WHAT IS JAVASCRIPT?

- JavaScript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages and make dynamic pages.

- It is an interpreted programming language.
- JavaScript was first known as LiveScript, but Netscape changed its name to JavaScript.
- The general-purpose core of the language has been embedded in Netscape, Internet Explorer, and other web browsers.
- JavaScript is a lightweight, interpreted programming language.
- Complementary to and integrated with HTML.

OVERVIEW

- **Client-Side JavaScript**
- **Client-Side comprises all the machines, on which are installed web browser for browsing the internet from the Server.**

- Client-side scripting involves any computer program or code or scripts or software that is designed to be executed on a client-side web browser of client computer.
- Client-side JavaScript is the most common form of the language.
- The script should be included in or referenced by an HTML document for the code to be interpreted by the browser.
- It means that a web page need not be a static HTML, but can include programs that interact with the user, control the browser, and dynamically create HTML content.

- For example, you might use JavaScript to check if the user has entered a valid e-mail address in a form field.

- **Client-side Scripting is generally used in Dynamic HTML(DHTML) to generate the dynamic web pages(interactive web pages).If the web page is dynamic ,the requested web page needs to undergo preprocessing, before sending it to the user. Depending on who performs the processing,**

DIFFERANCE IN CLIENT SIDE AND SERVER SIDE SCRIPTING

CLIENT SIDE SCRIPTING

1. **Computer programs or Scripts are executed by web browser,** when server sends the requested web page's scripts to the browser. Server doesn't play any role in script execution.
2. The server code can be viewed in the browser and can be easily changed without any authentication

SERVER SIDE SCRIPTING

1. **Computer programs or Scripts are executed by web Server,** when user issues a request for certain web page to the server through browser. Browser doesn't play any role in script execution.
2. The Server side Script are protected with login authentication so only valid users are allowed to see the source code only after he/she has logged in with the server.

CLIENT SIDE SCRIPTING

- Client side Scripts requires web browse
- The speed f execution is faster as compared to server side scriptingrs as an interface.
- Client side scripts are less flexible compar.
- Client side script is basically written using JavaScript.ed to server –side scripts

SERVER SIDE SCRIPTING

- Server side Scripts requires web Server software to execute.
- The speed f execution is slower as compared to server side scripting.
- Server side scripts are more flexible and provide the developers with the large number of programming option.
- Server side scripts are basically written using PERL(Practical Extraction Report Language),VBScript,Python,C++,java,Js cript,etc.

SYNTAX

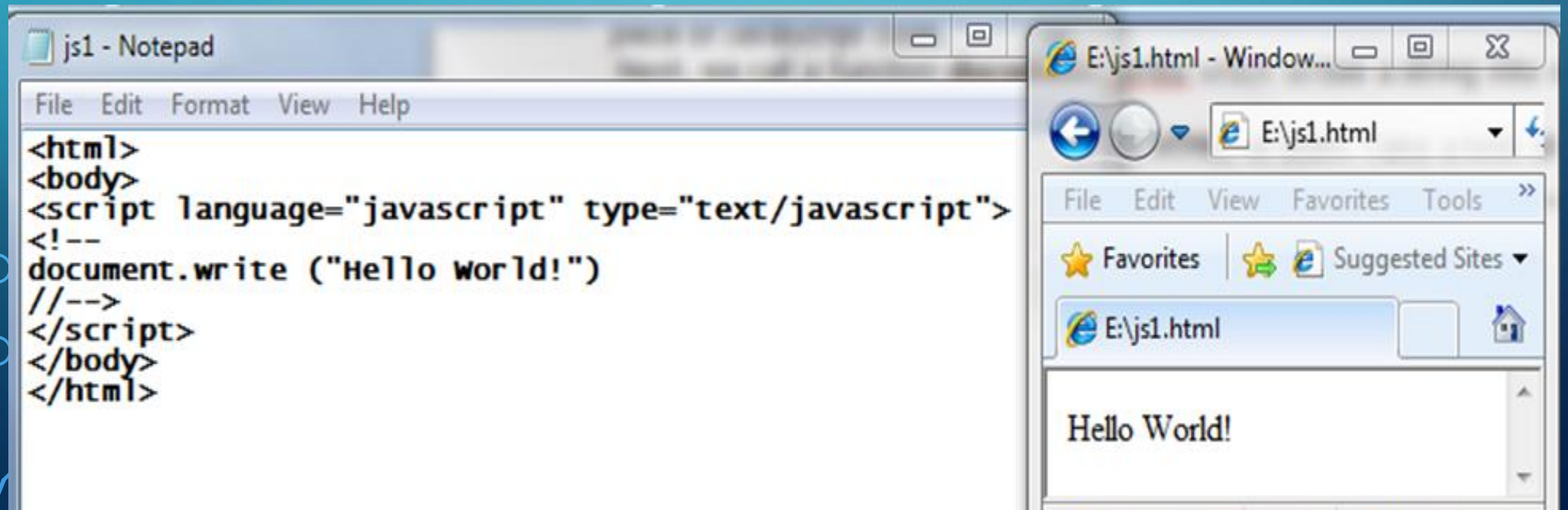
- JavaScript can be implemented using JavaScript statements that are placed within the **<script>... </script>** HTML tags in a web page.
- You can place the **<script>** tags, containing your JavaScript, anywhere within you web page, but it is normally recommended that you should keep it within the **<head>** tags.
- A simple syntax of your JavaScript will appear as follows.

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

- The script tag takes two important attributes:
-
- ☐ **Language:** This attribute specifies what scripting language you are using. Typically, its value will be javascript. Although recent versions of HTML (and XHTML, its successor) have phased out the use of this attribute.
-
- ☐ **Type:** This attribute is what is now recommended to indicate the scripting language in use and its value should be set to "text/javascript".
- So your JavaScript syntax will look as follows.
-
- <script language="javascript" type="text/javascript">
- JavaScript code
- </script>

YOUR FIRST JAVASCRIPTCODE

- Let us take a sample example to print out "Hello World". We added an optional. The comment ends with a "//-->". Here "//" signifies a comment in JavaScript, so we.
- Next, we call a function **document.write** which writes a string into our HTML document. **Javascript**



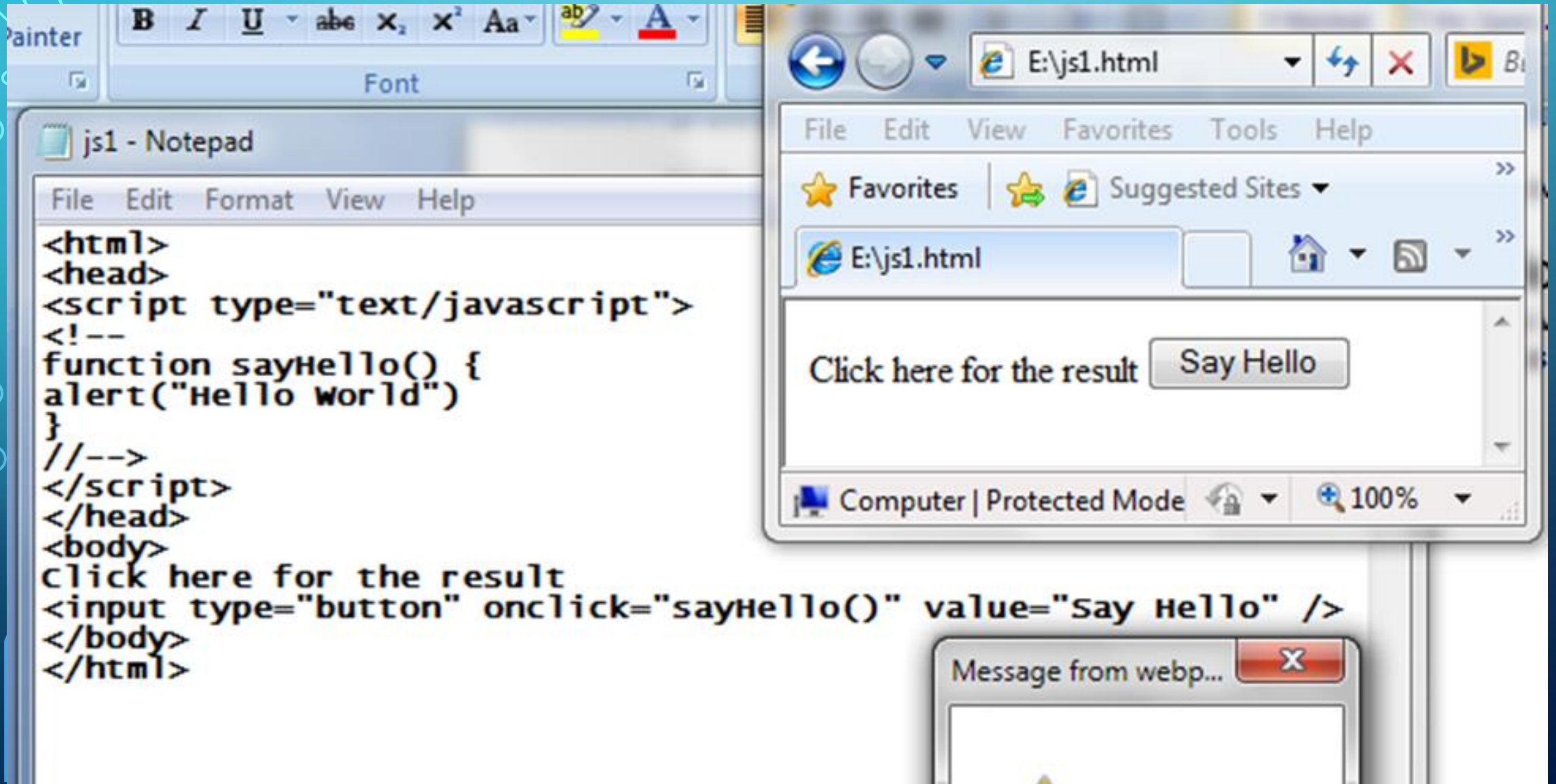
COMMENTS IN JAVASCRIPT

- JavaScript supports both C-style and C++-style comments. Thus:
- ☐ Any text between a // and the end of a line is treated as a comment and is ignored by JavaScript.
- ☐ Any text between the characters /* and */ is treated as a comment. This may span multiple lines.
- ☐ JavaScript also recognizes the HTML comment opening sequence <!--. JavaScript treats this as a single-line comment, just as it does the // comment.
- ☐ The HTML comment closing sequence --> is not recognized by JavaScript so it should be written as //-->.

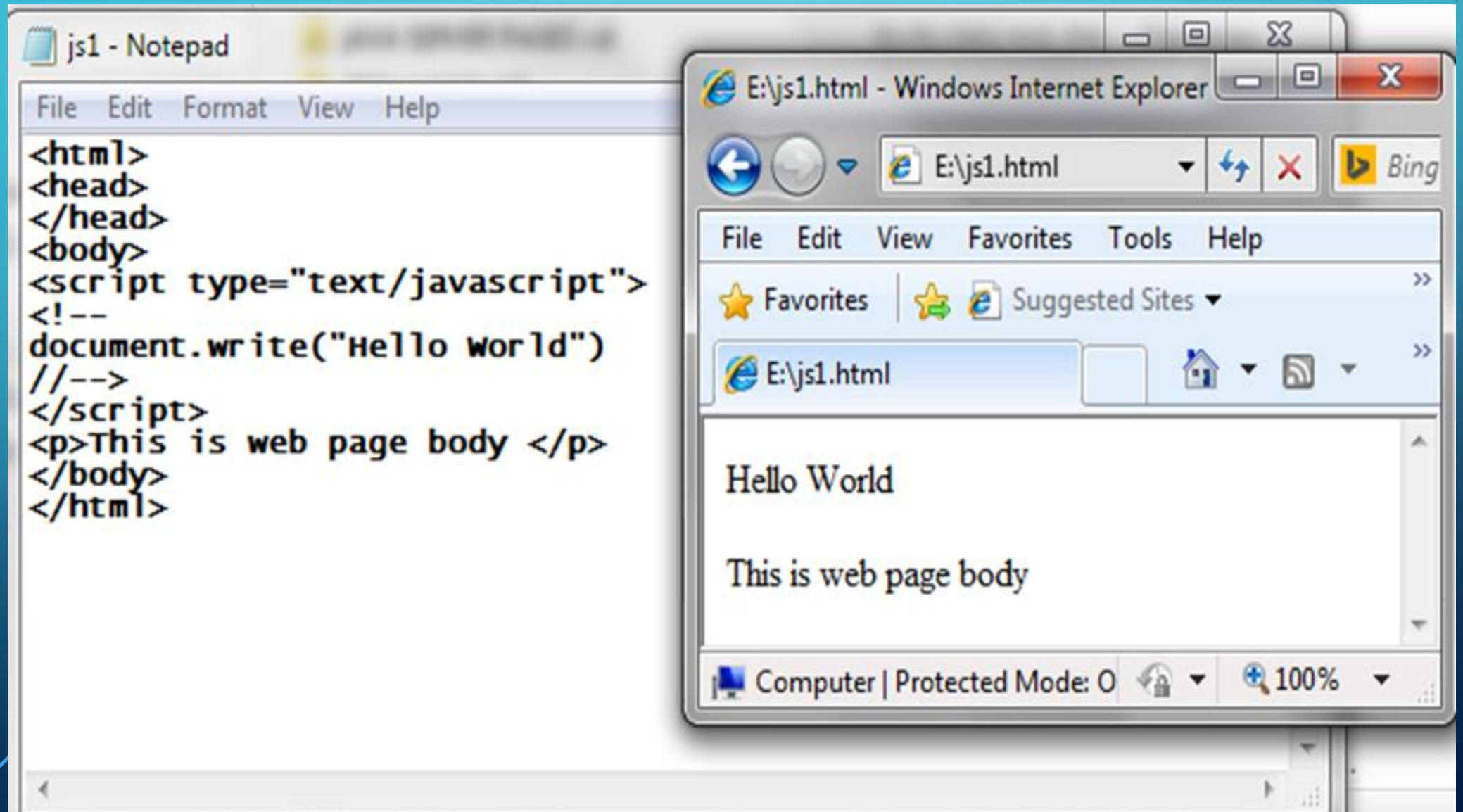
PLACEMENT

- There is a flexibility given to include JavaScript code anywhere in an HTML document. However the most preferred ways to include JavaScript in an HTML file are as follows:
-
- ☐ Script in <head>...</head> section.
-
- ☐ Script in <body>...</body> section.
-
- ☐ Script in <body>...</body> and <head>...</head> sections.
-
-

JAVASCRIPT IN <HEAD>...</HEAD> SECTION

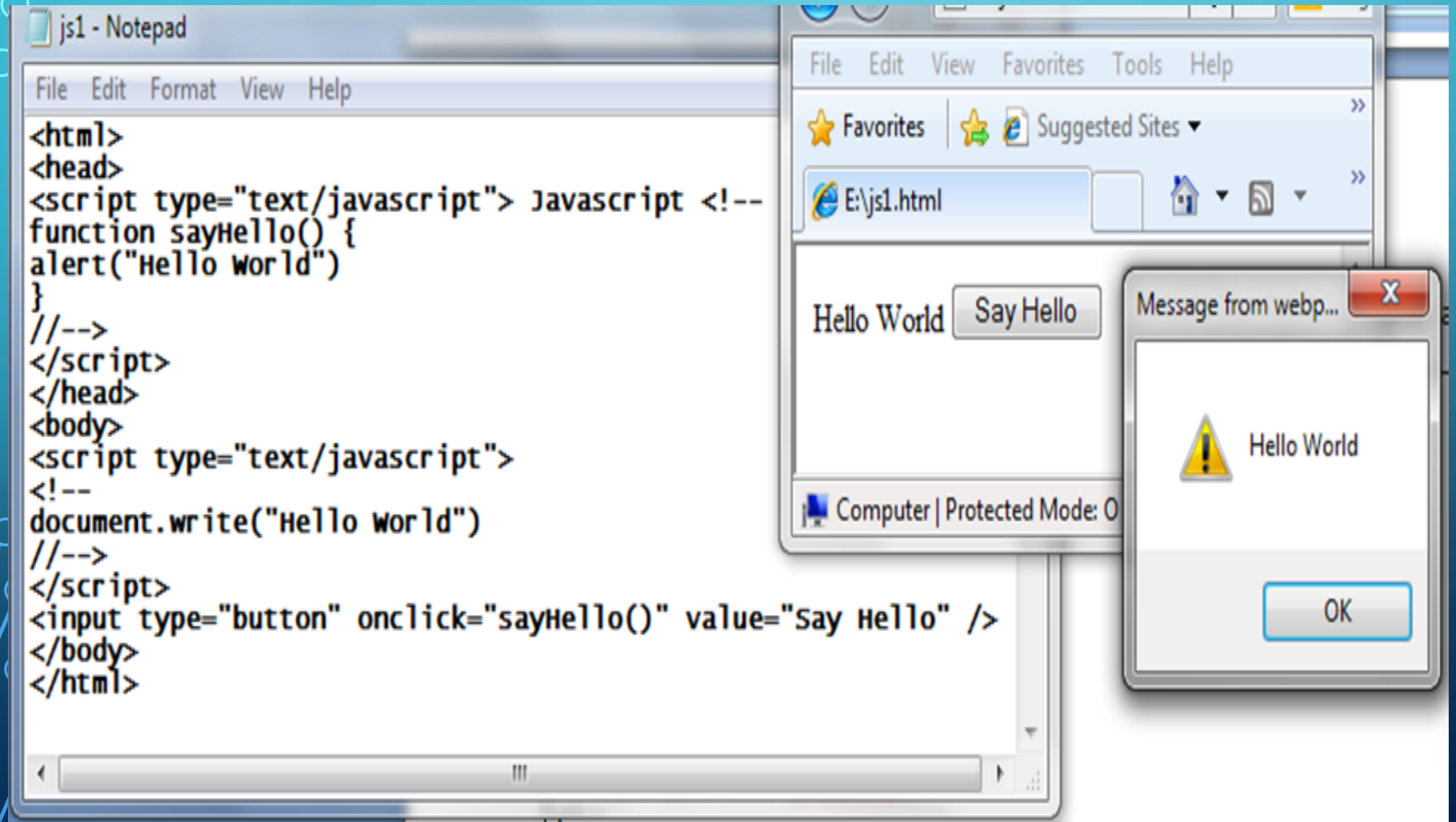


JAVASCRIPT IN <BODY>...</BODY> SECTION



JAVASCRIPT IN <BODY> AND <HEAD> SECTIONS

YOU CAN PUT YOUR JAVASCRIPT CODE IN <HEAD> AND <BODY> SECTION ALTOGETHER AS FOLLOWS.



VARIABLES
JAVASCRIPT DATA TYPES

- One of the most fundamental characteristics of a programming language is the set of data types it supports. These are the type of values that can be represented and manipulated in a programming language.
- JavaScript allows you to work with three primitive data types:
- **☐ Numbers, e.g., 123, 120.50 etc.**
- **☐ Strings of text, e.g. "This text string" etc.**
- **☐ Boolean, e.g. true or false.**
- JavaScript also defines two trivial data types**, null and undefined**, each of which defines only a single value. In addition to these primitive data types, JavaScript supports a composite data type known as **object**. We will cover objects in detail in a separate chapter.

JAVASCRIPT VARIABLES

- Like many other programming languages, JavaScript has variables. Variables can be thought of as named containers.
- Before you use a variable in a JavaScript program, you must declare it. Variables are declared with the **var** keyword as follows.

```
<script type="text/javascript">
<!--
var money;
var name;
//-->
</script>
```

JAVASCRIPT VARIABLE NAMES

- While naming your variables in JavaScript, keep the following rules in mind.
- ☐ You should not use any of the JavaScript reserved keywords as a variable name. These keywords are mentioned in the next section. For example, **break** or **boolean** variable names are not valid.
- ☐ JavaScript variable names should not start with a numeral (0-9). They must begin with a letter or an underscore character. For example, **123test** is an invalid variable name but **_123test** is a valid one.
- ☐ JavaScript variable names are case-sensitive. For example, **Name** and **name** are two different variables.

OPERATORS
WHAT IS AN OPERATOR?

- Let us take a simple expression **4 + 5 is equal to 9**. Here 4 and 5 are called **operands** and '+' is called the **operator**. JavaScript supports the following types of operators.
- ☐ Arithmetic Operators
- ☐ Comparison/Relational Operators
- ☐ Logical Operators
- ☐ Assignment Operators
- ☐ Conditional (or ternary) Operators
- Let's have a look at all the operators one by one.

JavaScript Operators

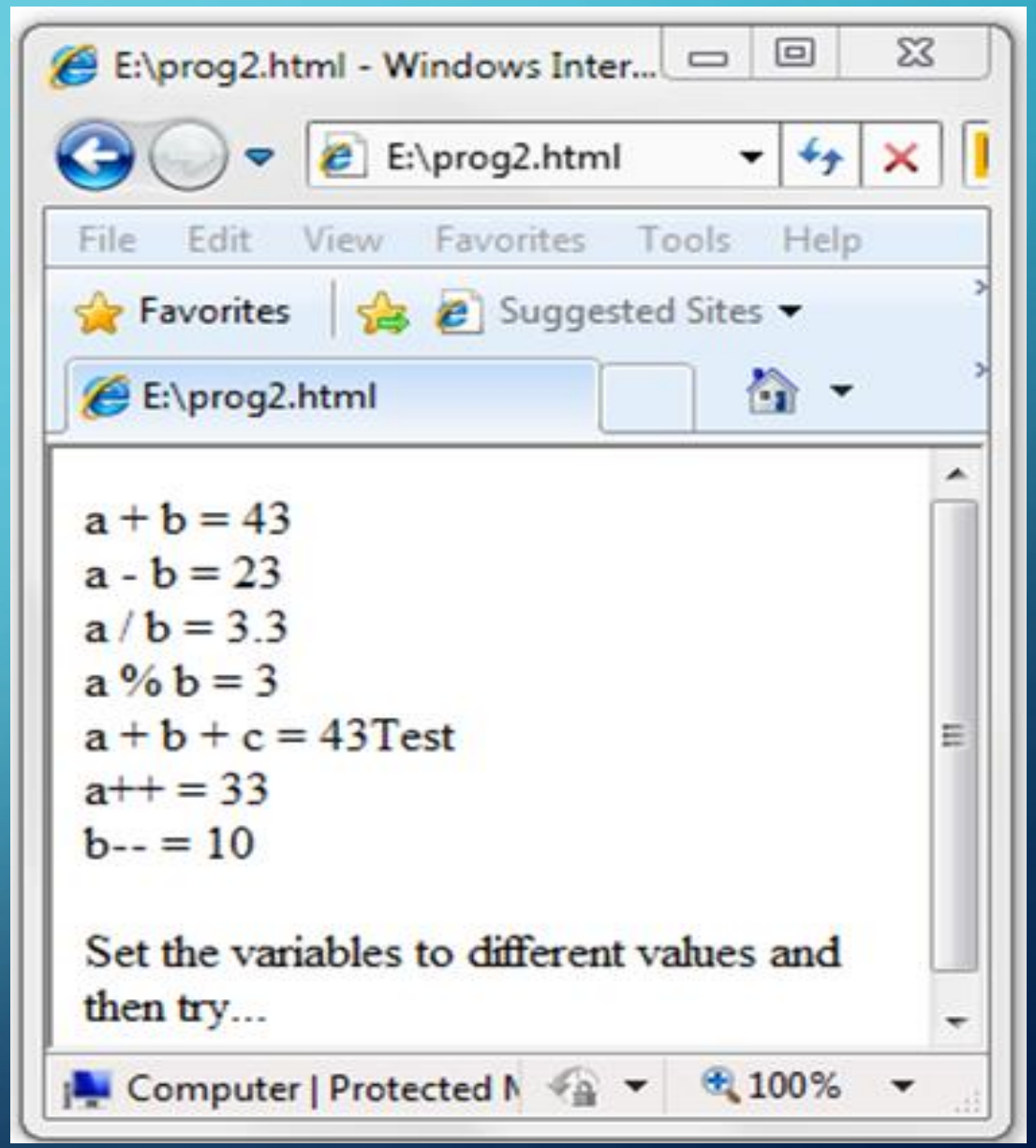
Arithmetic Operators

Operator	Description	Example	Result
+	Addition	x=2 y=2 x+y	4
-	Subtraction	x=5 y=2 x-y	3
*	Multiplication	x=5 y=4 x*y	20
/	Division	15/5 5/2	3 2,5
%	Modulus (division remainder)	5%2 10%8 10%2	1 2 0
++	Increment	x=5 x++	x=6
--	Decrement	x=5 x--	x=4

ARITHMETIC OPERATORS

```
<html>
<body>
<script
type="text/javascript">
<!--
var a = 33;
var b = 10;
var c = "Test";
var linebreak = "<br />";
document.write("a + b = ");
result = a + b;
document.write(result);
document.write(linebreak);
document.write("a - b = ");
result = a - b;
document.write(linebreak);
document.write("a / b = ");
result = a / b;
document.write(result);
document.write(linebreak);
document.write("a % b = ");
result = a % b;
document.write(result);
document.write(linebreak);
document.write("a + b + c = ");
result = a + b + c;
document.write(result);
document.write(linebreak);
```

```
a = a++;
document.write("a++ = ");
result = a++;
document.write(result);
document.write(linebreak);
b = b--;
document.write("b-- = ");
result = b--;
document.write(result);
document.write(linebreak);
//-->
</script>
<p>Set the variables to
different values and then
try...</p>
</body>
</html>
```



COMPARISON OPERATORS

- JavaScript supports the following comparison operators:
- Assume variable A
- holds 10 and
- variable B holds 20, then:

S.No	
1	**== (Equal)** Checks if the value of two operands are equal or not, if yes, then the condition becomes true. **Ex:** (A == B) is not true.
2	**!= (Not Equal)** Checks if the value of two operands are equal or not, if the values are not equal, then the condition becomes true. **Ex:** (A != B) is true.
3	**> (Greater than)** Checks if the value of the left operand is greater than the value of the right operand, if yes, then the condition becomes true. **Ex:** (A > B) is not true.
4	**< (Less than)** Checks if the value of the left operand is less than the value of the right operand, if yes, then the condition becomes true. **Ex:** (A < B) is true.
5	**>= (Greater than or Equal to)** Checks if the value of the left operand is greater than or equal to the value of the right operand, if yes, then the condition becomes true. **Ex:** (A >= B) is not true.
6	**<= (Less than or Equal to)** Checks if the value of the left operand is less than or equal to the value of the right operand, if yes, then the

LOGICAL OPERATORS

- JavaScript supports the following logical operators:
- Assume variable A holds 10 and variable B holds 20, then:

S.No	Operator and Description
1	**&& (Logical AND)** If both the operands are non-zero, then the condition becomes true. **Ex:** (A && B) is true.
2	**|| (Logical OR)** If any of the two operands are non-zero, then the condition becomes true. **Ex:** (A || B) is true.
3	**! (Logical NOT)** Reverses the logical state of its operand. If a condition is true, then the Logical NOT operator will make it false. **Ex:** ! (A && B) is false.

JAVASCRIPT POPUP BOXES

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

CONFIRM BOX

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

Example

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

PROMPT BOX

- A prompt box is often used if you want the user to input a value before entering a page.
- When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.
- If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.
- Syntax
- window.prompt("*sometext*","*defaultText*");

border_csss.html | box_css.html | Comparison_op.html | logical_op.html | pron

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

<h2>JavaScript Prompt</h2>

<button onclick="myFunction()">Try it</button>

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

<script>
function myFunction() {
  var txt;
  var person = prompt("Please enter your name:", "Harry Potter");
  if (person == null || person == "") {
    txt = "User cancelled the prompt.";
  } else {
    txt = "Hello " + person + "! How are you today?";
  }
  document.getElementById("demo").innerHTML = txt;
}
</script>

</body>
</html>
```

7. IF-ELSE

- JavaScript supports conditional statements which are used to perform different actions based on different conditions. Here we will explain the **if..else** statement.
- JavaScript supports the following forms of **if..else** statement:
-
- ☐ if statement
- ☐ if...else statement
-

- **1)if Statement**
- The '**if**' statement is the fundamental control statement that allows JavaScript to make decisions and execute statements conditionally.
- **Syntax**
- The syntax for a basic if statement is as follows:
- if (expression){
- Statement(s) to be executed if expression is true
- }

Example

Try the following example to understand how the **if** statement works.

```
<html>
<body>
<script type="text/javascript">
<!--
var age = 20;
if( age > 18 )
{
document.write("<b>Qualifies for driving</b>");
}
//-->
</script>
<p>Set the variable to different value and then try...</p>
</body>
</html>
```

- **2)if...else Statement**
- The '**if...else**' statement is the next form of control statement that allows JavaScript to execute statements in a more controlled way.
- **Syntax**
- The syntax of an **if-else** statement is as follows:

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

- **Example**
- Try the following code to learn how to implement an if-else statement in JavaScript.

```
<html>
<body>
<script type="text/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>
<p>Set the variable to different value and then try...</p>
```

Output

Does not qualify for driving

Set the variable to different value and then try...

SWITCH-CASE

- you can use a **switch** statement which handles exactly this situation, and it does so more efficiently than repeated **if...else if** statements.
- **Syntax**

```
switch (expression)
{
case condition 1: statement(s)
break;
case condition 2: statement(s)
break;
...
case condition n: statement(s)
break;
default: statement(s)
}
```

```
<html>
<body>
<script type="text/javascript">
<!--
var grade='A';
document.write("Entering switch block<br />");
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 />")
}
document.write("Exiting switch block");
//-->
</script>
```

Output

Entering switch block
Good job
Exiting switch block
Set the variable to different value and then try...

WHILE LOOP

- The most basic loop in JavaScript is the **while** loop which would be discussed in this chapter. The purpose of a **while** loop is to execute a statement or code block repeatedly as long as an **expression** is true. Once the expression becomes **false,** the loop terminates.

Syntax
The syntax of **while loop** in JavaScript is as follows:
while (expression){
Statement(s) to be executed if expression is true
}

Example

Try the following example to implement while loop.

```
<html>
<body>
<script type="text/javascript">
<!--
var count = 0;
document.write("Starting Loop ");
while (count <6){
document.write("Current Count : " + count + "<br />");
count++;
}
document.write("Loop stopped!");
//-->
</script>
</body>
</html>
```

Output

Starting Loop Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Current Count : 5
Loop stopped!
Set the variable to different value and then try...

THE DO...WHILE LOOP

- The **do...while** loop is similar to the **while** loop except that the condition check happens at the end of the loop. This means that the loop will always be executed at least once, even if the condition is **false**.
- **Syntax**
- The syntax for **do-while** loop in JavaScript is as follows:

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

Example

Try the following example to learn how to implement a **do-while** loop in JavaScript.

```
<html>
<body>
<script type="text/javascript">
<!--
var count = 0;
document.write("Starting Loop" + "<br />");
do{
document.write("Current Count : " + count + "<br
/>");
count++;
}while (count < 5);
document.write ("Loop stopped!");
//-->
</script>
<p>Set the variable to different value and then
try...</p>
</body>
</html>
```

Output

```
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Loop Stopped!
Set the variable to different
value and then try...
```

THE FOR LOOP

- The '**for**' loop is the most compact form of looping. It includes the following three important parts:
-
- ☐ The **loop initialization** where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.
-
- ☐ The **test statement** which will test if a given condition is true or not. If the condition is true, then the code given inside the loop will be executed, otherwise the control will come out of the loop.
-
- ☐ The **iteration statement** where you can increase or decrease your counter.
-
- You can put all the three parts in a single line separated by semicolons.
- **Syntax**

Syntax

The syntax of **for** loop is JavaScript is as follows:

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

Example

Try the following example to learn how a **for** loop works in JavaScript.

```
<html>
<body>
<script type="text/javascript">
<!--
var count;
document.write("Starting Loop" + "<br />");
for(count = 0; count <5; count++){
document.write("Current Count : " + count );
document.write("<br />");
}
document.write("Loop stopped!");
//-->
</script>
<p>Set the variable to different value and then try...</p>
</body>
</html>
```

Output

```
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Loop stopped!
Set the variable to different value and then try...
```

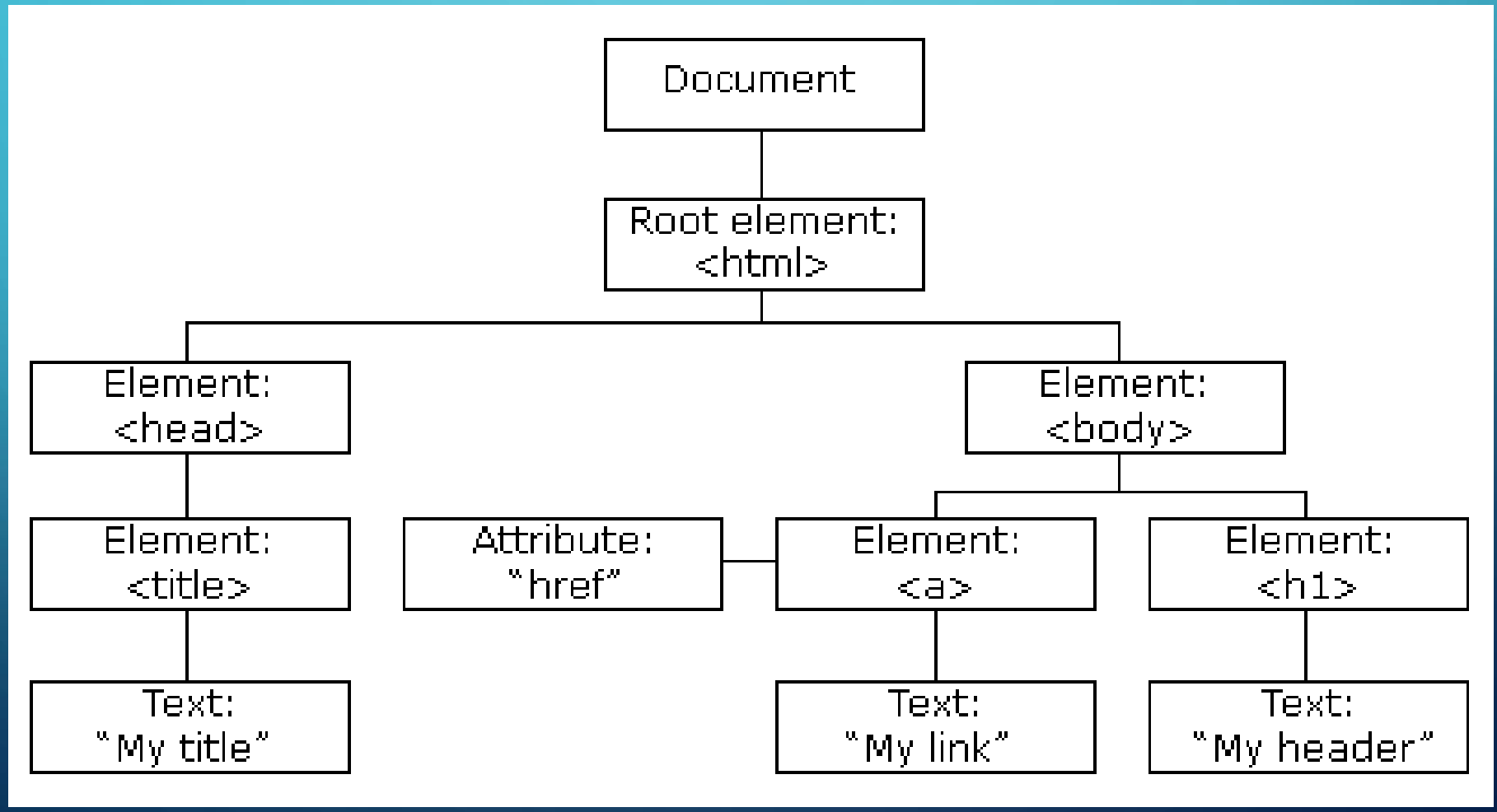
JavaScript HTML DOM

- With the HTML DOM, JavaScript can access and change all the elements of an HTML document.

The HTML DOM (Document Object Model)

- When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.
- The **HTML DOM** model is constructed as a tree of **Objects:**

HTML DOM


Document
Root element:
<html>
Element:
<head>
Element:
<body>
Element:
<title>
Attribute:
"href"
Element:
<a>
Element:
<h1>
Text:
"My title"
Text:
"My link"
Text:
"My header"

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

JAVASCRIPT - HTML DOM METHODS

- HTML DOM methods are **actions** you can perform (on HTML Elements).
- HTML DOM properties are **values** (of HTML Elements) that you can set or change.

The DOM Programming Interface

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

THE INNER HTML PROPERTY

- The easiest way to get the content of an element is by using the innerHTML property.

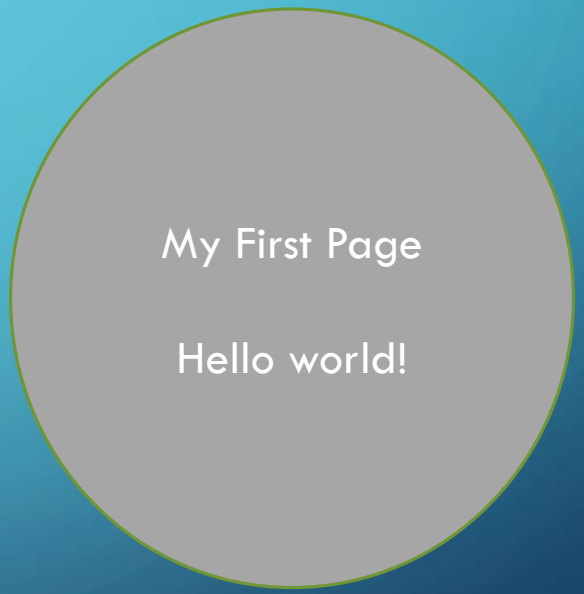
```
<!DOCTYPE html>
<html>
<body>

<h2>My First Page</h2>

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

<script>
document.getElementById("demo").innerHTML = "Hello World!";
</script>

</body>
</html>
```



THE HTML DOM DOCUMENT OBJECT

- The document object represents your web page.
- If you want to access any element in an HTML page, you always start with accessing the document object.
- Below are some examples of how you can use the document object to access and manipulate HTML.

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

CHANGING HTML ELEMENTS

Property	Description
element.innerHTML = *new html content*	Change the inner HTML of an element
element.attribute = *new value*	Change the attribute value of an HTML element
element.style.*property* = *new style*	Change the style of an HTML element
Method	Description
element.setAttribute(*attribute, value*)	Change the attribute value of an HTML element

FINDING HTML ELEMENTS

- There are several ways to do this:
- Finding HTML elements by id
- Finding HTML elements by tag name
- Finding HTML elements by class name
- Finding HTML elements by CSS selectors
- Finding HTML elements by HTML object collections

Finding HTML Element by Id

The easiest way to find an HTML element in the DOM, is by using the element id.
This example finds the element with id="intro":

Example

```
var myElement = document.getElementById("intro");
```

If the element is found, the method will return the element as an object (in myElement).
If the element is not found, myElement will contain `null`.

Finding HTML Elements by Tag Name

This example finds all <p> elements:

Example

```
var x = document.getElementsByTagName("p");
```

Finding HTML Elements by Class Name

If you want to find all HTML elements with the same class name, use getElementsByClassName().

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

Example

```
var x = document.getElementsByClassName("intro");
```

Finding elements by class name does not work in Internet Explorer 8 and earlier versions.

Finding HTML Elements by CSS Selectors

If you want to find all HTML elements that match a specified CSS selector (id, class names, types, attributes, values of attributes, etc), use the querySelectorAll() method.

This example returns a list of all <p> elements with class="intro".

Example

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

Finding HTML Elements by HTML Object Collections

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

Example

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

JavaScript Events

HTML events are **"things"** that happen to HTML elements.
When JavaScript is used in HTML pages, JavaScript can **"react"** on these events.

HTML Events

An HTML event can be something the browser does, or something a user does.
Here are some examples of HTML events:

- An HTML web page has finished loading
- An HTML input field was changed
- An HTML button was clicked

HTML allows event handler attributes, **with JavaScript code**, to be added to HTML elements.
With single quotes:

```
<element event='some JavaScript'>
```

With double quotes:

```
<element event="some JavaScript">
```

In the following example, an onclick attribute (with code), is added to a <button> element:

Example

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

Common HTML Events

Here is a list of some common HTML events:

Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

. FUNCTIONS

- A function is a group of reusable code which can be called anywhere in your program. This eliminates the need of writing the same code again and again. It helps programmers in writing modular codes.
- Functions allow a programmer to divide a big program into a number of small and manageable functions.

- **Function Definition**
- Before we use a function, we need to define it. The most common way to define a function in JavaScript is by using the **function** keyword, followed by a unique function name, a list of parameters (that might be empty), and a statement block surrounded by curly braces.

Syntax

The basic syntax is shown here.

```
<script type="text/javascript">
<!--
function functionname(parameter-list)
{
statements
}
//-->
</script>
```

Example

Try the following example. It defines a function called sayHello that takes no parameters:

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

- **Calling a Function**

To invoke a function somewhere later in the script, you would simply need to write the name of that function as shown in the following code.

```
<html>
<head>
<script type="text/javascript">
function sayHello()
{
document.write ("Hello there!");
}
</script>
</head>
<body>
<p>Click the following button to call the function</p>
<form>
<input type="button" onclick="sayHello()" value="Say Hello">
</form>
<p>Use different text in write method and then try...</p>
</body>
</html>
```

- **Function Parameters**
- Till now, we have seen functions without parameters. But there is a facility to pass different parameters while calling a function. These passed parameters can be captured inside the function and any manipulation can be done over those parameters. A function can take multiple parameters separated by comma.

- **Example**
- Try the following example. We have modified our **sayHello** function here. Now it takes two parameters.

```
<html>
<head>
<script type="text/javascript">
function sayHello(name, age)
{
document.write (name + " is " + age + " years
old.");
}
</script>
</head>
<body>
<p>Click the following button to call the function</p>
<form>
<input type="button" onclick="sayHello('Zara', 7)"
value="Say Hello">
</form>
<p>Use different parameters inside the function and
then try...</p>
</body>
```