one'
       log: 'two'
       log: 'three'
```

Andrew

problem: take
with

pseudo code:

check
nul

declare
for loop
if i =
set
if i =
che
if
and
other

if i >
if
s
s
else

return

Nicholas C

```
let vinicio = {
  name: 'Vinicio',
  next: {
    name: 'Sam',
    next: null
  }
}
```

traverse(list) = O(n)

n = list length

1. if
2. elif
3. return

```
const traverse = (node) => {
  console.log(node)
  if (typeof node !== 'object') {
    throw new TypeError('must be linked list')
  }
  if(node.next === null)
    return node;
  return traverse(node.next)
}

traverse(vinicio)
```

logs : vinicio

logs: { name: 'Sam', next: null }

returns { name: 'Sam', next: null }

Andrew 12/4/17

Create an object literal linked list as well as a function which traverses it.

declare object literal with nodes

declare a function
function should log current node,
function should call itself on the next node if the next is not null

```
"use strict";

let linkedList = {                                  O(1)
    'value' : 1,
    'next' : {
        'value' : 2,
        'next' : {
            'value' : 3,
            'next' : null
            }
        }
    }
}
function traverseList(list){
    console.log(list);                              O(1)
    if(list.next)
        return traverseList(list.next);
    }                                   ← O(# of nodes in linked list)
}
traverseList(linkedList)
```

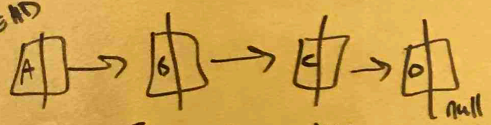
DAVID LINDAHL

- write out a linked list using object literal format
- write function to traverse list using
- document Big O

PSUEDOCODE: ⟷ **LEGIT CODE**

OBJECT with Key Value Pairs of value, & next for each node

HEAD



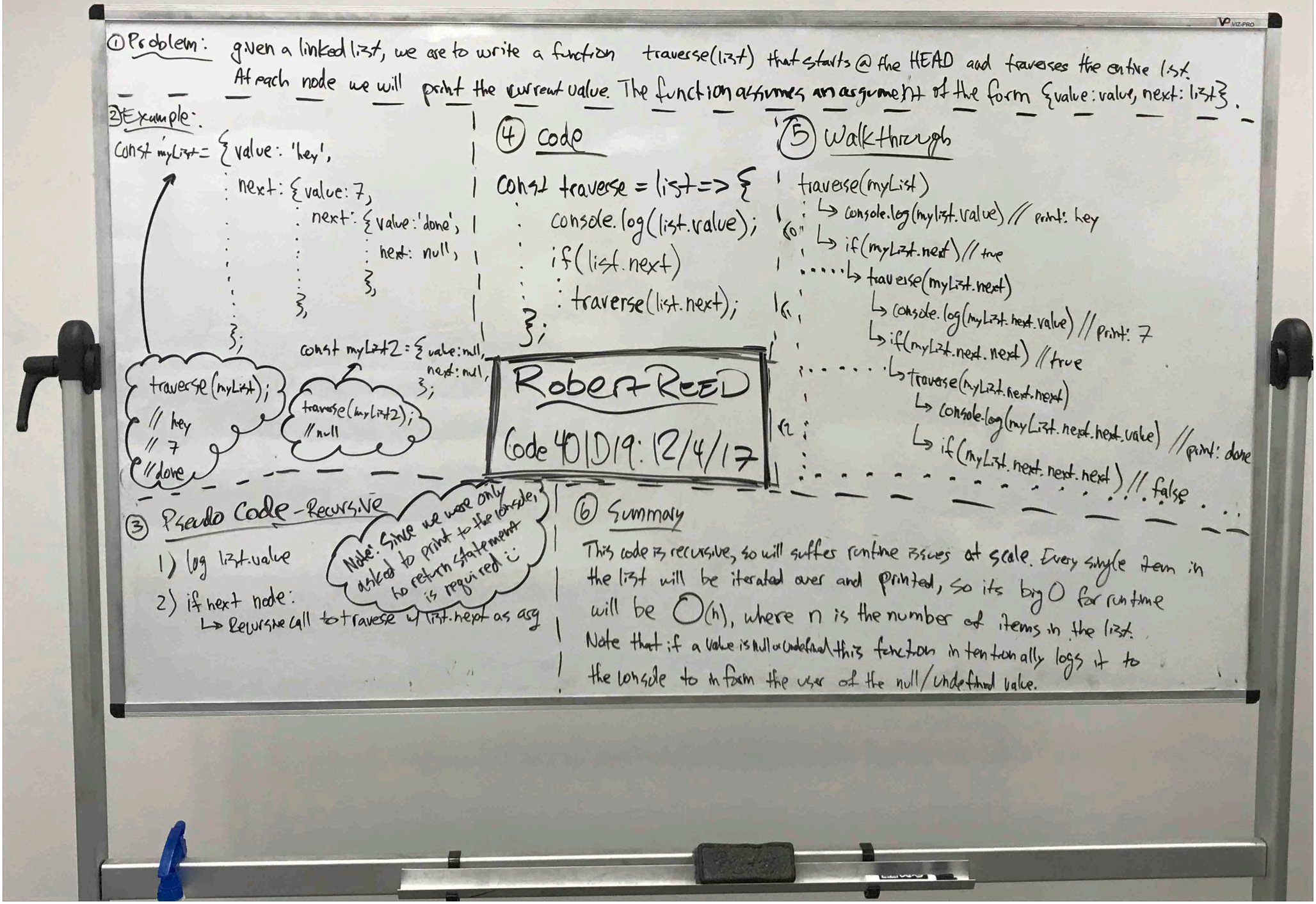
{ node1 : nextNode...
node.value, node.next

```
function ()
    while value !===null
        logger.log 'info' ; node.value

    break;
```

```
linkedList = {
    "nodeA : next.node ;
    "nodeB : next.node";
    "node C : next.node" } ;
```

```
function(linkedList )=> {
    while (this.nodeValue !===null ) {
        logger.log ('info' ; this.node.value);
        //OR console.log
        console.log (this.node.value);

    };
}
```

Big O = O(n)


① Problem: given a linked list, we are to write a function traverse(list) that starts @ the HEAD and traverses the entire list. At each node we will print the current value. The function assumes an argument of the form {value: value, next: list}.
② Example:
const myList = { value: 'hey', next: { value: 7, next: { value: 'done', next: null } } };
const myList2 = { value: null, next: null };
traverse(myList); // hey // 7 // done
traverse(myList2); // null
③ Pseudo Code - Recursive
1) log list.value
2) if next node:
↳ Recursive call to traverse w/ list.next as arg
Note: Since we were only asked to print to the console, no return statement is required :)
④ Code
const traverse = list => {
console.log(list.value);
if(list.next)
traverse(list.next);
};
Robert Reed
Code 401 D19: 12/4/17
⑤ Walkthrough
traverse(myList)
↳ console.log(myList.value) // print: hey
↳ if(myList.next) // true
↳ traverse(myList.next)
↳ console.log(myList.next.value) // print: 7
↳ if(myList.next.next) // true
↳ traverse(myList.next.next)
↳ console.log(myList.next.next.value) // print: done
↳ if(myList.next.next.next) // false
⑥ Summary
This code is recursive, so will suffer runtime issues at scale. Every single item in the list will be iterated over and printed, so its big O for runtime will be O(n), where n is the number of items in the list. Note that if a value is null or undefined this function intentionally logs it to the console to inform the user of the null/undefined value.

Part 1

Jeff Kusowski

assumptions
Valid LinkedList

```
let list = {
  "value" : 1,
  "next" : {
      "value" : 2,
      "next" : {
          "value" : 3,
          "next" : null
          }}};
```

Part 2

```
const traverse = (list) => {
  console.log(list.value);
  if (!list.next){
    return;
    }
  return traverse(list.next)
  }
traverse(list);
```

O(n)

RESTATE PROBLEM

1) REPRESENT A SINGLY LINKED LIST USI
OBJECT LITERAL

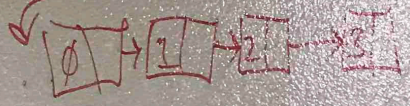
2 WRITE A FUNCTION TO TRAVERSE THE LIST

CONST TRAVERSE = (LIST) => {

3

EXAMPLE + PSEUDOCODE

CREATE LIST



TRAVERSE LIST

READ → 0 VAL 1 → LOG IT, READ → 1 VAL 2 → LOG IT, READ → 2 VAL 3 → LOG IT, READ → 3 VAL 4 → LOG IT

↳ DONE.

{
'KEY 0': 1, → NEXT: {
'KEY 1': 2, NEXT: {
'KEY 2': 3, NEXT: {
'KEY 3': 4
}}}

CONST TRAVERSE = (LIST) => {

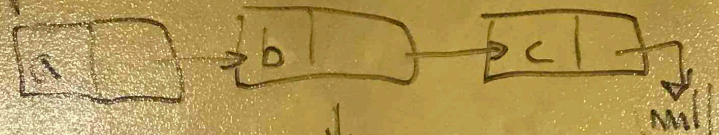
THIS.

Kerry Nordstrom
12/4/17

Restate:

Represent a linked list using an object literal

Part 1:




```
let list = {
  val: a
  next: {
    val: b
    next: {
      val: c
      next: null
    }
  }
}
```

list { value: a
next: { value
n c

Part 2:

```
const traverse = (list) => {
  while (list) {
    console.log (list.value)
    list = list.next
  }
}
```

RESTATE PROBLEM

1) REPRESENT A SINGLY LINKED LIST USING OBJECT LITERAL

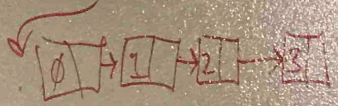
2 WRITE A FUNCTION TO TRAVERSE THE LIST

CONST TRAVERSE = (LIST) => {

3

EXAMPLE + BOILERPLATE

CREATE LIST



TRAVERSE.LIST

READ → 0 val 1 → LOG IT, READ → 1 val 2 → LOG IT, READ → 2 val 3 → LOG IT, READ → 3 val 4 → LOG IT

DONE.

FREDRIC (NOT FREDERIC)

{

'KEY 0': 1, NEXT: {

'KEY 1': 2, NEXT: {

'KEY 2': 3, NEXT: {

'KEY 3': 4

333

CONST TRAVERSE = (LIST) => {

THIS.

Seth Donihue

① Problem: - Represent a singly linked list using an object literal
- Write a function to traverse the linked list

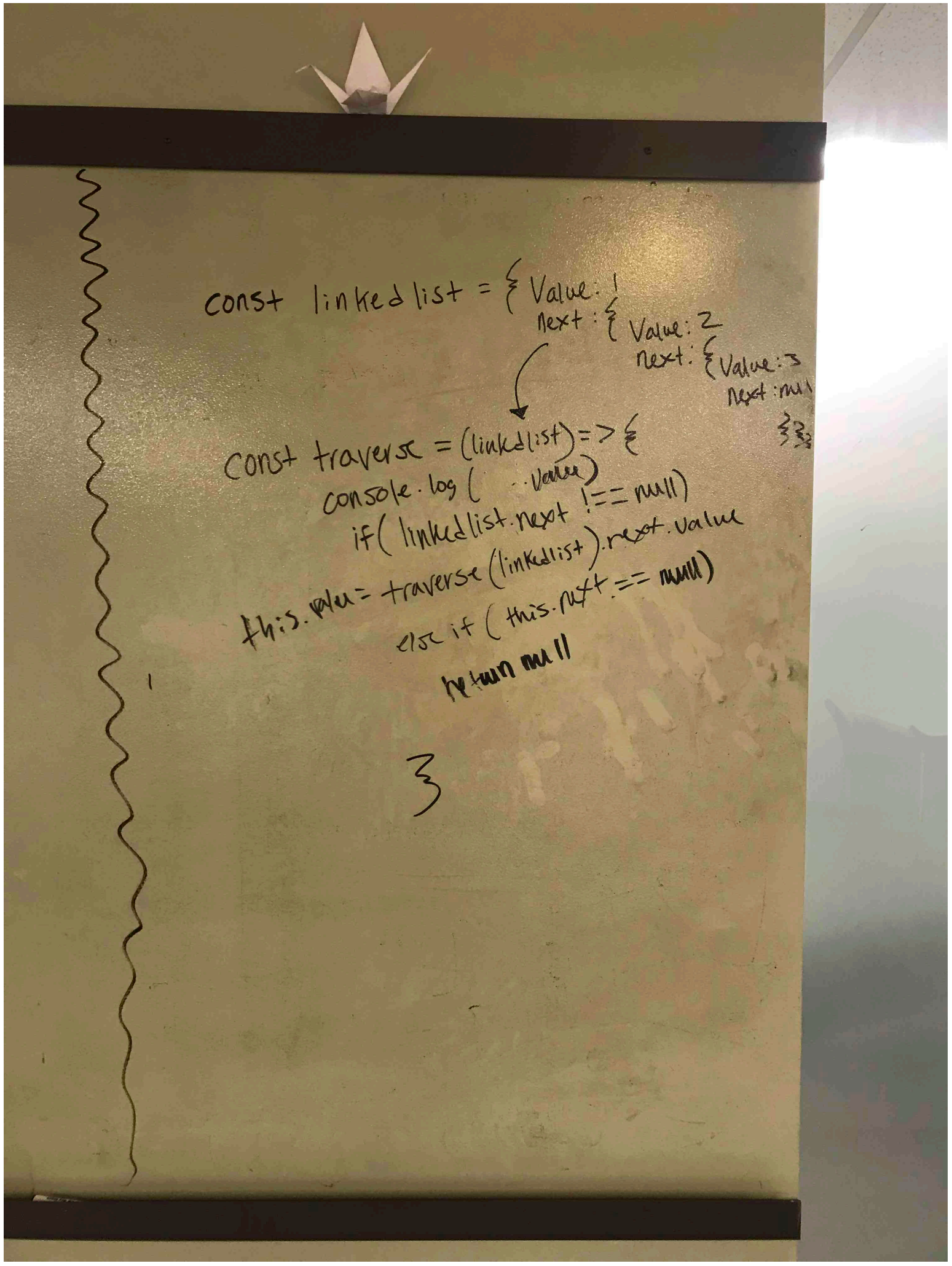
```
const traverse = (list) => {
```

Part 1: Input Linked List →

```
{"value": "value"
  "next": {"value": "value"
    "next": {"value": "value"
      "next": null
    }
  }
};
```

Part 2

```
Const travers = (list) => {
```


const linkedlist = { Value: 1
next: { Value: 2
next: { Value: 3
next: null
}}}
const traverse = (linkedlist) => {
console.log(.value)
if(linkedlist.next !== null)
this.value = traverse (linkedlist).next.value
else if (this.next == null)
return null
3

Problem Domain

Create a singly linked list with an object literal. Write a function to traverse the linked list

O(n) → n is the list of nodes

```
const traverse = list => {};
```

Catherine Looper
12/4/17

```
const LinkedList = {
  value = this.value;
  next = this.null;
};

const traverse (list) => {
  while (list) {
    if (list.next === null) {
      return list;
    } else {
      console.log(list.value);
      traverse(list.next);
    };
  };

  traverse(list);
```

1. Create an analogue of a "Singly-linked List" w/ an Object literal.
2. Write a function that traverses that entire L.L., printing out each node's payload.

```
traverse=(list) => {
  console.log list.data.toString();
  if (list.next !== null) traverse(list.next);
}
```

Phelan

```
{
'data': 'ASDF',
next: {
  data: 'JKLS',
  next: {
    data: 'XZYC',
    next: null
    }
  }
}
```