JavaScript Recap

JavaScript Recap

Let's do a brief recap about the JavaScript we learned in the pre-work.

JavaScript Recap

We will see a lot of analogies with what we learned in Ruby.

JavaScript Recap

Ruby and JavaScript are programming languages so they share many concepts.

JavaScript Recap

But there are some differences.

Ruby vs. JavaScript

Let's look at some general differences between JavaScript and Ruby.

Ruby vs. JavaScript

First, Ruby and JavaScript can run in different environments sometimes.

Ruby vs. JavaScript

JavaScript runs on all browsers. ✓

Ruby can't do that! ✗

Ruby runs on the server. ✓

JavaScript can do that too!! ✓

Ruby vs. JavaScript

This week we are going to see how we can run JavaScript like we've run Ruby.

Which is to say, in the terminal.

Ruby vs. JavaScript

`Node.js` allows us to do that.

Getting started with Node.js

You should already have Node.js installed.

nodejs.org

Getting started with Node.js

Just like in Ruby, we create JavaScript files. These will have the `.js` extension.

Getting started with Node.js

EXERCISE

Write a function that takes an array of numbers. Return every pair of positions where the addition of numbers sums to zero.

```
var exampleArray = [ 2, -5, 10,  5,  4, -10,  0 ];
```

Save the file as `file.js`.

Getting started with Node.js

EXAMPLE

```
// Positions:          0   1   2   3   4    5   6
var exampleArray = [ 2, -5, 10,  5,  4, -10,  0 ];

var results = process(exampleArray);
console.log(results);

//=> [ "1,3" , "2,5" , "6,6" ]
```

Getting started with Node.js

Let's see our failures!

(by failing, you learn)

IRON HACK

Ruby vs. JavaScript

In JavaScript functions are of greater importance.

Functions

To define a function you use the `function` keyword.

```
function eat (food) {
    console.log('Eating some ' + food);
}

eat('pizza');
```

Functions

Named functions are also values that can be passed around.

```
function eat (food) {
    console.log('Eating some ' + food);
}

console.log(eat);
```

Functions

But we'll talk more about that in a future session.

Ruby vs. JavaScript

Start your code by defining the function

EXERCISE

```
function process(data){

}
```

Ruby vs. JavaScript

Did you use curly braces to open and close sections?

In JavaScript, there is no `end` keyword. Braces are your loyal friends `{}`

Ruby vs. JavaScript

Let's create an array and variables to help us with the sum

EXERCISE

```
var sample_array = [2, -5, 10, 5, 4, -10, 0 ]
function process(data){
    var positions = [];
}
```

We'll use them later...

Ruby vs. JavaScript

Instructions need to end with a semicolon ;

Ruby vs. JavaScript

Although often if you forget the semicolon it will work anyway.

Be that as it may, try your best to not leave semicolons out

Ruby vs. JavaScript

Parentheses ( ) are *not* optional.

Ruby vs. JavaScript

You MUST declare your variables using `var`.

```
var positions = [];
```

If you don't use `var`, the variable bubbles up through the layers of scope.

Wait!!!

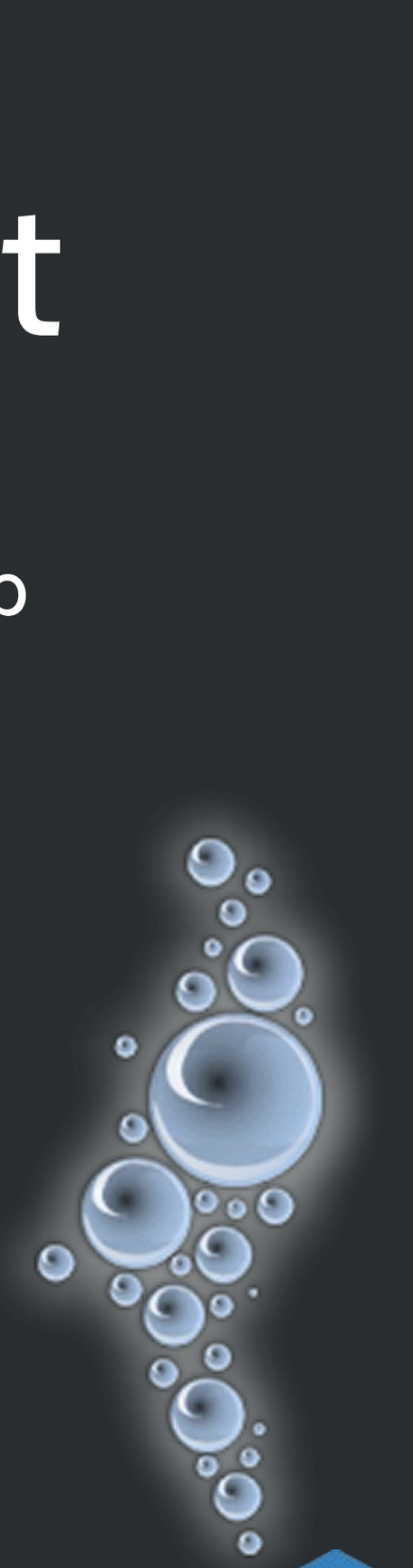
Let's take our time to explain this singularity...

IRON HACK

Ruby vs. JavaScript

If you don't use `var`, the variable bubbles up through the layers of scope.

Then, two different things could happen



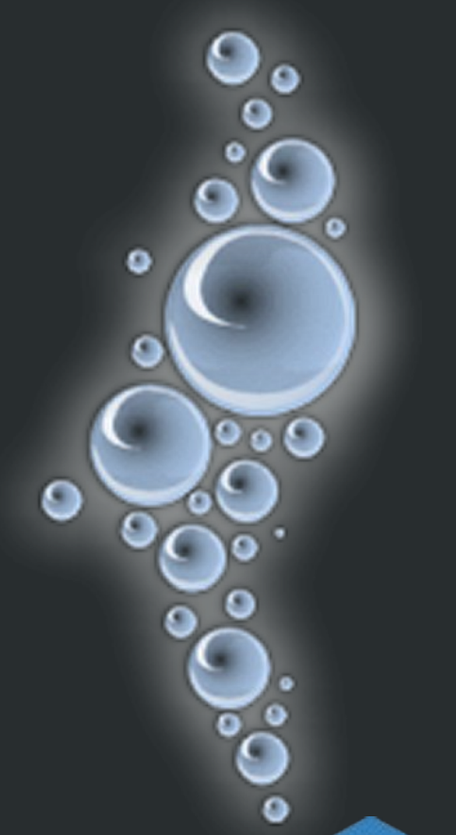
Ruby vs. JavaScript

1) It finds a variable by the given name and then attaches to it

```
//global scope
    function sum (data){
        var positions;
        function process(data){
            positions = [];
        }
    }
//more code
```

It actually exists from here

Though It's being used here



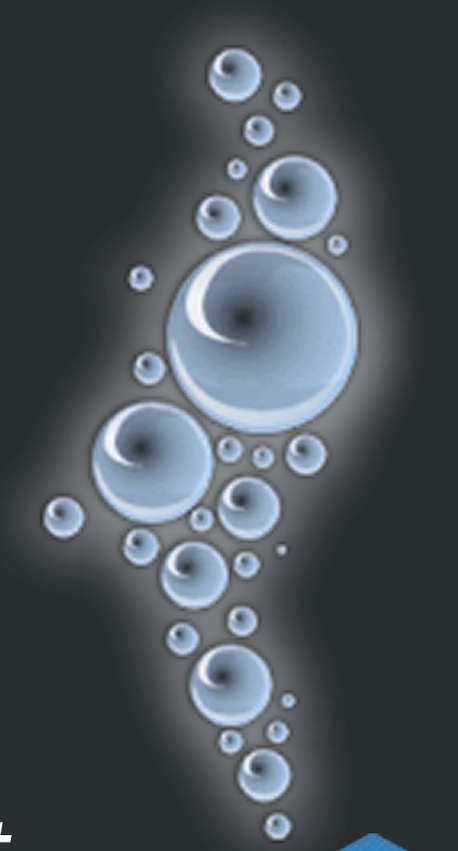
Ruby vs. JavaScript

2) It ends up in the global object (window, if you are doing it in the browser)

It actually exists from here

```
//global scope
    function sum (data){
        function process(data){
            positions = [];
        }
    }
//more code
```

Though It's being used here



This is the most common result when we forget to use `var`

Ruby vs. JavaScript

Let's go back to our exercise...

```
var sample_array = [2, -5, 10, 5, 4, -10, 0 ]
function process(data){
    var positions = [];
}
```

Ruby vs. JavaScript

You also need to call the function to execute it, remember to include your array as a parameter

```
process(sample_array);
```

Loops

You can use a `for` to loop in your program.

```
var i;

for (i = 1; i <= 42; i += 1) {
    console.log(i);
}
```

Loops

There are also array methods for looping similar to those in Ruby's `Enumerable`.

Loops

Except that JavaScript's versions use functions instead of blocks.

Loops

Here's the basic `.forEach` (instead of Ruby's `.each`).

```
var foods = [ 'pizza', 'cookies', 'bread' ];

foods.forEach(function (food) {
    console.log(food);
});
```

Loops

We've also got `.map`.

```
var foods = [ 'pizza', 'cookies', 'bread' ];

var capsFoods = foods.map(function (food) {
    return food.toUpperCase();
});

console.log(capsFoods);
// [ 'PIZZA', 'COOKIES', 'BREAD' ]
```

Loops

And `.reduce`.

```
var foods = [ 'pizza', 'cookies', 'bread' ];

var msg = foods.reduce(function (pre, food) {
    return pre + ' AND ' + food;
});

console.log(msg);
// pizza AND cookies AND bread
```

Loops

And `.filter` (instead of Ruby's `.select`).

```
var foods = [ 'pizza', 'cookies', 'bread' ];

var best = foods.filter(function (food) {
    return food !== 'bread';
});

console.log(best);
// [ 'pizza', 'cookies' ]
```

Ruby vs. JavaScript

Back to our exercise...
We need two loops to iterate on the array and make combinations between numbers

```
function process(data){
    var positions = [];
    data.forEach (function (value, i) {
        data.forEach (function (value2, j) {
        });
    });
}
 process(sample_array);
```

Conditions

Conditions work pretty much the same in JavaScript.

Conditions

The only difference is that you should use `===` for equal comparisons.

`if..else`

The `if..else` blocks in JavaScript use curly braces instead of `end`.

```
if (food === 'pizza') {
    console.log('Oh dear lord, pizza.');
} else if (food === 'cookies') {
    console.log('Mmmm cookies.');
} else {
    console.log('What the hell...');
}
```

Ruby vs. JavaScript

In our exercise...

```
var sample_array = [2, -5, 10, 5, 4, -10, 0 ]
function process(data){
    var positions = [];
    data.forEach (function (a, i) {
        data.forEach (function (b, j) {
            if (a + b === 0) {positions.push ( i +","+ j )}
        });
    });
}
process(sample_array);
```

Ruby vs. JavaScript

Unlike Ruby, JavaScript returns `undefined` *if you don't indicate what should the function return*

```
return positions;
```

Ruby vs. JavaScript

```
var sample_array = [2, -5, 10, 5, 4, -10, 0 ]
function process(data){
    var positions = []
    data.forEach (function (a, i) {
        data.forEach (function (b, j) {
            if (a + b === 0) {positions.push ( i +","+ j )}
        });
    });
    return positions;
}
process(sample_array);
```

Strings

Double quotes " " and single quotes ' ' are exactly the same in JavaScript.

Strings

```
// Escape double quotes in a double quote string
"My friend Izzy is \"special\"."

// Escape single quotes in a single quote string
'Don\'t mess with JavaScript\'s quotes.'
```

The only difference is which quotes must be *escaped*.

Ruby vs. JavaScript

Let's show our array directly in the console and create another console.log for a nice comment to our users

```
var sample_array = [2, -5, 10, 5, 4, -10, 0 ]
function process(data){
    var positions = [];
    data.forEach (function (a, i) {
        data.forEach (function (b, j) {
            if (a + b === 0) {positions.push ( i +","+ j )}
        });
    });
    console.log("You can sum these pairs of numbers and get zero:");
    console.log(positions);
}
process(sample_array);
```

Ruby vs. JavaScript

EXERCISE:

Check your code and notice that we are using two console.log statements. Why?

Change your code, concatenate the text to the array and use one console.log. What happened?

Use few minutes to print the result in the console with the appropriate format

Documentation

There is no official JavaScript documentation.

Documentation

But Mozilla

has your back

developer.mozilla.org/en-US/docs/Web/JavaScript

Exercise

Write a function that receives a string of numbers separated by colons. Have the function turn that string into an array of numbers and calculate their average.

```
var numbers = '80:70:90:100';
averageColon(numbers);
//=> 85
```