Javascript ‘The language’

What’s in this Course ?

Javascript Basics

1. What is Javascript ?
2. Most popular Frameworks / Libraries
3. Why Javascript ?
4. Javascript Syntax
5. Where to call Js file in HTML file
6. Variables & Datatypes
7. The Boolean Object (falsy values)
8. Operators
9. if...else Statement
10. Switch Case
11. Javascript Loops
12. Loop Control
13. Javascript Functions
14. Javascript Events
15. Objects
16. Arrays
17. Date Object

Javascript DOM Manipulation

1. DOM Nodes
2. DOM Selectors
3. DOM Styling
4. DOM Get / Set Attributes
5. DOM Manipulation

Let’s get started

JS

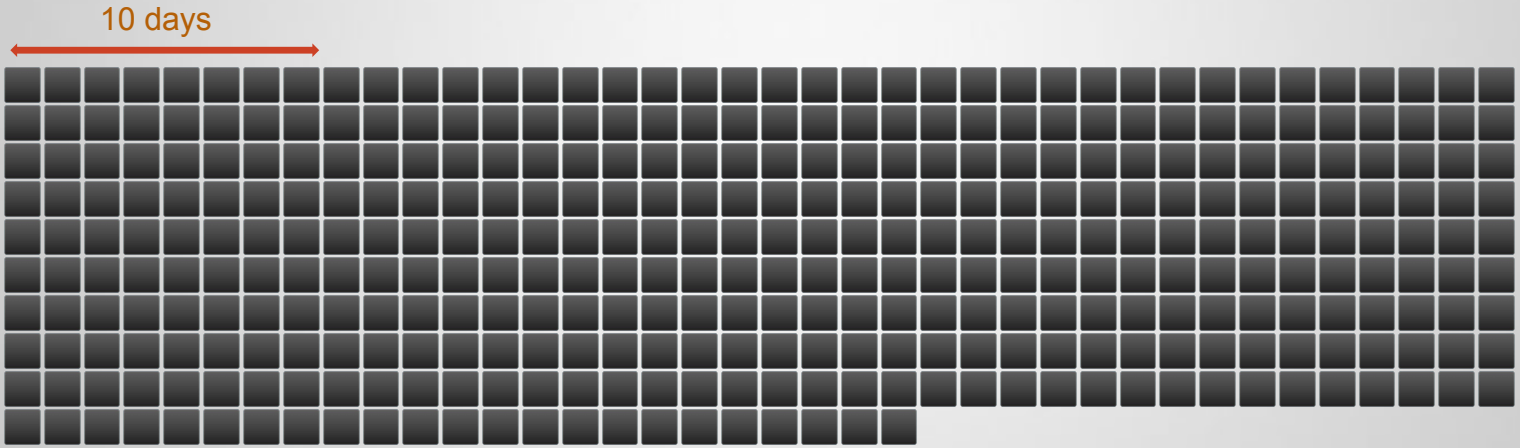
What is Javascript ?

- Javascript is a dynamic computer programming language.
- It is lightweight and most commonly used as a part of web pages.
- It is an interpreted programming language with object-oriented capabilities.
- Javascript is the most popular programming language in the world.
- Javascript is everywhere.
- Great thing about Javascript is that you will find tons of frameworks and Libraries.

What is Javascript ?

A bit of history:

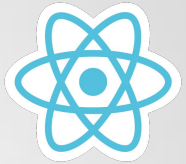
- JavaScript was created by Brendan Eich in 1995 at Netscape.
- Javascript was inspired by Java.
- Javascript developed in 10 days



What is Javascript ?

JS

Most popular Libraries / Frameworks



Why Javascript ?

- Client-side execution of the logic brings faster user experiences.
- For developers, JS is easy to learn and fast to get into active development.
- Easy and flexible syntax for newcomers.
- JavaScript is insanely popular.

> *“Any application that can be written in JavaScript, will eventually be written in JavaScript.”*
>
> *— **Jeff Atwood**, Author, Entrepreneur, Cofounder of StackOverflow*

Javascript Syntax

```
<script ...>
   JavaScript code goes here
</script>
```

Inside HTML document

```
<html>
   <body>
      <script language = "javascript" type = "text/javascript">
         <!--
            document.write("Hello World!")
         //-->
      </script>
   </body>
</html>
```

Javascript Syntax

- Case Sensitivity:
  - Javascript is a case-sensitive language.
- Comments in Javascript:
  - Javascript supports both C-style and C++-style comments

```
<script language = "javascript" type = "text/javascript">
   <!--
      // This is a comment. It is similar to comments in C++

      /*
      * This is a multi-line comment in JavaScript
      * It is very similar to comments in C Programming
      */
   //-->
</script>
```

Where to call Js file in HTML file ?

1. Script in <head>...</head> section.
2. Script in <body>...</body> section.
3. Script in <body>...</body> and <head>...</head> sections.
4. Script in an external file and then include in <head>...</head> section.

Load the code before closing the body tag

```
<html>
   <body>
      <!-- other HTML code ... -->
      <script
src='path/to/external/file.js'></script>
   </body>
</html>
```

Load the code inside the head tag using defer attribute

```
<html>
   <head>
     <script src='path/to/external/file.js'
defer></script>
   </head>
   <body>
     <!-- other HTML code ... -->
   </body>
</html>
```

Javascript Variables & Datatypes

- Javascript allows you to work with three primitive data types
  - **Numbers**: 1, 145, 300.50, etc.
  - **Strings**: "whatchu talkin bout willis" etc.
  - **Boolean**: true / false
- Javascript also comes with two trivial data types:
  - **Undefined**: Defined variable with no value.
  - **Null**: Assigned to a defined variable as a representation of no value.

Javascript Variables & Datatypes

Javascript Variable Scope

- **Scope**: A region of your program in which a variable is defined.

Javascript Variables & Datatypes

Javascript Variable names

Naming variables in Javascript follows a couple of rules that you should keep in mind:

- You cannot use any of the Javascript reserved keywords as a variable name, such as ***switch, export, public***, etc.
- A Javascript variable should not start with a number (0-9).
- A Javascript variable should not start with a symbol such as @, -, #, etc.
- A Javascript variable should start with a letter or an underscore character (_).

Javascript Variables & Datatypes

To avoid confusion

Here are some technical words you should keep in mind:

"	Double Quotation Mark	`	Back-tic
'	Single Quotation Mark	|	Pipe
-	Hyphen	/	Forward Slash
_	Underscore	\	Backslash
[ ]	Brackets	^	Caret
( )	Parentheses	~	Tilde
{ }	Curly Braces		

The Boolean Object (falsy values)

- A Boolean object has two values, either **true** or **false**.
- If the value is omitted, or the value is one of these types, the Object initial value will be false:
  - 0
  - -0
  - Null
  - False
  - NaN
  - Undefined
  - Empty string (“”)

Javascript Operators

Javascript has 5 types of operators:

1. **Arithmetic Operators:**

   Let's consider the following, **a** has a value 10 and **b** has a value 30, so:

- **+** Addition, a + b gives 40
- **-** Substruction, a - b gives -20
- ***** Multiplication, a * b gives 300
- **/** Division: Divide operands, b / a gives 3
- **%** Modulus: return the remainder of an integer division, b % a gives 0
- **++** Increment, a++ gives 11
- **--** Decrement, a-- gives 9

Note: Addition operator (+) works with strings as well, e.g "a" + 10 gives "a10".

Javascript Operators

Javascript has 5 types of operators:

2. **Comparison Operators:**

- **==** Equal: Check if two operands have the same value or not.
- **===** Equal: Do the same as == except it do check whether they have the same type or not.

```
a = 10;
b = "10";
a == b // returns true
a === b // returns false
```

- **!=** Not Equal: Check if two operands have different values.
- **!==** Not Equal: Check if two operands have different values and types.
- **>** Greater than: Check if the left operand is greater than the right operand.
- **<** Less than: Check if the left operand is less than the right operand.
- **>=** Greater than or Equal to: Check if the left operand is greater than or Equals to the right operand.
- **<=** Less than or Equal to: Check if the left operand is less than or Equals to the right operand.

Javascript Operators

Javascript has 5 types of operators:

3. **Logical Operators:**

   Let's consider the following, a = 10 and b = 20, so:

- **&&** Logical AND, if both operands are non-zero, the condition returns true. E.g. (a && b) is true.
- **||** Logical OR, if at least one of the operands is non-zero, the condition returns true. E.g. (a || b) is true.
- **!** Logical NOT, Reverse the logical state of its operands. If a condition is true, the Logical NOT makes it false.

  E.g. !(a && b) is false.

Javascript Operators

Javascript has 5 types of operators:

4. **Bitwise Operators:**

   Let's consider the following, a = 1 and b = 2, so:

- **&** Bitwise AND, applies a boolean AND operation on each bit of its integer arguments.
- **|** Bitwise OR, applies a boolean OR operation on each bit of its integer arguments.
- **^** Bitwise XOR, applies a boolean OR operation on each bit of its integer arguments. This is an Exclusive OR: Either operand one is true or operand two is true but not both are true.
- **~** Bitwise NOT, this is a unary operator and operates by reversing all bits in the operand.

Javascript Operators

Javascript has 5 types of operators:

5. **Assignment Operators:**

- **=** Simple assignment
- += Add and Assignment
- -= Subtract and Assignment
- *= Multiply and Assignment
- /= Divide and Assignment
- %= Modules and Assignment

Javascript Operators

Javascript has 5 types of operators:

Miscellaneous Operator

1. Conditional Operator (Ternary Operator)

   If condition is true **?** *value in case of true* **:** *value in case of false*

```
a = 10;
b = 10;
c = a == b ? "You are awesome" : "It's false, but you are awesome";
// c output: You are awesome
```

2. Typeof Operator

```
num = 10;
email = "myemail@domaine.com";
isValid = true;
Typeof num; // "number"
Typeof email; // "string"
Typeof isValid; // "boolean"
```

If...else Statement

Javascript if...else statement is similar to any other language:

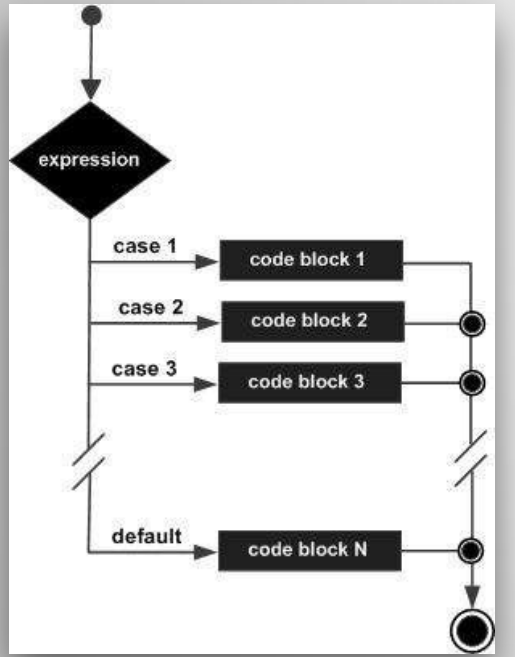
```
var lang = "Javascript";
if (lang == "Javascript")
console.log("is matching!");
// output: is matching!
```

```
var lang = "Javascript";
if (lang == "Java")
console.log("is matching!");
else console.log("not matching!");
// output: not matching!
```

```
var num = 20;
if (num <= 10) console.log("is less than or equal to 10!");
else if (num <= 15) console.log("is less than or equal to 15!");
else console.log("is greater than 15!");
// output: is greater than 15!
```

Switch Case

Javascript switch case statement is also similar to any other language, and it performs just like if... else statement, but more efficient.

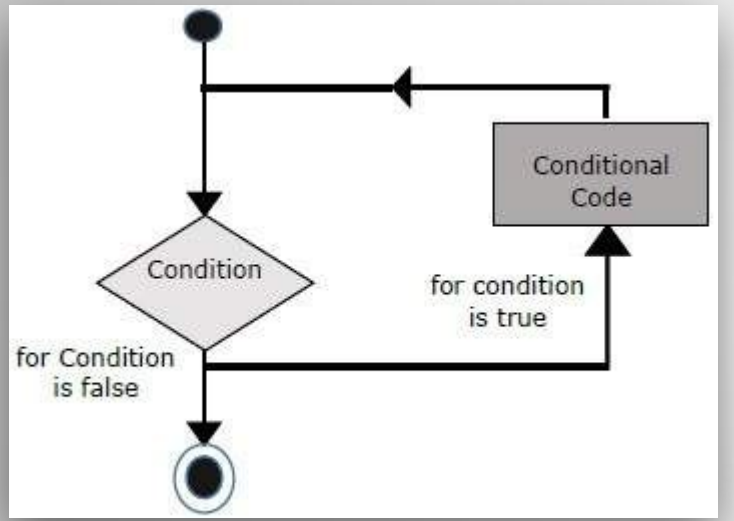



```
var animal = 'Sailfish';

switch(animal) {
  case 'Cheetah': document.write('The fastest animal on land.</br>');
  break;
  case 'Peregrine falcon': document.write('The fastest animal on
earth.</br>');
  break;
  case 'Sailfish': document.write('The fastest animal in the ocean.</br>');
  break;
  default: document.write('The animal you entered is not ranked !');
}
```

JS

Javascript Loops

The For Loop



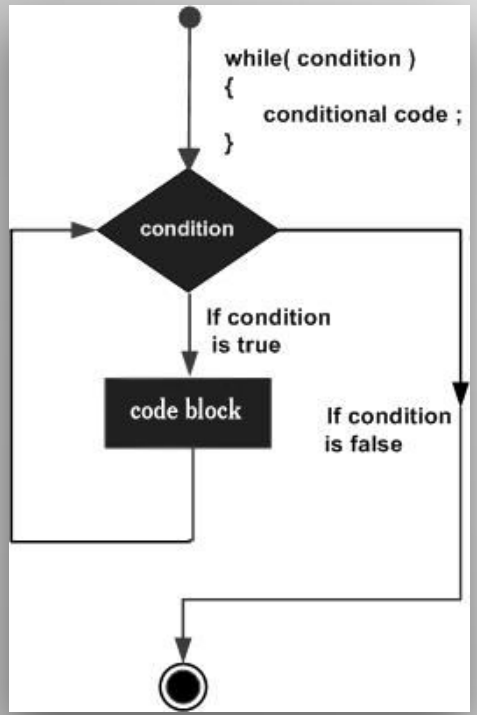

```
for (initialization; test condition; iteration statement)
{
   Statement(s) to be executed if test condition is true
}
```

```
var count;
document.write("Starting Loop" + "<br />");
for(count = 0; count < 10; count++) {
  document.write("Current Count : " + count + "<br />");
}
document.write("Loop stopped!");
```

JS

Javascript Loops

The while Loop



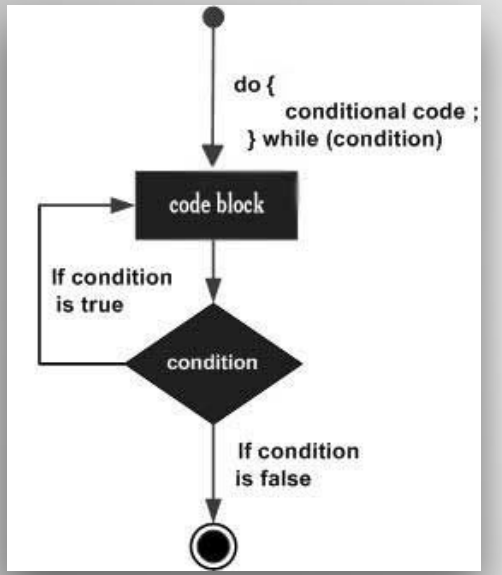

```
while (expression) {
   Statement(s) to be executed if expression is true
}
```

```
var count = 0;
document.write("-- Starting while Loop -- <br />");

while(count < 10) {
  document.write('Current count: ' + count + '<br />');
  count++;
}
document.write('-- While loop stopped --');
```

JS

Javascript Loops

The do...while Loop




```
do {
   Statement(s) to be executed;
} while (expression);
```

```
var count = 0;
document.write("-- Starting while Loop -- <br />");
do {
  document.write("Current Count : " + count + "<br />");
  count++;
}
while (count < 5);
document.write('-- While loop stopped --');
```

JS

Javascript Loops

For** Vs. **While** Vs. **Do ... While and when to use ?

For

How to perform

Allows you to initiate a counter variable, a check condition, and a way to increment your counter all in one line.

When to use it

Known number of iterations.

While

How to perform

Always evaluate the condition first. Then it executes the conditional code block.

When to use it

When the number of iterations is not known in advance.

Do ... While

How to perform

Always execute the code in the do{} block first and then evaluate the condition.

When to use it

Executes the instructions **once at start**, and afterwards it behaves just like the simple while.

Javascript Loops

For...in loop

Used to loop through an object's properties.

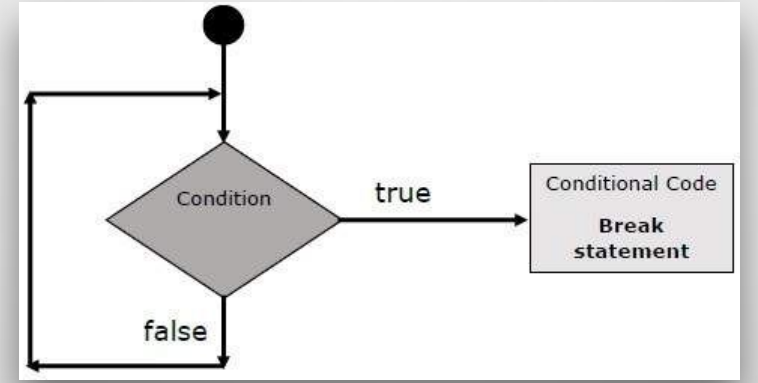
```
for (variableName in object) {
   statement or block to execute
}
```

In each iteration, one property from object is assigned to variableName and this loop continues till all the properties of the object are exhausted.

```
var aProperty;
document.write("Navigator Object Properties<br />");
for (aProperty in navigator) {
  document.write(aProperty + "<br />");
}
document.write ("Exiting from the loop!");
```

Javascript Loop Control

- Javascript provides full control to handle loops and switch statements.
- In some cases, we need to come out of the loop without reaching the end.
- Also, there are some cases when we want to skip a block of code and start the next iteration of the loop.
- Therefore... to handle such situations, Javascript provides **break** and **continue** statement.



Javascript Loop Control

break

```
var x = 1;
document.write("Entering the loop<br /> ");
while (x < 20) {
  if (x == 5) {
    break; // breaks out of loop completely
  }
  x = x + 1;
  document.write( x + "<br />");
}
document.write("Exiting the loop!<br /> ");
```

The break statement tells the interpreter to immediately to stop the loop.

continue

```
var x = 1;
document.write("Entering the loop<br />");
while (x < 10) {
  x = x + 1;
  if (x == 5) {
    continue;   // skip rest of the loop body
  }
  document.write( x + "<br />");
}
document.write("Exiting the loop!<br />");
```

The continue statement tells the interpreter to immediately start the next iteration of the loop and skip the remaining code block.

Javascript Functions

- A group of reusable code which can be called anywhere in your program.
- Eliminates the need of writing the same code again and again.
- Gives the ability to write modular codes.
- Allows a programmer to divide a big program into a number of small and manageable functions.

```
function functionname(parameter-list) {
  Statements...
}
```

- Calling a function to be executed is called - *invoke* -:

```
function sayHello() {
  alert("Hello, welcome to the js course");
}
```

- You can pass parameters to the function

```
function sayHello(name, occupation) {
  document.write (name + " works as " + occupation);
}
// invoking the function
sayHello("Najm", "Javascript developer");
// Output: Najm works as Javascript developer
```

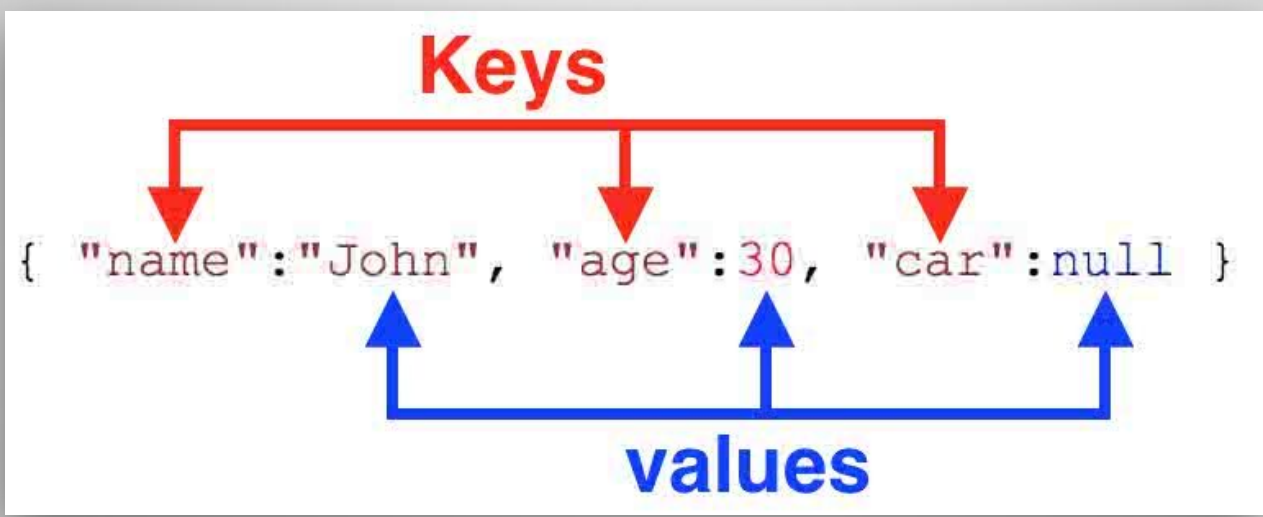
- A Javascript function can have an optional return statement, just if you want to return a value from the function.

```
function sum(a, b) {
  return a + b;
}
var res = sum(10, 20);
// Output: res = 30
```

Javascript Objects

- Javascript is an Object Oriented Programming (OOP) language.
- A programming language can be called object-oriented if it provides four basic capabilities to developers, such as:
    - **Encapsulation**: the capability to store related information, whether data or methods, together in an object.
    - **Aggregation**: the capability to store one object inside another object.
    - **Inheritance**: the capability of a class to rely upon another class (or number of classes) for some of its properties and methods.
    - **Polymorphism**: the capability to write one function or method that works in a variety of different ways.
- Object properties can be any of the three primitive data types.
- Object properties can be any of the abstract data types such as another object, array.
- An object value can be a function as well, and the property name will be the name of the function.

Javascript Objects



Javascript Objects

Create new Object and assign some values:

```
var book = new Object();  // Create the object
book.title = "The Da Vinci Code";  // Assign properties to the object
book.author = "Dan Brown";
book.price = 20;
book.priceCalc = function(nbr) {
  return nbr * book.price;
}
```

Access to Object values:

```
document.write("Book name is : " + book.title + "<br>");
document.write("Book author is : " + book.author + "<br>");
document.write("2 copies will be : " + book.priceCalc(2) + "$ <br>");
```

Javascript Objects

Using the **with** keyword as a shorthand to tell interpreter: use properties without the Object name and the dot.

```
var book = new Object();  // Create the object
book.title = "The Da Vinci Code";  // Assign properties to the object
book.author = "Dan Brown";
book.price = 20;
book.priceCalc = function(nbr) {
  return nbr * book.price;
}

with(book) {
  title = "Digital fortress";
  price = 40;
}

document.write("Book name is : " + book.title + "<br>");
document.write("Book author is : " + book.author + "<br>");
document.write("2 copies will be : " + book.priceCalc(2) + "$ <br>");
```

Javascript Objects

Loop through an Object using **for... in** loop. (Check out the loop description here)

```
var book = new Object();  // Create the object
book.title = "The Da Vinci Code";  // Assign properties to the object
book.author = "Dan Brown";
book.price = 20;
book.priceCalc = function(nbr) {
  return nbr * book.price;
}

for(var key in book) {
  document.write(key + ': ' + book[key] + '<br />');
}
```

Javascript Arrays

- Array Object let you store multiple values in a single variable. Here is how to create an Array:

```
var martialArts = new Array("Karate", "Kung Fu", "Taekwondo", "Muay Thai");
```

```
martialArts[0] is the first element: Karate
martialArts[1] is the second element: Kung Fu
martialArts[2] is the third element: Taekwondo
martialArts[3] is the fourth element: Muay Thai
```

- Getting the Array length:

```
martialArts.length // Output: 4
```

- Javascript Array has several methods. See the complete methods list here.

Javascript Arrays

- Let's consider the next Array:

```
var martialArts = new Array("Karate", "Kung Fu", "Taekwondo", "Muay Thai");
```

every() method: *Returns true if every element matches a given testing function*

```
var res = martialArts.every(function(el) {
  if (el.length > 5) return el; // test if length of word is greater than 5
});
```

filter() method: *Creates a new array with all elements that matches a given function filter.*

```
var res = martialArts.filter(function(el) {
  if (el.length > 6) return el;
});
```

forEach() method: *Call a given function for each.*

```
martialArts.forEach(function(el) {
  console.log(res);
});
```

Javascript Arrays

join() method: *Joins all elements of an array into a string.*

```
var res = martialArts.join("/");
```

indexOf() method: *Return the first occurrence index. Or -1 if none is found.*

```
var res = martialArts.indexOf("Kung Fu");
```

map() method: *Creates a new array depends on a given function.*

```
var res = martialArts.map(function(el){
  return "I love " + el;
});
```

pop() method: *Removes the last element from an array and returns that element.*

```
var res = martialArts.pop();
```

Javascript Arrays

shift() method: *Removes the first element from an array and returns that element.*

```
var res = martialArts.shift();
```

push() method: *Adds one or more elements to the end of an array and returns the new length of the array.*

```
martialArts.push("Capoeira");
```

unshift() method: *Adds one or more elements the beginning of an array and return the new length of the array.*

```
martialArts.unshift("Capoeira");
```

splice() method: *Adds and/or removes elements from an array.*

```
// Remove
martialArts.splice(1, 2);

// Remove then add
martialArts.splice(1, 2, "Wing Chun");
```

Javascript Arrays

slice() method: *Extracts a section of an array and returns a new array.*

```
var res = martialArts.slice(1, 2);
```

toString() method: *Returns a string representing the array and its elements.*

```
var res = martialArts.toString();
```

sort() method: *Sorts the elements of an array.*

```
martialArts.sort();
```

some() method: *Returns true if at least one element in this array satisfies the provided testing function.*

```
var res = martialArts.some(function(el){
  if (el.includes('z')) return el;
});
```

concat() method: *Returns a new array comprised of this array joined with other array(s) and/or value(s).*

```
var res = martialArts.concat(OtherMartialArts);
```

Javascript Arrays

reduce() method: *Apply a function simultaneously against two values of the array (from left-to-right) as to reduce it to a single value.*

```
var res = num.reduce(function(acc, el) {
  return acc + el;
});
```

Javascript Date

Constructor:

```
Date();
// Output: Sun Jun 14 2020 13:12:49 GMT+0100 (Central European Standard Time)
```

Static Methods:

```
Date.now();
// Output: 1592136923497

Date.parse("March 21, 2020");
// Output: 1584745200000

Date.UTC(2020, 02, 30);
// Output: 1585526400000
```

See the complete list of Date methods here.

EcmaScript 6

JS

Javascript - DOM Manipulation

DOM Nodes

What is DOM ?

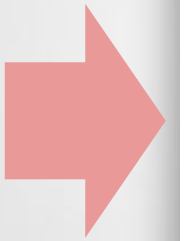
- Stands for **D**ocument **O**bject **M**odel.
- DOM is a representation of the HTML / XML document.
- HTML attributes such as `id`, `class`, `title`, `style`, etc. are also considered as nodes in DOM hierarchy but they don't participate in parent/child relationships. They are accessed as properties of the element node that contains them.

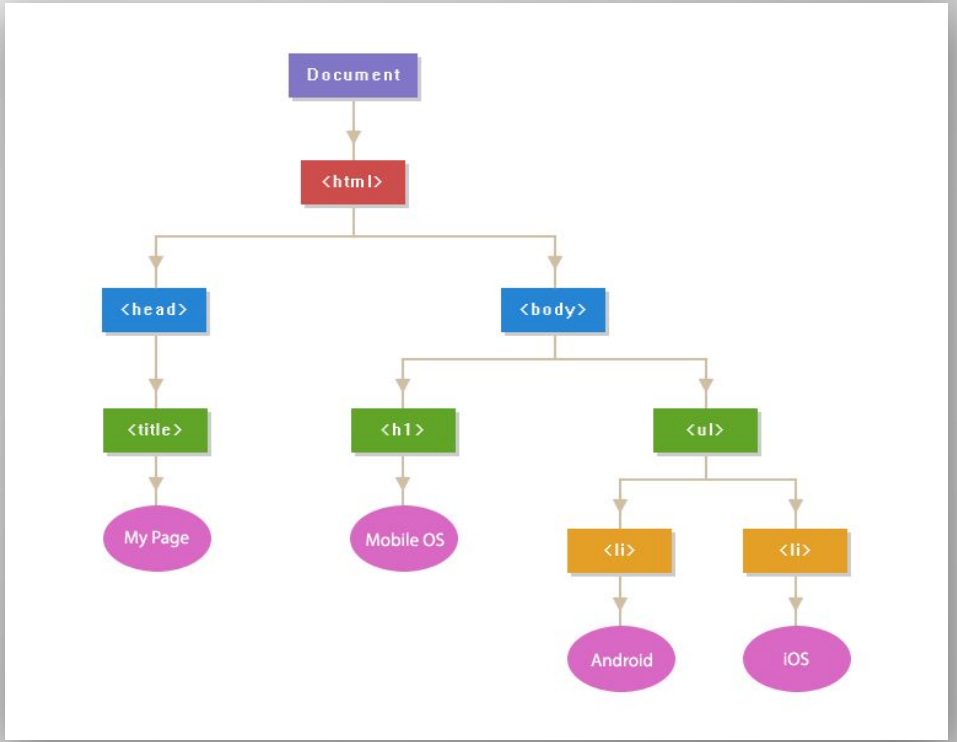
JS

Javascript - DOM Manipulation

DOM Nodes

```
<!DOCTYPE html>
<html>
<head>
    <title>My Page</title>
</head>
<body>
    <h1>Mobile OS</h1>
    <ul>
        <li>Android</li>
        <li>iOS</li>
    </ul>
</body>
</html>
```





JS

Javascript - DOM Manipulation

DOM Selectors

When you want to access HTML elements with Javascript, you have to find the elements first. Therefore, there are a couple of ways to do this:

Let's consider this HTML element:

```
<p class="intro" id="intro">Introduction</p>
```

- Finding HTML elements by id
  - ```
    var myElement = document.getElementById("intro");
    ```
- Finding HTML elements by tag name
  - ```
    var myElement = document.getElementsByTagName("p");
    ```
- Finding HTML Elements by Class Name
  - ```
    var myElement = document.getElementsByClassName("intro");
    ```
- Finding HTML Elements by CSS Selectors
  - ```
    var myElement = document.querySelectorAll("p.intro");
    ```

Javascript - DOM Manipulation

DOM Selectors - Events

- JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page.
- When the page loads, it is called an event.
- When the user clicks a button, that click too is an event.
- Pressing a key, closing a window, resizing a window are events as well.
- Events are a part of the ***DOM*** Level 3 and every HTML element contains a set of events which can trigger Javascript Code.

Javascript - DOM Manipulation

DOM Selectors - onclick Event

Triggered when a user clicks the left button of the mouse.

The click event is the most frequently used between all of the other events.

```
<button onclick="cheers()">Click me</button>
```

```
function cheers() {
  alert('Cheeeeeers!!!');
};

// Output: Cheeeeeers!!!
```

Javascript - DOM Manipulation

DOM Selectors - onsubmit Event

Triggered when a user tries to submit a form.

```
<form method="POST" action="some-action.php" onsubmit="validate()">
  .......
  <input type="submit" value="Submit" />
</form>
```

```
function validation() {
  alert('Submitted!!!');
}
```

Javascript - DOM Manipulation

DOM Selectors - *onmouseover Event / onmouseout Event*

Triggered when a user tries to submit a form.

```
<div onmouseover="over()" onmouseout="out()">
  <h2 id="heading">Initial text</h2>
</div>
```

```
function over() {
  document.getElementById('heading').innerHTML = "Mouse Over";
};

function out() {
  document.getElementById('heading').innerHTML = "Mouse Out";
};
```

You can see the full events list in this link.

Javascript - DOM Manipulation

DOM Selectors - *addEventListener*

At some point, when you want to keep your code cleaner, you need to move all Javascript tricks into your Javascript code.

```
<button onclick="cheers()">Click me</button>
```

```
function cheers() {
  alert('Cheeeeeers!!!');
};

// Output: Cheeeeeers!!!
```

```
<button id="cheers-btn">Click me</button>
```

```
function cheers() {
  alert('Cheeeeeers!!!');
};

var btn = document.getElementById('cheers-btn');

btn.addEventListener('click', cheers);
```

Javascript - DOM Manipulation

DOM Styling

- Javascript add an online css style.
- Inline styles are applied directly to the specific HTML element using the style attribute.
- In JavaScript the style property is used to get or set the inline style of an element.

```
▼<div class="wrapper" style="
  background: red;
">
```

Javascript - DOM Manipulation

DOM Styling - Set properties

- Many CSS properties, such as `font-size`, `background-image`, `text-decoration`, etc.
- In JavaScript hyphen is a reserved operator and it is interpreted as a minus sign.
- The CSS property names that contain one or more hyphens are converted to a camel case word.

```
Font-size         → fontSize
Border-left-style → borderLeftStyle
```

Javascript - DOM Manipulation

DOM Styling - Get properties

```
<body>
    <p id="intro" style="color:red; font-size:20px;">This is a paragraph.</p>
    <p>This is another paragraph.</p>

    <script>
      // Selecting element
      var elem = document.getElementById("intro");

      // Getting style information from element
      alert(elem.style.color);  // Outputs: red
      alert(elem.style.fontSize);  // Outputs: 20px
      alert(elem.style.fontStyle);  // Outputs nothing
    </script>
</body>
```

Javascript - DOM Manipulation

DOM Styling - Get properties

- To get the values of all CSS properties that are actually used to render an element you can use the `window.getComputedStyle()`

```
<body>
    <p id="intro" style="color:red; font-size:20px;">This is a paragraph.</p>
    <p>This is another paragraph.</p>
    <script>
      // Selecting element
      var elem = document.getElementById("intro");
      // Getting computed style information
      var styles = window.getComputedStyle(elem);
      alert(styles.getPropertyValue("color"));  // Outputs: rgb(255, 0, 0)
      alert(styles.getPropertyValue("font-size"));  // Outputs: 20px
      alert(styles.getPropertyValue("font-weight"));  // Outputs: 700
      alert(styles.getPropertyValue("font-style"));  // Outputs: italic
    </script>
</body>
```

Javascript - DOM Manipulation

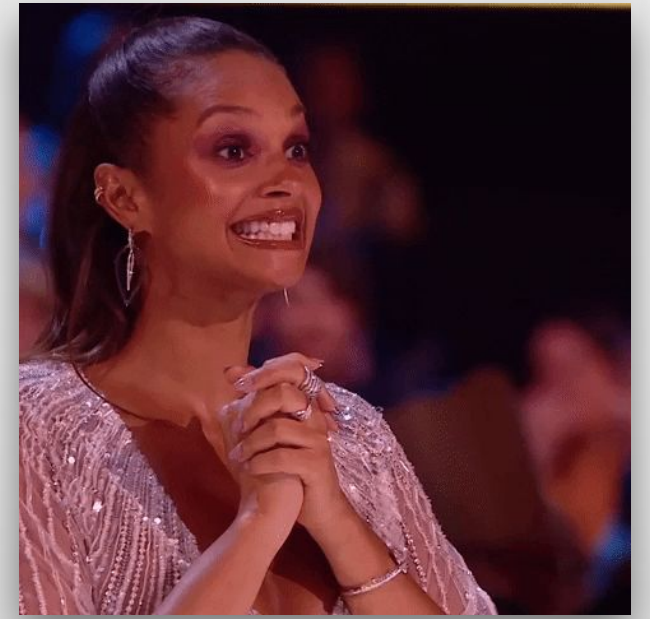
DOM Styling - Adding CSS Classes to Elements

- You can also get or set CSS classes to the HTML elements using the `className` property.

```
<div id="info" class="disabled">Something very important!</div>

<script>
    // Selecting element
    var elem = document.getElementById("info");

    elem.className = "note";  // Add or replace all classes with note class
    elem.className += " highlight";  // Add a new class highlight
</script>
```

Javascript - DOM Manipulation

Or even a better way



Javascript - DOM Manipulation

DOM Styling - Adding CSS Classes to Elements

- You can also `add` / `remove` / `toggle` methods.

```
<div id="info" class="disabled">Something very important!</div>
    <script>
      // Selecting element
      var elem = document.getElementById("info");

      elem.classList.add("hide");  // Add a new class
      elem.classList.add("note", "highlight");  // Add multiple classes
      elem.classList.remove("hide"); // Remove a class
      elem.classList.remove("disabled", "note"); // Remove multiple classes
      elem.classList.toggle("visible"); // If class exists remove it, if not add it

      // Determine if class exist
      if(elem.classList.contains("highlight")) {
        alert("The specified class exists on the element.");
      }
    </script>
```

Javascript - DOM Manipulation

DOM Get / Set Attributes - Getting attributes

- `Attributes` are special words used inside the start tag of an HTML element to control the tag's behavior or provides additional information about the tag.

```
<a href="https://www.google.com/" target="_blank" id="myLink">Google</a>

<script>
    // Selecting the element by ID attribute
    var link = document.getElementById("myLink");

    // Getting the attributes values
    var href = link.getAttribute("href");
    alert(href); // Outputs: https://www.google.com/

    var target = link.getAttribute("target");
    alert(target); // Outputs: _blank
</script>
```

Javascript - DOM Manipulation

DOM Get / Set Attributes - Setting attributes

- If the attribute already exists on the element, the value is updated; otherwise a new attribute is added with the specified name and value.

```
<button type="button" id="myBtn">Click Me</button>

<script>
    // Selecting the element
    var btn = document.getElementById("myBtn");

    // Setting new attributes
    btn.setAttribute("class", "click-btn");
    btn.setAttribute("disabled", "");
</script>
```

Javascript - DOM Manipulation

DOM Get / Set Attributes - Updating attributes

```
<a href="#" target="_blank" id="myLink">Go to Google</a>

<script>
    // Selecting the element
    var link = document.getElementById("myLink");

    // Changing the href attribute value
    link.setAttribute("href", "https://www.google.tn");
</script>
```

Javascript - DOM Manipulation

DOM Get / Set Attributes - Remove attributes

```
<a href="https://www.google.com/" id="myLink">Google</a>

<script>
    // Selecting the element
    var link = document.getElementById("myLink");

    // Removing the href attribute
    link.removeAttribute("href");
</script>
```

Javascript - DOM Manipulation

DOM Manipulation - Adding new elements to DOM

- The `appendChild()` method adds the new element at the end of any other children of a specified parent node.
- Use `insertBefore()` to add the new element at the beginning.

```
<div id="main">
    <h1 id="title">Hello World!</h1>
    <p id="hint">This is a simple paragraph.</p>
</div>
<script>
// Creating a new div element
var newDiv = document.createElement("div");
// Creating a text node
var newContent = document.createTextNode("Hi, how are you doing?");
// Adding the text node to the newly created div
newDiv.appendChild(newContent);
// Adding the newly created element and its content into the DOM
var currentDiv = document.getElementById("main");
document.body.appendChild(newDiv, currentDiv);
</script>
```

Javascript - DOM Manipulation

DOM Manipulation - Getting or Setting HTML Contents to DOM

- You can also get or set the contents of the HTML elements easily with the `innerHTML` property.

```
<div id="main">
    <h1 id="title">Hello World!</h1>
    <p id="hint">This is a simple paragraph.</p>
</div>

<script>
// Getting inner HTML conent
var contents = document.getElementById("main").innerHTML;
alert(contents); // Outputs inner html contents

// Setting inner HTML contents
var mainDiv = document.getElementById("main");
mainDiv.innerHTML = "<p>This is <em>newly inserted</em> paragraph.</p>";
</script>
```

Note: the `innerHTML` property replaces all existing content of an element.

Javascript - DOM Manipulation

Note: the `innerHTML` property replaces all existing content of an element.

Javascript - DOM Manipulation

DOM Manipulation - Getting or Setting HTML Contents to DOM

```
<!-- beforebegin -->
<div id="main">
    <!-- afterbegin -->
    <h1 id="title">Hello World!</h1>
    <!-- beforeend -->
</div>
<!-- afterend -->
<script>
// Selecting target element
var mainDiv = document.getElementById("main");
// Inserting HTML just before the element itself, as a previous sibling
mainDiv.insertAdjacentHTML('beforebegin', '<p>This is paragraph one.</p>');
// Inserting HTML just inside the element, before its first child
mainDiv.insertAdjacentHTML('afterbegin', '<p>This is paragraph two.</p>');
// Inserting HTML just inside the element, after its last child
mainDiv.insertAdjacentHTML('beforeend', '<p>This is paragraph three.</p>');
// Inserting HTML just after the element itself, as a next sibling
mainDiv.insertAdjacentHTML('afterend', '<p>This is paragraph four.</p>');
</script>
```

Javascript - DOM Manipulation

DOM Manipulation - Removing Existing Elements from DOM

```
<div id="main">
    <h1 id="title">Hello World!</h1>
    <p id="hint">This is a simple paragraph.</p>
</div>

<script>
var parentElem = document.getElementById("main");
var childElem = document.getElementById("hint");
parentElem.removeChild(childElem);
</script>
```

Javascript - DOM Manipulation

DOM Manipulation - Removing Existing Elements from DOM

- It is also possible to remove the child element without exactly knowing the parent element.
- Use the `parentNode` property to find its parent element.

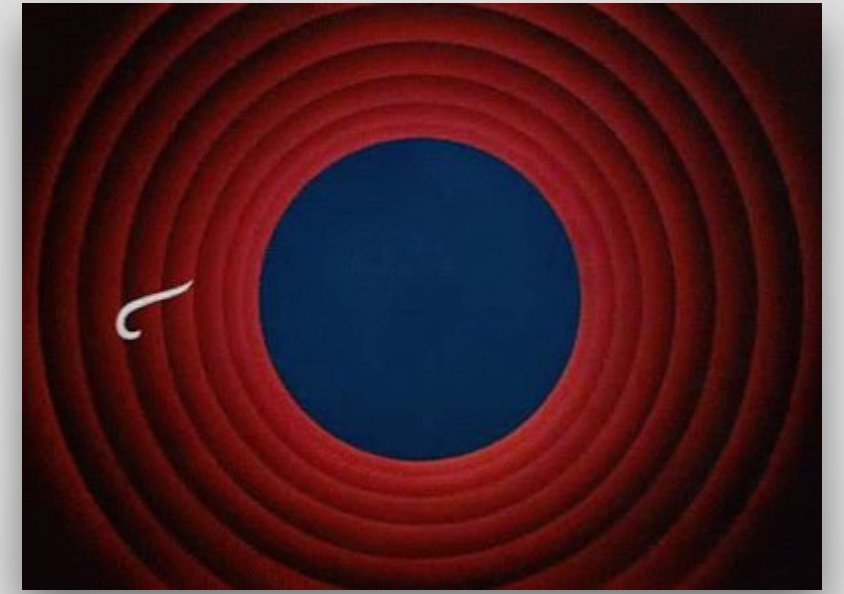
```
<div id="main">
    <h1 id="title">Hello World!</h1>
    <p id="hint">This is a simple paragraph.</p>
</div>

<script>
var childElem = document.getElementById("hint");
childElem.parentNode.removeChild(childElem);
</script>
```

Javascript - DOM Manipulation

DOM Manipulation - Replacing Existing Elements from DOM

- the `replaceChild()` method. This method accepts two parameters: the node to **insert** and the node to be **replaced**.
- It has the syntax like `parentNode.replaceChild(newChild, oldChild);`

```
<div id="main">
    <h1 id="title">Hello World!</h1>
    <p id="hint">This is a simple paragraph.</p>
</div>
<script>
var parentElem = document.getElementById("main");
var oldPara = document.getElementById("hint");
// Creating new elememt
var newPara = document.createElement("p");
var newContent = document.createTextNode("This is a new paragraph.");
newPara.appendChild(newContent);
// Replacing old paragraph with newly created paragraph
parentElem.replaceChild(newPara, oldPara);
</script>
```



JS