

Lec- 7

JavaScript1 Language Fundamental

JavaScript was initially created to “*make webpages alive*”.

- JavaScript is a high-level programming language
- Alongside HTML and CSS, JavaScript is one of the three core technologies of the World Wide Web
- allow client-side script to interact with the user and make dynamic pages.
- All the modern browsers come with built-in support for JavaScript.

In-browser JavaScript is able to:

- Add new HTML to the page, change the existing content.
- modify styles.
- React to user actions, run on mouse clicks, pointer movements, key presses.
- download and upload files
- Get and set cookies, ask questions to the visitor, show messages.
- Remember the data on the client-side
- You can validate user input before sending the page off to the server

JavaScript - Syntax

- JavaScript can be implemented using JavaScript statements that are placed within the `<script>... </script>`.
- JavaScript ignores spaces, tabs, and newlines
- Semicolons are **Optional**
- Case-sensitive language.

Note: Old JavaScript examples may use a type attribute:

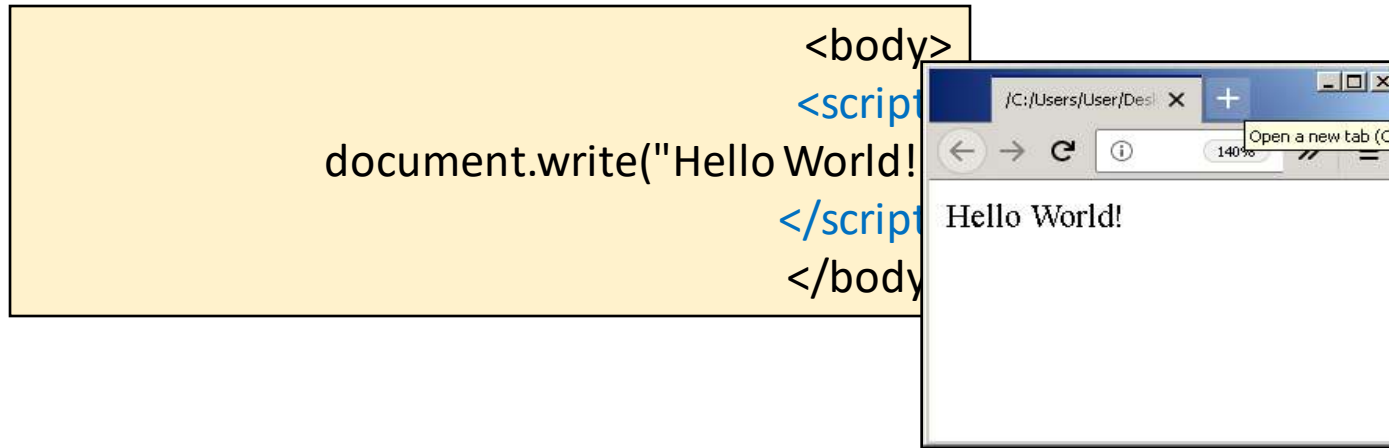
`<script type="text/javascript"> ... </script>`

The type attribute is not required. JavaScript is the default scripting language in HTML.

Placement in HTML File

- Scripts can be placed in the <body>, or in the <head> section of an HTML page, or in both.
- Scripts can also be placed in external files

JS: Hello World



The document .write is a javascript command telling the browser that what follows within the parentheses is to be written into the document.

* JavaScript accepts both double and single quotes.

Values

- **Numbers**

- Integers (whole numbers) e.g. 0, 1, 2, 3, 4
- Floats (fractional numbers) e.g. 3.14

- **Strings**

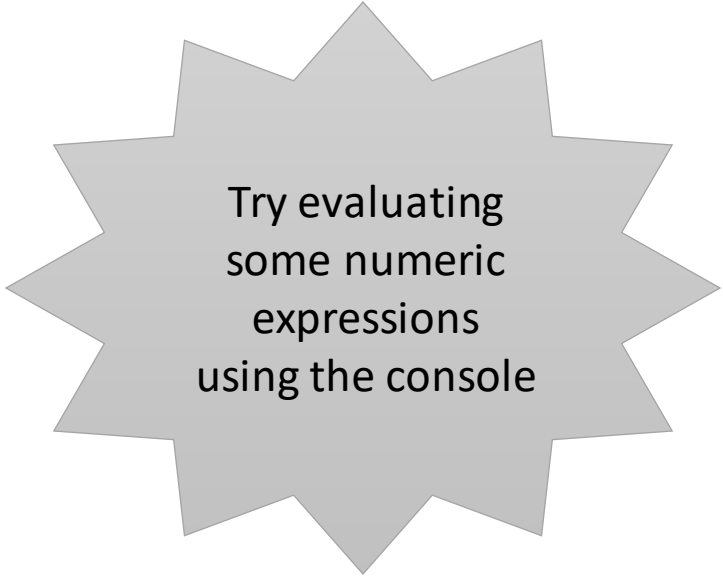
- "A string of characters"
- 'This is also a string, but in single quotes'
- "Escape quotes with backslash like this: \" ";

- **Booleans**

- true
- false

Numeric operators

- Addition: $5 + 3$
- Subtraction: $7 - 6$
- Multiplication: $5.3 * 2.7$
- Division: $20 / 4$
- Modulo: $8 \% 2$
- Increment: $5++$
- Decrement: $2--$



Try evaluating
some numeric
expressions
using the console

String operators

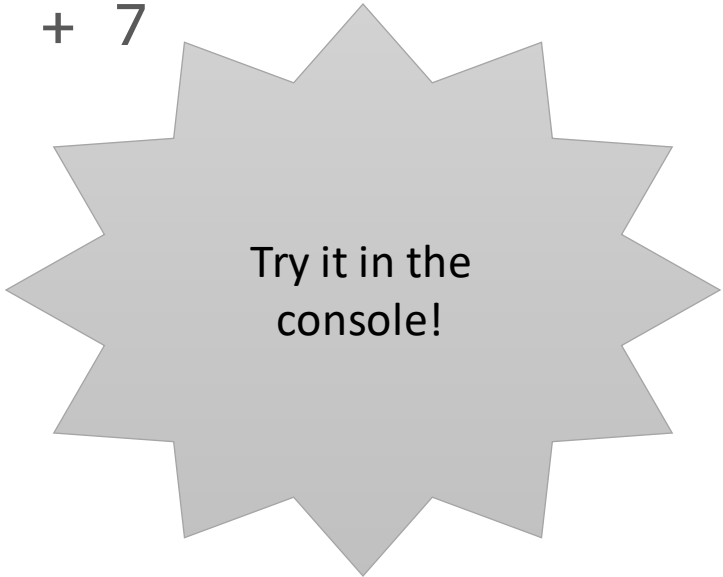
- Concatenation

```
// Evaluates to "this is a concatenated string"
```

```
"this is " + "a concatenated string"
```

- What happens if you add a number to a string?

```
"Here is a string with a number " + 7
```



Try it in the
console!

Variables

Store a value with a name for reference elsewhere in the program

Declaration:

```
var myVariable;
```

Assignment statements:

```
myVariable = true;
```

Declaration and initial assignment:

```
var x = 0;
```

```
var myString = "This is a string";
```

Assignment shorthands

Shorthands for variable assignment include:

`+=`

`-=`

`*=`

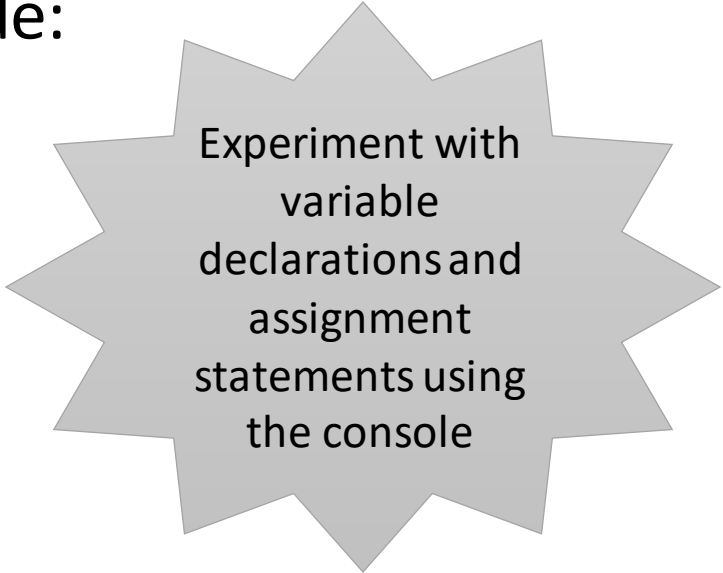
`/=`

`%=`

`x += 3`

is equivalent to

`x = x + 3`



Experiment with
variable
declarations and
assignment
statements using
the console

Statements

- Optionally separated by semi-colons
- Use curly braces { } to group statements into blocks

```
var radius = 3;  
var circumference = 2 * Math.PI * radius;  
console.log("result is " + circumference)
```

Comments

```
// This is a comment until the end of this line only  
var aVariable = "Not a comment";
```

```
/*  
 * This is a comment spanning several lines,  
 * until the star slash  
 */  
// Use comments to disable/enable statements  
// var anotherVariable = "Disabled code";
```

Functions

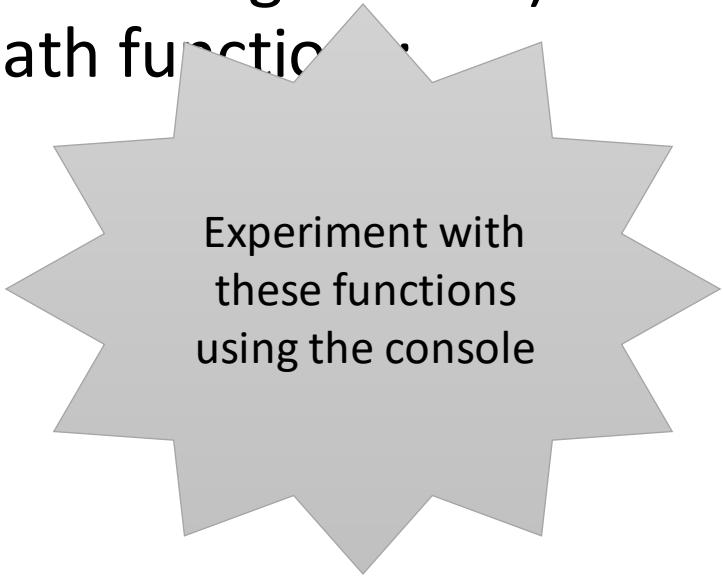
- A block of statements that can be named and called
- Can take parameters e.g. radius
- Can perform an action or return a result (or both!)

```
function calculateCircumference (radius) {  
    var circumference = 2 * Math.PI * radius;  
    return circumference;  
}
```

```
// The function is called elsewhere in the program, we  
    pass in the value 3 for the radius  
var myCircumference = calculateCircumference(3);
```

Built-in libraries

- `Math.PI` is a constant (a variable that never changes value) from the built-in `Math` library. Some additional `Math` functions:
 - `Math.round(4.7)`
 - `Math.sqrt(9)`
 - `Math.max(1, 5)`
 - `Math.min(6, 7)`
 - `Math.floor(5.6)`
 - `Math.random()`
- `console.log()` is a built-in function (in Chrome) that prints values to the console



Experiment with
these functions
using the console

Comparison operators

- Expressions based on comparison operators evaluate to a boolean:
 - Equal:
 - `3.5 == 2 // (evaluates to false)`
 - Not equal:
 - `"aString" != "anotherString" // (true)`
 - Greater than / (or equal):
 - `6 > 6 // (false)`
 - `6 >= 6 // (true)`
 - Less than / (or equal):
 - `5 < 3 // (false)`
 - `5 <= 3 // (false)`

Boolean operators

- Combine boolean expressions using logical operators:
 - AND
 &&
 - OR
 ||
 - NOT
 !

Conditional statements

Implement alternative behaviours based on conditions

```
if (temperature < 20) {  
    console.log("It is cold");  
} else if (temperature >= 20 && temperature < 29) {  
    console.log("It is warm");  
} else {  
    console.log("it is hot");  
}
```

Loops

While loop

```
var maxLimit = 20, counter = 0, value = 0;
while (value !== 6 && counter < maxLimit) {
  // ensure variables in loop condition can change
}
```

For loop

```
for (var i = 0; i < 10; i++){
  // print 0,1,2,3,4,5,6,7,8,9
  console.log(i);
}
```

Update the dice game to keep rolling until the result is 6.

Display the number of rolls it took to win.

Sample solution: <http://jsfiddle.net/AnnaGerber/epaAs/2/>

Arrays

An ordered list of values

```
var myArray = [1,6,10];  
var anotherArray =  
["first value", 5, "third value", false];  
  
// Access values – indexed from 0  
myArray[0] // 1  
anotherArray[2] // "third value"
```

What is the DOM?

- *"The Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."*
- The DOM defines a standard for accessing documents.

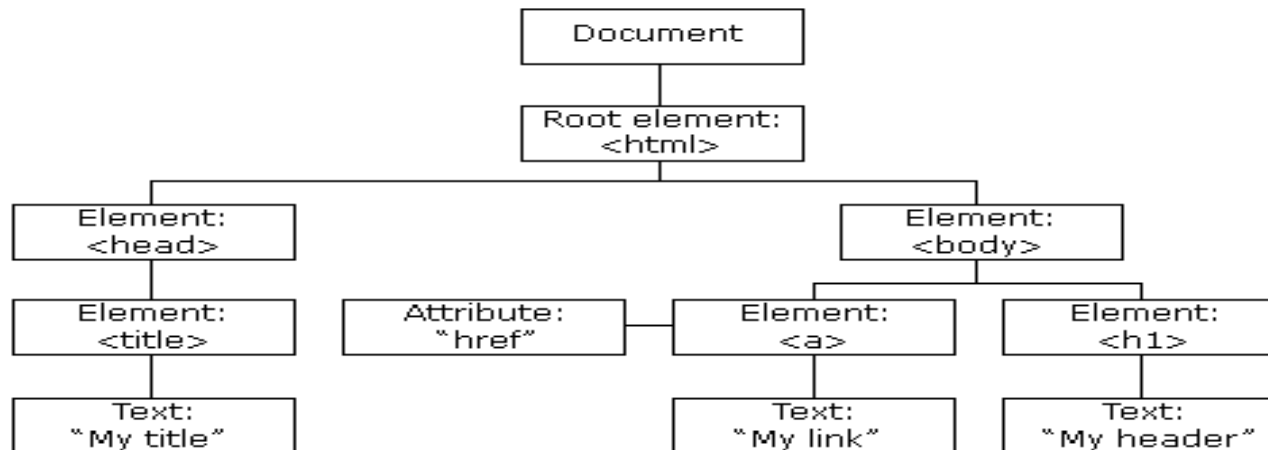
What is the HTML DOM?

- 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, access change, add, or delete HTML elements.

HTML DOM (Document Object Model)

- When a web page is loaded, the browser creates a **Document Object Model** of the page.
- The **HTML DOM** model is constructed as a tree of **Objects**:
- The HTML DOM Tree of Objects

The HTML DOM Tree of Objects



JavaScript - HTML DOM Methods

- **HTML DOM methods are actions you can perform (on HTML Elements).**
- **HTML DOM properties are values (of HTML Elements) that you can set or change.**
- The HTML DOM can be accessed with JavaScript (and with other programming languages).
- In the DOM, all HTML elements are defined as **objects**.
- The programming interface is the properties and methods of each object.
- **A property is a value that you can get or set (like changing the content of an HTML element).**
- **A method is an action you can do (like add or deleting an HTML element).**

The following example changes the content (the innerHTML) of the <p> element with id="demo":

```
<html>
<body>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = "Hello World!";
</script>
</body>
</html>
```

JavaScript Can Change HTML element

- If you want to access/change any element in an HTML page, first you should find the specific element.

Method	Description
<code>document.getElementById(id)</code>	Find an element by element id
<code>document.getElementsByTagName(name)</code>	Find elements by tag name
<code>document.getElementsByClassName(name)</code>	Find elements by class name

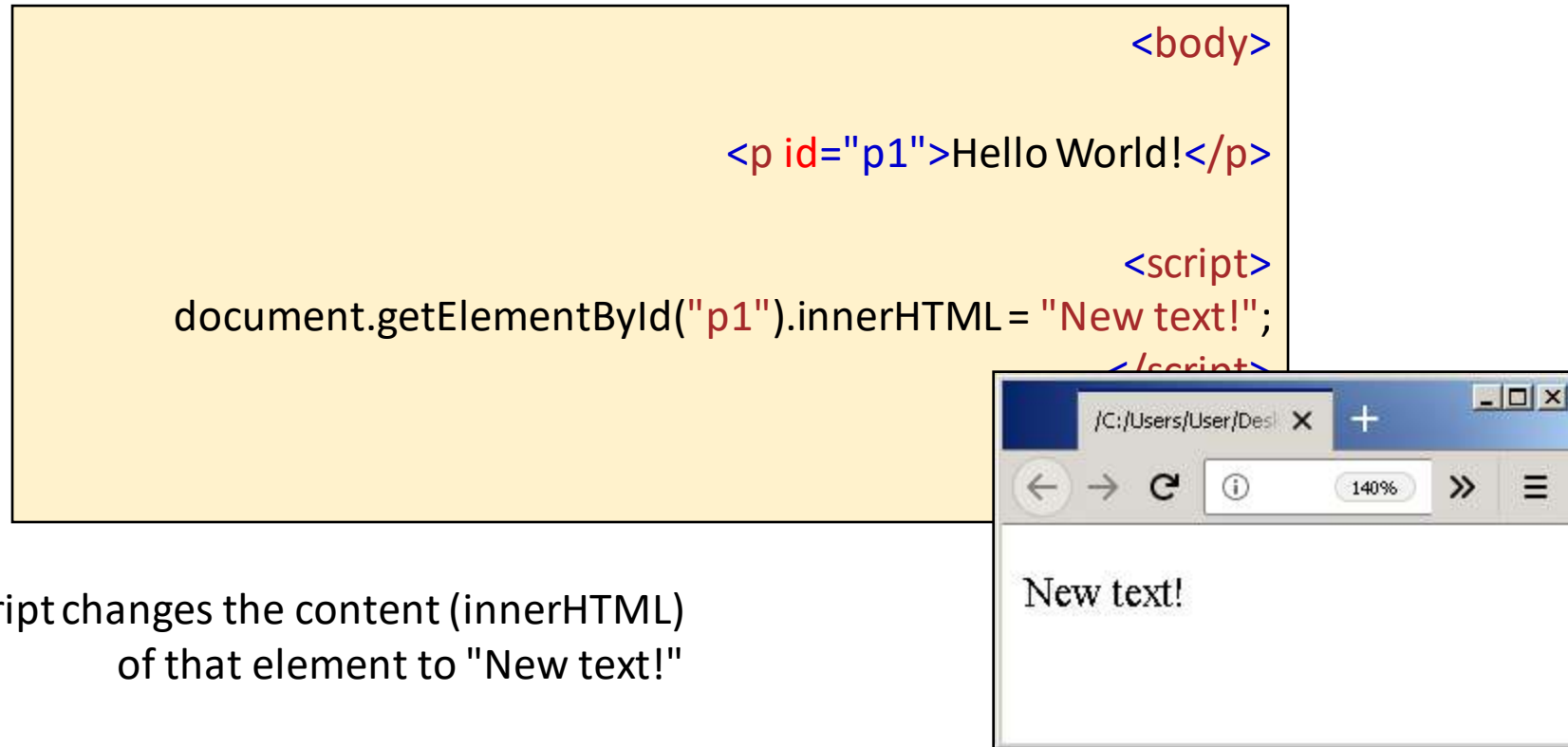
Changing HTML Elements

- If you want to access/change any element in an HTML page, first you should find the specific element.

Method	Description
<code>element.innerHTML = new html content</code>	Change the inner HTML of an element
<code>element.attribute = new value</code>	Change the attribute value of an HTML element
<code>element.setAttribute(attribute, value)</code>	Change the attribute value of an HTML element
<code>element.style.property = new style</code>	Change the style of an HTML element

Example of Changing HTML Content (by ID)

- The easiest way to modify the content of an HTML element is by using the **innerHTML** property.
- This example changes the content of a <p> element:



The diagram illustrates a JavaScript script that changes the content of an HTML element. It consists of two main parts: a code editor and a browser preview.

Code Editor: A yellow rectangular box contains the following HTML and JavaScript code:

```
<body>  
  
<p id="p1">Hello World!</p>  
  
<script>  
document.getElementById("p1").innerHTML = "New text!";  
</script>
```

Browser Preview: A small window on the right shows a browser interface. The address bar displays a file path: `/C:/Users/User/Desktop/`. The browser shows the text "New text!" in the main content area, indicating that the JavaScript code successfully updated the innerHTML of the element with ID "p1".


A JavaScript changes the content (innerHTML)
of that element to "New text!"

Example of Changing HTML Content (by Tag)

- This example changes the content of a <p> element by Tag's name

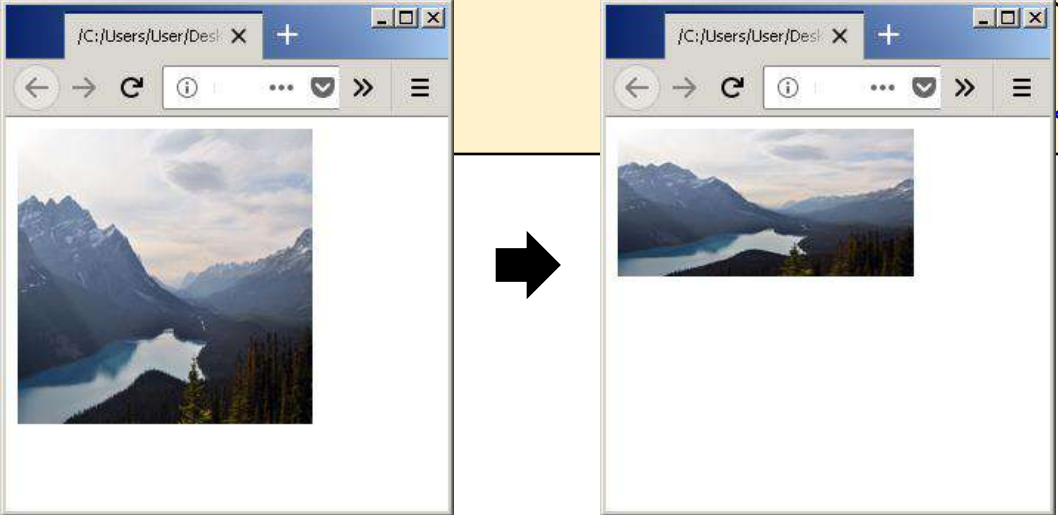
```
<body>  
  
<p>Hello World!!</p>  
<p>Hello World!!</p>  
  
<script>  
document.getElementsByTagName("p")[0].innerHTML = "New text!";  
</script>
```

The Tag can be accessed by index numbers. The index starts at 0.



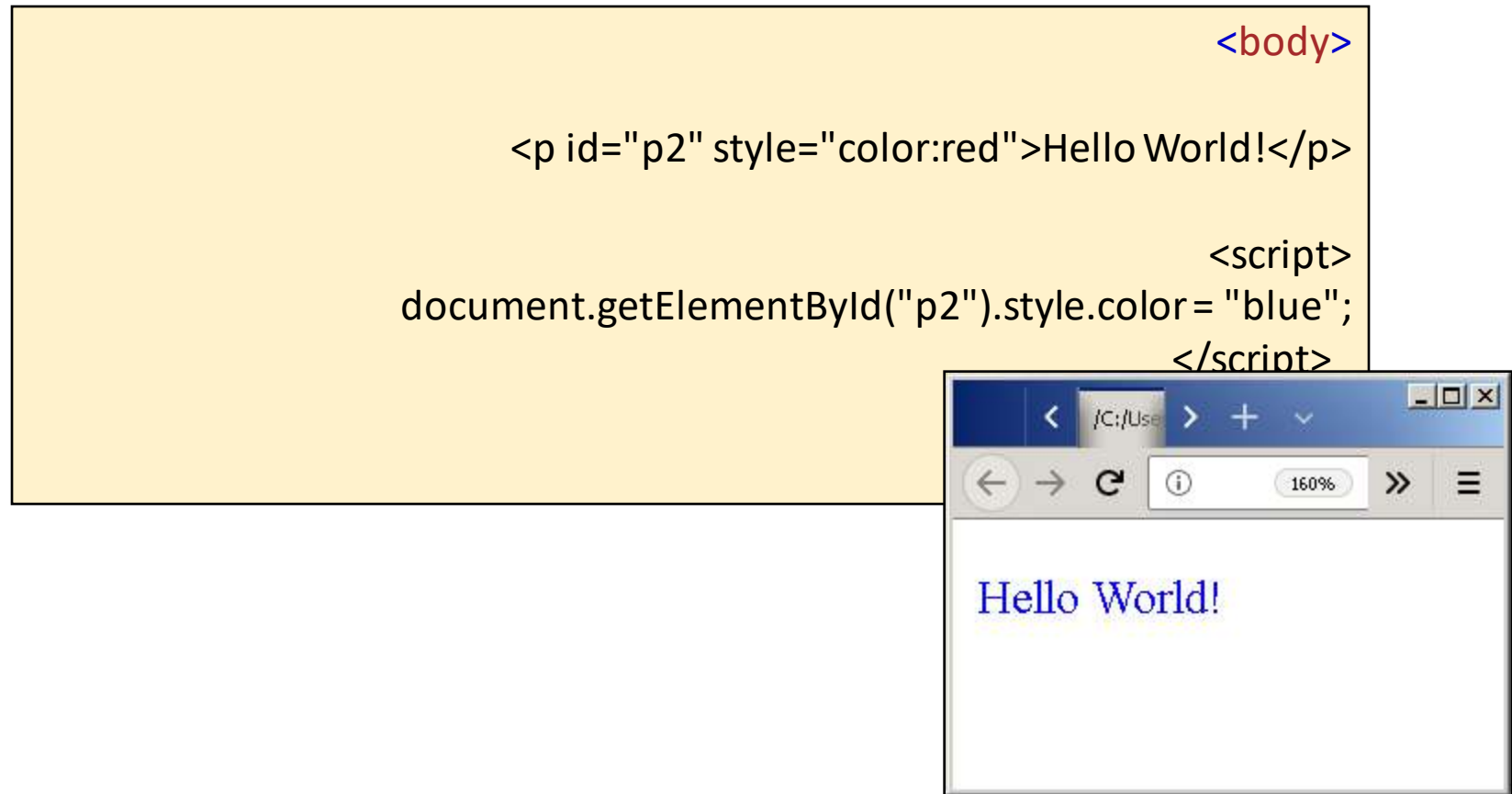
- **Example of Changing the Value of an Attribute**
- This example changes an HTML image by changing the attribute **height** of an `` tag

```
<body>  
  
  
  
<script>  
    document.getElementById("myImage").height = "100";  
</script>  
  
</body>
```



before changing the height after changing the height

- **Example of Changing the style of HTML elements**
- The following example changes the style of a <p> element:



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

In the example above, getElementById is a method, while innerHTML is a property.

The getElementById Method

- The most common way to access an HTML element is to use the id of the element.
- The easiest way to get the content of an element is by using the innerHTML property.
- The innerHTML property is useful for getting or replacing the content of HTML elements.
- The innerHTML property can be used to get or change any HTML element, including <html> and <body>.
- The document object represents your web page.
- If you want to access any element in an HTML page, you always start with accessing the document object.
-

Finding HTML Elements by CSS Selectors

- If you want to find all HTML elements that match a specified CSS selector (id, class names, types, attributes, values of attributes, etc), use the `querySelectorAll()` method.
- This example returns a list of all `<p>` elements with `class="intro"`.

Example

- `var x = document.querySelectorAll("p.intro");`
-

- This example finds the form element with id="frm1", in the forms collection, and displays all element values:

- Example

- ```
var x = document.forms["frm1"];
var text = "";
var i;
for (i = 0; i < x.length; i++) {
 text += x.elements[i].value + "
";
}
document.getElementById("demo").innerHTML = text;
```

## Changing HTML Style

To change the style of an HTML element, use this syntax:

- `document.getElementById(id).style.property = new style`
- The following example changes the style of a `<p>` element:
- Example
- `<html>`  
`<body>`

```
<p id="p2">Hello World!</p>
```

```
<script>
```

```
document.getElementById("p2").style.color = "blue";
```

```
</script>
```

```
<p>The paragraph above was changed by a script.</p>
```

```
</body>
```

```
</html>
```

Lec.  
JavaScript cont.,

# Features of JavaScript

- JavaScript does have some **special features** that make it such a popular language.
- Javascript is a very handy language to learn today since almost every website uses it. The coolest part of javascript is, you don't need prior knowledge of any programming language. You just need to have some basic knowledge about JavaScript. And for that basic knowledge.
- JavaScript is divided into two main features, they are as follows –
  1. General JavaScript Features
  2. Modern JavaScript Features

# General JavaScript Features:

## **1. Validating User's Input**

- JavaScript is very useful while using forms. It has the capability to validate user input for errors and also saves time. If the user leaves a required field empty or the information is incorrect, JavaScript checks for them before sending the data over to the server.

## **2. Simple Client-side Calculations**

- Since JavaScript is a client-side technology, it can perform basic calculations on the browser. The browser does not need to ask server time for every task. This is especially helpful when a user needs to perform these calculations repeatedly. In these cases, connecting to the server would take a lot more time than performing the actual calculations.

## **3. Greater Control**

- JavaScript provides greater control to the browser rather than being completely dependent on the web servers. JavaScript provides various browsers with additional functionalities that help reduce server load and network traffic.

## **4. Platform Independent**

- Since browsers interpret JavaScript, it solves the problem of compilation and compatibility.

## General JavaScript Features cont,

### 5. Handling Dates and Time

- Unlike other programming languages, JavaScript has built-in functions to determine the date and time. Thus it is very easy to code only by using methods like **.getDate()**.

### 6. Generating HTML Content

JavaScript has very handy features to dynamically generate HTML content for the web. It allows us to add text, links, images, tables, etc after an event occurrence (**eg – mouse click**).

### 7. Detecting the User's Browser and OS

- JavaScript is very capable in the detection of the user's browser and OS information. Though JavaScript runs on every platform, there may occur a situation where we need the user's browser before processing. This can be helpful for writing code that results in different outputs in different browsers.

## 1. Let/Const

- JavaScript has introduced the keywords **'let'** and **'const'** that are available to replace **'var'**. Unlike **'var'**, they are important due to their blocked scope i.e we can only access them in the block we defined them in. Whereas **'var'**, even if we initialize it inside a function, we can access it outside of the function.

## 2. Arrow Functions

- These functions are very useful in simplifying the syntax and tamp down the lines of codes for the web page or web application. Since these are light-weight in syntax, they can be very easily used in anonymous

```
1. <html>
2. <body>
3.
4. <script>
5. function hello(str){ //function declaration
6. return "Hello " + str;
7. }
8. document.write("By a regular function:
");
9. document.write(hello("DataFlair

")); //function call
10.
11. var helloAgain = str => "Hello " + str; //arrow function
12.
13. document.write("By an arrow function:
");
14. document.write(helloAgain("DataFlair")); //function call
15. </script>
16.
17. </body>
```

### 3. Template Literal

This is a common feature in other programming languages that allows you to save variables directly into strings. This proves to be an important tool for developers as it permits them to focus more on the development of the application rather than spending the time on syntax.

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

<h2>JavaScript Template Literals</h2>

<p>Template literals use back-ticks (`) to define a string:</p>

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

<p>Template literals are not supported in Internet Explorer.</p>

<script>
let text = `Hello world!`;
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

#### JavaScript Template Literals

Template literals use back-ticks (`) to define a string:

Hello world!

Template literals are not supported in Internet Explorer.



## 4. New Array Functions

- Though array functions are not necessary for any programming language, they do simplify things for the developer. This also compacts the code and makes it much easier to understand. A regular array and an associative array.

## 5. Default Parameters

- This JavaScript feature helps to avoid collapsing the whole code for a simple mistake. It is very useful when the developer needs to check the working of a function without any parameters.

## 6. Property Shorthand

- Built-in methods like **.get()** are available for the developer's use. These methods help avoid writing the same code every time and cut back on various lines of code. These inborn methods are really supportive of cutting back the developing time and cost.

# Using Events

- The HTML DOM allows you to execute code when an event occurs.
- Events are generated by the browser when "things happen" to HTML elements:

- An element is clicked on
- The page has loaded
- Input fields are changed

example

- This example changes the style of the HTML element with id="id1", when the user clicks button:

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

<button onclick="document.getElementById('demo').innerHTML=Date()">The time is?
</button>

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

</body>
</html>
```

The time is?

The time is?

Fri Jul 15 2022 23:12:30 GMT+0300 (Arabian Standard Time)

## Reacting to Events

- A JavaScript can be executed when an event occurs, like when a user clicks on an HTML element.
- To execute code when a user clicks on an element, add JavaScript code to an HTML event attribute: **onclick=JavaScript**
- When a user clicks the mouse
- When a web page has loaded
- When an image has been loaded
- When the mouse moves over an element
- When an input field is changed
- When an HTML form is submitted
- When a user strokes a key
- Example
- ```
<!DOCTYPE html>
<html>
<body>
<h1 onclick="this.innerHTML = 'Ooops!'">Click on this text!</h1>
</body>
</html>
```

The onload and onunload Events

- The onload and onunload events are triggered when the user enters or leaves the page.
- The onload event can be used to check the visitor's browser type and browser version, and load the proper version of the web page based on the information.
- The onload and onunload events can be used to deal with cookies.
- Example
- `<body onload="checkCookies()">`
- The onmouseover and onmouseout Events
- The onmouseover and onmouseout events can be used to trigger a function when the user mouses over, or out of, an HTML element:

The Browser Object Model (BOM)

- There are no official standards for the **Browser Object Model (BOM)**.
- Since modern browsers have implemented (almost) the same methods and properties for JavaScript interactivity, it is often referred to, as methods and properties of the BOM.

The Window Object

- The window object is supported by all browsers. It represents the browser's window.
- All global JavaScript objects, functions, and variables automatically become members of the window object.
- **Global variables are properties of the window object.**
- **Global functions are methods of the window object.**
- Even the document object (of the HTML DOM) is a property of the window object:

```
    window.document.getElementById("header");
```

Window Size

- Two properties can be used to determine the size of the browser window.
- Both properties return the sizes in pixels:
- **window.innerHeight** - the inner height of the browser window (in pixels)
- **window.innerWidth** - the inner width of the browser window (in pixels)
- The browser window (the browser viewport) is NOT including toolbars and scrollbars.

For Internet Explorer 8, 7, 6, 5:

- document.documentElement.clientHeight
- document.documentElement.clientWidth
- or
- document.body.clientHeight
- document.body.clientWidth

JavaScript Popup Boxes

- JavaScript has three kind of popup boxes: Alert box, Confirm box, and Prompt box.
- **Alert Box:** is often used if you want to make sure information comes through to the user.
- When an alert box pops up, the user will have to click "OK" to proceed.
- Syntax

```
window.alert("sometext");
```

Confirm Box

- A confirm box is often used if you want the user to verify or accept something.
- When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.
- If the user clicks "OK", the box returns true. If the user clicks "Cancel", the box returns false.

Syntax

```
window.confirm("sometext");
```

The `window.confirm()` method can be written without the `window` prefix.

Example

```
if (confirm("Press a button!")) {  
    txt = "You pressed OK!";  
}  
  
else {  
    txt = "You pressed Cancel!";  
}
```


Lec- 9

jQuery

or to use JavaScript on your website.

e many lines of JavaScript code to
t you can call with a single line of code.

things from JavaScript, like AJAX calls and

S:

jQuery Syntax

The jQuery syntax is tailor-made for **selecting** elements. The syntax consists of one or more **selectors** (which select one or more elements) followed by one or more **actions** (which perform an action on the selected elements).

(*action*).*selector* \$Basic syntax is:

- A \$ sign to define/access jQuery
- A (*selector*) to select one or more elements
- An *action* to perform an action on the selected elements

Examples

```
.tnamele tnerruc eht sedih - (edih.(siht))$  
.stnamele <p> lla sedih - (edih.("p"))$  
."tset"=ssalc htiw stnamele lla sedih - (edih.("tset."))$  
."tset"=di htiw tnamele eht sedih - (edih.("tset#"))$
```

jQuery Selectors

jQuery selectors allow to select and manipulate HTML element(s). are used to "find" (or select) HTML elements based on their name, id, classes, types, attributes, values of attributes and much more.

It's based on the existing [CSS Selectors](#), and in addition, it has some own custom selectors.

All selectors in jQuery start with the dollar sign and parentheses: \$().

`$(selector).action();`

for example

`$('p').hide();`

The element Selector

.The jQuery element selector selects elements based on the element name

:You can select all `<p>` elements on a page like this

`("p")$`

Example

:When a user clicks on a button, all `<p>` elements will be hidden

```
$(document).ready(function(){  
  $("button").click(function(){  
    $("p").hide();  
  });  
});
```

Example

```
<!DOCTYPE html>
<html>
<head>
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.
min.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("p").hide();
  });
});
</script>
</head>
<body>

<h2>This is a heading</h2>
<p>This is a paragraph.</p>
<p>This is another paragraph.</p>
<button>Click me to hide paragraphs</button>
</body>
</html>
```

This is a heading

This is a paragraph.

This is another paragraph.

Click me to hide paragraphs

The #id Selector

.tnele ciceps eht dnif ot gat LMTH na fo etubirtta di eht sesu rotceles **id#**The jQuery

.An id should be unique within a page, so you should use the #id selector when you want to find a single, unique element

:To find an element with a specific id, write a hash character, followed by the id of the HTML element

```
$("#test")
```

Example

:When a user clicks on a button, the element with id="test" will be hidden

```
$(document).ready(function(){  
    $("button").click(function(){  
        $("#test").hide();});});
```

The .class Selector

.The jQuery .class selector finds elements with a specific class

\$(".test"):To find elements with a specific class, write a period character, followed by the name of the class

Example

:When a user clicks on a button, the elements with class="test" will be hidden

Example

```
$(document).ready(function(){  
    $("button").click(function(){  
        $(".test").hide();});});
```

Syntax	Description
<code>\$("*")</code>	Selects all elements
<code>\$(this)</code>	Selects the current HTML element
<code>\$("#p.intro")</code>	Selects all <code><p></code> elements with <code>class="intro"</code>
<code>\$("#p:first")</code>	Selects the first <code><p></code> element
<code>\$("#ul li:first")</code>	Selects the first <code></code> element of the first <code></code>
<code>\$("#ul li:first-child")</code>	Selects the first <code></code> element of every <code></code>
<code>\$("[href]")</code>	Selects all elements with an href attribute
<code>\$("#a[target='_blank']")</code>	Selects all <code><a></code> elements with a target attribute value equal to <code>"_blank"</code>
<code>\$("#a[target!='_blank']")</code>	Selects all <code><a></code> elements with a target attribute value NOT equal to <code>"_blank"</code>
<code>\$(":button")</code>	Selects all <code><button></code> elements and <code><input></code> elements of <code>type="button"</code>
<code>\$("#tr:even")</code>	Selects all even <code><tr></code> elements
<code>\$("#tr:odd")</code>	Selects all odd <code><tr></code> elements

Functions In a Separate File

If your website contains a lot of pages, and you want your jQuery functions to be easy to maintain, you can put your jQuery functions in a separate .js file.

When we demonstrate jQuery in this tutorial, the functions are added directly into the `<head>` of the HTML document. This is not ideal for larger websites with many pages, as the functions would be repeated on every page. A better approach is to load the functions from a separate file. This can be done by using the `<script src="">` tag to load an external JavaScript file. The `<script src="">` tag is used to load an external JavaScript file. The `<script src="">` tag is used to load an external JavaScript file.

Example:

```
<head>  
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>  
<script src="my_jquery_functions.js"></script>  
</head>
```