



**We are on a mission to address the digital skills gap for 10 Million+ young professionals, train and empower them to forge a career path into future tech**



# JavaScript Basics

MAY, 2023

## Scripting languages : Introduction

- **Scripting is a language** for writing instructions in a run-time environment. They are **interpreted rather than compiling**.

### Examples:

- JavaScript
- PHP
- Python
- VBScript, etc.,

## Javascript : Introduction

- JavaScript is an **object-based, lightweight** and **interpreted** programming language.
- JavaScript allows you to **add interactivity** to a web page.
- It is **open source** and **cross-platform**.
- JavaScript consists of **three main parts**:
- ECMAScript provides the core functionality.(i.e. ECMA is a standard)
- **The Document Object Model (DOM)** provides interfaces for interacting with elements on web pages
- **The Browser Object Model (BOM)** provides the browser API for interacting with the web browser.

## Javascript : Introduction

- **Client-side JavaScript:**
  - Client-side JavaScript is the most **common form** of the language.
  - For the code to be interpreted by the browser, the **script must be included** in or linked from an HTML document.
  - It means that a web page can now incorporate programmes that interact with the user, control the browser, and dynamically create HTML content, rather than just static HTML.
  - In comparison to typical CGI server-side scripts, the JavaScript client-side method has numerous advantages. **For ex. Use JavaScript to verify that a user has provided a valid email address in a form field.**
  - When the user submits the form, the Script code gets executed, and the form is submitted to the Web Server **iff all of entries are valid as per the script.**



## Javascript : Advantages

- **Less server interaction** – Before transmitting the page to the server, we can validate the user's input. This reduces server traffic, resulting in a lower load on your server.
- **Immediate feedback to the visitors** – They don't have to wait for the page to reload to see if they lost something.
- **Increased interactivity** – We can make interfaces that respond when the user moves their mouse over them or activates them with the keyboard.
- **Richer interfaces** – To provide a Rich Interface to your site users, we can utilise JavaScript to incorporate features like drag-and-drop components and sliders.
- **Simplicity** - JavaScript is easy to understand and learn.

## Javascript : Advantages

- **Interoperability** - We can embed it into any webpage or inside the script of another programming language.
- **Versatility** - JavaScript can now be used for both front-end and back-end development. NodeJS is used for back-end development, whereas AngularJS, ReactJS, and other libraries are used for front-end development.
- **Less Overhead** - By reducing the code length, JavaScript enhances the efficiency of websites and web apps. The usage of numerous built-in methods for loops, DOM access, and other tasks reduces the amount of overhead in the code.

## Javascript : Advantages

- **File reading and writing at the client-side are not supported by JavaScript.** For security reasons, this has been preserved.
- JavaScript doesn't have any **multi-threading** or **multiprocessor** capabilities.
- JavaScript only **allows for a single inheritance, not multiple inheritance.** This object-oriented language feature may be required by some programmes.
- A single code error can **cause the entire JavaScript code on the page to stop rendering.**



# HTML Vs CSS Vs JavaScript



HTML (Structure)



HTML + CSS (Presentation)



HTML + CSS + JavaScript (Functionality)

## Javascript : Syntax

- A JavaScript consists of JavaScript statements that are placed within the **<script>... </script>** HTML tags in a web page.
- The **<script>** element, which contains JavaScript, can be placed anywhere on the page, however it is advised to put it within the **<head>** tags.
- The **<script>** tag alerts the browser program to begin interpreting all the text between these tags as a script.
- **Syntax of JavaScript** will be as follows:

```
<script>  
  
    Javascript Code  
  
</script>
```

## First JavaScript

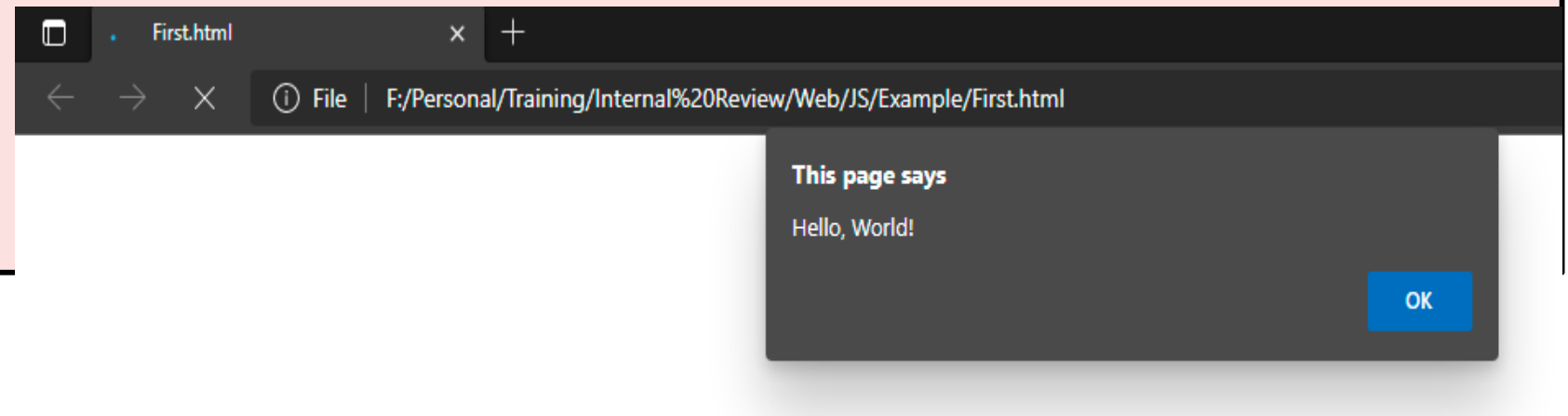
- The script tag takes **two important** attributes:
- **language:** The scripting language you're using is specified by this attribute. The most common value is javascript. The use of this feature has been phased out in current versions of HTML (and its successor, XHTML).
- **type:** The value of this attribute should be set to "**application/javascript**" in order to identify the scripting language in use.

## JavaScript Basics

# First JavaScript

- The JavaScript code in the **<script>** element is interpreted from **top to bottom**.

```
<!DOCTYPE html>
<html>
<head>
<script type="application/javascript">
    alert('Hello, World!');
</script>
</head>
<body>
</body>
</html>
```



## JavaScript Placement in HTML File

- There is a flexibility given to include **JavaScript code anywhere** in an HTML document. But there are following **most preferred ways** to include JavaScript in your HTML file.
- Script in **<head>...</head> section**: If you want to have a **script run on some event**, such as when a user clicks somewhere, then you will place that script in the head.
- Script in **<body>...</body> section**: If you need a **script to run as the page loads** so that the script generates content in the page, the script goes in the <body> portion of the document.
- Script in **<body>...</body> and <head>...</head> sections**.
- Script in and **external file** and then include in <head>...</head> section.

## Javascript : First example program

### JavaScript in External File:

- The script tag provides a mechanism to allow you to **store JavaScript in an external file** and then include it into your HTML files. To use JavaScript from an external file source, you need to write your all JavaScript source code in a simple text **file with extension ".js"** and then include that file as shown below.

```
<!DOCTYPE html>
<html>
<head>
  <script type="text/javascript" src="filename.js" ></script>
</head>
<body>
</body>
</html>
```



## Javascript : First example program

### JavaScript in External File:

- **For example**, you can keep following content in filename.js file and then you can use **alert('Hello, World!')** in your HTML file after including filename.js file:

```
//filename.js  
alert('Hello, World!');
```

```
<!DOCTYPE html>  
<html>  
<head>  
  <script type="text/javascript" src="filename.js" ></script>  
</head>  
<body>  
</body>  
</html>
```

## Basic Elements

- **JavaScript is case-sensitive**
- Everything in JavaScript is case-sensitive, including variables, function names, class names, and operators. It indicates that the counter and Counter variables are not the same.
- **Identifiers**
- An identifier is the name of a variable, function, parameter, or class. An identifier consists of one or more characters in the following format:
- The first character must be a letter (a-z, or A-Z), an underscore(\_), or a dollar sign (\$).
- The other characters can be letters (a-z, A-Z), numbers (0-9), underscores (\_), and dollar signs (\$).

## JavaScript Basics

# Basic Elements

- **Comments**

- JavaScript supports both single-line and block comments.

```
// this is a single-line comment
```

- A single-line comment starts with two forward-slash characters (//), for example:

```
/*  
 * This is a block comment that can * span  
 * multiple lines  
*/
```

- **Statements**

- JavaScript does not require to end a statement with a **semicolon (;)**, it is recommended to always use the semicolon to end a statement.

```
var a = 10;  
var b = 20;
```

## Basic Elements

- We can use a code block that begins with a left curly brace ({) and ends with the right curly brace (}) to combine multiple statements as follows:

```
if( a > b)
{
  console.log('a is greater than b');
  return 1;
}
```

- **Expressions**

- An expression is a piece of code that evaluates to a value.

**For example:**

```
c=2 + 1
```

- The above expression assign 3 to c.

## Basic Elements

- **Keywords & Reserved words**

- JavaScript defines a list of keywords and reserved words that have special uses.
- We cannot use the keywords and reserved words as the identifiers.

abstract	arguments	await	boolean	break	byte	case	catch
char	class	const	continue	debugger	default	delete	do
double	else	enum	eval	export	extends	false	final
finally	float	for	function	goto	if	implements	import
in	instanceof	int	interface	let	long	native	new
null	package	private	protected	public	return	short	static
super	switch	synchronized	this	throw	throws	transient	true
try	typeof	var	void	volatile	while	with	yield

## Basic Elements: Variables

### Variables

Variables are containers for storing data (values).

There are **3 ways** to declare a JavaScript variable:

Using var

Using let

Using const

### Using var

- Creating a variable in JavaScript is called "**declaring**" a **variable**. After the declaration, the variable has no value (technically it has the value of undefined). To assign a value to the variable using **assignment operator**.

```
var message;  
message = "Hello";
```



## Basic Elements: Variables

- You can also assign a value to the variable when you declare it:

```
var message = "Hello";
```

- You can declare many variables in one statement. Start the statement with var and separate the variables by **comma**:

```
var FirstName= "Sachin", LastName = "Tendulkar";
```

- Note : Using var, we can redeclare variables**
- The **general rules** for constructing names for variables (unique identifiers) are:
  - Names can contain letters, digits, underscores, and dollar signs.
  - Names must begin with a letter
  - Names can also begin with \$ and \_ (but we will not use it in this tutorial)
  - Names are case sensitive (y and Y are different variables)
  - Reserved words (like JavaScript keywords) cannot be used as names

## Basic Elements: Variables

### Using let

- The let keyword was introduced in ES6 (2015).

Variables defined with let **cannot be redeclared**.

- Variables defined with let **must be declared before use**.
- Variables defined with let **have block scope**.
- **Example:**

```
let x = "Sachin";  
let x = 0;  
// SyntaxError: 'x' has already been declared
```

- With var we can:

```
var x = "Sachin";  
var x = 0;
```

## Basic Elements: Variables

### Block Scope

- ES6 introduced two important new JavaScript keywords: **let** and **const**.
- These two keywords provide **block scope** in JavaScript.
- Variables declared inside a `{ }` block cannot be accessed from outside the block:

```
{  
  let x = 2;  
}  
  
// x cannot be used here
```

```
{  
  var x = 2;  
}  
  
// x can be used here
```

## Basic Elements: Variables

### Using const

- The const keyword was introduced in **ES6 (2015)**.
- Variables defined with const **cannot be redeclared**.
- Variables defined with const **cannot be reassigned**.

```
const PI = 3.141592653589793;  
PI = 3.14;    // This will give an error  
PI = PI + 10; // This will also give an error
```

```
const PI = 3.141592653589793;
```

**Correct**

```
const PI ;  
PI = 3.141592653589793;
```

**InCorrect**

## Basic Elements: Variables

### Block Scope

- Declaring a variable with `const` is **similar to `let`** when it comes to Block Scope.
- **For Example:** The `x` declared in the block is not the same as the `x` declared outside the block

```
const x = 10; // Here x is 10
{
    const x = 2; // Here x is 2
}
// Here x is 10
```

## Basic Elements : Const

### Redeclaring

- Redeclaring a JavaScript **var** variable is **allowed** anywhere in a program:

```
var x = 2;    // Allowed  
x = 4;       // Allowed
```

- Redeclaring an existing **var** or **let** variable to **const**, in the same scope, is **not allowed**:

```
var x = 2;    // Allowed  
const x = 2;  // Not allowed  
{  
  let x = 2;  // Allowed  
  const x = 2; // Not allowed  
}  
{  
  const x = 2; // Allowed  
  const x = 2; // Not allowed  
}
```



## Basic Elements : Operators

### Arithmetic Operators:

- Arithmetic operators are used to perform arithmetic on numbers:

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation (ES2016)
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

## Basic Elements : Operators

### Assignment Operators:

- Arithmetic operators are used to perform arithmetic on numbers:

Operator	Example	Same As
=	$x = y$	$x = y$
+=	$x += y$	$x = x + y$
-=	$x -= y$	$x = x - y$
*=	$x *= y$	$x = x * y$
/=	$x /= y$	$x = x / y$
%=	$x \% = y$	$x = x \% y$
**=	$x ** = y$	$x = x ** y$

## Basic Elements : Operators

### Comparison Operators:

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

## Basic Elements : Operators

### Logical Operators:

Operator	Description
&&	logical and
	logical or
!	logical not

### Type Operators:

Operator	Description
typeof	Returns the type of a variable
instanceof	Returns true if an object is an instance of an object type

## Basic Elements : Operators

### Bitwise Operators:

Operator	Description	Example	Same as	Result	Decimal
&	AND	5 & 1	0101 & 0001	0001	1
	OR	5   1	0101   0001	0101	5
~	NOT	~ 5	~0101	1010	10
^	XOR	5 ^ 1	0101 ^ 0001	0100	4
<<	Zero fill left shift	5 << 1	0101 << 1	1010	10
>>	Signed right shift	5 >> 1	0101 >> 1	0010	2
>>>	Zero fill right shift	5 >>> 1	0101 >>> 1	0010	2

## Basic Elements : Data Types

- JavaScript variables can hold different data types: **numbers, strings, objects** and more:

```
let length = 16; // Number  
let lastName = "Johnson"; // String  
let x = {firstName:"Rahul", lastName:"Dravid"}; // Object
```

- JavaScript Types are Dynamic**
  - JavaScript has dynamic types. i.e. the same variable can be used to hold different data types:

```
let x; // Now x is undefined  
x = 5; // Now x is a Number  
x = "John"; // Now x is a String
```

## Basic Elements : Data Types

### Strings

- A string (or a text string) is a **series of characters** like "Sachin".
- Strings are written with quotes. **single or double quotes** can be used.

```
let Name1 = "Sachin"; // Using double quotes
```

```
let Name2 = 'Sachin'; // Using single quotes
```

- We can use quotes inside a string, as long as they don't match the quotes surrounding the string:

```
let Name1 = "It's alright"; // Single quote inside double quotes
```

```
let Name2 = "He is called 'Johnny'"; // Single quotes inside double quotes
```

```
let Name3 = 'He is called "Johnny"'; // Double quotes inside single quotes
```

## Basic Elements : Data Types

### Numbers

- Numbers can be written **with, or without** decimals:

```
let x1 = 34.00; // Written with decimals  
let x2 = 34;    // Written without decimals
```

### Booleans

- Booleans can only have two values: **true** or **false**.

```
let x = 5;  
let y = 5;  
let z = 6;  
(x == y) // Returns true  
(x == z) // Returns false
```



## Basic Elements : Data Types

### Arrays

- JavaScript arrays are written with square brackets.
- Array items are separated by commas.
- Array indexes are **zero-based**, which means the first item is [0], second is [1], and so on.

```
const cars = ["Saab", "Volvo", "BMW"];
```

### Objects

- JavaScript objects are written with curly braces {}.
- Object properties are written as **name:value pairs**, separated by commas.

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

## Basic Elements : typeof Operator

- **typeof operator** can be used to find the type of a JavaScript variable.

The **typeof operator** returns the type of a **variable** or **an expression**:

```
typeof ""           // Returns "string"  
typeof "Sachin"     // Returns "string"  
typeof "Sachin Guru" // Returns "string"
```

```
typeof 0            // Returns "number"  
typeof 314          // Returns "number"  
typeof 3.14         // Returns "number"  
typeof (3)          // Returns "number"  
typeof (3 + 4)      // Returns "number"
```

## JavaScript Basics

# Basic Elements : Output

- JavaScript can "display" data in **different ways**:
- Writing into the HTML output using **document.write()**.

Writing into an alert box, using **window.alert()**.

Writing into the browser console, using **console.log()**.

Writing into an HTML element, using **innerHTML**. (**Will Discuss Later**)

## Basic Elements : Output

### Using document.write()

- For testing purposes, it is convenient to use document.write():

```
<!DOCTYPE html>  
<html>  
<body>  
<script>  
document.write(5 + 6);  
</script>  
</body>  
</html>
```

## Basic Elements : Output

### Using window.alert()

- You can use an alert box to display data
- The window object is the global scope object, that means that variables, properties, and methods by default belong to the window object.

```
<!DOCTYPE html>
<html>
<body>
<script>
window.alert(5 + 6);
</script>
</body>
</html>
```

## Basic Elements : Output

### 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>
```

## Basic Elements : Control Flow Statements

### if Statement

```
if (condition) {  
    // block of code to be executed if the condition is true  
}
```

```
<script type="application/javascript">  
var age = 20;  
if( age > 18 ){  
    document.write("<b>Qualifies for driving</b>");  
}  
</script>
```

## Basic Elements : Control Flow Statements

### if-else Statement

```
if (condition) {  
    // block of code to be executed if the condition is true  
} else {  
    // block of code to be executed if the condition is false  
}
```

```
<script type="application/javascript">  
var age = 15;  
if( age > 18 ){  
    document.write("<b>Qualifies for driving</b>");  
}else{  
    document.write("<b>Does not qualify for driving</b>");  
}  
</script>
```



## Basic Elements : Control Flow Statements

**if...else if... statement:**

```
if (condition1) {  
    // block of code to be executed if condition1 is true  
} else if (condition2) {  
    // block of code to be executed if the condition1 is false and condition2 is true  
} else {  
    // block of code to be executed if the condition1 is false and condition2 is false  
}
```

## Basic Elements : Control Flow Statements

if...else if... statement:

```
<script type="application/javascript">
var book = "maths";
if( book == "history" ){
    document.write("<b>History Book</b>");
}else if( book == "maths" ){
    document.write("<b>Maths Book</b>");
}else if( book == "economics" ){
    document.write("<b>Economics Book</b>");
}else{
    document.write("<b>Unknown Book</b>");
}
</script>
```

## Basic Elements : Control Flow Statements

### switch-case Statement

```
switch(expression) {  
  case x:  
    // code block  
    break;  
  case y:  
    // code block  
    break;  
  default:  
    // code block  
}
```

## Basic Elements : Control Flow Statements

### Switch-case Statement

```
<script type="application/javascript">
var grade='A';
switch (grade) {
  case 'A': document.write("Good job<br />");
    break;
  case 'B': document.write("Pretty good<br />");
    break;
  case 'C': document.write("Passed<br />");
    break;
  case 'D': document.write("Not so good<br />");
    break;
  case 'F': document.write("Failed<br />");
    break;
  default: document.write("Unknown grade<br />")
}
</script>
```

## Basic Elements : Control Flow Statements

### while Loop

```
while (condition) {  
    // code block to be executed  
}
```

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

## Basic Elements : Control Flow Statements

### do...while Loop

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

```
<script type="application/javascript">  
var count = 0;  
document.write("Starting Loop" + "<br />");  
do{  
    document.write("Current Count : " + count + "<br />");  
    count++;  
}while (count < 0);  
document.write("Loop stopped!");  
</script>
```

## Basic Elements : Control Flow Statements

### for Loop

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

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

## Basic Elements : Control Flow Statements

### break Statement

```
<script type="application/javascript">
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 /> ");
</script>
```



## Basic Elements : Control Flow Statements

### label break Statement

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

document.write("Entering the loop!<br /> ");

outerloop: // This is the label name
for (var i = 0; i < 5; i++){

    document.write("Outerloop: " + i + "<br />");

    innerloop:
    for (var j = 0; j < 5; j++) {

        if (j > 3 ) break ;      // Quit the innermost loop
        if (i == 2) break outerloop; // Do the same thing
        if (i == 4) break outerloop; // Quit the outer loop
        document.write("Innerloop: " + j + " <br />");

    }

}

document.write("Exiting the loop!<br /> ");

</script>
```

## Basic Elements : Control Flow Statements

### continue Statement

```
<script type="application/javascript">  
  
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 /> ");  
  
</script>
```

## Basic Elements : Control Flow Statements

### label continue Statement

```
<script type="application/javascript">
document.write("Entering the loop!<br /> ");
outerloop: // This is the label name
for (var i = 0; i < 3; i++){
    document.write("Outerloop: " + i + "<br />");
    for (var j = 0; j < 5; j++) {
        if (j == 3){
            continue outerloop;
        }
        document.write("Innerloop: " + j + "<br />");
    }
}
document.write("Exiting the loop!<br /> ");
</script>
```

## Basic Elements : 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).
- **Syntax:**

```
function functionName(parameter-list){  
    statements  
}
```

- A JavaScript function is defined with the **function keyword**, followed by a name, followed by parentheses ().
- Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).
- The parentheses may include **parameter names separated by commas**: (parameter1, parameter2, ...)
- The code to be executed, by the function, is placed inside curly brackets: {}

## Basic Elements : Functions

### Called Function

```
<script type="application/javascript">  
function sayHello(){  
    alert("Hello there");  
}</script>
```

### Calling Function

```
<script type="application/javascript">  
sayHello();  
</script>
```

## Basic Elements : DialogBoxes

### Confirmation Dialog Box:

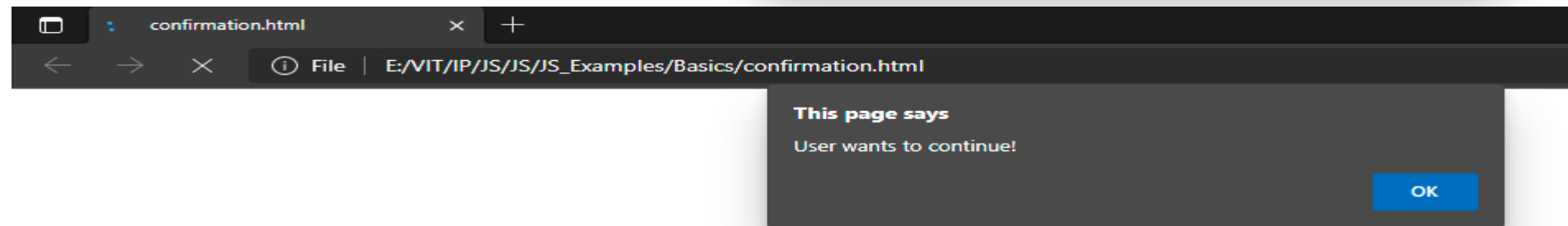
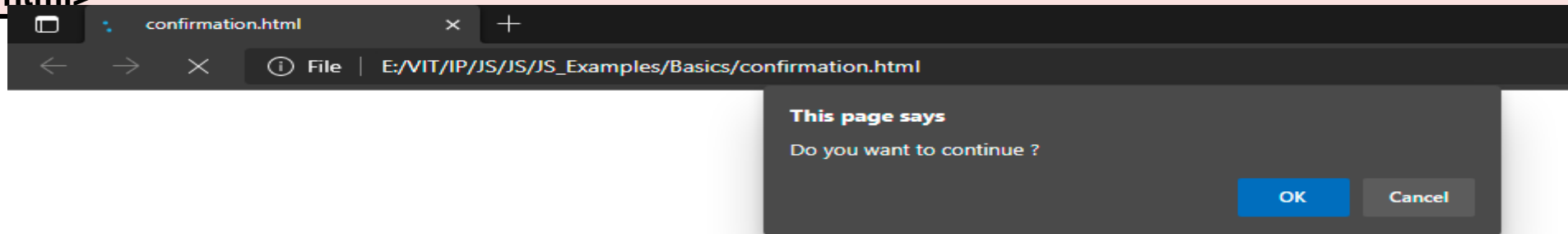
- A confirmation dialog box is mostly used to take user's consent on any option. It displays a dialog box with two buttons: **OK** and **Cancel**.
- If the user clicks on **OK** button the window method **confirm()** will return true.

```
<html>
<head>
<script>
  function myConfirmbox(){
    var retVal = confirm("Do you want to continue ?");
    if( retVal == true ){
      alert("User wants to continue!");
    }
    else{
      alert("User does not want to continue!");
    }
  }
</script>
</head>
```

## Basic Elements : DialogBoxes

### Confirmation Dialog Box:

```
<body>
<script>
    myConfirmbox();
</script>
</body>
</html>
```



## Basic Elements : DialogBoxes

### Prompt Dialog Box:

- The prompt dialog box is very useful when you want to pop-up a text box to get user input.
- This dialog box with two buttons: **OK** and **Cancel**.
- The window method **prompt()** returns the entered value when the user clicks on **OK** button.

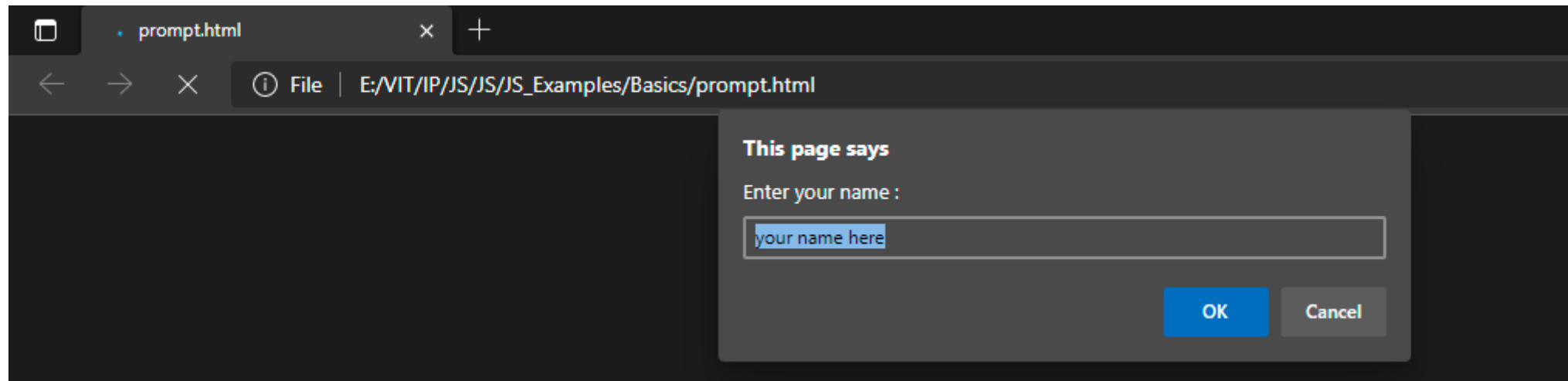
```
<html>
<head>
<script>
function myPrompt(){
    var retVal = prompt("Enter your name : ", "your name here");
    alert("You have entered : " + retVal );
}
</script>
</head>
```



## Basic Elements : DialogBoxes

### Prompt Dialog Box:

```
<body>  
<script>  
  myPrompt();  
</script>  
</body></html>
```



## Basic Elements : DialogBoxes

### Prompt Dialog Box:

- The prompt dialog box is very useful when you want to pop-up a text box to get user input.

This dialog box with two buttons: **OK** and **Cancel**.

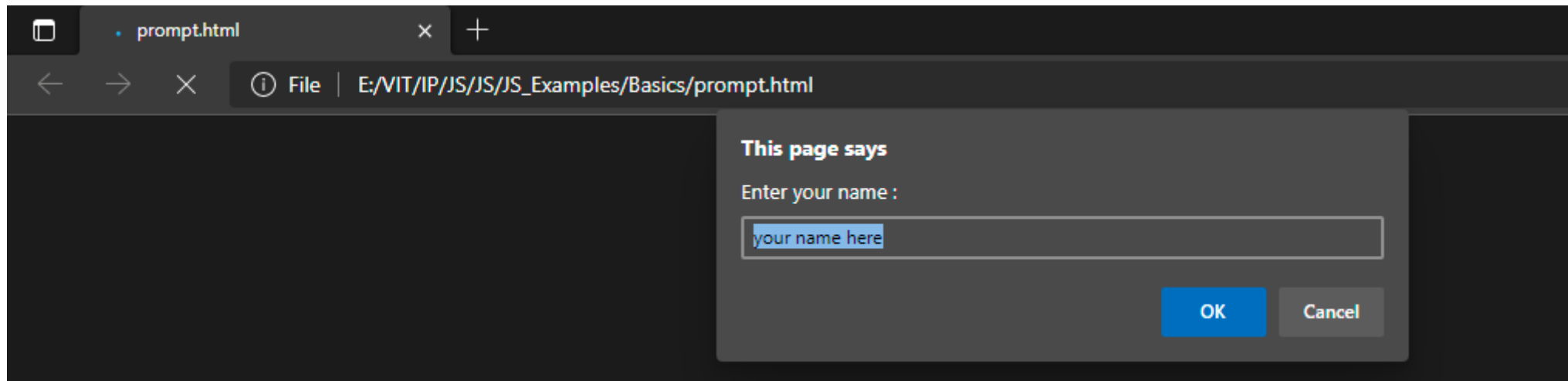
- The window method **prompt()** returns the entered value when the user clicks on **OK** button.

```
<html>
<head>
<script>
function myPrompt(){
    var retVal = prompt("Enter your name : ", "your name here");
    alert("You have entered : " + retVal );
}
</script>
</head>
```

## Basic Elements : DialogBoxes

### Prompt Dialog Box:

```
<body>  
<script>  
  myPrompt();  
</script>  
</body></html>
```



# JavaScript Objects



## JavaScript Objects

# Introduction

- **"Everything"** in JavaScript is an Object. Even **primitive data types** (except null and undefined) can be treated as objects.
  - Booleans can be objects (or primitive data treated as objects)
  - Numbers can be objects (or primitive data treated as objects)
    - Strings are also objects (or primitive data treated as objects)
  - Dates are always objects
  - Maths and Regular Expressions are always objects
    - Arrays are always objects

## Object Properties

- Properties are the values associated with an object.

### Syntax:

- `objectName.propertyName`
- `objectName["propertyName"]`

## JavaScript Objects

# Introduction

### Object Definition

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

```
const person = {firstName:"Sachin", lastName:"Tendulkar", age:50};
```

### Object Properties:

Property	Property Value
firstName	Sachin
lastName	Tendulkar
age	50

### Example:

```
person.lastName; // Tendulkar
```

Or

```
person["lastName"];
```

## JavaScript Objects

# Introduction

### Object Methods

- Objects can also have methods.
- Methods are actions that can be performed on objects.

Methods are stored in properties as function definitions.

#### Accessing Object Methods:

- You access an object method with the following syntax:

```
objectName.methodName()
```

## Introduction

### Example:

```
person = {firstName: "Sachin", lastName : "Tendulkar", fullName : function() {  
    return this.firstName + " " + this.lastName;}  
};
```

OR

```
person = {firstName:"Sachin", lastName:"Tendulkar", age:50};  
person.fullName=function(){  
    return person.firstName+ "+"+person.lastName;  
}
```

OR

```
person = {firstName:"Sachin", lastName:"Tendulkar", age:50};  
person.fullName=function(){  
    return this.firstName+ "+"+this.lastName;  
}
```

**`document.write("The person full name is: " + person.fullName());`**



## JavaScript Objects

# Strings

- A JavaScript string is **zero or more characters** written inside quotes.
- Normally, JavaScript strings are **primitive values**, created from literals:

**Example:** `let firstName = "Rahul";`

- But strings can also be **defined as objects** with the **keyword new**:

**Example:** `let firstName = new String("Rahul");`

## Strings

- **Comparison of == and === operator**

- == operator compares two string values.

```
let x = "Sachin";  
let y = new String("Sachin");  
  
// (x == y) is true because x and y have equal values
```

- === operator checks equality in **both data type and value but not with objects.**

```
let x = "Sachin";  
let y = new String("Sachin");  
  
// (x === y) is false because x and y have different types (string and object)
```

```
let x = new String("John");  
let y = new String("John");  
  
// (x == y) is false because x and y are objects
```

```
let x = new String("John");  
let y = new String("John");  
  
// (x === y) is false because x and y are objects
```

## JavaScript Objects

# Strings

- **String Length:**
- To find the length of a string, use the **built-in length property**.

```
let text = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";  
text.length; // Will return 26
```

# String Methods

## Possible String Methods:

Method	Description
<b>charAt()</b>	Returns the character at the specified index.
<b>concat()</b>	Combines the text of two strings and returns a new string.
<b>indexOf()</b>	Returns the index within the calling String object of the first occurrence of the specified value, or -1 if not found.
<b>lastIndexOf()</b>	Returns the index within the calling String object of the last occurrence of the specified value, or -1 if not found.
<b>match()</b>	Used to match a regular expression against a string.
<b>replace()</b>	Used to find a match between a regular expression and a string, and to replace the matched substring with a new substring.
<b>search()</b>	Executes the search for a match between a regular expression and a specified string.
<b>slice()</b>	Extracts a section of a string and returns a new string.
<b>split()</b>	Splits a String object into an array of strings by separating the string into substrings.

## String Methods

### Possible String Methods:

<b>substr()</b>	Returns the characters in a string beginning at the specified location through the specified number of characters.
<b>substring()</b>	Returns the characters in a string between two indexes into the string.
<b>toLocaleLowerCase()</b>	The characters within a string are converted to lower case while respecting the current locale.
<b>toLocaleUpperCase()</b>	The characters within a string are converted to upper case while respecting the current locale.
<b>toLowerCase()</b>	Returns the calling string value converted to lower case.
<b>toString()</b>	Returns a string representing the specified object.
<b>toUpperCase()</b>	Returns the calling string value converted to uppercase.
<b>valueOf()</b>	Returns the primitive value of the specified object.

## String Methods: Example #1

### The slice() Method

- **slice()** extracts a part of a string and returns the extracted part in a new string.
- The method takes **2 parameters**: the start position, and the end position (end not included).

```
let str = "Welcome to Javascript World";  
str.slice(11, 21) // Returns Javascript
```

```
let str = "Welcome to Javascript World";  
str.slice(-16, -6) // Returns Javascript
```

```
let str = "Welcome to Javascript World";  
str.slice(11); // Returns Javascript World
```

```
let str = "Welcome to Javascript World";  
str.slice(-16); // Returns Javascript World
```

## String Methods: Example #2

### The substring() Method

- substring() is similar to slice().
- If start > stop, then function swaps both arguments.
- If any argument is negative or is NaN, it is treated as 0.

```
let str = "Welcome to Javascript World";  
str.substring(11, 21) // Returns Javascript
```

```
let str = "Welcome to Javascript World";  
str.substring(11); // Returns Javascript World
```

## String Methods: Example #3

- **The substr() Method**

- substr() is similar to slice().
- The difference is that the second parameter specifies the length of the extracted part.

```
let str = "Welcome to Javascript World";  
str.substr(11, 10) // Returns Javascript
```

- If you omit the second parameter, substr() will slice out the rest of the string.

```
let str = "Welcome to Javascript World";  
str.substr(11); // Returns Javascript World
```

```
let str = "Welcome to Javascript World";  
str.substr(-5); // Returns World
```



## String Methods: Example #4

- **Replacing String**

- The `replace()` method replaces a specified value with another value in a string:

```
let text = "Welcome to Nepal";  
let newText = text.replace("Nepal", "India"); // Returns Welcome to India
```

- The `replace()` method does not change the string it is called on. It returns a new string.
- By default, the `replace()` method replaces only the first match:

```
let text = "Welcome to Nepal and Nepal Country";  
let newText = text.replace("Nepal", "India");  
// Returns Welcome to India and Nepal Country
```

## String Methods

- To replace **case insensitive**, use a regular expression with an **/i flag** (insensitive).
- To replace all matches, use a regular expression with a **/g flag** (global match):

```
let text = "Welcome to Nepal";  
let newText = text.replace("Nepal", "India"); // Returns Welcome to India
```

```
let text = "Welcome to Nepal";  
let newText = text.replace("/NEPAL/i", "India"); // Returns Welcome to India
```

```
let text = "Welcome to Nepal and Nepal Country";  
let newText = text.replace("/Nepal/g", "India");  
// Returns Welcome to India and India Country
```

## JavaScript Objects

# Number Methods

- The Number object represents numerical data, either **integers** or **floating-point numbers**.

```
let x = 3.14; // A number with decimals  
let y = 3;    // A number without decimals
```

Method	Description
<code>constructor()</code>	Returns the function that created this object's instance. By default this is the Number object.
<code>toExponential()</code>	Forces a number to display in exponential notation, even if the number is in the range in which JavaScript normally uses standard notation.
<code>toFixed()</code>	Formats a number with a specific number of digits to the right of the decimal.
<code>toLocaleString()</code>	Returns a string value version of the current number in a format that may vary according to a browser's locale settings.
<code>toPrecision()</code>	Defines how many total digits (including digits to the left and right of the decimal) to display of a number.
<code>toString()</code>	Returns the string representation of the number's value.
<code>valueOf()</code>	Returns the number's value.

## Number Methods: Example

### The toString() Method:

- The toString() method returns a **number as a string**.
- All number methods can be used on any type of numbers (literals, variables, or expressions):

```
let x = 123; (OR)
let x = new Number(123);
x.toString();           // returns 123 from variable x
(123).toString();       // returns 123 from literal 123
(100 + 23).toString();  // returns 123 from expression 100 + 23
```

## Regular Expressions and RegExp Object

- A regular expression is a **sequence of characters** that forms a search pattern.
- The search pattern can be used for **text search** and **text replace operations**.
- When you search for data in a text, you can use this search pattern to **describe what you are searching for**.
- A regular expression can be a **single character**, or a **more complicated pattern**.
- Regular expressions can be used to perform all types of **text search** and **text replace operations**.

## Regular Expressions and RegExp Object

### Syntax:

```
var pattern = new RegExp(pattern, attributes);  
      (OR)  
var pattern = /pattern/attributes;
```

Here,

- **pattern**: A string that specifies the **pattern of the regular expression**.
- **attributes**: An optional string containing any of the "**g**", "**i**", and "**m**" attributes that specify **global, case-insensitive, and multiline matches**, respectively.

## Regular Expressions: Quantifiers

- Possible quantifiers

Expression	Description
[...]	Any one character between the brackets.
[^...]	Any one character not between the brackets.
[0-9]	It matches any decimal digit from 0 through 9.
[a-z]	It matches any character from lowercase a through lowercase z.
[A-Z]	It matches any character from uppercase A through uppercase Z.
[a-Z]	It matches any character from lowercase a through uppercase Z.
[aeiou]	matches a single character in the given set
[^aeiou]	matches a single character outside the given set
(foo bar baz)	matches any of the alternatives specified

## Regular Expressions: Quantifiers

- Possible quantifiers

Expression	Description
<code>p+</code>	It matches any string containing at least one p.
<code>p*</code>	It matches any string containing zero or more p's.
<code>p?</code>	It matches any string containing one or more p's.
<code>p{N}</code>	It matches any string containing a sequence of <b>N</b> p's
<code>p{2,3}</code>	It matches any string containing a sequence of two or three p's.
<code>p{2, }</code>	It matches any string containing a sequence of at least two p's.
<code>p\$</code>	It matches any string with p at the end of it.
<code>^p</code>	It matches any string with p at the beginning of it.
<code>p+</code>	It matches any string containing at least one p.



## RegExp Object Methods

- **Possible Methods**

Method	Description
<code>exec()</code>	Executes a search for a match in its string parameter. If it finds a match, it returns the matched text ; otherwise, it returns null.
<code>test()</code>	Tests for a match in its string parameter. This method returns true if it finds a match, otherwise it returns false.
<code>toSource()</code>	Returns an object literal representing the specified object; you can use this value to create a new object.
<code>toString()</code>	Returns a string representing the specified object.

## RegExp Object Methods: Example

- The **exec method** searches string for text that matches regexp. If it finds a match, it returns the **matched text** ; otherwise, it returns **null**.

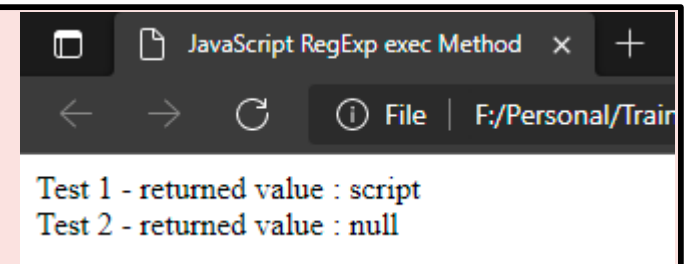
### Syntax:

**RegExpObject.exec( string );** //string : The string to be searched

```
<script type="text/javascript">
  var str = "Javascript is an interesting scripting language";
  var re = new RegExp( "script", "g" );

  var result = re.exec(str);
  document.write("Test 1 - returned value : " + result);

  re = new RegExp( "pushing", "g" );
  var result = re.exec(str);
  document.write("<br />Test 2 - returned value : " + result);
</script>
```



# JavaScript Events



## Introduction

- **JavaScript's interaction** with **HTML** is handled through **events** that occur when the user or browser manipulates a page.
- 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.
- **Examples of HTML events:**
  - 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

## JavaScript Events

# Introduction

### • Possible Events

Event	Description
<code>onchange</code>	Script runs when the element changes
<code>onsubmit</code>	Script runs when the form is submitted
<code>onreset</code>	Script runs when the form is reset
<code>onselect</code>	Script runs when the element is selected
<code>onblur</code>	Script runs when the element loses focus
<code>onfocus</code>	Script runs when the element gets focus
<code>onkeydown</code>	Script runs when key is pressed
<code>onkeypress</code>	Script runs when key is pressed and released
<code>onkeyup</code>	Script runs when key is released
<code>onclick</code>	Script runs when a mouse click
<code>ondblclick</code>	Script runs when a mouse double-click

## JavaScript Events

# Introduction

### • Possible Events

Event	Description
<code>onmousedown</code>	Script runs when mouse button is pressed
<code>onmousemove</code>	Script runs when mouse pointer moves
<code>onmouseout</code>	Script runs when mouse pointer moves out of an element
<code>onmouseover</code>	Script runs when mouse pointer moves over an element
<code>onmouseup</code>	Script runs when mouse button is released
<code>onload</code>	The browser has finished loading the page
<code>onkeydown</code>	The user pushes a keyboard key

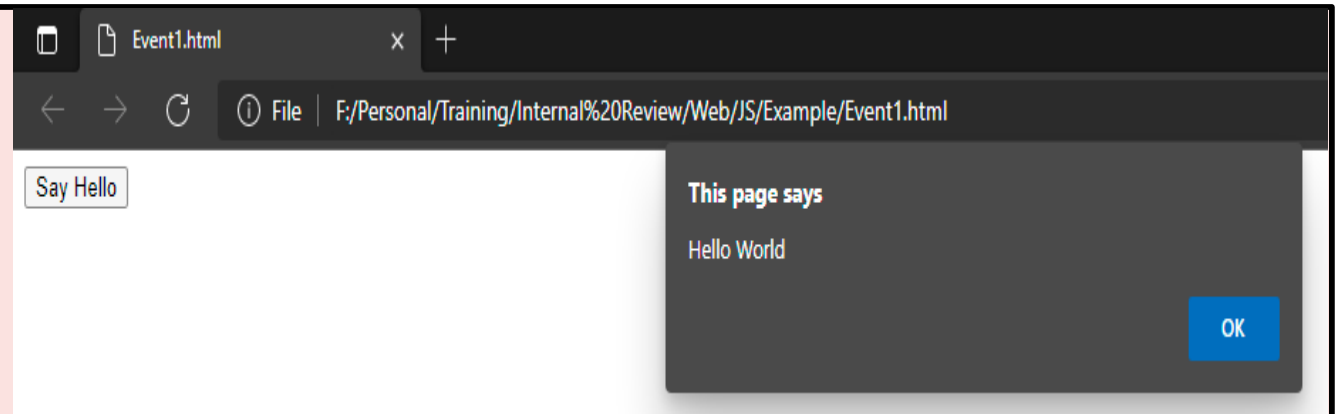
## JavaScript Events

### Example: #1

#### onclick Event Type:

- This is the most frequently used event type which occurs **when a user clicks mouse left button**. You can put your validation, warning etc against this event type.

```
<html>
<head>
<script type="text/javascript">
function sayHello() {
    alert("Hello World")
}
</script>
</head>
<body>
<input type="button" onclick="sayHello()" value="Say Hello" />
</body>
</html>
```



## JavaScript Events

### Example: #2

- The onmouseover and onmouseout events can be used to trigger a function when the user mouses over, or out of, an HTML element:

```
<html>
<head>
<script type="text/javascript">
function over() { alert("Mouse Over"); }
function out() { alert("Mouse Out"); }
</script>
</head>
<body>
<p>Bring your mouse inside the division to see the result:</p>
<div onmouseover="over()" onmouseout="out()">
<h2> This is inside the division </h2>
</div>
</body>
</html>
```



# Document Object Model (DOM)



## DOM(Document Object Model)

### Introduction

- The DOM is a W3C (World Wide Web Consortium) standard.
- It defines a standard for accessing documents:

"The W3C **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 W3C DOM standard is separated into **3 different parts**:
  - **Core DOM** - standard model for all document types
  - **XML DOM** - standard model for XML documents
  - **HTML DOM** - standard model for HTML documents

## DOM(Document Object Model)

# Introduction

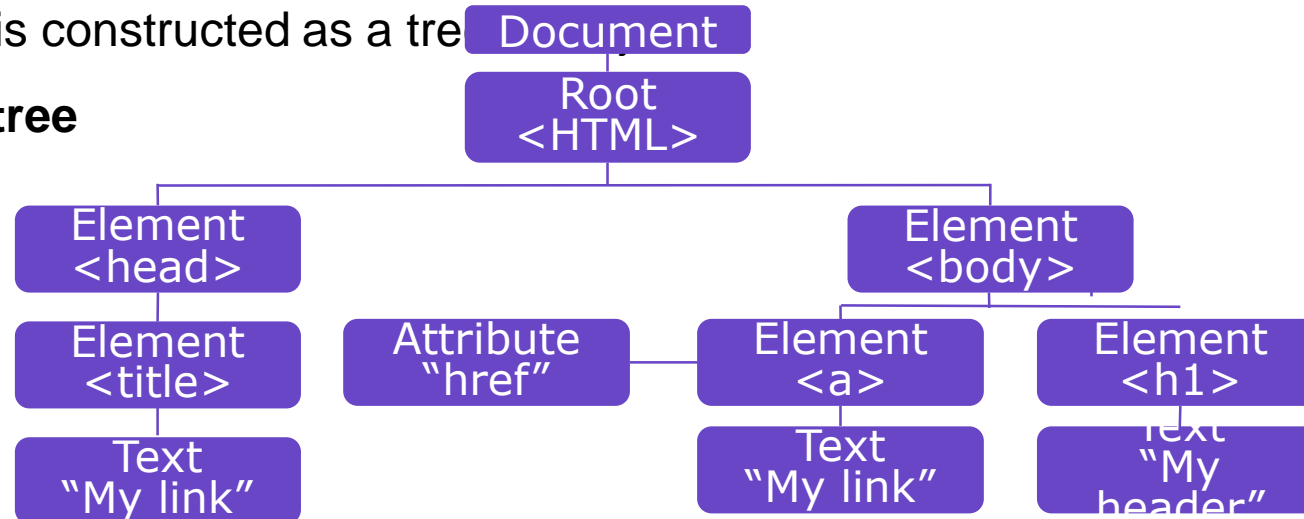
- The HTML DOM is a standard object model and **programming interface for HTML**.
- It **defines**:

- The HTML elements as **objects**
- The **properties** of all HTML elements
- The **methods** to access all HTML elements
- The **events** for all HTML elements

### Note:

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 model is constructed as a tree
- **For Ex: sample tree**



## DOM(Document Object Model)

# HTML DOM Methods and Property

- 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 innerHTML Property

- The innerHTML property is the simplest way to **access the content of an element**. The innerHTML property can be used to **access or replace** HTML element content.
- Any element's HTML content, including **<html>** and **<body>**, can be **retrieved or changed** using the innerHTML property.

## DOM(Document Object Model)

# HTML DOM Methods

- In the **HTML DOM object model**, the document object **represents a web page**.
- The document object is the **owner of all other objects** in a web page.
- Always start with accessing the document object to **access objects** in an HTML page.

## Finding HTML Elements

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

## DOM(Document Object Model)

# HTML DOM Methods

### Changing HTML Elements

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

Method	Description
<code>document.createElement()</code>	Create an HTML element
<code>document.removeChild()</code>	Remove an HTML element
<code>document.appendChild()</code>	Add an HTML element
<code>document.replaceChild()</code>	Replace an HTML element
<code>document.write(<i>text</i>)</code>	Write into the HTML output stream

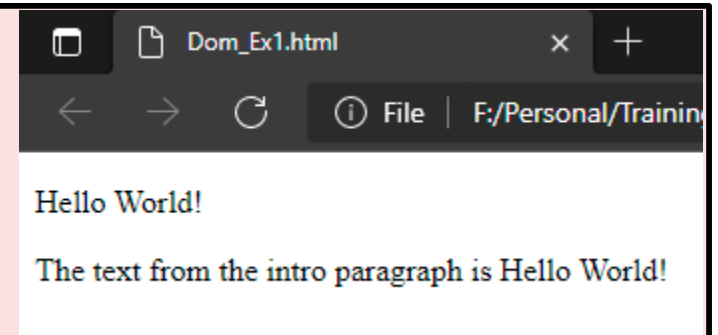
## DOM(Document Object Model)

# HTML DOM Methods

### Finding HTML Elements by Id

- The easiest way to find HTML elements in the DOM is by using the **element id**.

```
<!DOCTYPE html>
<html>
<body>
<p id="intro">Hello World!</p>
<p id="demo"></p>
<script>
myElement = document.getElementById("intro");
document.getElementById("demo").innerHTML =
"The text from the intro paragraph is " + myElement.innerHTML;
</script>
</body>
</html>
```



## DOM(Document Object Model)

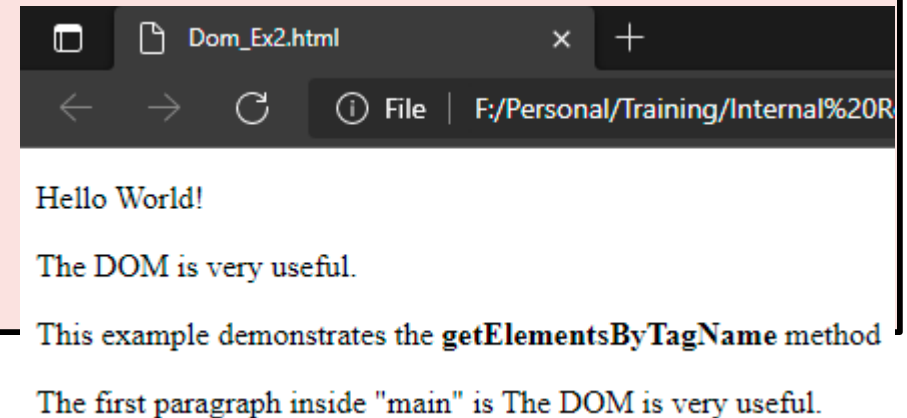
# HTML DOM Methods

## Finding HTML Elements by Tag Name

- This example finds the element with id="main", and then finds all <p> elements inside "main":

```
<!DOCTYPE html>
<html>
<body>
<p>Hello World!</p>
<div id="main">
<p>The DOM is very useful.</p>
<p>This example demonstrates the
  <b>getElementsByTagName</b> method</p>
</div>
<p id="demo"></p>
```

```
<script>
var x = document.getElementById("main");
var y = x.getElementsByTagName("p");
document.getElementById("demo").innerHTML =
'The first paragraph inside "main" is' + y[0].innerHTML;
</script>
</body>
</html>
```





## DOM(Document Object Model)

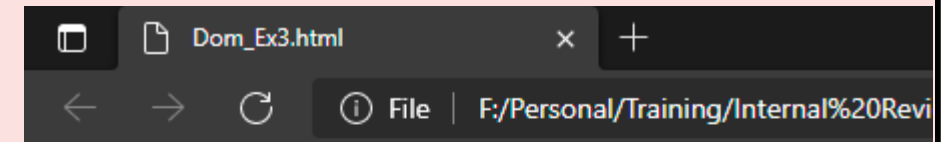
# HTML DOM Methods

### Finding HTML Elements by Class Name

- This example returns a list of all elements with class="intro".

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript HTML DOM</h2>
<p>Finding HTML Elements by Class Name.</p>
<p class="intro">Hello World!</p>
<p class="intro">This example demonstrates the
  <b>getElementsByClassName</b> method.</p>
<p id="demo"></p>
```

```
<script>
const x = document.getElementsByClassName("intro");
document.getElementById("demo").innerHTML =
'The first paragraph (index 0) with class="intro" is: ' + x[0].innerHTML;
</script>
</body>
</html>
```



## JavaScript HTML DOM

Finding HTML Elements by Class Name.

Hello World!

This example demonstrates the **getElementsByClassName** method.

The first paragraph (index 0) with class="intro" is: Hello World!

## DOM(Document Object Model)

# HTML DOM Events

### Changing HTML Elements

Method	Description
<code>document.getElementById(<i>id</i>).onclick=function(){<i>code</i>}</code>	Adding event handler code to an onclick event

## DOM(Document Object Model)

# HTML DOM Events : Assign Events

- A function named **displayDate()** is assigned to an HTML element with the **id="myBtn"**.

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<p>Click "Try it" to execute the displayDate() function.</p>
<button id="myBtn">Try it</button>

<p id="demo"></p>
<script>
function displayDate() {
    document.getElementById("demo").innerHTML = Date();
}
document.getElementById("myBtn").onclick = function(){displayDate()};
</script>
</body>
</html>
```

## DOM(Document Object Model)

# HTML DOM Events: onchange Event

- The onchange event occurs when the **value of an element has been changed**. For radiobuttons and checkboxes, the onchange event occurs when the checked state has been changed.
- The upperCase() function will be called when a user changes the content of an input field.

```
<html>
<head>
<script>
function upperCase(){
    var x = document.getElementById("fname");
    x.value = x.value.toUpperCase();
}
</script>
</head>
```

## DOM(Document Object Model)

# HTML DOM Events: onchange Event

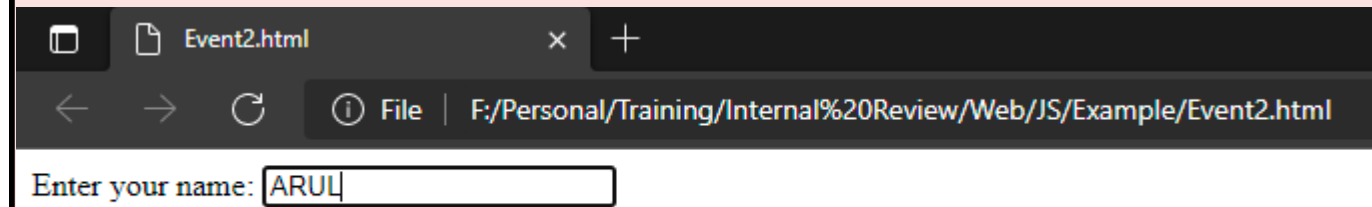
```
<body>
```

```
Enter your name: <input type="text" id="fname" onchange="upperCase()">
```

```
<p>When you leave the input field, a function is triggered which transforms the input text to upper case.</p>
```

```
</body>
```

```
</html>
```



When you leave the input field, a function is triggered which transforms the input text to upper case.

## HTML DOM Events : The onfocus and Onblur Events

```
<!DOCTYPE html>
<html>
<body>
<p>When you enter the input field, background color changes to yellow. When you leave the input field, the
background color changes to red.</p>
Enter your name: <input type="text" id="myInput" onfocus="focusFunction()" onblur="blurFunction()">
<script>

// Focus = Changes the background color of input to yellow
function focusFunction() {
    document.getElementById("myInput").style.background = "yellow";
}
// No focus = Changes the background color of input to red
function blurFunction() {
    document.getElementById("myInput").style.background = "red";
}
</script>
</body>
</html>
```

## DOM(Document Object Model)

# HTML DOM Event Listener

- The **addEventListener()** method attaches an event handler to the specified element.
- The **addEventListener()** method attaches an event handler to an element without overwriting existing event handlers.

```
<!DOCTYPE html>
<html>
<body>
<button id="myBtn">Try it</button>
<script>
document.getElementById("myBtn").addEventListener("click", myFunction);
function myFunction() {
alert ("Hello World!");
}
</script>
</body>
</html>
```

## DOM(Document Object Model)

# HTML DOM Event Listener

- The **addEventListener()** method allows you to add many events to the same element.
- We can add events of different types to the same element:

```
<!DOCTYPE html>
<html>
<body>
<button id="myBtn">Try it</button>

<p id="demo"></p>
<script>
var x = document.getElementById("myBtn");
x.addEventListener("mouseover", myFunction);
x.addEventListener("click", mySecondFunction);
x.addEventListener("mouseout", myThirdFunction);
```

```
function myFunction() {
document.getElementById("demo").innerHTML += "Moused
over!<br>"; }
function mySecondFunction() {
document.getElementById("demo").innerHTML +=
"Clicked!<br>"; }
function myThirdFunction() {
document.getElementById("demo").innerHTML += "Moused
out!<br>";
}
</script>
</body>
</html>
```



## HTML DOM Event Listener : Passing parameters

```
<!DOCTYPE html>
<html>
<body>
<p>5 and 7 are two arguments passed to function that performs a Multiplication.</p>
<button id="myBtn">Try it</button>
<p id="demo"></p>

<script>
let p1 = 5;
let p2 = 7;
document.getElementById("myBtn").addEventListener("click", function() {
  myFunction(p1, p2);});
function myFunction(a, b) {
  document.getElementById("demo").innerHTML = a * b;
}
</script>
</body>
</html>
```

## DOM(Document Object Model)

## HTML DOM Event Listener : Bubbling

```
<!DOCTYPE html>
<html>
<head>
<style>
#myDiv1 {
background-color: coral;
padding: 50px;
}
#myP1 {
background-color: white;
font-size: 20px;
border: 1px solid;
padding: 20px;
}
</style>
</head>
```

```
<body>
<div id="myDiv1">
<h2>Bubbling:</h2>
<p id="myP1">Click me!</p>
</div><br>
<script>
document.getElementById("myP1").addEventListener("click", function() {
alert("You clicked the white element!"); });
document.getElementById("myDiv1").addEventListener("click", function() {
alert("You clicked the orange element!"); });
</script>
</body>
</html>
```

Event bubbling is **a method of event propagation** in the HTML DOM API when an event is in an element inside another element, and both elements have registered a handle to that event.

## DOM(Document Object Model)

# HTML DOM Collections

- The **getElementsByTagName()** method returns an HTMLCollection object.
- An HTMLCollection object is an **array-like list (collection)** of HTML elements.
- The following code selects all **<p>** elements in a document.
- The **length** property defines the **number of elements in an HTMLCollection**.

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript HTML DOM</h2>
<p>Hello World!</p>
<p>Hello Norway!</p>
<p id="demo"></p>
<p id="demo1"></p>
<button onclick="myFunction()">Try it</button>
```

## DOM(Document Object Model)

# HTML DOM Collections

```
<script>
const myCollection = document.getElementsByTagName("p");
document.getElementById("demo").innerHTML = "This document contains " + myCollection.length + "
  paragraphs.";
document.getElementById("demo1").innerHTML = "The innerHTML of the second paragraph is: " +
  myCollection[1].innerHTML;
function myFunction() {
  const myCollection = document.getElementsByTagName("p");
  for (let i = 0; i < myCollection.length; i++) {
    myCollection[i].style.color = "red";
  }
}
</script>
</body>
</html>
```

## DOM(Document Object Model)

# HTML DOM Collections

## Finding HTML Elements by HTML Object Collections

```
<!DOCTYPE html>
<html>
<body>
<form id="frm1" action="">
First name: <input type="text" name="fname"
  value="Sachin"><br>
Last name: <input type="text" name="lname"
  value="Tendulkar"><br><br>
<input type="submit" value="Submit">
</form>
```

```
<p>Click "Display" to display the value of each element in the
  form.</p>
<button onclick="myFunction()">Display</button>
<p id="demo"></p>
<script>
function myFunction() {
  var x = document.getElementById("frm1");
  var text = "";
  var i;
  for (i = 0; i < x.length ;i++) {
    text += x.elements[i].value + "<br>";
  }
  document.getElementById("demo").innerHTML = text;
}
</script>
</body>
</html>
```

## DOM(Document Object Model)

# The DOM CSS

## Changing HTML Style

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

```
document.getElementById(id).style.property = new style
```

## 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

**My Heading 1**

Click Me!

**My Heading 1**

Click Me!

```
<!DOCTYPE html>
<html>
<body>
<h1 id="id1">My Heading 1</h1>
<button type="button" onclick="document.getElementById('id1').style.color =
'red'">
Click Me!</button>
</body>
</html>
```

## DOM(Document Object Model)

# The DOM CSS

```
<style>
.imgbox {
  float: left;
  text-align: center;
  width: 120px;
  height: 145px;
  border: 1px solid gray;
  margin: 4px;
  padding: 0;
}

.thumbnail {
  width: 110px;
  height: 90px;
  margin: 3px;
}
```

```
.box {
  width: 110px;
  padding: 0;
}
</style>
```

## The DOM CSS

```
<div style="text-align:center">
  <div style="width:394px;height:160px;margin-left:auto;margin-right:auto;text-align:left;border:1px solid gray;">

    <div class="imgbox" id="imgbox1">Box 1<br>
      
      <input class="box" type="button" onclick="removeElement()" value="Remove">
    </div>

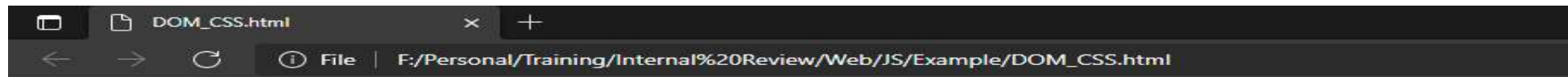
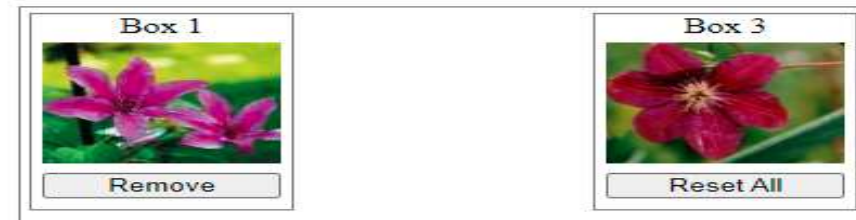
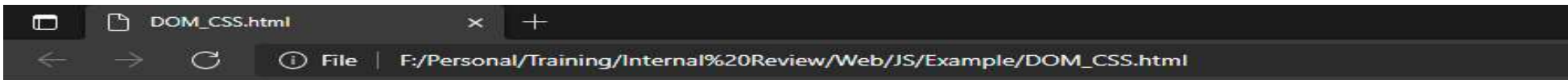
    <div class="imgbox" id="imgbox2">Box 2<br>
      
      <input class="box" type="button" onclick="changeVisibility()" value="Hide">
    </div>

    <div class="imgbox">Box 3<br>
      
      <input class="box" type="button" onclick="resetElement()" value="Reset All">
    </div>
  </div>
</div>
```



# DOM(Document Object Model)

## The DOM CSS



## DOM(Document Object Model)

# The DOM Forms

### Data Validation

- Data validation is the process of ensuring that user input is clean, correct, and useful.

#### Typical validation tasks are:

- Has the user filled in all required fields?
- Has the user entered a valid date?
  - Has the user entered text in a numeric field?
- The purpose of data validation is to ensure correct user input.
- Validation can be defined by many different methods, and deployed in many different ways.
- **Server side validation** is performed by a web server, after input has been sent to the server.
- **Client side validation** is performed by a web browser, before input is sent to a web server.

## DOM(Document Object Model)

# The DOM Forms

## HTML Constraint Validation

HTML constraint validation is based on:

- Constraint validation HTML Input Attributes
- Constraint validation CSS Pseudo Selectors
- Constraint validation DOM Properties and Methods.

Attribute	Description
<b>disabled</b>	Specifies that the input element should be disabled
<b>max</b>	Specifies the maximum value of an input element
<b>min</b>	Specifies the minimum value of an input element
<b>pattern</b>	Specifies the value pattern of an input element
<b>required</b>	Specifies that the input field requires an element
<b>type</b>	Specifies the type of an input element


## DOM(Document Object Model)

# The DOM Forms

### Automatic HTML Form Validation

- HTML5 enable the features form validation can be performed automatically by the browser.
- If a form field (fname) is empty, the required attribute prevents this form from being submitted:

#### JavaScript Validation

 Please fill out this field.

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Validation</h2>
<form action=" " method="post">
  <input type="text" name="fname" required>
  <input type="submit" value="Submit">
</form>
</body>
</html>
```

## DOM(Document Object Model)

# The DOM Forms

## JavaScript Form Validation

```
<!DOCTYPE html>
<html>
<head>
<script>

function validateForm() {
  let x = document.forms["Form1"]["fname"].value;
  if (x == "") {
    alert("Name must be filled out");
    return false;
  }
}
</script>
</head>
<body>
```

```
<h2>JavaScript Validation</h2>

<form name="Form1" action="" onsubmit="return
  validateForm()" method="post">
  Name: <input type="text" name="fname">
  <input type="submit" value="Submit">
</form>

</body>
</html>
```

### JavaScript Validation

Name:

This page says

Name must be filled out

# HTML5 Canvas



## Canvas

# HTML5 Canvas

- The HTML **canvas** element provides HTML a bitmapped surface to work with.
- It is used to draw graphics on the web page.
- The HTML 5 **<canvas>** tag is used to draw graphics using scripting language like **JavaScript**.
- The **<canvas>** element is only a container for graphics, we must need a scripting language to draw the graphics.
- The **<canvas>** element allows for dynamic and scriptable rendering of 2D shapes and bitmap images.
- Canvas has several methods for drawing paths, boxes, circles, text, and adding images.

## Canvas

# HTML5 Canvas

### To create a HTML canvas

- A canvas is a rectangle like area on an HTML page.
- It is specified with canvas element.
- By default, the <canvas> element has no border and no content, it is like a container.

```
<canvas id = "mycanvas" width ="200" height ="100"> </canvas>
```

- Always specify an **id** attribute (to be referred to in a script), and a **width** and **height** attribute to define the size of the canvas.
- To add a border, use the **style** attribute.



## Canvas

### Basic empty canvas

```
<!DOCTYPE>  
<html>  
<body>  
<canvas id="myCanvas" width="300" height="200" style="border:2px solid;">  
</canvas>  
</body>  
</html>
```



## Canvas

# Canvas - Line

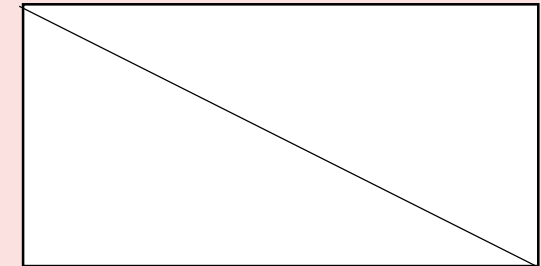
- After creating the rectangular canvas area, we must add a JavaScript to do the drawing.
- If you want to draw a straight line on the canvas, we can use the following two methods.
  - **moveTo(x,y):** It is used to define the starting point of the line.
  - **lineTo(x,y):** It is used to define the ending point of the line.
- If we draw a line which starting point is (0,0) and the end point is (200,100), use the stroke method to draw the line.

•

## Canvas

# Drawing Line on Canvas

```
<!DOCTYPE html>
<html>
<body>
<canvas id="myCanvas" width="200" height="100" style="border:1px solid #d3d3d3;">
Your browser does not support the HTML canvas tag.</canvas>
<script>
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.moveTo(0,0);
ctx.lineTo(200,100);
ctx.stroke();
</script>
</body></html>
```



## Canvas

# Canvas - Circle

- If we want to draw a circle on the canvas, we can use the `arc()` method:

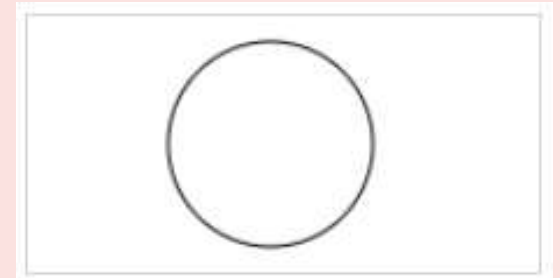
```
arc(x, y, r, start, stop)
```

- To sketch circle on HTML canvas, use one of the `ink()` methods, like `stroke()` or `fill()`.

## Canvas

# Drawing Circle on Canvas

```
<!DOCTYPE html>
<html>
<body>
<canvas id="myCanvas" width="200" height="100" style="border:1px solid #d3d3d3;">
Your browser does not support the HTML canvas tag.</canvas>
<script>
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.beginPath();
ctx.arc(95,50,40,0,2*Math.PI);
ctx.stroke();
</script>
</body></html>
```



## Canvas

### Canvas - Text

- There are property and methods used for drawing text on the canvas.
- **font property:** It is used to define the font property for the text.
- **fillText(text,x,y) method:** It is used to draw filled text on the canvas. It looks like bold font.
- **strokeText(text,x,y) method:** It is also used to draw text on the canvas, but the text is unfilled.

## Canvas

### Drawing text on canvas

```
<!DOCTYPE html>
<html>
<body>
<canvas id="myCanvas" width="200" height="100" style="border:1px solid #d3d3d3;">
Your browser does not support the HTML canvas tag.</canvas>
<script>
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.font = "30px Arial";
ctx.fillText("Hello World",10,50);
</script>
</body>
</html>
```

A rectangular canvas with a thin grey border. Inside the canvas, the text "Hello World" is written in a large, bold, black serif font, centered horizontally and vertically.

## Canvas

### Stroke text on canvas

```
<!DOCTYPE html>
<html>
<body>
<canvas id="myCanvas" width="200" height="100" style="border:1px solid #d3d3d3;">
Your browser does not support the HTML canvas tag.</canvas>
<script>
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.font = "30px Arial";
ctx.strokeText("Hello World",10,50);
</script>
</body>
</html>
```



Hello World



## Canvas

# Canvas - Gradient

- Gradients can be used to fill rectangles, circles, lines, text, etc.
- Shapes on the canvas are not limited to solid colors.
- There are **two different types** of gradients:
  - **createLinearGradient(x,y,x1,y1)** - creates a linear gradient
  - **createRadialGradient(x,y,r,x1,y1,r1)** - creates a radial/circular gradient
- Once we have a gradient object, we must add two or more color stops.
- The **addColorStop()** method specifies the color stops, and its position along the gradient. Gradient positions can be anywhere between 0 to 1.
- To use the gradient, set the **fillStyle** or **strokeStyle** property to the gradient, then draw the shape (rectangle, text, or a line).

## Canvas

# Draw Linear Gradient on canvas

```
<!DOCTYPE html>
<html>
<body>
<canvas id="myCanvas" width="200" height="100"
style="border:1px solid #d3d3d3;">
Your browser does not support the HTML canvas
tag.</canvas>
<script>
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
// Create gradient
var grd = ctx.createLinearGradient(0,0,200,0);
```

```
grd.addColorStop(0,"red");
grd.addColorStop(1,"white");
// Fill with gradient
ctx.fillStyle = grd;
ctx.fillRect(10,10,150,80);
</script>
</body>
</html>
```



## Canvas

# Draw Circular Gradient on canvas

```
<!DOCTYPE html>
<html>
<body>
<canvas id="myCanvas" width="200" height="100"
style="border:1px solid #d3d3d3;">
Your browser does not support the HTML canvas
tag.</canvas>
<script>
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
// Create gradient
var grd = ctx.createRadialGradient(75,50,5,90,60,100);
```

```
grd.addColorStop(0,"red");
grd.addColorStop(1,"white");
// Fill with gradient
ctx.fillStyle = grd;
ctx.fillRect(10,10,150,80);
</script>
</body>
</html>
```



## Canvas

### Canvas - Image

- canvas **drawImage()** function is used to display an image or video on canvas.
- This function can be used to display the whole image or just a small part of the image.
- But, image has to be loaded first to load it further on canvas.

```
context.drawImage(img, x, y, swidth, sheight, sx, sy, width, height);
```

- **img**: the image or video to draw on canvas.
- **x**: the x-coordinate where image has to be placed.
- **y**: the y-coordinate where image has to be placed.
- **swidth**: the width of the clipped image (optional).
- **sheight**: the height of the clipped image (optional).
- **sx**: x-coordinate where to start the clipping (optional).
- **sy**: y-coordinate where to start the clipping (optional).
- **width**: the width of the image to use (optional).
- **height**: the height of the image to use (optional).

## Canvas

### Draw Image on canvas

```
<!DOCTYPE html>
<html>
<body>
<p>Image to use:</p>

<p>Canvas:</p>
<canvas id="myCanvas" width="279" height="214"
style="border:1px solid #d3d3d3;">
Your browser does not support the HTML5 canvas tag.
</canvas>
```

```
<script>
window.onload = function() {
  var c = document.getElementById("myCanvas");
  var ctx = c.getContext("2d");
  var img = document.getElementById("dog");
  ctx.drawImage(img, 10, 10);
}
</script>
</body>
</html>
```

## Canvas

# Draw Image on canvas

Image to use:



Canvas:



## Canvas

### Draw Image on canvas

```
<!DOCTYPE html>
<html>
<body>
<p>Image to use:</p>

<p>Canvas:</p>
<canvas id="myCanvas" width="279" height="214"
style="border:1px solid #d3d3d3;">
Your browser does not support the HTML5 canvas tag.
</canvas>
```

```
<script>
window.onload = function() {
    var c = document.getElementById("myCanvas");
    var ctx = c.getContext("2d");
    var img = document.getElementById("dog");
    ctx.drawImage(img, 10, 10, 150, 180);
}
</script>
</body>
</html>
```

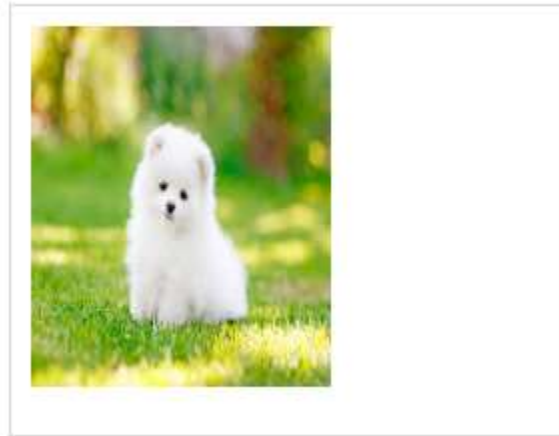
## Canvas

# Draw Image on canvas

Image to use:



Canvas:





## Canvas

### Draw Image on canvas

```
<!DOCTYPE html>
<html>
<body>
<p>Image to use:</p>

<p>Canvas:</p>
<canvas id="myCanvas" width="279" height="214"
style="border:1px solid #d3d3d3;">
Your browser does not support the HTML5 canvas tag.
</canvas>
```

```
<script>
window.onload = function() {
    var c = document.getElementById("myCanvas");
    var ctx = c.getContext("2d");
    var img = document.getElementById("dog");
    ctx.drawImage(img, 100,60,50,60,10,10,50,60);;
}
</script>
</body>
</html>
```

## Canvas

# Draw Image on canvas

Image to use:



Canvas:



# Javascript

## Quiz

**1. What is the correct JavaScript syntax to change the content of the HTML element below?**

**`<p id="demo">This is a demonstration.</p>`**

**a) `document.getElementById("demo").innerHTML="Hai"`**

**b) `document.getElement("p").innerHTML="Hai"`**

**c) `#demo.innerHTML="Hai"`**

**d) `document.getElement("p").innerHTML="Hai"`**

**a) `document.getElementById("demo").innerHTML="Hai"`**



## Javascript

### Quiz

**2. Which event occurs when the user clicks on an HTML element?**



a) onclick

b) onchange

c) onmouseclick

d) onmouseover

a) onclick

## Javascript

### Quiz

**3. To which object does the location property belong?**



a) Window

b) Position

c) Location

d) None of these

a) Window

## Javascript

## Quiz

4. The statement `a===b` refers to \_\_\_\_\_



a) Both a and b are equal in value, type and reference address

b) Both a and b are equal in value

c) Both a and b are equal in value and type

d) There is no such statement

c) Both a and b are equal in value and type

# Javascript

## Quiz

5. What will be returned from following JavaScript code?



```
<p id="demo"></p>
<script>
function myFunction()
{
    document.getElementById("demo").innerHTML=
    Math.abs(-7.25);
}
</script>
```

a) -7.25

b) 7

c) 7.25

d) -7

c) 7.25

## Javascript

### Quiz

**6. The HTML canvas is a**



**a) Three-dimensional grid**

**b) Two-dimensional grid**

**c) One-dimensional grid**

**d) None of the above**

**b) Two-dimensional grid**



## Javascript

### Quiz

**7. Which of the following element is used for canvas graphics?**



a) `<graphic>`

b) `<canvas>`

c) `<css>`

d) `<paint>`

**b) `<canvas>`**

## Javascript

## Quiz

8. To draw text on a canvas, which property and method is used.



a) font

b) fillText(text,x,y)

c) strokeText(text,x,y)

d) Both A & B

c) strokeText(text,x,y)

# THANK YOU