Javascript: Basics

Introduction

- JavaScript is a scripting language most often used for client- side web development.
- The JavaScript supported in the browsers typically support additional objects.
  - e.g., Window, Frame, Form, DOM object, etc.
- Different brands or/and different versions of browsers may support different implementation of JavaScript.
  - They are not fully compatible
- JScript is the Microsoft version of JavaScript

What can we do with JavaScript?

- To create interactive user interface in a web page (e.g., menu, pop-up alert, windows, etc.)

- Manipulating web content dynamically
  - Change the content and style of an element
  - Replace images on a page without page reload
  - Hide/Show contents

- Generate HTML contents on the fly
- Form validation
- AJAX (e.g. Google complete)
- etc.

Why Study JavaScript?

- JavaScript is one of the **3 languages** all web developers **must** learn:

  1. **HTML** to define the content of web pages
  2. **CSS** to specify the layout of web pages
  3. **JavaScript** to program the behavior of web pages

- Web pages are not the only place where JavaScript is used.
- Many desktop and server programs use JavaScript. Node.js is the best known.
- Some databases, like MongoDB and CouchDB, also use JavaScript as their programming language.

Syntax

- JavaScript can be implemented using JavaScript statements that are placed within the **<script>... </script>**

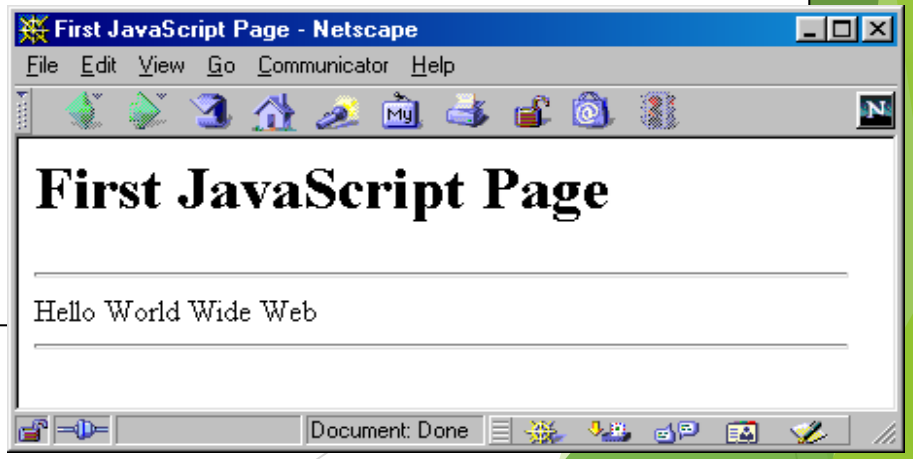
- You can place the **<script>** tags, containing your JavaScript, anywhere within your web page, but it is normally recommended that you should keep it within the **<head>** tags.

- The <script> tag alerts the browser program to start interpreting all the text between these tags as a script.

A Simple Script

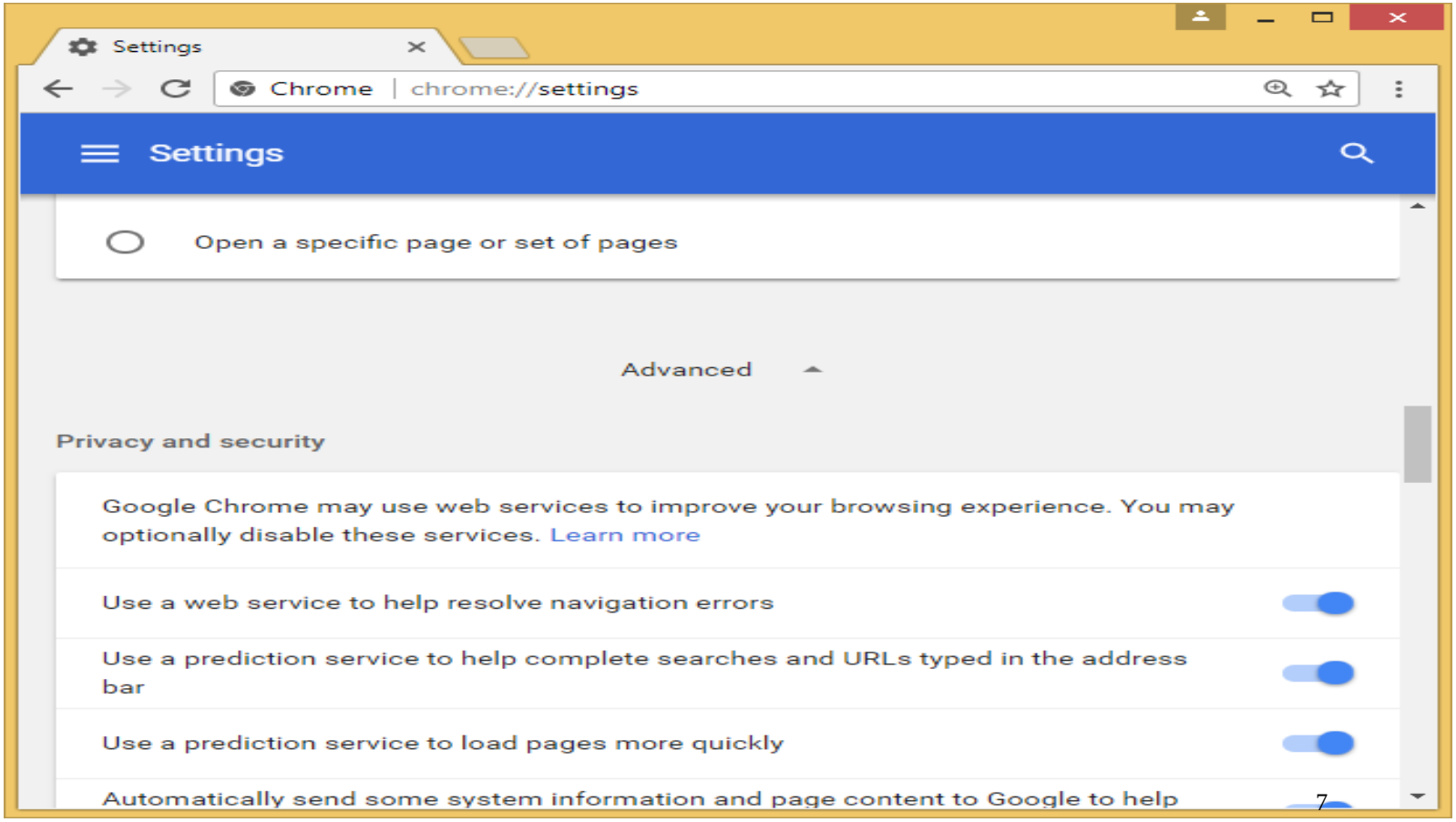
Ex1.html

```
<html>
<head><title>First JavaScript Page</title></head>
<body>
<h1>First JavaScript Page</h1>
<script type="text/javascript">
  document.write("<hr>");
  document.write("Hello World from JavaScript!");
  document.write("<hr>");
</script>
</body>
</html>
```



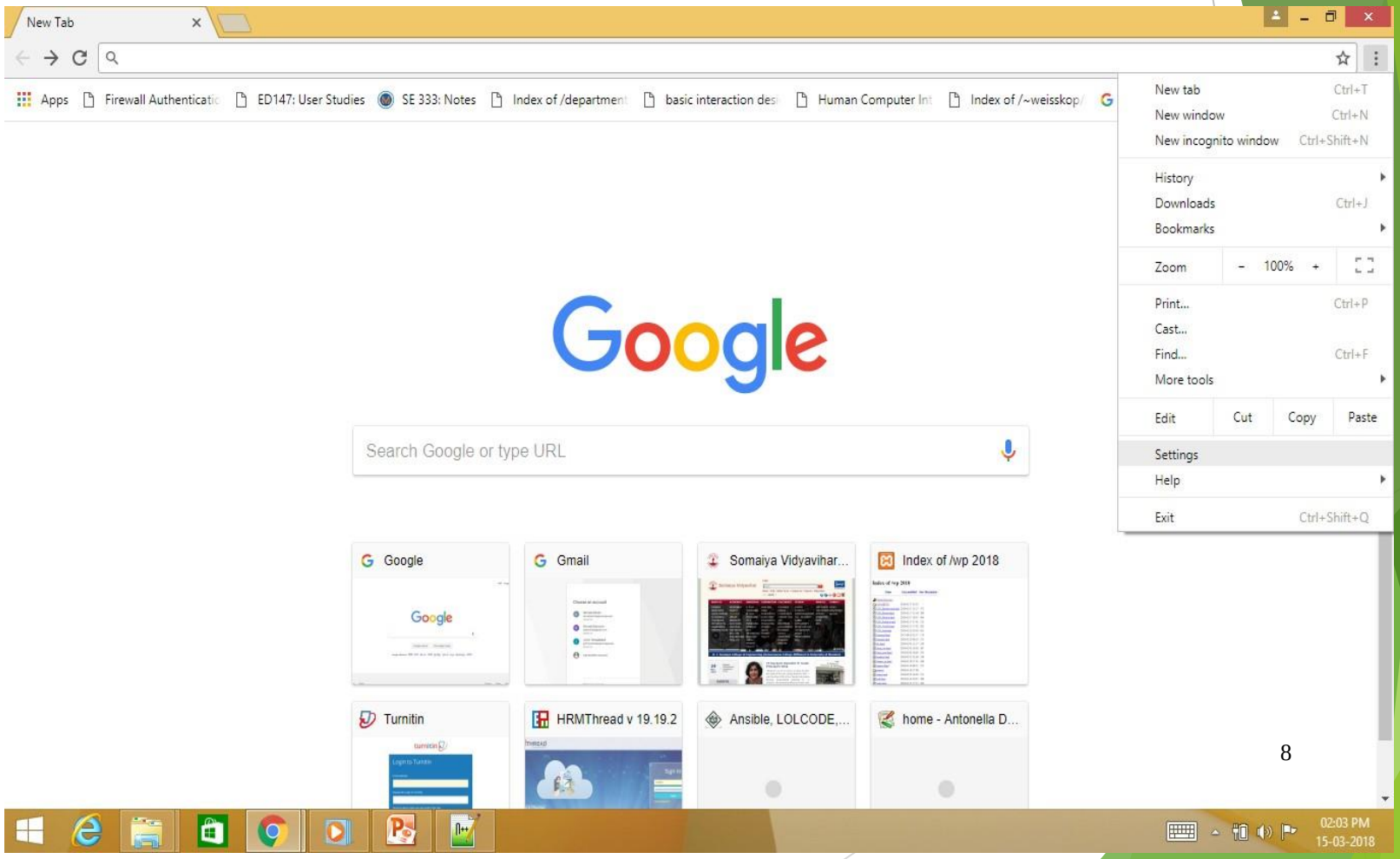
Enabling - Chrome

Step 2 : Advance Setting -> Privacy & Security



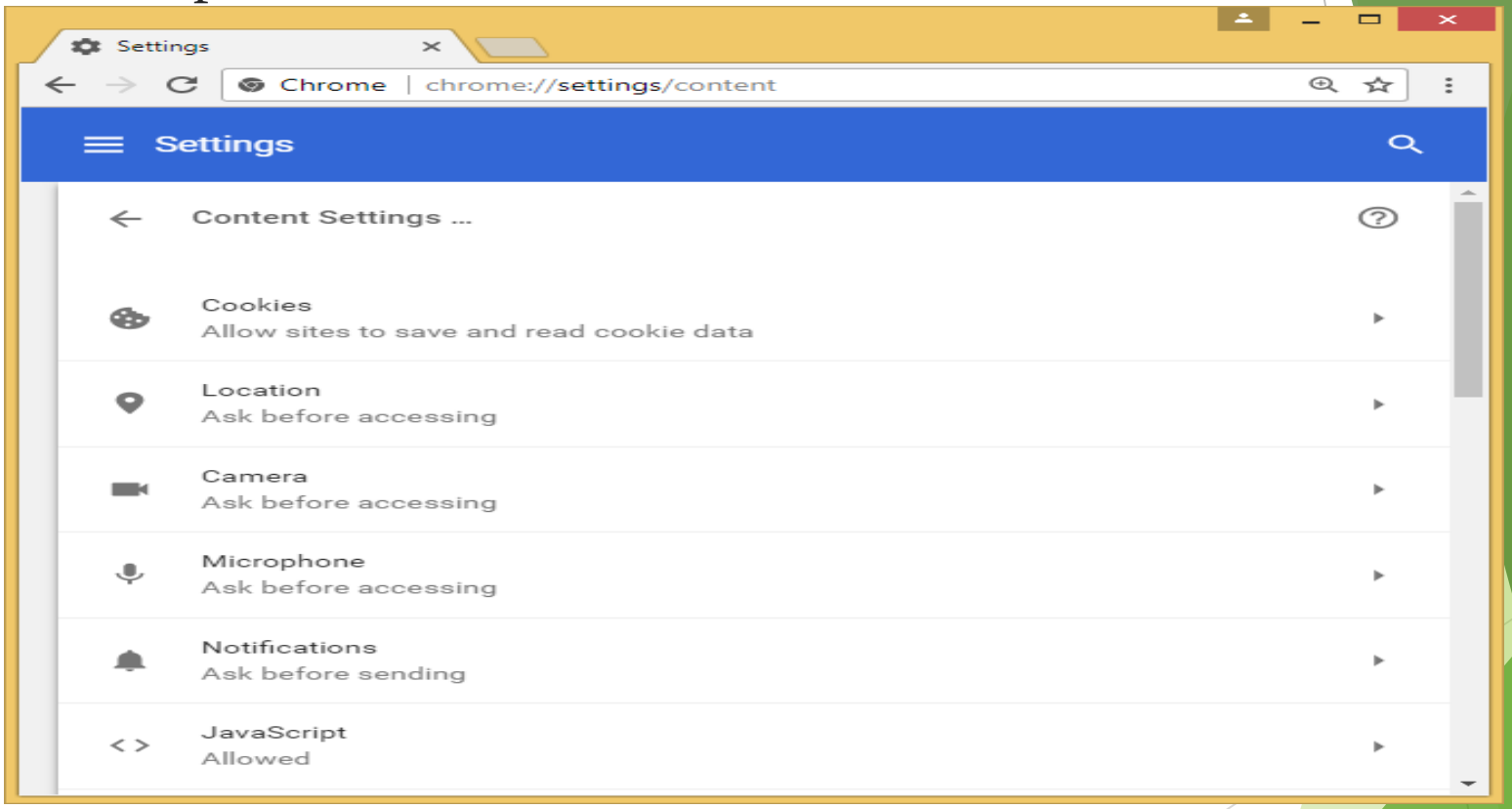
Enabling - Chrome

- Step 1 : Choose Setting



Enabling - Chrome

Step 3 : Privacy & Security -> Content Settings-Enable Javascript



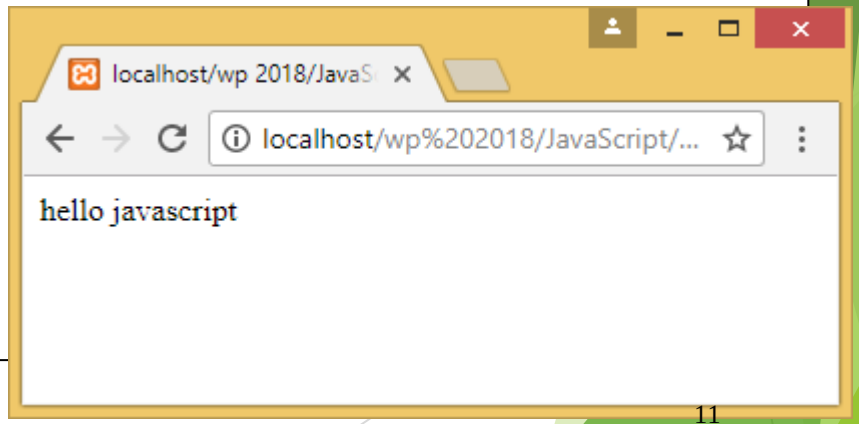
Comments

- Single Line Comment
  - It is represented by double forward slashes (//). It can be used before and after the statement.
- Multiline Comment
  - It can be used to add single as well as multi line comments. So,
    it is more convenient.
  - /* your code */

A Simple Script

Ex2.html

```
<html>
<body>
<script>
/* Sample Program for
comment */
// It is single line comment
document.write("hello javascript");  //Printing Message
   on Output device


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



Basic Features

- You can use as many <script> tags as you like in both the
<head> and <body> and they are executed sequentially.
- Script Execution
  - Top-down
  - <head> before <body>
  - Can’t forward reference outside a <script> tag
- JavaScript is case sensitive
- Whitespace is generally ignored in JavaScript statement
  - X = X + 1 same as X=X+1

Statements

- A script is made up of individual statements.
- JavaScript statement are terminated by returns / semi-colons (;)
- Prefer to use semi-colons
- Statements are group together using curly braces { }.
  - Used for control structures and functions

JavaScript Can Change HTML Content

- One of many JavaScript HTML methods is getElementById().
- This example uses the method to "find" an HTML element (with id="demo") and changes the element content (innerHTML) to "Hello JavaScript":
- Example

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

JavaScript Can Change HTML Styles (CSS)

- Changing the style of an HTML element, is a variant of changing an HTML attribute:
- Example

```
document.getElementById("demo").style.fontSize = "35px";
```

JavaScript Can Hide HTML Elements

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

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

JavaScript Can Show HTML Elements

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

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

JavaScript Where To

- JavaScript in <head>

```
<head>
<script>
function myFunction() {
  document.getElementById("demo").innerHTML =
  "Paragraph
changed.";
}
</script>
</head>
```

JavaScript Where To

- JavaScript in <body>

```
<body>
<h1>A Web Page</h1>

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

<button type="button" onclick="myFunction()">Try it</button>
<script>
function myFunction() {
 document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>
</body>
```

External JavaScript

- Scripts can also be placed in external files:

External file:

myScript.js function

```
myFunction() {
 document.getElementById("demo").innerHTML =
"Paragraph changed.";
}
```

- External scripts are practical when the same code is used in many different web pages.
- JavaScript files have the file extension.js.
- To use an external script, put the name of the script file in the src
  (source) attribute of a <script> tag:

Example

JavaScript Display Possibilities

JavaScript can "display" data in different ways:

- Writing into an HTML element, using innerHTML.
- Writing into the HTML output using document.write().
- Writing into an alert box, using window.alert().
- Writing into the browser console, using console.log().

JavaScript Display Possibilities

- document.getElementById("demo").innerHTML = 5
  + 6;
- document.write(5 + 6);
- <button type="button" onclick="document.write(5 + 6)">Try it</button>
- window.alert(5 + 6);
- console.log(5 + 6);

Variables

- Variables are containers for storing data values.
- These unique names are called identifiers.
- Identifiers can be short names (like x and y) or more
  descriptive names (age, sum, total).

Cont…

- 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 (likeJavaScript keywords) cannot be usedas
    names

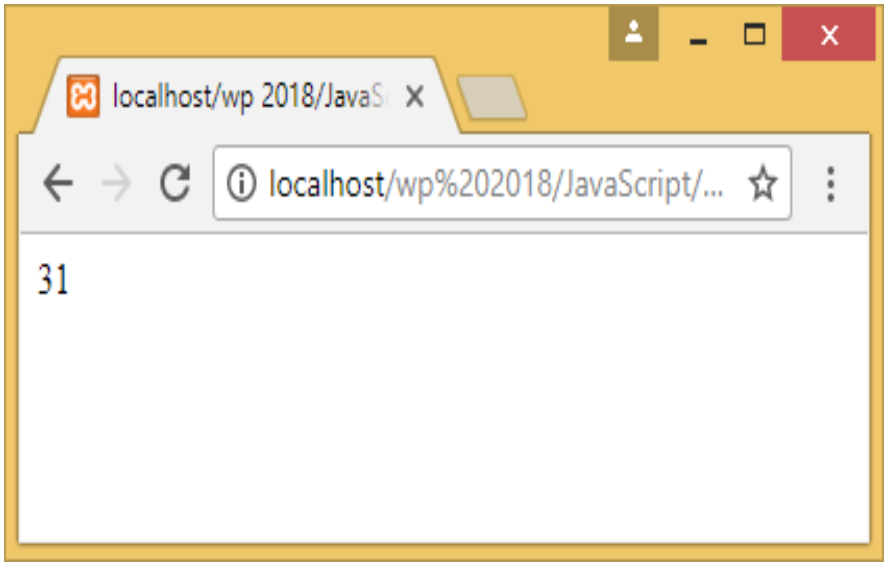
Cont…

- Define a variable using the var statement
  - var x;
- If undefined a variable will be defined on its first use
- Variables can be assigned at declaration time
  - var x = 5;
- Commas can be used to define many variables at once
  - var x, y = 5, z;

A Simple Script

Ex3.html

```
<html>
<body>
<script>
   var myvar = 31;
     document.write(myvar);
     document.write(Myvar);
</script>
</body>
</html>
```



JavaScript Reserved Words

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

JavaScript Reserved Words

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

Words marked with* are new in ECMAScript 5 and 6.

Data Types

- Every variable has a data type that indicates what kind of data the variable holds
- Basic data types in JavaScript
  - Strings
  - Numbers
  - Boolean
  - Array
- JavaScript has dynamic types. This means that the same variable can be used to hold different data types.

Numbers

- JavaScript has only one type of numbers.
- Numbers can be written with, or without decimals:

  E.g.

```
var x1 = 34.00;
var x2 = -34;
```

- Extralarge or extra small numbers can be written with scientific (exponential) notation.

  E.g.

```
var y = 123e5;
```

Strings

- A string (or a text string) is a series of characters
- Strings are written with quotes. You can use single or double quotes
  - E.g. (“thomas”, ‘x’, “Who are you?”)
- Strings may include special escaped characters
  - E.g. ‘This isn\’t hard’
- Strings may contain some formatting characters
  - “Here are some newlines \n\n\n and tabs \t\t\t yes!”
- Whitespace has no meaning outside of quotation marks

```
var foo = ’ h e l l o world ’;
```

Boolean

- Booleans can only have two values: true or false.
- Booleans are often used in conditional testing.

Type Conversion

- JavaScript **typeof** operator find the type of a JavaScript variable
- E.g.

```
typeof (3)          // Returns "number"
typeof (3 + 4)      // Returns "number"
```

Arrays

- An array is a special variable, which can hold more than one value at a time.
- Creating array

```
var cars = [];
```

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

- Access the elements of an array by referring **index** number.

```
var name = car[1];
```

- Array index starts from zero (0).

Cont…

- The **length** property of an array returns the length of an array (the number of array elements).
  - E.g. cars.length;
- Add a new element to an array is using the push method

  E.g. cars.push(“swift”);

Arithmetic Operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
++	Increment
--	Decrement

Assignment Operator

Operator	Example	Equivalent
=	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

Comparison Operator

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

Logical Operator

Operator	Description
&&	logical and
||	logical or
!	logical not

Conditional Statement

```
if (hour < 18)
{
    greeting = "Good
}   day";


if (hour< 10) {
    greeting = "Good
morning";
} else if (hour < 20) {
    greeting = "Good day";
} else {
    greeting = "Good
    evening";
}
```

```
var hour = new
Date().getHours();
var greeting;
 if (hour < 18)
{
        greeting = "Good day";
 }
else
{
        greeting = "Good
evening";
 }
```

Loops

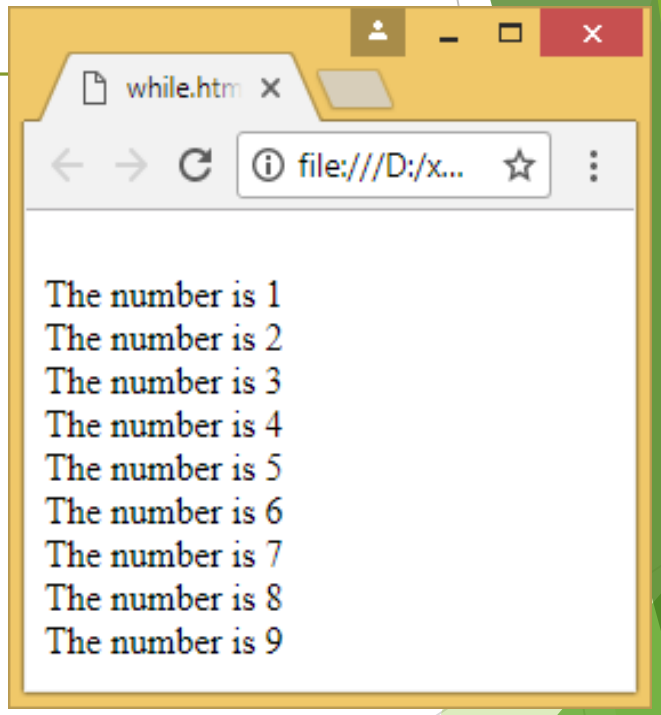
- Loops can execute a block of code as long as a specified condition is true.
- while loop loops through a block of code as long as a specified condition is true

Syntax

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

Exampl
e

```
<html>
<body>
<script>
var text = "";
var i = 1;
while (i <
10)
{
   document.write("<br> The number is " +
   i);
   i++;
}
</script>
</body>
</html>
```



Example

```
<html>
<body>
<script>
var text = "";
var i = 10;
while (i <
10)
{
   document.write("<br> The number is " +
   i);
   i++;
}
document.write(“Outside while loop “ + i);
</script>
</body>
</html>
```

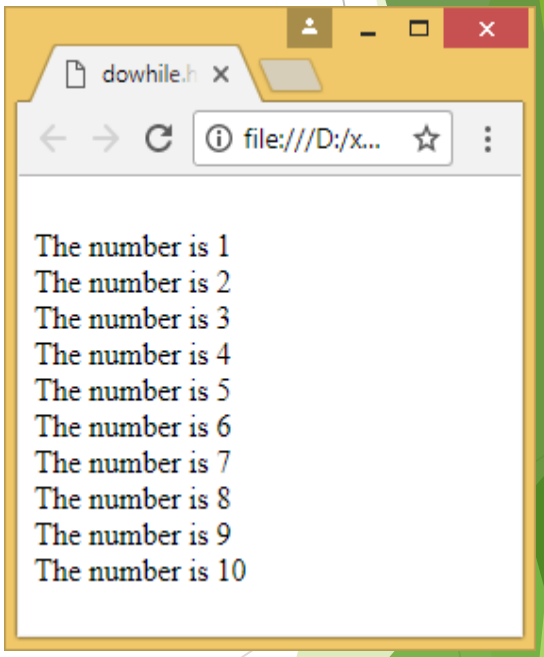
Loops

- Do-while loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true

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

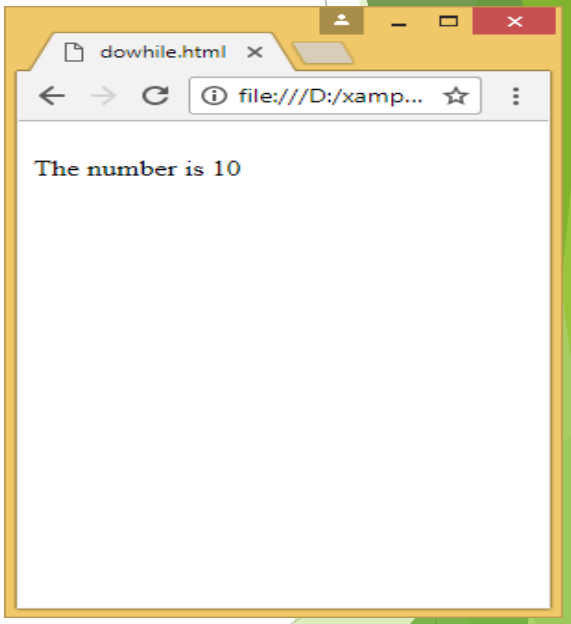
Example

```
<html>
<body>
<script>
var i = 1;
do
{
      document.write("<br> The number is " + i);
      i++;
} while (i < 10);
</script>
</body>
</html>
```



Example

```
<html>
<body>
<script>
var i = 10;
do
{
      document.write("<br> The number is " + i);
      i++;
} while (i < 10);
</script>
</body>
</html>
```



For Loop

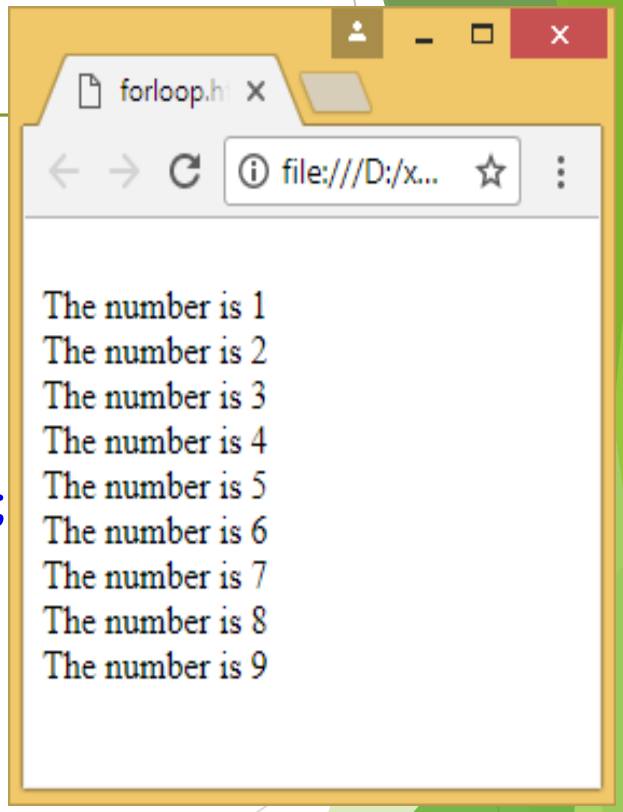
- The for loop is often the tool you will use when you want to create a loop
- Syntax

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

  - **Statement 1** is executed before the loop (the code block) starts.
  - **Statement 2** defines the condition for running the loop (the code block).
  - **Statement 3** is executed each time after the loop (the code block) has been executed.

Example

```
html>
<body>
<script>
var i = 1;
for(i=1;i<10;i++)
{
   document.write("<br> The number is " + i);
}
</script>
</body>
</html>
```



Switch Statement

- Use the switch statement to select one of many blocks of code to be executed.
  - Syntax

```
switch(expression) {
  case n:
    code block
    break;
  case n:
    code block
    break;
  default:
    code block
}
```

  - The switch expression is evaluated once.
  - The value of the expression is compared with the values of each case.
  - If there is a match, the associated block of code is executed.

Example

```
<script>  let day;
switch (new Date().getDay()) {  case 0:
        day = "Sunday";  break;
    case 1:
        day = "Monday";
        break;
    case 2:
        day = "Tuesday";
        break;
   case 3:
        day = "Wednesday";  break;
    case 4:
        day = "Thursday";
        break;
    case 5:
        day = "Friday";
        break;
    case 6:
        day = "Saturday";
}
document.write("Today is " + day);
</script>
```

JavaScript Functions

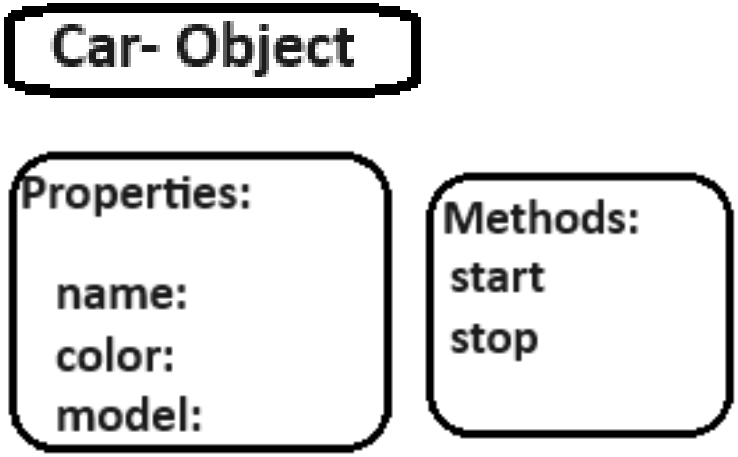
- A JavaScript function is a block of code designed to perform a particular task.
- A JavaScript function is executed when "something" invokes it (calls it).
- **function definition:**
- function functionName (parameters)
- {
- // function body;
- }
- Function calling:
- nameOfThefunction (arguments);

Example

- <!DOCTYPE html>
- <html> <body>
- <h1>JavaScript Functions</h1>
- Call a function which performs a calculation and returns the result:
-
- <script>
- let x = myFunction(4, 3); //function calling
- document.getElementById("demo").innerHTML = x;
- function myFunction(a, b) //function definition
- {
- return a * b;
- }
- </script> </body></html>

JavaScript Object Notation(JSON)

- Object –
- Logically all the data stored together under one name
- It has properties and Methods.
  - Properties are the attributes of an Object.
  - Methods are actions that can be performed on objects.



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

Object creation

- The properties are written as name:value pairs

  (name and value separated by a colon).

EX. let car = {type:“Baleno", model:"500", color:"white"};

Creating objects in JavaScript

1. By object literal
   - You define properties and methods directly within curly braces {}
2. By creating instance of Object directly (using new keyword)
3. By using an object constructor (using new keyword)

Creating object by literal

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

<h2>JavaScript Objects</h2>
<p>Creating a JavaScript Object:</p>

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

<script>
const person = {firstName:"John", lastName:"Doe", age:50};

document.getElementById("demo").innerHTML =
person.firstName + " is " + person.age + " years old.";
</script>

</body>
</html>
```

By creating instance of Object

- The syntax of creating object directly is given below:

1. var objectname=new Object();

- Here, **new keyword** is used to create object.
- Example:
- <html>
- <body>
- <script>
- var emp=new Object();
- emp.id=101;
- emp.name="Ravi Joshi";
- emp.salary=50000;
- document.write(emp.id+" "+emp.name+" "+emp.salary);
- </script>
- </body></html>

- Creating with new keyword

```
var person = new Object();
person.name = "saya";
person.age = 46;
```

By using an Object constructor

- Here, you need to create function with arguments. Each argument value can be assigned in the current object by using ‘this’ keyword.
- The **this keyword** refers to the current object.
- Example:
- <script>
- function emp(eid,ename,esalary){
- this.id=eid;
- this.name=ename;
- this.salary=esalary;
- }
- Var e=new emp(103,"Vimal Jaiswal",30000);
- document.write(e.id+" "+e.name+" "+e.salary);
- </script>

```
<!DOCTYPE html>
<html><body>
<h2>JavaScript Object Constructors</h2>
<p id="demo"></p>

<script>
// Constructor function for Person objects
function Person(first, last, age, eye) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eye;
}

// Create two Person objects
const myFather = new Person("John", "Doe", 50, "blue");
const myMother = new Person("Sally", "Rally", 48, "green");

// Display age
document.getElementById("demo").innerHTML =
"My father is " + myFather.age + ". My mother is " + myMother.age + "."+"My
father name is " + myFather.firstName;

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

Javascript method

- getElementById()
- GetElementsByClassName()
- getElementsByName
- getElementsByTagName
- JS innerHTML property

Document.getElementById()

- The **document.getElementById()** method returns the element of specified id.
- Example:
- document.getElementById("demo").innerHTML = "Hello JavaScript";
- <!DOCTYPE html>
- <html>
- <body>
- <h2>What Can JavaScript Do?</h2>
- JavaScript can change HTML content.
- <button type="button" onclick='document.getElementById("demo").innerHTML = "Hello JavaScript!"'>Click Me!</button>
- </body>
- </html>

document.getElementsByName()

- This method returns all the element of specified name.
- syntax :

  document.getElementsByName("name")

Example:

```
1.  <script type="text/javascript">
2.  function totalelements()
3.  {
4.  var allgenders=document.getElementsByName("gender");
5.  alert("Total Genders:"+allgenders.length);
6.  }
7.  </script>
8.  <form>
9.  Male:<input type="radio" name="gender" value="male">
10. Female:<input type="radio" name="gender" value="female">
11.  <input type="button" onclick="totalelements()" value="Total Genders">
12. </form>
```

document.getElementsByTagName()

- This method returns all the element of specified tag name.
- Syntax:
- document.getElementsByTagName(“p")
- Example:

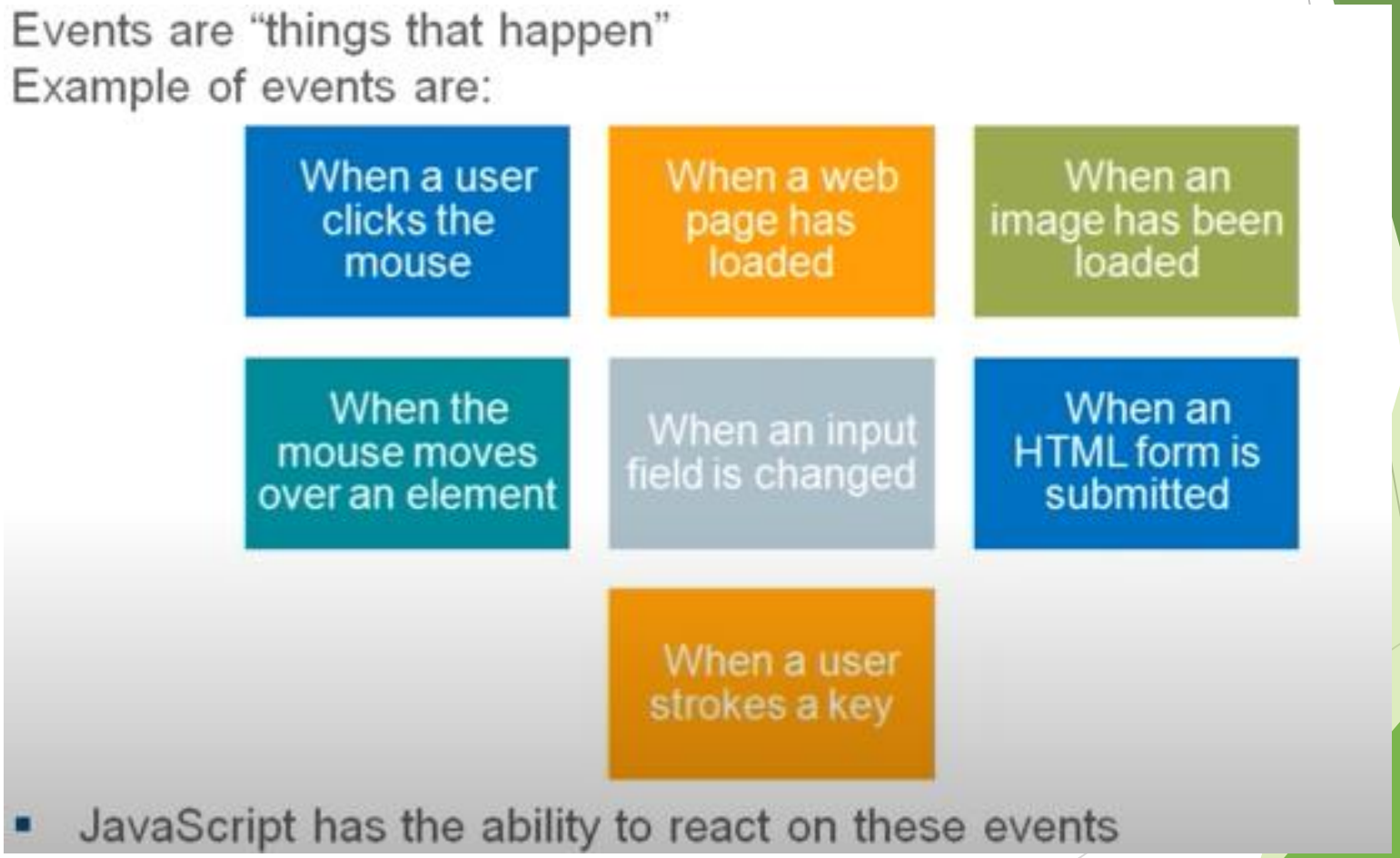
```
<body>
    <h1>This is a heading</h1>
    <p>This is a paragraph.</p>
    <p>This is another paragraph.</p>

    <script>
        // Get all <p> elements
        let paragraphs = document.getElementsByTagName("p");

        // Loop through the collection and change the text color
        for (let i = 0; i < paragraphs.length; i++) {
            paragraphs[i].style.color = "blue";
        }
    </script>
</body></html>
```

ParseInt()- String to Int conversion
ParseFloat()- String to float conversion

- **Example:**
- <!doctype html>
- <html>
- <head><title></title></head>
- <body>
- <script>
- var num1=prompt(“Enter the number");
- document.write(typeof num1);
- document.write(num1+9);
- document.write("
");
- document.write(parseInt(num1)+9);
- </script>
- </body>
- </html>
- var num1=parseInt(prompt(“Enter the number"));

Document Object Model: Events

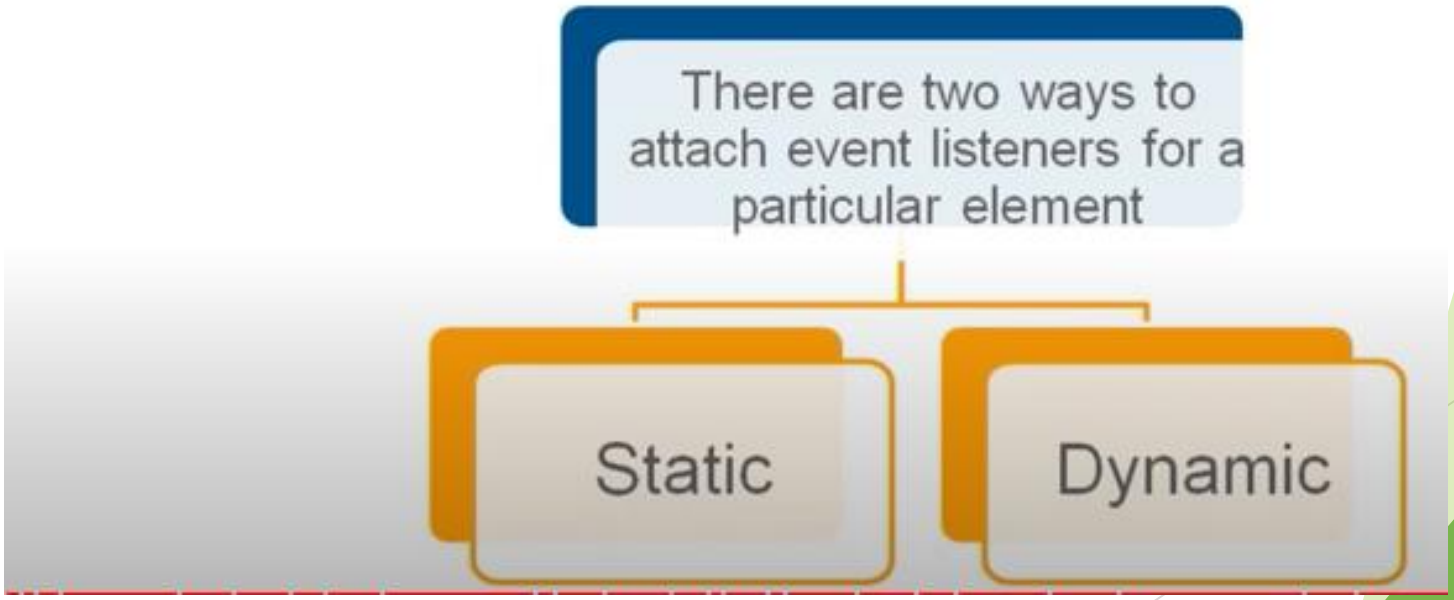
Events are "things that happen"
Example of events are:



- JavaScript has the ability to react on these events

Document Object Model: Event Listeners

- Event Listeners are: The type of event that is occurring
- We can attach any number of event listeners to an element



Statically

- In JavaScript, you can add event listeners both statically (in your HTML markup) and dynamically (in your JavaScript code).
- In HTML, you can add event listeners directly to elements using attributes like onclick, onmouseover, etc.
- Example:
- <button onclick="myFunction()">Click me</button>

Dynamic method

- In JavaScript, you can add event listeners to elements dynamically using the addEventListener() method
- Syntax:
- Element.addEventListener(event,function)

```
<script>
document.getElementById("myPara").addEventListener("click",
paraClicked);
function paraClicked()
{
 document.getElementById("myPara").setAttribute("style", "color:
cyan");
}
</script>
```

Common HTML Events

- Mouse events:

Event Performed	Event Handler	Description
click	onclick	When mouse click on an element
mouseover	onmouseover	When the cursor of the mouse comes over the element
mouseout	onmouseout	When the cursor of the mouse leaves an element
mousedown	onmousedown	When the mouse button is pressed over the element
mouseup	onmouseup	When the mouse button is released over the element
mousemove	onmousemove	When the mouse movement takes place.

Keyboard events:

Event Performed	Event Handler	Description
Keydown & Keyup	onkeydown & onkeyup	When the user press and then release the key

Form events:

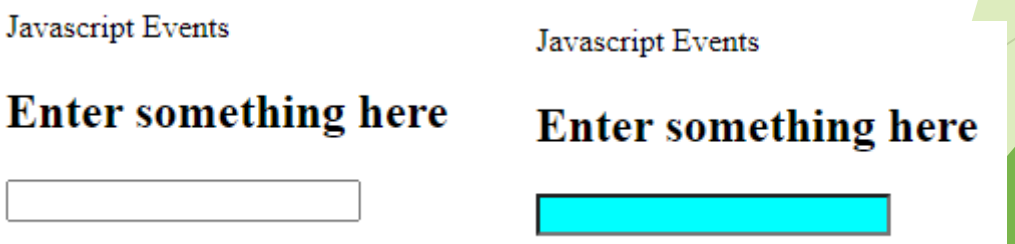
Event Performed	Event Handler	Description
focus	onfocus	When the user focuses on an element
submit	onsubmit	When the user submits the form
blur	onblur	When the focus is away from a form element
change	onchange	When the user modifies or changes the value of a form element

Click Event

```
<html>
<head
<meta charset  "UTF-8”>
<meta name ="viewport" content-width=“device-width”>
<title> Javascript Events </title> </head>
<body>
<script language="Javascript" type="text/Javascript">
 function clickevent()
 {
        document.write("This is Javascript Event demo");
 }
 </script>
<form>
<input type="button" onclick="clickevent()" value=“Click here"/>
</form>
</body></html>
```

Mouse over Event

- <body>
- <script type="text/Javascript">
- function mouseoverevent()
- {
- document.write("This is Mouse Over Event");
- }
- </script>
- Keep cursor over me
- </body>
- </html>



What can JavaScript Do?

Event handlers can be used to handle, and verify, user input, user actions, and browser actions:

- Things that should be done every time a page loads
- Things that should be done when the page is closed
- Action that should be performed when a user clicks a button
- Content that should be verified when a user inputs data
- And more ...

What can JavaScript Do?

Many different methods can be used to let JavaScript work with
events:

- HTML event attributes can execute JavaScript code directly
- HTML event attributes can call JavaScript functions
- You can assign your own event handler functions to HTML elements
- You can prevent events from being sent or being handled
- And more ...

Form validation

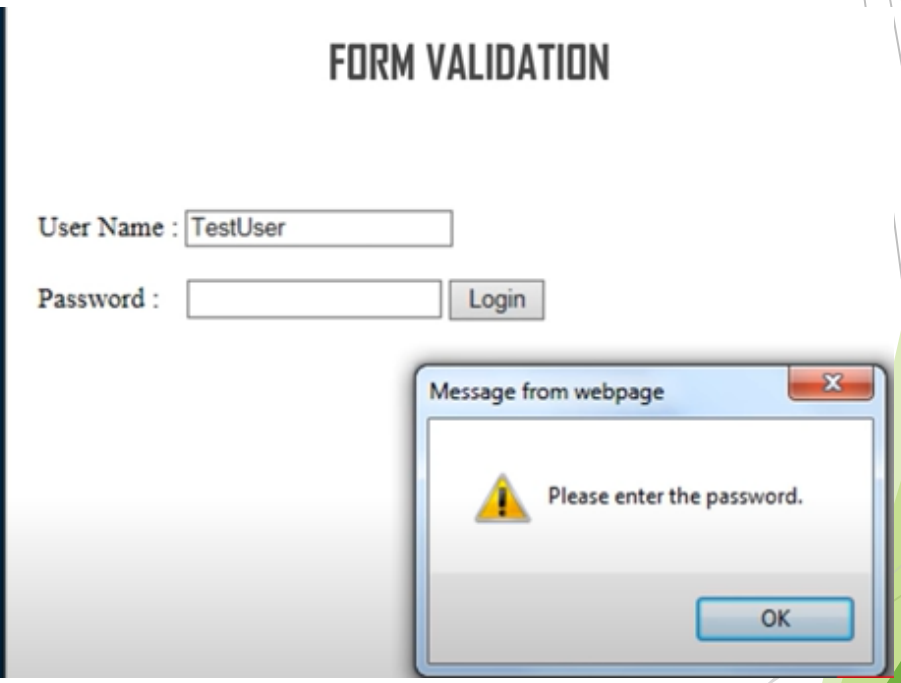
- HTML forms can be validated by JavaScript
- The main goal of Data Validation is to ensure correct user input
- Validation can be defined by many different methods, and deployed in many different ways :
  - **Server-side validation** is performed by a web server, after input has been sent to the server
  - **Client-side validation** is performed by a web browser, before input is sent to a web server
- Saves a lot of unnecessary calls to the server as all processing is handled by the web browser

Typical Validations

- Check if the input field is empty
- Check if the input field is a numeric value
- Check if the input field is a valid email id

JAVASCRIPT VALIDATION

- Name
- Password
- Email
- Date
- Mobile Number



Example

Name:

Password:

Register

```
<form name="myform" method="post"
onsubmit="return validateform()" action="dom1.html">

Name:<input type="text" name="name"/><br><br>

Password:<input type="password" name="pass"/><br>

<input type="submit" value="Register"/>

</form>
```

```
<script type="text/javascript">
function validateform()
	{
		var n=document. myform.name.value;
		var p=document.myform.pass.value;
		if (n == null || n == "")
		{
		alert("Name cant be blank");
		return false;
		}else if (p.length <6)
		{
		alert("Password must be longer than 6 digit");
		return false;
		}
	}
</script>
```

Complete example:

```
<html><head><title>Javascript Validation</title>
</head>
<body>
<script type="text/javascript">
function validateform()
	{
		var n=document. myform.name.value;
		var p=document.myform.pass.value;
		if (n == null || n == "")
		{
		alert("Name cant be blank");
		return false;
		}else if (p.length <6)
		{
		alert("Password must be long");
		return false;
		}
	}
</script>
 <form name="myform" method="post" onsubmit="return validateform()" action="dom1.html">
Name:<input type="text" name="name"/><br><br>
Password:<input type="password" name="pass"/><br>
<input type="submit" name="Register"/>
</form>
</body>
</html>
```

JavaScript Errors - Throw and Try to Catch

- The **try** statement lets you test a block of code for errors.
- The **catch** statement lets you handle the error.
- The **throw** statement lets you create custom errors.
- The **finally** statement lets you execute code, after try and catch, regardless of the result.
- When executing JavaScript code, different errors can occur.
- Errors can be coding errors made by the programmer, errors due to wrong input, and other unforeseeable things.

JavaScript Errors - Throw and Try to Catch

- Example<!DOCTYPE html>
- <html>
- <body>
- <h2>JavaScript Error Handling</h2>
-
- <script>
- try {
- adddlert("Welcome guest!");
- }
- catch(err) {
- document.getElementById("demo").innerHTML = err.message;
- }
- </script></body></html>

JavaScript Throws Errors, finally

- When an error occurs, JavaScript will normally stop and generate an error message.
- The technical term for this is: JavaScript will **throw an exception (throw an error)**.
- JavaScript will actually create an Error object with two properties: name and message.
- The **finally** statement lets you execute code, after try and catch, regardless of the result

JavaScript Debugging

Code Debugging

- Programming code might contain syntax errors, or logical errors.
- Many of these errors are difficult to diagnose.
- Often, when programming code contains errors, nothing will happen. There are no error messages, and you will get no indications where to search for errors.
- Searching for (and fixing) errors in programming code is called code debugging.

JavaScript Debugging

JavaScript Debuggers

- Debugging is not easy. But fortunately, all modern browsers have a built-in JavaScript debugger.
- Built-in debuggers can be turned on and off, forcing errors to be reported to the user.
- With a debugger, you can also set breakpoints (places where code execution can be stopped), and examine variables while the code is executing.

Best Practices

Avoid global variables, avoid new, avoid ==, avoid eval()

Avoid Global Variables

- Minimize the use of global variables.
- This includes all data types, objects, and functions.
- Global variables and functions can be overwritten by other scripts.
- Use local variables instead

Best Practices

Always Declare Local Variables

- All variables used in a function should be declared as local variables.
- Local variables must be declared with the var keyword or the let keyword, otherwise they will become global variables.
- Strict mode does not allow undeclared variables.

Best Practices

Declarations on Top

It is a good coding practice to put all declarations at the top of
each script or function.

This will:

- Give cleaner code
- Provide a single place to look for local variables
- Make it easier to avoid unwanted (implied) global variables
- Reduce the possibility of unwanted re-declarations
- By default, JavaScript moves all declarations to the top

Best Practices

Initialize Variables

It is a good coding practice to initialize variables when you declare them.

This will:

- Give cleaner code
- Provide a single place to initialize variables
- Avoid undefined values

Best Practices

Never Declare Number, String, or Boolean Objects

- Always treat numbers, strings, or booleans as primitive values. Not as objects.
- Declaring these types as objects, slows down execution speed, and produces nasty side effects

Best Practices

Don't Use new Object()

- Use {} instead of new Object()
- Use "" instead of new String()
- Use 0 instead of new Number()
- Use false instead of new Boolean()
- Use [] instead of new Array()
- Use /()/ instead of new RegExp()
- Use function (){} instead of new Function()

Best Practices

Don't Use new Object(): Examples

```
var x1 = {};           // new object
var x2 = "";           // new primitive string
var x3 = 0;            // new primitive number
var x4 =               // new primitive boolean
false;  var x5         // new array object
= [];  var x6          // new regexp object
= /()/;  var x7 = function(){}; // new function object
```

JavaScript Date Objects

JavaScript new Date()

Example:

- var d = new Date();

  Sat Aug 17 2019 18:46:55 GMT+0530 (India Standard Time)

- Date objects are created with the new Date() constructor.
- There are 4 ways to create a new date object:

  new Date()

  new Date(year, month, day, hours, minutes, seconds, milliseconds)

  new Date(milliseconds)

  new Date(date string)

JavaScript Date Formats

JavaScript Date Input

- There are generally 3 types of JavaScript date input formats:

Type	Example
ISO Date	"2015-03-25" (The International Standard)
Short Date	"03/25/2015"
Long Date	"Mar 25 2015" or "25 Mar 2015"