/Week 3 - Day 2

Section 1

JavaScript Total Recall

Setup

For this lab, create a `HTML, CSS & JS REPL` from repl.it -- you can name it `"JavaScript Review Lab"` .

I. Variables & Datatypes

To answer these questions, you can add them in a multiline comment section inside of `script.js`

like this:

```
/*
1. How do we assign a value to a variable? A. With the assignment operator
2. How do we change the value of a...
...
*/
```

A. Q + A

1. How do we assign a value to a variable?
2. How do we change the value of a variable?
3. How do we assign an existing variable to a new variable?
4. Remind me, what are `declare` , `assign` , and `define` ?
5. What is pseudocoding and why should you do it?
6. What percentage of time should be spent thinking about how you're going to solve a problem vs actually typing in code to solve it?

B. Strings

For all other questions that involve writing code, you can solve them via the following instructions.

1. Create a variable called `firstVariable`

2. Assign it the value of the string `"Hello World"`
3. Change the value of this variable to some number
4. Store the value of `firstVariable` in a new variable called `secondVariable`
5. Change the value of `secondVariable` to any string.
6. What is the value of `firstVariable` ?
7. Create a variable called `yourName` and set it equal to your name as a string. Then, write an expression that takes the string "Hello, my name is " and the variable `yourName` so that it returns a new string with them concatenated.

   ex: `Hello, my name is Jean Valjean`

C. Booleans

- Using the provided variable definitions, replace the blanks so that all log statements print `true` in the console. Answers should be all be valid JS syntax and not weird things that don't make sense but happen to print `true` to the console

```
const a = 4;
const b = 53;
const c = 57;
const d = 16;
const e = 'Kevin';

console.log(a __ b);
console.log(c __ d);
console.log('Name' __ 'Name');
// FOR THE NEXT TWO, USE ONLY && OR ||
console.log(true __ false);
console.log(false __ false __ false __ false __ false __ true);
console.log(false __ false)
console.log(e ___ 'Kevin');
console.log(a __ b ___ c); // note: a < b < c is NOT CORRECT (and is not a valid JS expressior
console.log(a __ a ___ d); // note: the answer is a simple arithmetic equation, not something
console.log(48 __ '48');
```

D. The farm

1. Declare a variable `animal` . Set it to be either "cow" or something else
2. Write code that will print out "mooooo" if the it is equal to `cow`
3. Change your code so that if the variable `animal` is anything other than a cow, it will print "Hey! You're not a cow."
4. Commit

E. Driver's Ed

1. Make a variable that holds a person's age; be semantic
2. Write code that will print out "Here are the keys!", if the age is 16 years or older, or, if the age is younger than 16, a message should print "Sorry, you're too young."

II. Loops

Remember: **USE `let` when you initialize your for loops!**

This is GOOD: for(**let i = 0;** i < 100; i++)

This is NO GOOD: for(i = 0; i < 100; i++)

A. The basics

1. Write a loop that will print out all the numbers from 0 to 10, inclusive
2. Write a loop that will print out all the numbers from 10 up to and including 400
3. Write a loop that will print out every third number starting with 12 and going no higher than 4000

B. Get even

1. Print out the numbers that are within the range of 1 - 100
2. Adjust your code to add a message next to even numbers only that says: "<-- is an even number"

C. Give me Five

1. For the numbers 0 - 100, print out "I found a `number`. High five!" if the number is a multiple of five

   Example Output:

```
I found a 5. High five!
I found a 10. High five!
```

1. Add to the code from above to print out "I found a `number`. Three is a crowd" if the number is a multiple of three

   Example Output:

```
I found a 3. Three is a crowd
I found a 5. High five!
I found a 6. Three is a crowd
I found a 9. Three is a crowd
I found a 10. High five!
```

1. For numbers divisible by *both three and five*, be sure your code prints both messages

D. Savings account

1. Write code that will save the sum of all the numbers between 1 - 10 to a variable called `[illegible]`

1. Write code that will save the sum of all the numbers between 1 - 10 to a variable called `bank_account`.

   Check your work! Your bank_account should have $55 in it.

2. You got a bonus! Your pay is now doubled each week. Write code that will save the sum of all the numbers between 1 - 100 multiplied by 2.

   Check your work! Your bank_account should have $10,100 in it.

III. Arrays & Control flow

A. Talk about it:

1. What are the things in an array called?
2. Do Arrays guarantee those things will be in order?
3. What real-life thing could you model with an array?

B. Easy Does It

1. Create an array that contains three quotes and store it in a variable called `quotes`

C. Accessing elements

Given the following array `const randomThings = [1, 10, "Hello", true]`

1. How do you access the 1st element in the array?
2. Change the value of `"Hello"` to `"World"`
3. Check the value of the array to make sure it updated the array. How? Why, yes! `console.log()`;

D. Change values

Given the following array `const ourClass = ["Salty", "Zoom", "Sardine", "Slack", "Github"]`

1. What would you write to access the 3rd element of the array?
2. Change the value of "Github" to "Octocat"
3. Add a new element, "Cloud City" to the array

E. Mix It Up

Note: You don't really need `.splice()` for these. You *could* use it, but there are simpler array methods that are more appropriate.

Given the following array: `const myArray = [5, 10, 500, 20]`

1. Add the string `"Aegon"` to the end of the array. Add another string of your choice to the end of the array.
2. Remove the `5` from the beginning of the array.
3. Add the string `"Bob Marley"` to the beginning of the array.
4. Remove the string of your choice from the end of the array.
5. Reverse this array using `Array.prototype.reverse()`. Did you mutate the array? What does *mutate* mean? Did the `.reverse()` method return anything?

F. Biggie Smalls

Create a variable that contains an integer.

Write an `if ... else` statement that:

1. `console.log()` s "little number" if the number is entered is less than **100**
2. `console.log()` s `big number` if the number is greater than or equal to 100.

G. Monkey in the Middle

Write an `if ... else if ... else` statement:

1. `console.log()` `little number` if the number entered is less than **5**.
2. If the number entered is more than 10, log `big number`.
3. Otherwise, log "monkey".

H. What's in Your Closet?

Below, we've given you examples of Kristyn and Thom's closets modeled as data in JavaScript.

```
const kristynsCloset = [
  "left shoe",
  "cowboy boots",
  "right sock",
  "GA hoodie",
  "green pants",
  "yellow knit hat",
  "marshmallow peeps"
];

// Thom's closet is more complicated. Check out this nested data structure!!
const thomsCloset = [
  [
    // These are Thom's shirts
    "grey button-up",
    "dark grey button-up",
    "light blue button-up",
    "blue button-up",
  ],[
    // These are Thom's pants
    "grey jeans",
    "jeans"
```

```
    "PJs"
  ],[
    // Thom's accessories
    "wool mittens",
    "wool scarf",
    "raybans"
  ]
];
```

1. What's Kristyn wearing today? Using bracket notation to access items in `kristynsCloset`, log the sentence "Kristyn is rocking that " + *the third item in Kristyn's closet* + " today!" to the console.
2. Kristyn just bought some sweet shades! Add `"raybans"` to her closet **after `"yellow knit hat"`.**
3. Kristyn spilled coffee on her hat... modify this item to read `"stained knit hat"` instead of yellow.
4. Put together an outfit for Thom! Using **bracket notation**, access the first element in Thom's `shirts` array.
5. In the same way, access one item from Thom's pants array.
6. Access one item from Thom's accessories array.
7. Log a sentence about what Thom's wearing. Example: `"Thom is looking fierce in a grey button-up, jeans and wool scarf!"`
8. Get more specific about what kind of PJs Thom's wearing this winter. Modify the name of his PJ pants to `Footie Pajamas`.

IV. Functions

A. `printGreeting`

Do you think you could write a function called `printGreeting` with a parameter `name` that returns a greeting with the argument **interpolated** into the greeting?

Like so?

```
console.log(printGreeting("Slimer"));
```

```
=> Hello there, Slimer!
```

You think you could? I think so too. Feel free to skip this problem, because you've already done it. If you've done the problem twice, **read entire problems carefully before doing them from now on**.

B. `printCool`

Write a function `printCool` that accepts one parameter, `name` as an argument. The function should print the name and a message saying that that person is cool.

```
console.log(printCool("Captain Reynolds"));
```

```
=> "Captain Reynolds is cool";
```

C. `calculateCube`

Write a function `calculateCube` that takes a single number and prints the volume of a cube made from that number.

```
console.log(calculateCube(5));
```

=> 125

D. `isVowel`

Write a function `isVowel` that takes a character (i.e. a string of length 1) and returns true if it is a vowel, false otherwise. The vowel could be upper or lower case. **Test your function on every vowel and make sure it's working.** In general, when you write functions, take a minute to test them with different values to make sure they behave the way you want.

```
console.log(isVowel("a"));
```

```
=> true
```

E. `getTwoLengths`

Write a function `getTwoLengths` that accepts two parameters (strings). The function should **return** an *array* of numbers where each number is the length of the corresponding string.

```
console.log(getTwoLengths("Hank", "Hippopopalous"));
```

```
=> [4, 13]
```

F. `getMultipleLengths`

Write a function `getMultipleLengths` that accepts a single parameter as an argument: an **array** of **strings**. The function should **return** an **array** of **numbers** where each number is the length of the corresponding string.

```
console.log(getMultipleLengths(["hello", "what", "is", "up", "dude"]));
```

```
=> [5, 4, 2, 2, 4]
```

G. `maxOfThree`

Define a function `maxOfThree` that takes three numbers as arguments and returns the largest of them. If all numbers are the same, it doesn't matter which one is returned. If the two largest numbers are the same, one of them should be returned. Be sure to test it with larger values in each of the three locations.

```
console.log(maxOfThree(6, 9, 1));
```

=> 9

Did you use Google and find `Math.max()` ? If so, great job! Very resourceful—keep looking stuff up! However, for this particular question, we need you to submit a solution that **does not** use `Math.max()` .

H. `printLongestWord`

Write a function `printLongestWord` that accepts a single argument, an **array** of **strings**. The method should return the longest word in the array. In case of a tie, the method should return the word that appears first in the array.

```
console.log(printLongestWord(["BoJack", "Princess", "Diane", "a", "Max", "Peanutbutter", "big",
```

`=> "Peanutbutter"`

Objects

Let's set up an object data structure. Let's say we have a website that sells products, and we have a user of our website, and we want to store that user's data. The object data structure is a good way to organize the data from our user.

A. Make a user object

1. Create an object called `user` .
2. Write in to the object the key-value pairs for `name` , `email` , `age` , and `purchased` . Set the value of `purchased` to an empty array `[]` . Set the other values to whatever you would like.

B. Update the user

1. Our user has changed his or her email address. Without changing the original `user` object, update the `email` value to a new email address.
2. Our user has had a birthday! Without changing the original `user` object, increment the `age` value using the postfix operator. Hint: `age++`

C. Adding keys and values

You have decided to add your user's location to the data that you want to collect.

1. Without changing the original `user` object, add a new key `location` to the object, and give it a value or some-or-other location (a string).

D. Shopaholic!

1. Our user has purchased an item! They have purchased some "carbohydrates". Using `.push()` , add the string "carbohydrates" to the `purchased` array.
2. Our user has purchased an item! They have purchased some "peace of mind". Using `.push()` , add the string "peace of mind" to the `purchased` array.

purchased array.

3. Our user has purchased an item! They have purchased some "Merino jodhpurs". Using `.push()`, add the string "Merino jodhpurs" to the `purchased` array.
4. Console.log just the "Merino jodhpurs" from the `purchased` array.

E. Object-within-object

Remember that you can add an object to an existing object in the same way that you can add any new property/value pair.

If we want to give our user a `friend` with a `name` and `age`, we could write:

```
user.friend = {
    name: "Grace Hopper",
    age: 85
}
```

When we console.log `user`, we would see the `friend` object added to our user object.

1. Write a `friend` object into your `user` object and give the friend a name, age, location, and purchased array (empty for now)
2. Console.log just the friend's name
3. Console.log just the friend's location
4. CHANGE the friend's age to 55
5. The `friend` has purchased "The One Ring". Use `.push()` to add "The One Ring" to the friend's `purchased` array.
6. The `friend` has purchased "A latte". Use `.push()` to add "A latte" to the friend's `purchased` array.
7. Console.log just "A latte" from the friend's `purchased` array.

F. Loops

1. Write a *for loop* that iterates over the User's `purchased` array (NOT the friend's purchased array), and prints each element to the console.
2. Write a *for loop* that iterates over the Friend's `purchased` array, and prints each element to the console.

G. Functions can operate on objects

1. Write a single function `updateUser` that takes no parameters. When the function is run, it should:
2. it should increment the user's age by 1
3. make the user's name uppercase

The function does not need a `return` statement, it will merely modify the user object.

2. Write a function `oldAndLoud` that performs the exact same tasks as `updateUser`, but instead of hard-coding it to only work on our `user` object, make it take a parameter `person`, and have it modify the object that is passed in as an argument when the function is called. Call your `oldAndLoud` function with `user` as the argument.

Requirements Complete! Hungry for More?

Cat Combinator

1. Mama cat

- Define an object called `cat1` that contains the following properties:
  - name
  - breed
  - age (a number)
- console.log the cat's age
- console.log the cat's breed

2. Papa cat

- Define an object called `cat2` that also contains the properties:
  - name
  - breed
  - age (a number)

3. Combine Cats!

The cats are multiplying!

Write a function `combineCats` that has two parameters `mama` , and `papa` . The function will take two arguments -- each a cat object.

- Pass `cat1` and `cat2` as arguments to the `combineCats` function. The function should console.log them.

Example:

```
combineCats(cat1, cat2)
```

{ name: "Joe", age: 19, breed: "Mog" }

{ name: "Jam", age: 45, breed: "Siamese" }

This is to demonstrate that functions can take objects as arguments

You could also invoke the `combineCats` function by writing the objects straight into the parentheses:

```
combineCats({ name: "Craig", age: 20, breed: "unknown" }, { name: "Linda", age: 20, breed: "unde
```

- Make it so the `combineCats` function will return a combination of the two incoming cats
  - The result should be an object wherein the
    - name is a concatenation of the parents' names
    - the age is 1
    - the breed is each of the parents' breeds with a hyphen in between

Example:

```
console.log(combineCats(cat1, cat2));
```

Result:

```
{ name: 'JoeJam', age: 1, breed: 'Mog-Siamese' }
```

This is to demonstrate that a function can return an object

---

4. Cat brain bender

If `combineCats` returns an **object**, and if `combineCats` takes **objects** as **arguments**, then it stands to reason that:

`catCombinator` can use **itself** as its own argument.

Take a second to stew on that . . .

What is the result of:

```
console.log(combineCats(combineCats(cat1, cat2), combineCats(cat1, cat2)));
```

Whoa . . .

The above console.log is **two levels** deep of combineCats.

- Write a console.log that is **three levels** deep of combineCats. combineCats should have two arguments, each which are combineCats, each which have two arguments, each which are combineCats.

Your output should look something like:

```
{ name: 'JoeJamJoeJamJoeJamJoeJam',
  age: 1,
  breed: 'Mog-Siamese-Mog-Siamese-Mog-Siamese-Mog-Siamese' }
```

Section 2