**UDACITY’s Hackreactor JAVASCRIPT:**

**Git:**

\_Git init . first

\_Config global user.email “my-real-email”

\_Then Git add .

\_If the remote branch & local branch are different, use git pull origin master first, then commit and push again

**Scope:**

\_Variable inside the curly braces can have access to those on the outer scope and global scope

\_No var === global

EXECUTION CONTEXT vs Lexical cope:

\_In-memory scope structure built as the program executes. It builds all the ones outside first, and then go in

<https://classroom.udacity.com/courses/ud015/lessons/2593668697/concepts/25411890600923>

\_If you call a function many times, each time it will create a new execution context (memory space in the memory). Will not replace anything

\_Function object: Any object that can be called as if it was a function

THIS:

Q: What is “this” bound to?

A: A function that is looked up when it’s being evoked

\_”This” behaves like parameters

No binding of a parameter until the function gets called

\_If there’s no variable.method form (which ‘this’ will refer to the variable), then it will refer to the global scope

\_To bind “this” to a function: Use .call(parameters), so “this” will be bounded to that function, not the globak object

\_red.method.call(yellow,green,blue) => will give me a yellow, green,blue because .call overrides the “.method” rule

\_If no value is passed to the function for setTimeOut, the output is undefined

If there’s a keyword “this” and it isn’t bound to anything like this

Var fn = function (one,two){

Log(this,one,two);

};

* This will be bound to global

Q: WHAT IS THE MOMENT THAT INFLUENCES WHEE THE “THIS” KEYWORD WILL GET BOUND?

A: The moment of CALL TIME

This behaves like a parameter

Q: Method vs Function?

A: Piece of code that can be performed on objects..

Identical to a function, except: method might not return a value

Ex:

var str = "Apple, Banana, Kiwi";

var res = str.slice(7, 13);

Q: To add a function as a method?

A: Add that function as a property of an object

Ex:

Var fn = function (one,two){

Log(this,one,two);

};

var r={}, g ={}, b={};

**r.method = fn;**

* So when you invoke the r.method(g,b) method, r wil get invoked first, then g and b

Don’t do this:

setTimeout(r.method, 1000)

Q: Why?

A: Because if it’s a call-back, you have little control over how it’s called. Totall dependent on the system

Q: So what now?

A: Pass a different function

setTimeout(function(){

r.method

}, 1000)

Q: What’s the output?

Var fn = function (one,two){

Log(this,one,two);

};

setTimeout(function(){

r.method

}, 1000)

log(one)

A: undefined because we only defined it within the context of fn, not in global scope

**PROTOTYPE CHAIN:**

\_If you create an object from a prototype, and you try to log a property that only exist on its prototype, it goes up the chain

ex: var gold = {a:1}

var rose = Object.create(gold)

rose.b = 2;

log(rose.a) => (notice “a” isn’t a property on “rose”)

2 ways to create prototype chian:

1. **Extend** : extend 1 time
2. **Object.create:** create ongoing lookup

Ex: var gold ={a:1}

1. Var blue = extend ( {}, gold)

Blue.b = 2;

* Blue will only copy 1 time the properties of the gold object. So if I add a z property after I already copied “blue”, it won’t be copied and returns undefined

1. Var rose = Object.create(gold);

Rose.b = 2;

* Ongoing lookup, so it will still return value of z with this method