**Javascript**

**JavaScripts Data Types & Values**

* Objects( complex as you want)
* Primitives
  + String
  + Number
  + Boolean
  + Null
  + Undefined
* Symbol(ES6)
* Object

**While Loop**

* **Sintax**

**Var i= 0;**

**While(i>0){**

**Console.log(“text” + i);**

**i -=1;**

**}**

**Break and Continue**

The break statement allows to stop the execution of a loop. For example, we can create a loop that loops forever using while(true) and use the break statement to break inside the loop instead by checking that a certain condition was met.

The continue statement skips the rest of the loop and jumps back to the beginning of the loop. For example, if we would want to print only odd numbers using a for statement, we can do the following:

**Scopes**

JavaScript, each function gets its own **scope**. **Scope** is basically a collection of variables as

well as the rules for how those **variables** are accessed by name. Only code inside that

**function** can access that **function's** ***scoped*****variables**.

A **variable** name has to be unique within the same **scope** -- there can't be two different ***a***

**variables** sitting right next to each other. But the same **variable** name a could appear in

different **scopes**.

Also, a **scope** can be nested inside another **scope**, just like if a clown at a birthday party

blows up one balloon inside another balloon. If one  is nested inside another, code

inside the innermost **scope** can access variables from either **scope**.

**Conditions**

**Switch Statement**

 The switch statement is used to select between more than two different options, and to run the same code for more than one option.

**Operators**

*1- Assignement* ***=; a =2****;*

*2- Math signs* ***+-\*/;***

*3 - Increment / Decrement* ***++/-- => a++ = a + 1;***

*4 - Compound Assignement* ***+=, -=, \*=, /= => a+=2 = a + 2;***

*5 - Object property access =>* ***console.log;***

*6 - Equality* ***==, ===, !=, !==;***

*7.- Comparison* ***>, <***

*8.- Logical* ***&& (and), || (or),***

**Arrays**

Array is an object that hold value ( of any type) in numerically indexed positions

var arr = [

“hello world”,

42,

True

];

arr[0]; // “hello world”

arr[1]; // 42

* **Sintaxe**

**var todos** = [‘item1’, ‘item2’, ‘item3’] - list of todos**,**

or

**var todos** = **new Array(1,2,3);**

**console.log(todos) => [‘item1’, ‘item2’, ‘item3’]**

* **Add new todos/arrays**

**todos.push(‘new item’)**

* **Pop arrays(remove from the array)**

array.pop() => will popped the array from the end so

var myStack=[1,2,3]

myStack.pop()

console.log(myStack);=>3 21

* **Shift arrays( remove from the array)**

array.shift() => will remove from the the array from the beggining so

var myStack=[1,2,3]

myStack.shift()

console.log(myStack);=>1 23

* **Unshift arrays ( from the beggining)**

var myArray=[1,2,3]

myArray.unshift(0);

console.log(myArray) =>0,1,2,3

* **Change todos**

**todos[0] –** returns the first item;

**todos[0] = ‘item 1 updated’**

**todos =** [‘item1 updated’, ‘item2’, ‘item3’]

* **Delete todos**

**Todos.splice(0, 1) –** first parameter is the item to delete ( being 0 the first one), the second parameter is how my items to delete after**.**

* **Other things**

var myArray = []

myArray[2] = "hello"

console.log(myArray); => ,,hello =>Mesmo que a array esteja vazia, podemos declarar um valor directamente para o 3 lugar por exemplo.

**Functions**

A function is generally a named section of code that can be "called" by name, and the code

inside it will be run each time.

**function** printAmount()

Functions can optionally take arguments (aka parameters) -- values you pass in. And they

can also optionally return a value back.

function printAmount(amt) {

console.log( amt.toFixed( 2 ) );

}

* **Sintaxe**

**function** sayHiTo(person){

console.log(‘hi’, person);

}

sayHiTo(‘gordon’) =>Hi, gordon

* **Display todos**

Var todos = [‘item1’, ‘item2’, ‘item3’, ‘item4’]

**function** displayTodos(){

console.log(‘My todos:’, todos);

}

**displayTodos()** => My todos: [‘item1’, ‘item2’, ‘item3’, ‘item4’]

* **Add todos**

**function** addTodo(){

todos.push(‘new todo’)

}

addTodo()

**displayTodos()** => My todos: [‘item1’, ‘item2’, ‘item3’, ‘item4’, ‘newtodo’]

Other option is to put a value in the function:

**function** addTodo(todo){

todos.push(todo);

displayTodos();

}

So now when calling the fuction we have to give it the value

addTodo(‘hello there’)

**displayTodos()** => My todos: [‘item1’, ‘item2’, ‘item3’, ‘item4’, ‘newtodo’, ‘hello there’]

* **Changing todos**

**function** changeTodo(position, newValue){

todos[position] = newValue;

}

**changeTodo([0], ‘john’)** => [‘john’, ‘item2’, ‘item3’, ‘item4’, ‘newtodo’, ‘hello there’]

* **Delete todos**

**function** deleteTodo(position){

todos.splice(position, 1);

display Todos

}

**deleteTodo(0)** => [ ‘item2’, ‘item3’, ‘item4’, ‘newtodo’, ‘hello there’]

If you are inside of a function, you can look out and see data , but the opposite isn’t true. If you are outside, you can´t look in. circle a function. And then arrow where it goes get the data. A arrow can´t go inside a function only out. This is a scope.

Var myName= ‘Ricardo’;

Function sayName(){

var secret = ‘Moreira’;

console.log(myName);

}

sayName() => Ricardo

console.log(secret) =>error

**Objects**

Obect is a compound value where you can set properties(named locations) that each hold their own values of any type.

* **Example**

**var obj = {**

**a: “hello world”,**

**b : 42,**

**c: “42”,**

**d: true**

**};**

* **Acess the object**

**Obj.a = obj[“a”] = “hello world”**

Two objects are always different unless they are assigned equal.

So {}==={} =>is false

myname= { color :'red', name: 'ricardo'};

myname1= { color :'red', name: 'ricardo'};

but myname === myname1 => false

but if myname = myname1 => then becomes true

var obj = {

a:”hello world”,

b: 42

};

var b = “a”;

obj[b]; // “hello world”

obj[“b”]; // 42

* **Comparisons**

**{} ==={} => false**

**Var housea = {};**

**housea === housea =>true;**

housea = {

color: ‘blue’;

}

houseb = {

color: ‘blue’;

}

housea.color === houseb.color **=>true because it looks for the value, for the primitive not for the object.**

* **Sintaxe**

**Var myObject = {**

**property 1: value 1,**

**property 2 : value2,**

**property3: value3,**

**}**

myObject

{ **property** 1: value 1, **property** 2 : value2, **property3**: value3}

**myObject.property1 =>** value1

Values can be anything.

* **Functions inside a object**

Var ricardo = {

Name: ‘Ricardo’,

sayName: **function**() {

console.log(**this.name**)

}

}

**Ricardo.sayName()** => Ricardo

A **function** inside a object is called a **method**. So a method is a **property** that is equal to a **function**. So say Name is a **method** on the ricardo **object**. Inside a **object** a **function** don´t need to be named because ( in this case) sayName will be the name of the **function**. It´s called an **anonymous function.**

* **Store todos**

**Var todoList = {**

**todos: [‘item1’, ‘item2’, ‘item3’]**

**};**

* **Display todos**

Var todoList = {

todos: [‘item1’, ‘item2’, ‘item3’]

**displayTodos: function(){**

**console.log(‘My Todos’, this.todos);**

**}**

**};**

**todoList.displayTodos() =>** [‘item1’, ‘item2’, ‘item3’]

* **Add todos**

Var todoList = {

todos: [‘item1’, ‘item2’, ‘item3’]

displayTodos: function(){

console.log(‘My Todos’, this.todos);

**},**

**addTodo: function(todo){**

**this.todos.push(todo);**

**this.displayTodos();**

**}**

};

todoList.addTodo(‘Shopping’) => todos: [‘item1’, ‘item2’, ‘item3’, ‘shopping’]

* **Change Todos**

Var todoList = {

todos: [‘item1’, ‘item2’, ‘item3’]

displayTodos: function(){

console.log(‘My Todos’, this.todos);

},

addTodo: function(todo){

this.todos.push(todo);

this.displayTodos();

**},**

**changeTodo: function(position, newValue){**

**this.todos[position] = newValue;**

**this.displayTodos();**

**}**

**};**

**todoList.changeTodo(0, ‘karting’) =>** [‘karting’, ‘item2’, ‘item3’]

* **Delete Todos**

Var todoList = {

todos: [‘item1’, ‘item2’, ‘item3’]

displayTodos: function(){

console.log(‘My Todos’, this.todos);

},

addTodo: function(todo){

this.todos.push(todo);

this.displayTodos();

},

changeTodo: function(position, newValue){

this.todos[position] = newValue;

this.displayTodos();

}**,**

**deleteTodo: function(position) {**

**this.todos.splice(position, 1);**

**this.displayTodos();**

**};**

**todoList.deleteTodo(1) =>** todos: [‘item1’, ‘item3’]

* **Add Objects**

Var todoList = {

**todos: []**

displayTodos: function(){

console.log(‘My Todos’, this.todos);

},

addTodo: function(todoText){

**this.todos.push({**

**todoText: todoText,**

**completed: false**

**});**

this.displayTodos();

},

changeTodo: function(position, newValue){

this.todos[position] = newValue;

this.displayTodos();

},

deleteTodo: function(position) {

this.todos.splice(position, 1);

this.displayTodos();

};

todoList.addTodo(‘object’) => My Todos =[todo text: object; completed: false]

* **Change todoText Property**

Var todoList = {

todos: []

displayTodos: function(){

console.log(‘My Todos’, this.todos);

},

addTodo: function(todoText){

this.todos.push({

todoText: todoText,

completed: false

});

this.displayTodos();

},

**changeTodo: function(position, todoText){**

**this.todos[position].todoText =todoText;**

**this.displayTodos();**

**},**

deleteTodo: function(position) {

this.todos.splice(position, 1);

this.displayTodos();

};

**todoList.changeTodo(0, ‘eating’) =>** todos: [‘eating]

* **Toggle change completed property**

Var todoList = {

todos: []

displayTodos: function(){

console.log(‘My Todos’, this.todos);

},

addTodo: function(todoText){

this.todos.push({

todoText: todoText,

completed: false

});

this.displayTodos();

},

**changeTodo: function(position, todoText){**

**this.todos[position].todoText =todoText;**

**this.displayTodos();**

**},**

deleteTodo: function(position) {

this.todos.splice(position, 1);

this.displayTodos();

},

**toggleCompleted: function(position){**

**var todo = this.todos[position];**

**todo.completed = !todo.completed;**

**displayTodos();**

**}**

};

**todoList.toggleCompleted(0); =>** The first object change the property and will become true

**Loops**

Each time the loop block executes, that's called an **iteration**.

If the condition is initially false , ***a while loop*** will never run, but ***a do..while*** loop will run just the first time. The **for loop** has three clauses: the initialization clause ( var i=0 ), the conditional test

clause ( i <= 9 ), and the update clause ( i = i + 1 ). So if you're going to do counting with

your loop iterations, **for** is a more compact and often easier form to understand and write.

**For Loop**

* **Syntaxe**

**for ( initialization; condition; final-expression){**

**Console.log(“hey”);**

**}**

**for ( var i =0 ; I < 3; i++ ){**

**console.log(“hey”);**

**}**

***i++ = i +1***

* **Looping over arrays**

**var testArray** = [‘item1’, ‘item2’, ‘item3’];

**for** ( var i =0 ; I < 3; i++ ){

console.log(testArray[i]);

}

But because we can add value to the array we write :

**for** ( var i =0 ; I < testArray.length; i++ ){

console.log(testArray[i]);

}

If we haven´t put the [i], it will print the array 3 times, the full array

**Or**

**function** logName(name){

Console.log(name);

}

testArray.forEach(logName) => prints outs [‘item1’, ‘item2’, ‘item3’]

**Or**

testArray.forEach(function(name){

console.log.name(name);

})

**ForEach**

**Function for Each (myArray, myFunction) {**

**for(var I=0; i<myArray.length; i++){**

**myFunction(myArray[i]);**

**}**

**}**

**forEach**(testArray, function(array) {

Console.log(student);

}

**forEach**(testArray, logName)

* **Looping – Should show .todoText**

Var todoList = {

todos: []

**displayTodos: function(){**

**console.log(‘My Todos:’);**

**for ( var i = 0; I <this.todos.lenght; I ++){**

**console.log(this.todos[i].length.todoText);**

**},**

addTodo: function(todoText){

this.todos.push({

todoText: todoText,

completed: false

});

this.displayTodos();

},

changeTodo: function(position, todoText){

this.todos[position].todoText =todoText;

this.displayTodos();

},

deleteTodo: function(position) {

this.todos.splice(position, 1);

this.displayTodos();

},

toggleCompleted: function(position){

var todo = this.todos[position];

todo.completed = !todo.completed;

displayTodos();

}

};

* **Looping – Should show if todo is empty**

Var todoList = {

todos: []

displayTodos: function(){

**If(this.todos.lenght===0){**

**Console.log(“Your todo list is empty”);**

**} else {**

**console.log(‘My Todos:’);**

**for ( var i = 0; I <this.todos.lenght; I ++){**

**console.log(this.todos[i].length.todoText);**

}

};

addTodo: function(todoText){

this.todos.push({

todoText: todoText,

completed: false

});

this.displayTodos();

},

changeTodo: function(position, todoText){

this.todos[position].todoText =todoText;

this.displayTodos();

},

deleteTodo: function(position) {

this.todos.splice(position, 1);

this.displayTodos();

},

toggleCompleted: function(position){

var todo = this.todos[position];

todo.completed = !todo.completed;

displayTodos();

}

};

* **Looping – Should show if todo is completed**

Var todoList = {

todos: []

displayTodos: function(){

If(this.todos.lenght===0){

Console.log(“Your todo list is empty”);

} else {

console.log(‘My Todos:’);

for ( var i = 0; I <this.todos.lenght; I ++){

**if (this.todos[i].completed === true) {**

**console.log(‘(x)’, this.todos[i].todoText);**

**} else {**

**Console.log(‘()’, this.tofos[i].todoText);**

**}**

**}**

**}**

**},**

addTodo: function(todoText){

this.todos.push({

todoText: todoText,

completed: false

});

this.displayTodos();

},

changeTodo: function(position, todoText){

this.todos[position].todoText =todoText;

this.displayTodos();

},

deleteTodo: function(position) {

this.todos.splice(position, 1);

this.displayTodos();

},

toggleCompleted: function(position){

var todo = this.todos[position];

todo.completed = !todo.completed;

displayTodos();

}

};

* **Toggle – If everything true make everything false and otherwise make everything true**

Var todoList = {

todos: []

displayTodos: function(){

If(this.todos.lenght===0){

Console.log(“Your todo list is empty”);

} else {

console.log(‘My Todos:’);

for ( var i = 0; I <this.todos.lenght; I ++){

if (this.todos[i].completed === true) {

console.log(‘(x)’, this.todos[i].todoText);

} else {

Console.log(‘()’, this.tofos[i].todoText);

}

}

}

},

addTodo: function(todoText){

this.todos.push({

todoText: todoText,

completed: false

});

this.displayTodos();

},

changeTodo: function(position, todoText){

this.todos[position].todoText =todoText;

this.displayTodos();

},

deleteTodo: function(position) {

this.todos.splice(position, 1);

this.displayTodos();

},

toggleCompleted: function(position){

var todo = this.todos[position];

todo.completed = !todo.completed;

this.displayTodos();

}

},

**toggleAll: function(){**

**var totalTodos = this.todos.length;**

**var completeTodos =0;**

**for(var i =0; I < totalTodos; i++){**

**if(this.todos[i].completed ===true) {**

**completedTodos++;**

**}**

**}**

**if(completedTodos === totalTodos){**

**for(var i=0; i<totalTodos; i++){**

**this.todos[i].completed = false;**

**}**

**/Case 2 Otherwise make everything true**

**} else {**

**for (var i=0; i<totalTodos; i++){**

**this.todos[i].completed = true;**

**}**

**}**

this.displayTodos();

}

};

**This**

1. The value of **this** is usually determined by a functions execution context. Execution context simply means how a **function** is called.
2. It´s important to know that **this** may be different each time the **function** is called

When the keyword this is used inside of a declared object, the value of this is set to the closest parent object the method is called on.

Var person = {

first : ‘Ricardo’,

last: ‘Moreira’,

full: function(){

console.log(**this**.first + ‘ ’+ **this**.last);

}

};

**person.full()** => Ricardo Moreira

AND

Var person = {

first : ‘Ricardo’,

last: ‘Moreira’,

full: function(){

console.log(**this**);

}

};

**person.full()** => Object{…}

var person = {  
 first: 'John',  
 last: 'Smith',  
 full: function() {  
 console.log(**this**.first + ' ' + **this**.last);  
 },  
 personTwo: {  
 first: 'Allison',  
 last: 'Jones',  
 full: function() {  
 console.log(**this**.first + ' ' + **this**.last);  
 }  
 }  
};

**person.full() =>** 'John Smith'

**person.personTwo.full() =>** 'Allison Jones'

**Add Event Listener**

Catching a element

Var element= $0;

**Element** = element that was tagged

**Element**.addEventListener(‘click’, function(){

Console.log(“the element”);

});