

JS

Introduction to

JAVASCRIPT

Full Course

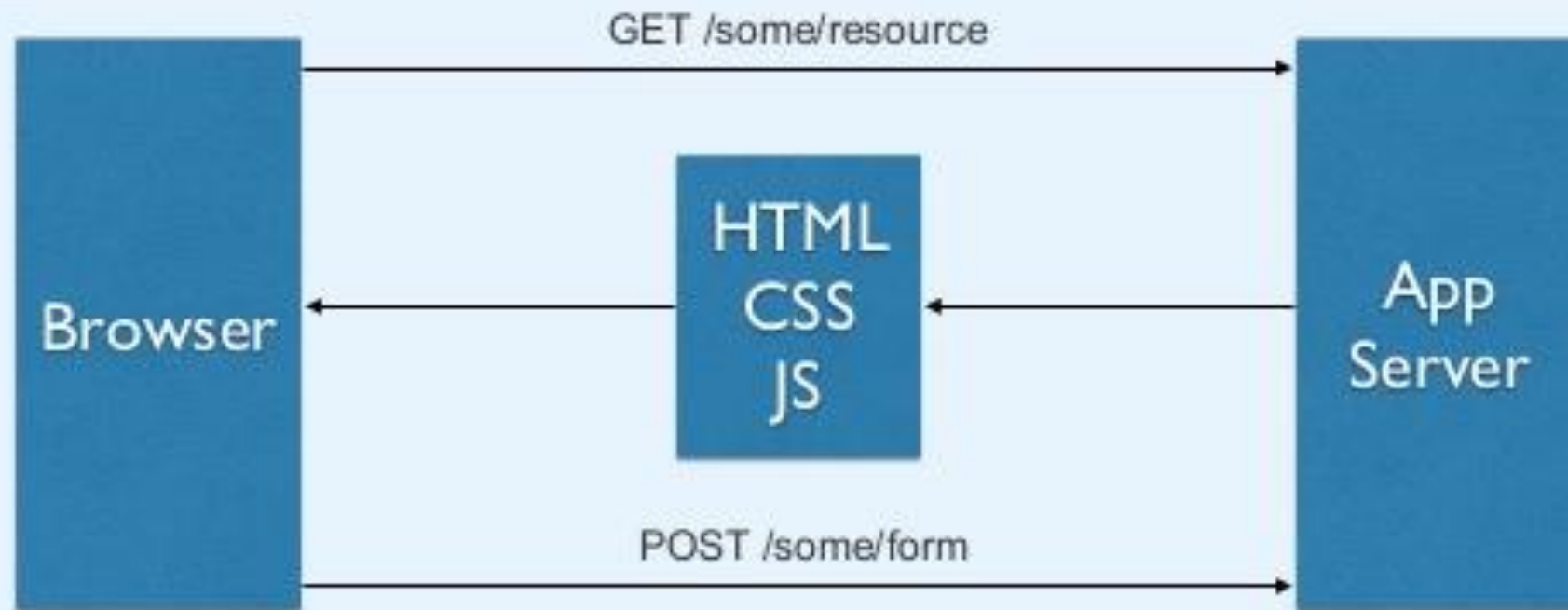
By: Dr. Zeeshan Bhatti

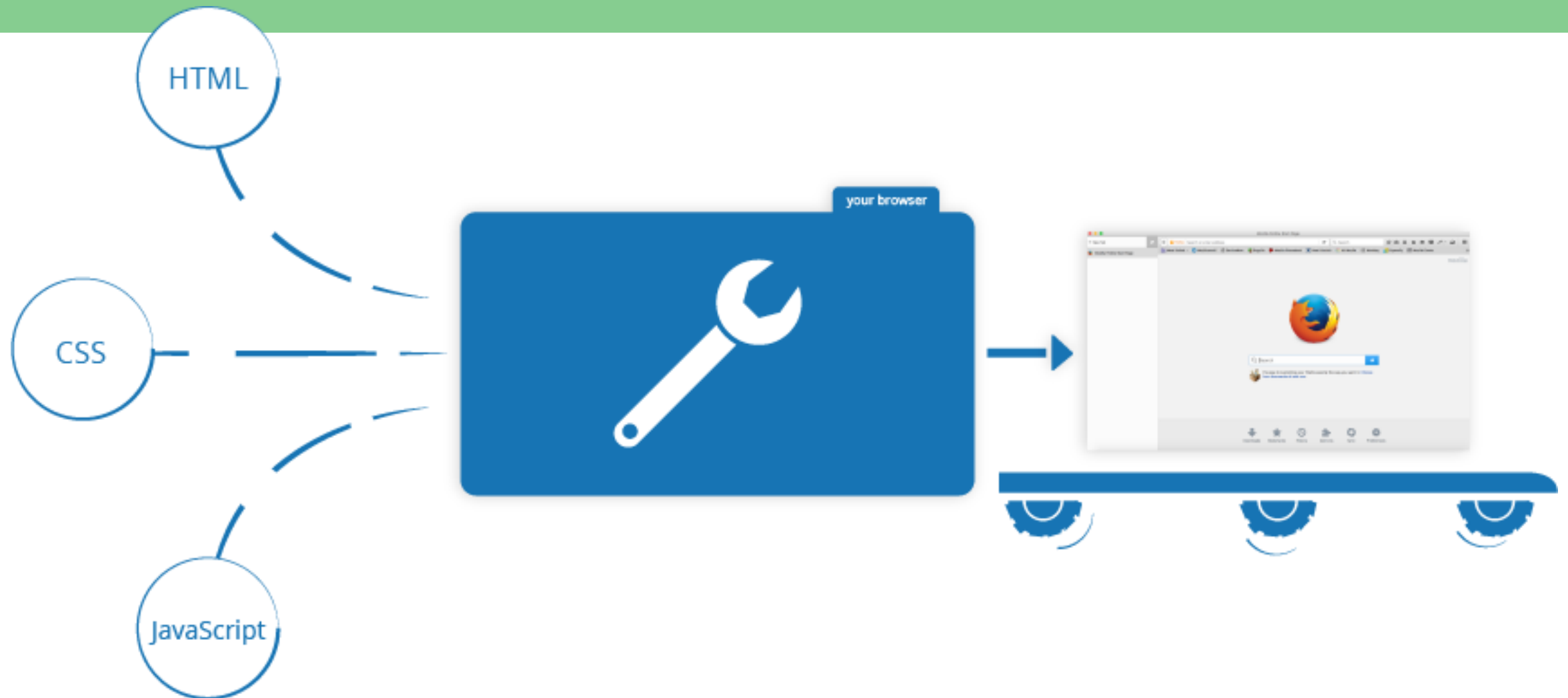


Introduction to JavaScript

Lesson - 1

Traditional Web Apps





<SCRIPT>
ALERT
VAR
FUNCTION
IF/ELSE
typeof
REPLACE



RETURN
ARRAY
SETINTERVAL
WHILE
THIS
FOREACH
</SCRIPT>

What is JavaScript?

More interactive web pages



Interpreted language

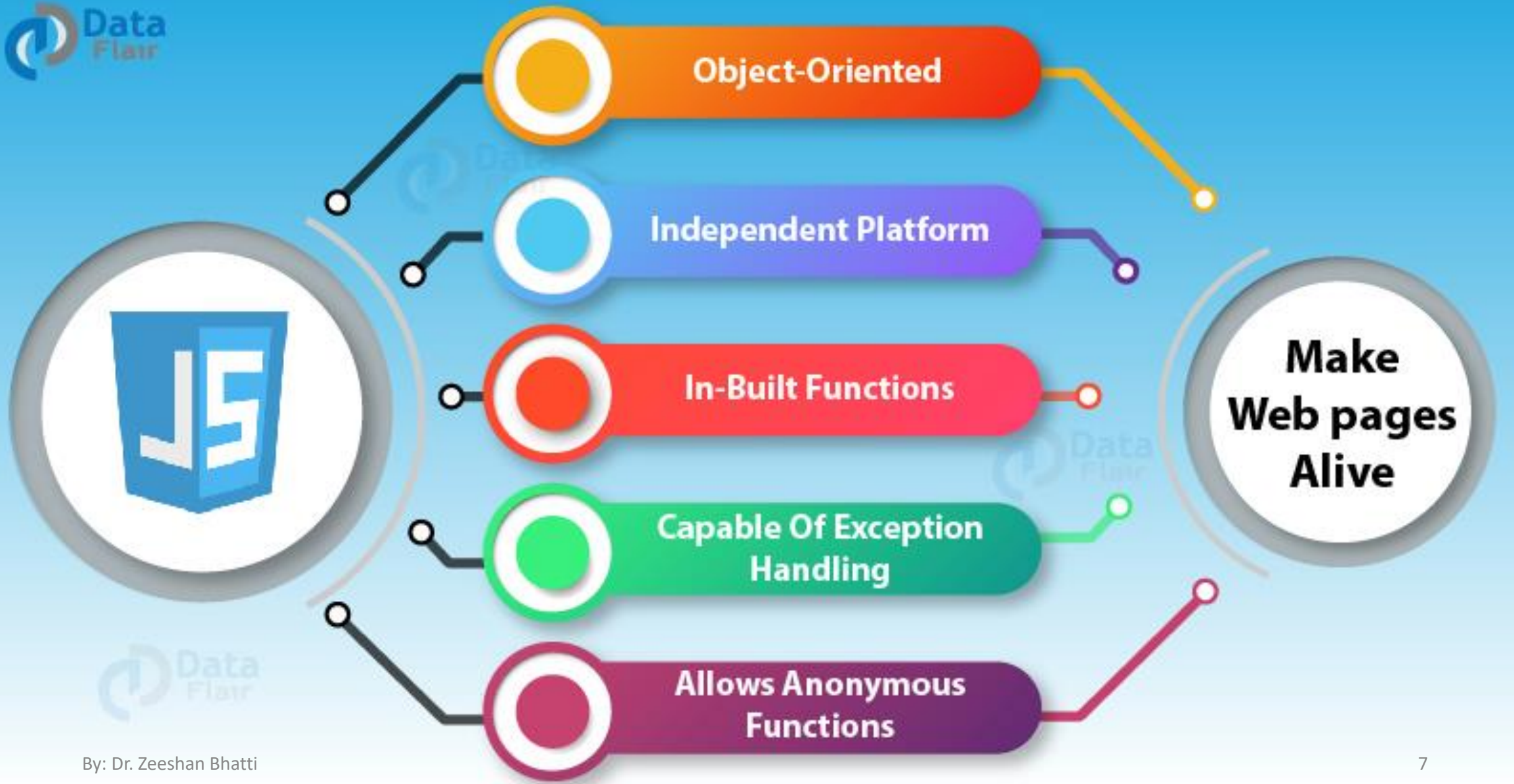


Interpreter

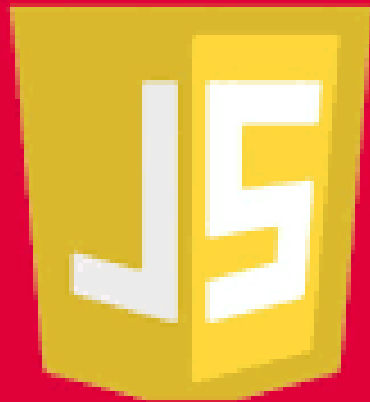


Runs on the client's computer





What is



What is JS ?

Browser Side Scripting
Language

Adds interactivity to
HTML

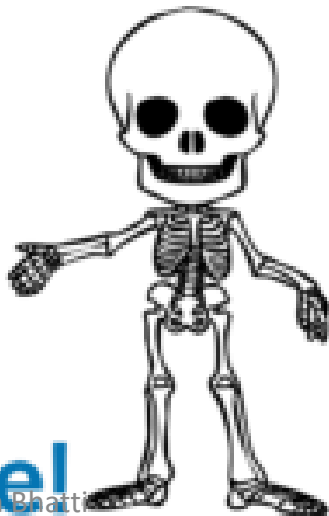
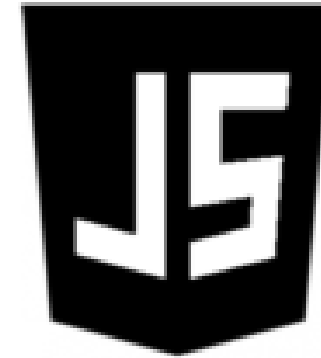
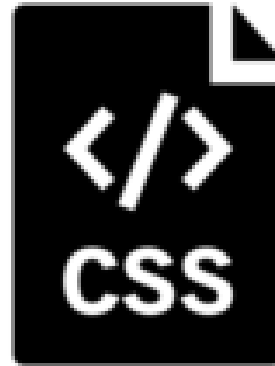
Lightweight Programming
Language

Embedded into HTML

Interpreted language

Free of Cost

What is JS



JavaScript?

- Easy-to-use programming language that can be embedded in the header of your web pages.
- It can enhance the dynamics and interactive features of your page by allowing you to perform
 - calculations,
 - check forms,
 - write interactive games,
 - add special effects,
 - customize graphics selections,
 - create security passwords and more.

JavaScript

- JavaScript is a scripting language
- A scripting language is a lightweight programming language
- JavaScript was designed to add interactivity to HTML pages
- JavaScript is usually embedded directly into HTML pages
- JavaScript is an interpreted language (means that scripts execute without preliminary compilation)
- Everyone can use JavaScript without purchasing a license

What can a JavaScript do?

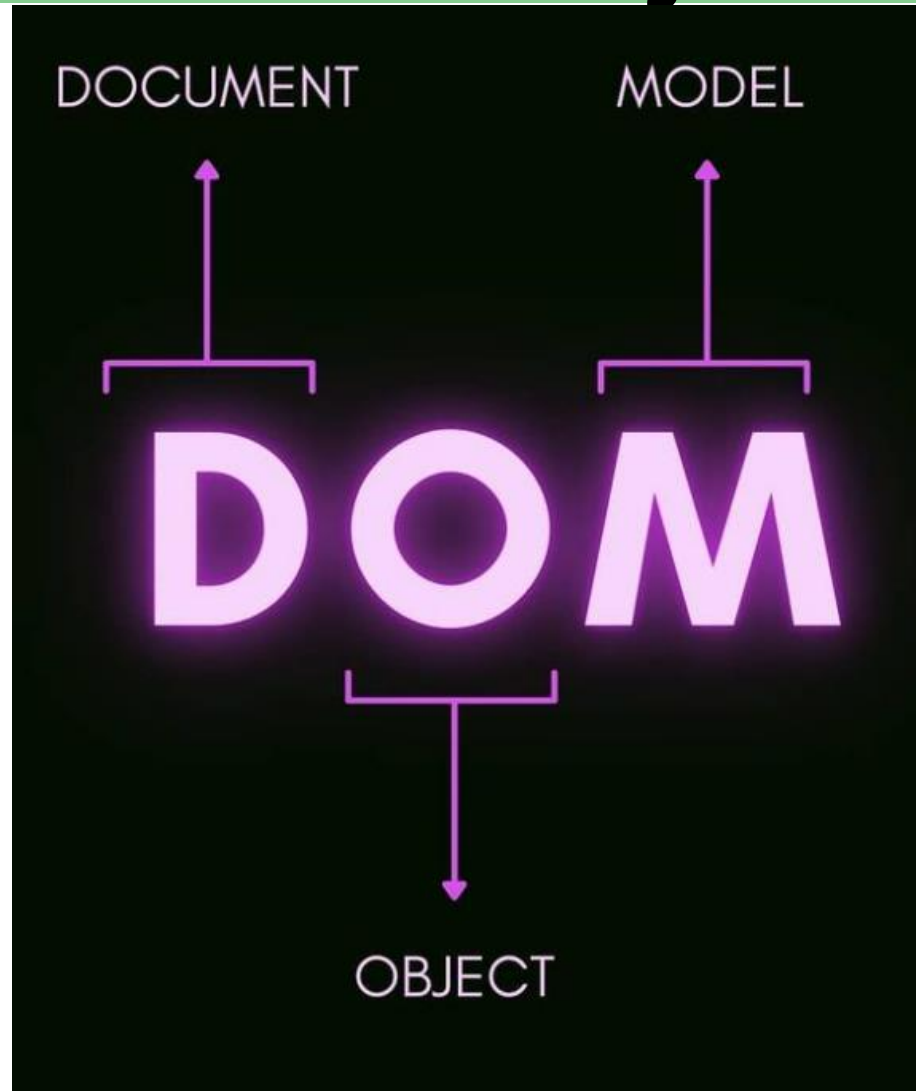
- **JavaScript can put dynamic text into an HTML page** - A JavaScript statement like this: `document.write("<h1>" + name + "</h1>")` can write a variable text into an HTML page
- **JavaScript can react to events** - A JavaScript can be set to execute when something happens, like when a page has finished loading or when a user clicks on an HTML element
- **JavaScript can read and write HTML elements** - A JavaScript can read and change the content of an HTML element
- **JavaScript can be used to validate data** - A JavaScript can be used to validate form data before it is submitted to a server. This saves the server from extra processing

JavaScript DOM

Lesson - 2

DOM

Document Object Model



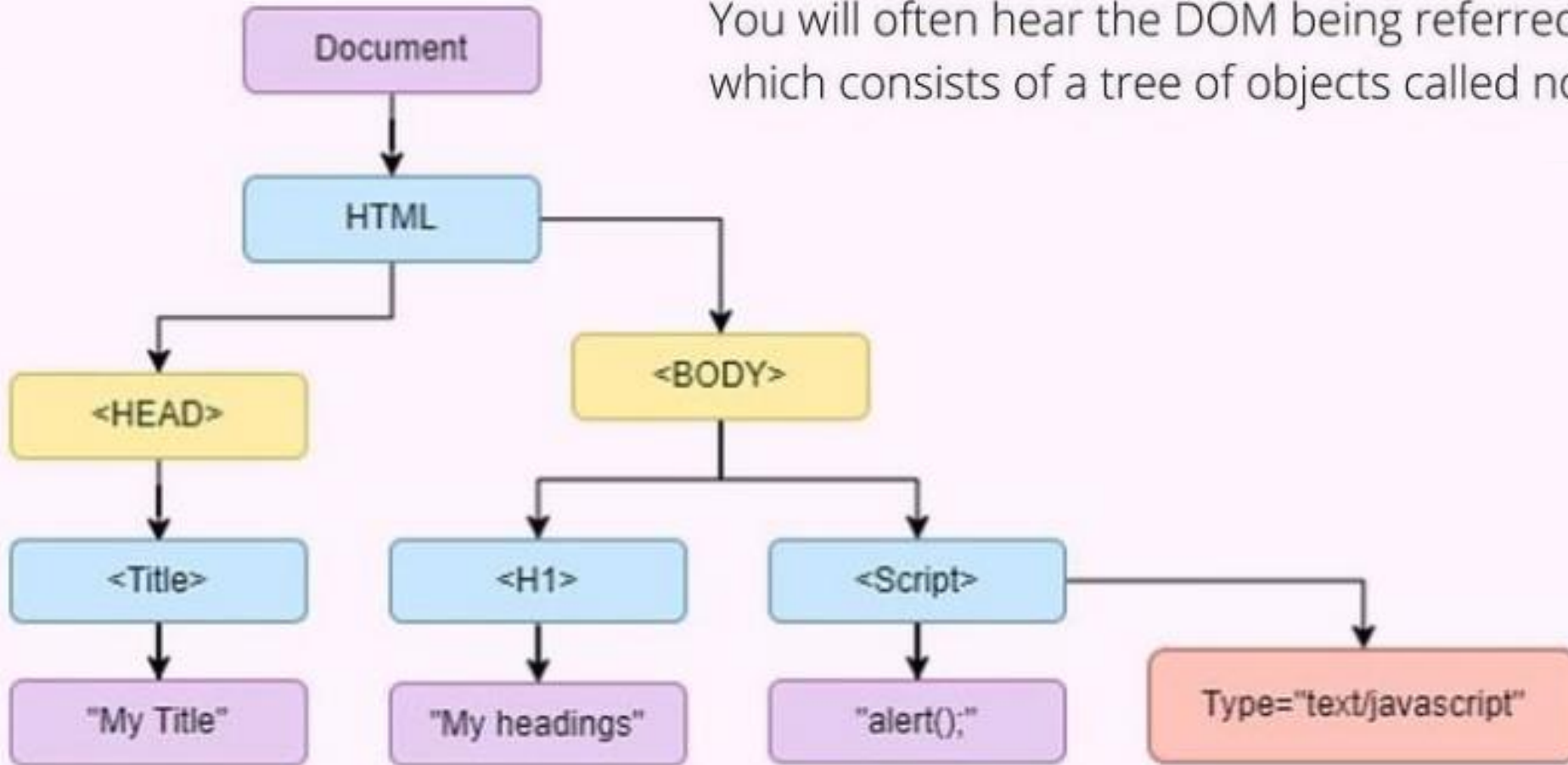
Document Object Model

- Every web page resides inside a browser window which can be considered as an object.
- A Document object represents the HTML document that is displayed in that window.
- The Document object has various properties that refer to other objects which allow access to and modification of document content.
- The way that document content is accessed and modified is called the Document Object Model, or DOM. The Objects are organized in a hierarchy.

- **Objects and Properties:** Your web page document is an object.
- Any table, form, button, image, or link on your page is also an object.
- Each object has certain properties (information about the object).
- For example, the background color of your document is written **document.bgcolor**.
- You would change the color of your page to red by writing the line:
document.bgcolor="red"
- The contents (or value) of a textbox named "password" in a form named "entryform" is
document.entryform.password.value.

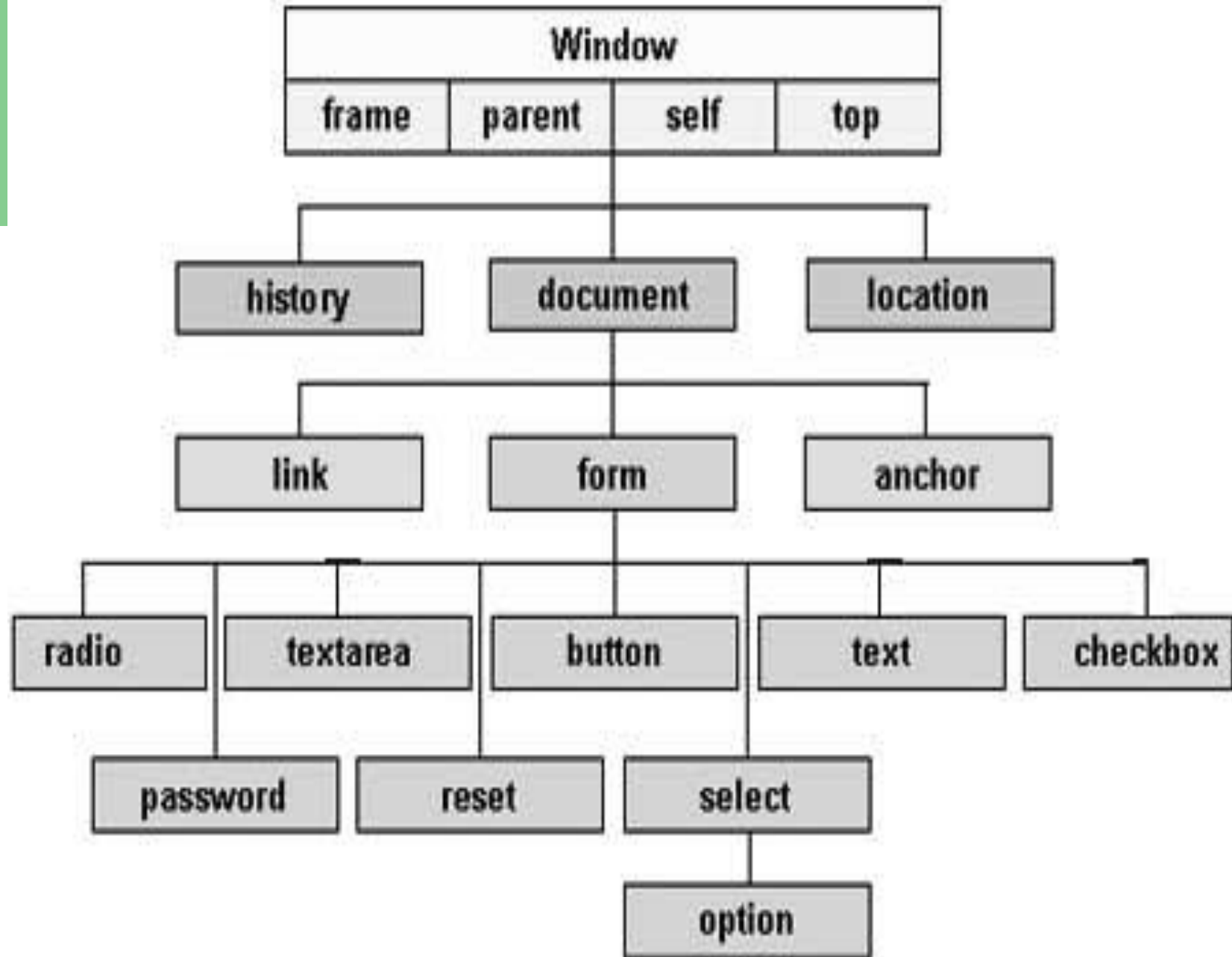
DOM TREE

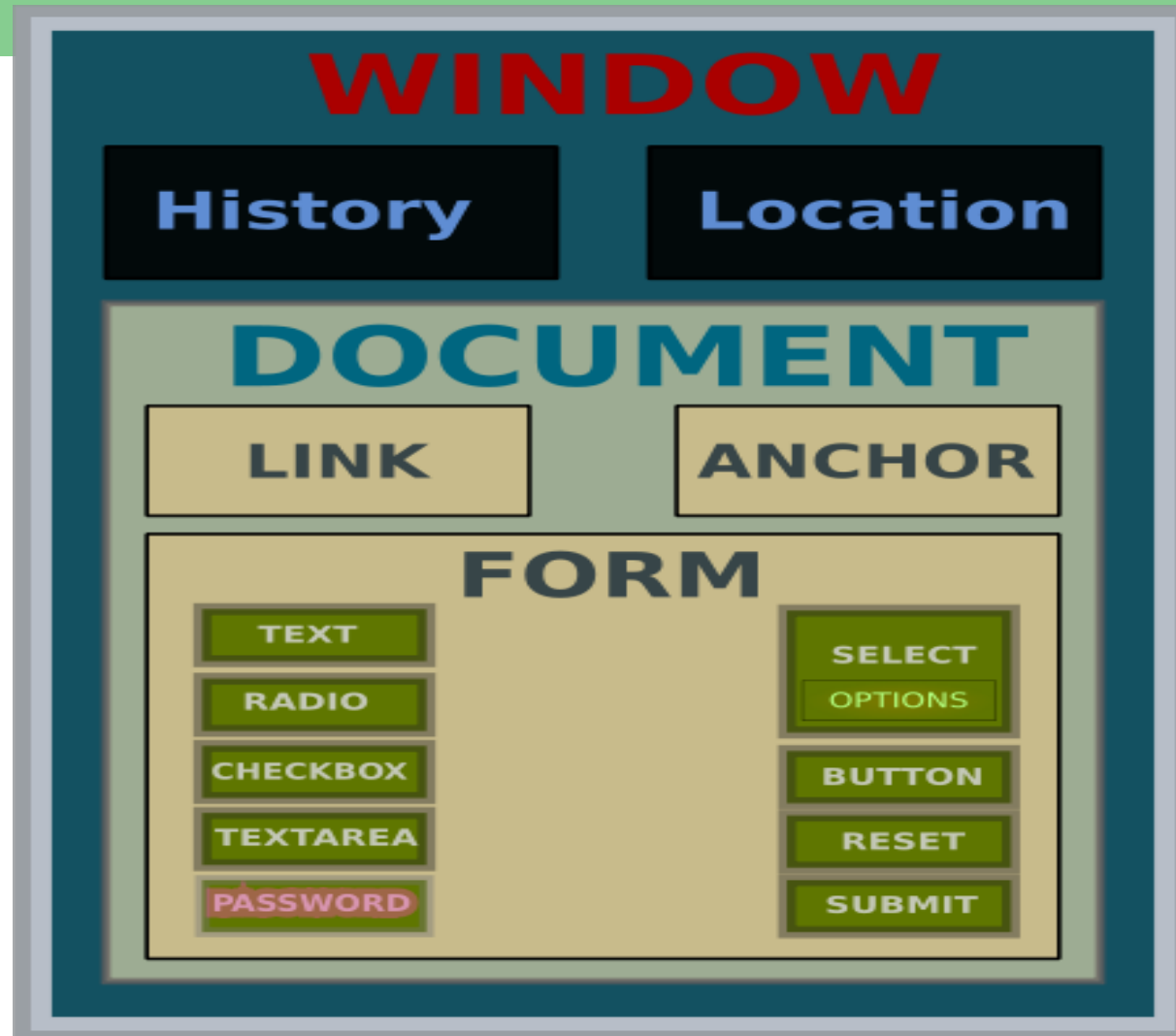
You will often hear the DOM being referred to as the DOM tree, which consists of a tree of objects called nodes.



DOM

TREE (extended)





Methods to Grab Elements

Document

getElementById	Returns the element with id elementId.
getElementsByTagName	Returns the NodeList of elements with tag tagName.
createElement	Create an element of the specified type.
createTextNode	Create a text node with specified text
createAttribute	Create an attribute of the specified name.
addEventListener	Attach an event handler of event of event to the document that runs function.

- **Methods:** Most objects have a certain collection of things that they can **do**.
- Different objects can do different things, just as a door can open and close, while a light can turn on and off. A new document is opened with the method **document.open()**
- You can write "Hello World" into a document by typing **document.write("Hello World")** .
- **open()** and **write()** are both *methods* of the object: document.

Methods to Play with **Nodes**

Node

appendChild(<i>node</i>)	Append Node as a child Node
removeChild(<i>node</i>)	Remove child node <i>node</i> .
hasChildNodes()	Returns true if the node has child nodes
childNodes()	Returns a NodeList of an element's child nodes.
parentNode()	Returns parent node as node object
insertBefore(<i>new</i>, <i>existing</i>)	Insert <i>newNode</i> as a child node before <i>existingNode</i>

DOM Events

Mouse

onclick

onmouseover

onmouseout

Form

onchange

onfocus

onblur

oninput

Keyboard

onkeydown

onkeyup

onkeypress

DOM Events

- **Events:** Events are how we trigger our functions to run. The easiest example is a button, whose definition includes the words **onClick="run_my_function()"**.
- The onClick event, as its name implies, will run the function when the user clicks on the button. Other events include **OnMouseOver**, **OnMouseOut**, **OnFocus**, **OnBlur**, **OnLoad**, and **OnUnload**.

What is the HTML DOM?

- The HTML DOM is a standard **object** model and **programming interface** for HTML. It defines:
 - The HTML elements as **objects**
 - The **properties** of all HTML elements
 - The **methods** to access all HTML elements
 - The **events** for all HTML elements
- In other words: **The HTML DOM is a standard for how to get, change, add, or delete HTML elements.**

How to Use JavaScript

Lesson - 3

JavaScript Power

With the object model, JavaScript gets all the power it needs to create dynamic HTML:

- JavaScript can change all the HTML elements in the page
- JavaScript can change all the HTML attributes in the page
- JavaScript can change all the CSS styles in the page
- JavaScript can remove existing HTML elements and attributes
- JavaScript can add new HTML elements and attributes
- JavaScript can react to all existing HTML events in the page
- JavaScript can create new HTML events in the page

Applications of JavaScript

1. Web Development



2. Web Applications



3. Presentations



4. Server Applications



5. Web Servers



6. Games



Understanding JavaScript structure and syntax.

- **How to use JavaScript into an HTML page:** The HTML `<script>` tag is used to insert a JavaScript into an HTML page. The example below shows how to use JavaScript to write text on a web page:

```
<html>
```

```
    <body>
```

```
        <script type="text/javascript">
```

```
            document.write("Hello World!");
```

```
        </script>
```

```
    </body>
```

```
</html>
```

The **document.write** command is a standard JavaScript command for writing output to a page.

- The example below shows how to add HTML tags to the JavaScript:

```
<html>  
  <body>  
    <script type="text/javascript">  
      document.write("<h1>Hello World!</h1>");  
    </script>  
  </body>  
</html>
```

To insert a JavaScript into an HTML page, we use the `<script>` tag. Inside the `<script>` tag we use the `type` attribute to define the scripting language.

So, the `<script type="text/javascript">` and `</script>` tells where the JavaScript starts and ends:

- By entering the `document.write` command between the `<script>` and `</script>` tags, the browser will recognize it as a JavaScript command and execute the code line.
- In this case the browser will write Hello World! to the page:

Note: If we had not entered the `<script>` tag, the browser would have treated the `document.write("Hello World!")` command as pure text, and just write the entire line on the page.

How to Handle Browsers

- Browsers that do not support JavaScript, will display JavaScript as page content. To prevent them from doing this, and as a part of the JavaScript standard, the HTML comment tag should be used to "hide" the JavaScript.
- Just add an HTML comment tag `<!--` before the first JavaScript statement, and a `-->` (end of comment) after the last JavaScript statement, like this:

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

The two forward slashes at the end of comment line (`//`) is the JavaScript comment symbol. This prevents JavaScript from executing the `-->` tag

Where to Put the JavaScript

- JavaScript in a page will be executed immediately while the page loads into the browser. This is not always what we want. Sometimes we want to execute a script when a page loads, other times when a user triggers an event.
- **Scripts in <body>**
- *Scripts are to be executed when the page loads and go in the body section.* If you place a script in the body section, it generates the content of a page.

```
<html>
  <head> </head>
  <body>
    <script type="text/javascript">
      document.write("This message is written by JavaScript");
    </script>
  </body>
</html>
```

Scripts in <head>

Scripts to be executed when they are called, or when an event is triggered, go in the head section. If you place a script in the head section, you will ensure that the script is loaded before anyone uses it.

```
<html>
```

```
  <head>
```

```
    <script type="text/javascript">
```

```
      function message() {
```

```
        alert("This alert box was called with the onload event"); }
```

```
    </script>
```

```
  </head>
```

```
  <body onload="message()"> </body>
```

```
</html>
```

Scripts in <head> and <body>

- You can place an unlimited number of scripts in your document, so you can have scripts in both the body and the head section.

```
<html>
```

```
    <head>
```

```
        <script type="text/javascript"> .... </script>
```

```
    </head>
```

```
    <body>
```

```
        <script type="text/javascript"> .... </script>
```

```
    </body>
```

JavaScript in the body section will be executed WHILE the page loads.
JavaScript in the head section will be executed when CALLED.

JavaScript Can Change HTML Content

- One of many JavaScript HTML methods is `getElementById()`.
- The example below "finds" an HTML element (with `id="demo"`), and changes the element content (`innerHTML`) to "Hello JavaScript":

```
document.getElementById("demo").innerHTML = "Hello  
JavaScript";
```

example

```
<!DOCTYPE html>
<html>
<body>

<h2>What Can JavaScript Do?</h2>

<p id="demo">JavaScript can change HTML content.</p>

<button type="button"
onclick='document.getElementById("demo").innerHTML =
"Hello JavaScript!">Click Me!</button>

</body>
</html>
```

What Can JavaScript Do?

JavaScript can change HTML content.

Click Me!



What Can JavaScript Do?

Hello JavaScript!

Click Me!

JavaScript Can Change HTML Attribute Values

- In this example JavaScript changes the value of the **src (source)** attribute of an **** tag:

```
<!DOCTYPE html>  
<html>  
<body>
```

```
<h2>What Can JavaScript Do?</h2>
```

```
<p>JavaScript can change HTML attribute values.</p>
```

```
<p>In this case JavaScript changes the value of the src (source) attribute of an image.</p>
```

```
<button onclick="document.getElementById('myImage').src='pic_bulbon.gif'">Turn on the  
light</button>
```

```

```

```
<button onclick="document.getElementById('myImage').src='pic_bulboff.gif'">Turn off the  
light</button>
```

```
</body>  
</html>
```

By: Dr. Zeeshan Bhatti

What Can JavaScript Do?

JavaScript can change HTML attribute values.

In this case JavaScript changes the value of the src (source) attribute of an image.



Turn on the light

Turn off the light

JavaScript Can Change **HTML** **Styles (CSS)**

- Changing the style of an HTML element, is a variant of changing an HTML attribute:
- `document.getElementById("demo").style.fontSize = "35px";`

JavaScript Can **Hide** HTML Elements

- Hiding HTML elements can be done by changing the display style:

```
document.getElementById("demo").style.display = "none";
```

JavaScript Can **Show** HTML Elements

- Showing hidden HTML elements can also be done by changing the display style

```
document.getElementById("demo").style.display = "block";
```

JS



Introduction to

JAVASCRIPT

Full Course

By: Dr. Zeeshan Bhatti

Outputs in Javascript

Lesson - 4

JavaScript Display Possibilities

- JavaScript can "display" data in different ways:
- Writing into an HTML element, using `innerHTML`.
- Writing into the HTML output using `document.write()`.
- Writing into an alert box, using `window.alert()`.
- Writing into the browser console, using `console.log()`.

Using innerHTML

- To access an HTML element, JavaScript can use the `document.getElementById(id)` method.
- The id attribute defines the HTML element. The `innerHTML` property defines the HTML content:

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My First Paragraph</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = 5 +
6;
</script>

</body>
</html>
```

Using document.write()

- For testing purposes, it is convenient to use `document.write()`:
- The `document.write()` method should only be used for testing.

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<script>
document.write(5 + 6);
</script>

</body>
</html>
```


Using window.alert()

- You can use an alert box to display data:
- You can skip the **window** keyword.
- `<script>`
 `alert(5 + 6);`
 `</script>`

```
<!DOCTYPE html>  
<html>  
<body>
```

```
<h1>My First Web Page</h1>  
<p>My first paragraph.</p>
```

```
<script>  
window.alert(5 + 6);  
</script>
```

```
</body>  
</html>
```

Using console.log()

- For debugging purposes, you can call the `console.log()` method in the browser to display data.

```
<!DOCTYPE html>
<html>
<body>

<script>
    console.log(5 + 6);
</script>

</body>
</html>
```

JavaScript Print

- JavaScript does not have any **print** object or print methods.
- You cannot access output devices from JavaScript.
- The only exception is that you can call the **window.print()** method in the browser to print the content of the current window.

```
<button onclick="window.print()">Print this  
page</button>
```

Change `textContent`

- Setting the `textContent` property of an Element is one way to output text on a web page

```
<p id="paragraph"></p>
```

- To change its `textContent` property, we can run the following JavaScript

```
document.getElementById("paragraph").textContent = "Hello, World";
```

- This will select the element that with the id `paragraph` and set its text content to "Hello, World":

```
<p id="paragraph">Hello, World</p>
```

JavaScript Variables

Lesson - 5

JavaScript Variables

Defining a Variable

```
var myVariable = "This is a variable!";
```

- This is an example of defining variables. This variable is called a "string" because it has ASCII characters (A-Z, 0-9, !@#\$, etc.)

```
var number1 = 5;
```

```
number1 = 3;
```

```
console.log(number1); // 3
```

```
window.alert(number1); // 3
```

Operations on Variables

- To add, subtract, multiply, divide, etc., we do like so:

```
number1 = number1 + 5; // 3 + 5 = 8
```

```
number1 = number1 - 6; // 8 - 6 = 2
```

```
var number2 = number1 * 10; // 2 (times) 10 = 20
```

```
var number3 = number2 / number1; // 20 (divided by) 2 = 10;
```

We can also add strings which will concatenate them, or put them together. For example:

```
var myString = "I am a " + "string!"; // "I am a string!"
```

Types of Variables

- **var** myInteger = 12; // 32-bit number (from -2,147,483,648 to 2,147,483,647)
- **var** myLong = 9310141419482; // 64-bit number (from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807)
- **var** myFloat = 5.5; // 32-bit floating-point number (decimal)
- **var** myDouble = 9310141419482.22; // 64-bit floating-point number
- **var** myBoolean = true; // 1-bit true/false (0 or 1)
- **var** myBoolean2 = false;
- **var** myNotANumber = NaN;
- **var** NaN_Example = 0/0; // NaN: Division by Zero is not possible
- **var** notDefined; // undefined: we didn't define it to anything yet
- window.alert(aRandomVariable); // undefined
- **var** myNull = null; // null

Let Variable

- In this example, x, y, and z, are variables, declared with the let keyword:

```
let x = 5;
```

```
let y = 6;
```

```
let z = x + y;
```

JavaScript **const**

- If you want a general rule: always declare variables with **const**.
- If you think the value of the variable can change, use **let**.
- In this example, **price1**, **price2**, and **total**, are variables:
- ```
const price1 = 5;
const price2 = 6;
let total = price1 + price2;
```

# Arrays

```
var myArray = []; // empty array
```

- An array is a set of variables. For example:

```
var favoriteFruits = ["apple", "orange", "strawberry"];
```

```
var carsInParkingLot = ["Toyota", "Ferrari", "Lexus"];
```

```
var employees = ["Billy", "Bob", "Joe"];
```

```
var primeNumbers = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31];
```

```
var randomVariables = [2, "any type works", undefined, null, true, 2.51];
```

```
myArray = ["zero", "one", "two"];
```

```
window.alert(myArray[0]); // 0 is the first element of an array
```

*// in this case, the value would be "zero"*

```
myArray = ["John Doe", "Billy"];
```

```
elementNumber = 1;
```

```
window.alert(myArray[elementNumber]); // Billy
```

# Conditions and Loops in JavaScript

## Lesson -6

# Ternary operators

- Can be used to shorten if/else operations. This comes in handy for returning a value quickly (i.e. in order to assign it to another variable).

For example:

```
var animal = 'kitty';
```

```
var result = (animal === 'kitty') ? 'cute' : 'still nice';
```

- In this case, result gets the 'cute' value, because the value of animal is 'kitty'. If animal had another value, result would get the 'still nice' value.
- Compare this to what the code would look like with if/else conditions

```
var animal = 'kitty';
var result = '';
if (animal === 'kitty') {
 result = 'cute';
} else {
 result = 'still nice';
}
```

# Switch statement

- Switch statements compare the value of an expression against 1 or more values and executes different sections of code based on that comparison.

```
var value = 1;
switch (value) {
 case 1:
 console.log('I will always run');
 break;
 case 2:
 console.log('I will never run');
 break;
}
```

# If / Else If / Else Control

- In its most simple form, an if condition can be used like this:

```
var i = 0;
if (i < 1) {
 console.log("i is smaller than 1");
}
```

- The condition `i < 1` is evaluated, and if it evaluates to **true** the block that follows is executed. If it evaluates to **false**, the block is skipped.
- An if condition can be expanded with an **else** block

```
if (i < 1) {
 console.log("i is smaller than 1");
} else {
 console.log("i was not smaller than 1");
}
```

- Supposing the **else** block contains nothing but another if block (with optionally an **else** block) like this:

```
if (i < 1) {
 console.log("i is smaller than 1");
} else {
 if (i < 2) {
 console.log("i is smaller than 2");
 } else {
 console.log("none of the previous conditions was true");
 }
}
```



Some important footnotes about the above examples:

- If any one condition evaluated to **true**, no other condition in that chain of blocks will be evaluated, and all
- corresponding blocks (including the **else** block) will not be executed.
- The number of **else if** parts is practically unlimited. The last example above only contains one, but you can
- have as many as you like.
- The *condition* inside an if statement can be anything that can be coerced to a boolean value, see the topic on
- boolean logic for more details;
- The if-else-if ladder exits at the first success. That is, in the example above, if the value of i is 0.5 then the
- first branch is executed. If the conditions overlap, the first criteria occurring in the flow of execution is
- executed. The other condition, which could also be true is ignored.
- If you have only one statement, the braces around that statement are technically optional, e.g this is fine:

```
if (i < 1) console.log("i is smaller than 1");
```

- And this will work as well:

```
if (i < 1)
```

```
console.log("i is smaller than 1");
```

# JavaScript Loops

JavaScript supports different kinds of loops:

- **for** - loops through a block of code a number of times
- **for/in** - loops through the properties of an object
- **for/of** - loops through the values of an iterable object
- **while** - loops through a block of code while a specified condition is true
- **do/while** - also loops through a block of code while a specified condition is true

# For Loop

The **for** loop has the following syntax:

```
for (statement 1; statement 2; statement 3) {
 // code block to be executed
}
```

**Statement 1** is executed (one time) before the execution of the code block.

**Statement 2** defines the condition for executing the code block.

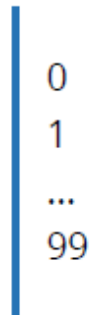
**Statement 3** is executed (every time) after the code block has been executed.

```
for (let i = 0; i < 5; i++) {
 text += "The number is " + i + "
";
}
```

- **Standard usage**

```
for (var i = 0; i < 100; i++) {
 console.log(i);
}
```

Expected output:

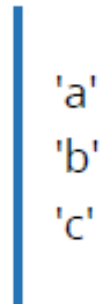


```
0
1
...
99
```

Commonly used to read the length of an array.

```
var array = ['a', 'b', 'c'];
for (var i = 0; i < array.length; i++) {
 console.log(array[i]);
}
```

Expected output:



```
'a'
'b'
'c'
```

# For...of Loop

```
const iterable = [0, 1, 2];
for (let i of iterable) {
 console.log(i);
}
```

Expected output:

```
|
0
1
2
```

- for...of will treat a string as a sequence of Unicode characters

```
const string = "abc";
for (let chr of string) {
 console.log(chr);
}
```

Expected output:


```
| a b c
```

# While Loop

- A standard while loop will execute until the condition given is false:

```
var i = 0;
while (i < 100) {
 console.log(i);
 i++;
}
```

Expected output:



0  
1  
...  
99

# Do-while Loop

- A do...while loop will always execute at least once, regardless of whether the condition is true or false:

```
var i = 101;
do {
 console.log(i);
} while (i < 100);
```

Expected output:



101

```
var availableName;
do {
 availableName = getRandomName();
} while (isNameUsed(name));
```

# Functions in JavaScript

## Lesson -7



# JavaScript Functions

- A JavaScript function is a block of code designed to perform a particular task.
- A JavaScript function is executed when "something" invokes it (calls it).

```
function myFunction(p1, p2) {
 return p1 * p2; // The function returns the product
of p1 and p2
}
```

**Syntax:**

```
function name(parameter1, parameter2, parameter3) {
 // code to be executed
}
```

# Function Arguments

- Functions can take inputs in form of variables that can be used and assigned inside their own scope. The following function takes two numeric values and returns their sum:

```
function addition (argument1, argument2){
 return argument1 + argument2;
}
```

```
console.log(addition(2, 3)); // -> 5
```

# Return Statement

- The return statement can be a useful way to create output for a function.

*//An example function that will take a string as input and return  
//the first character of the string.*

```
function firstChar (stringIn){
 return stringIn.charAt(0);
}
```

```
console.log(firstChar("Hello world"));
```

*Console output will be:*

> H

# Return Statement

- When JavaScript reaches a **return** statement, the function will stop executing.
- Functions often compute a return value. The return value is "returned" back to the "caller":

```
let x = myFunction(4, 3); // Function is called, return
value will end up in x
```

```
function myFunction(a, b) {
 return a * b; // Function returns the
product of a and b
}
```

# JavaScript Objects

## Lesson -8

# Objects

- An object is a group of values; unlike arrays, we can do something better than them:


```
myObject = {};
john = {firstname: "John", lastname: "Doe", fullname: "John Doe"};
billy = {
 firstname: "Billy",
 lastname: undefined,
 fullname: "Billy"
};
window.alert(john.fullname); // John Doe
window.alert(billy.firstname); // Billy
```

Rather than making an array ["John Doe", "Billy"] and calling myArray[0], we can just call john.fullname and billy.fullname.

# Objects

## Real Life Objects, Properties, and Methods

- In real life, a car is an **object**.
- A car has **properties** like weight and color, and **methods** like start and stop:

| Object                                                                             | Properties                                                                                                                        | Methods                                                                                                     |
|------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------|
|  | <code>car.name = Fiat</code><br><code>car.model = 500</code><br><code>car.weight = 850kg</code><br><code>car.color = white</code> | <code>car.start()</code><br><code>car.drive()</code><br><code>car.brake()</code><br><code>car.stop()</code> |

- All cars have the same **properties**, but the property **values** differ from car to car.
- All cars have the same **methods**, but the methods are performed **at different times**.

# JavaScript Objects

- Objects are variables too. But objects can contain many values.
- This code assigns **many values** (Fiat, 500, white) to a **variable** named car:

```
const car = {type:"Fiat", model:"500", color:"white"};
```

- The values are written as **name:value** pairs (name and value separated by a colon).



# Object Definition

- You define (and create) a JavaScript object with an object literal:

```
const person = {firstName:"John", lastName:"Doe", age:50,
eyeColor:"blue"};
```

Spaces and line breaks are not important. An object definition can span multiple lines:

```
const person = {
 firstName: "John",
 lastName: "Doe",
 age: 50,
 eyeColor: "blue"
};
```

# Object Properties

- The name:values pairs in JavaScript objects are called properties:

| Property  | Property Value |
|-----------|----------------|
| firstName | John           |
| lastName  | Doe            |
| age       | 50             |
| eyeColor  | blue           |

# Accessing Object Properties

- You can access object properties in two ways:

*objectName.propertyName*

Or

*objectName["propertyName"]*

Example1

`person.lastName;`

Or

`person["lastName"];`

# JSON

## Lesson -9

# JavaScript JSON

- JSON is a format for storing and transporting data.
- JSON is often used when data is sent from a server to a web page.

## What is JSON?

- JSON stands for **JavaScript Object Notation**
- JSON is a lightweight data interchange format
- JSON is language independent \*
- JSON is "self-describing" and easy to understand
- \* The **JSON** syntax is derived from JavaScript object notation syntax, but the JSON format is text only. Code for reading and generating **JSON** data can be written in any programming language.

- JSON Example

```
{"firstName": "John", "lastName": "Doe"}
```

# JSON Syntax Rules

- Data is in name/value pairs
- Data is separated by commas
- Curly braces hold objects
- Square brackets hold arrays
- JSON data is written as name/value pairs, just like JavaScript object properties.
- A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value:
- `"firstName": "John"`

# JSON Objects

- JSON objects are written inside curly braces.
- Just like in JavaScript, objects can contain multiple name/value pairs:

```
{ "firstName": "John", "lastName": "Doe" }
```



# JSON Array

- JSON arrays are written inside square brackets.
- Just like in JavaScript, an array can contain objects:

```
"employees":[
 {"firstName":"John", "lastName":"Doe"},
 {"firstName":"Anna", "lastName":"Smith"},
 {"firstName":"Peter", "lastName":"Jones"}
]
```

- In the example above, the object "employees" is an array. It contains three objects.
- Each object is a record of a person (with a first name and a last name).

# Converting a JSON Text to a JavaScript Object

- A common use of JSON is to read data from a web server, and display the data in a web page.
- For simplicity, this can be demonstrated using a string as input.
- First, create a JavaScript string containing JSON syntax:
- ```
let text = '{ "employees" : [' +  
  '{ "firstName":"John" , "lastName":"Doe" },' +  
  '{ "firstName":"Anna" , "lastName":"Smith" },' +  
  '{ "firstName":"Peter" , "lastName":"Jones" } ]}';
```

- Then, use the JavaScript built-in function `JSON.parse()` to convert the string into a JavaScript object:
- `const obj = JSON.parse(text);`

```
<!DOCTYPE html>
<html>
<body>

<h2>Create Object from JSON String</h2>

<p id="demo"></p>

<script>
let text = '{"employees":[' +
'{"firstName":"John","lastName":"Doe" },' +
'{"firstName":"Anna","lastName":"Smith" },' +
'{"firstName":"Peter","lastName":"Jones" }]}';

const obj = JSON.parse(text);
document.getElementById("demo").innerHTML =
obj.employees[1].firstName + " " + obj.employees[1].lastName;
</script>

</body>
</html>
```

Create Object from JSON String

Anna Smith



Thankyou

**IF YOU LIKED
MY VIDEO**

**Don't Forget to
Subscribe**

