**All the examples of code are given, also see the code.**

**43. Placing scripts**

You include a JavaScript file in an HTML file the same way you include an external CSS file—with an opening and closing tag.

<script src="whatever.js"></script>

For the same reason that it's a good idea to put JavaScript code at the end of the body section, it's a good idea to place the markup that includes JavaScript files at the end of the body section. You can include as many external JavaScript files as you like. You can include JavaScript files and also embed other JavaScript code in the HTML.

**44. Commenting**

Single line comment double forward slash

e.g. // This is a comment, ignored by the browser

JavaScript multi-line comment tags are the same as CSS comment tags. Open with /\*. Close with \*/.

e.g. /\* This is a comment,

ignored by the browser \*/

**Events**

What are events in JavaScript?

JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page. When the page loads, it is called an event. When the user clicks a button, that click too is an event. Other examples include events like pressing any key, **closing** a window, resizing a window, etc.

**Most important thing about event you should keep in mind:**

**You can trigger any Function call with any events ‘**see example of chap 46 and 48**’, and you can directly target the most of the attributes of that element where you are putting the event ‘**see example of chap 47**’, and you can also target other element with their id’s or tag name or class Name’s etc.**

**45. Events: link**

Inline event-handling means that you combine bits of JavaScript with HTML markup. Here's a line that displays a link and then displays an alert when the user clicks it.

<a href="#" onClick="alert('Hi');">Click</a>

onClick= says, "When the button is clicked, execute the following JavaScript." onClick isn't case-sensitive. You could write onclick, ONCLICK, or OnClIcK and it would work. But the convention is onClick, so we'll stick with that.

href=“JavaScript:void(0)”

Simple **JavaScript Void 0** Simple Example. **void** is an operator that is used to return a null value so the browser will not be able to load a new page. An important thing to note about the **void** operator is that it requires a value and cannot be used by itself. Here is a simple way to use **void** to cancel out the page load

**46. Events: button**

onClick

<input type="button" value="Click" onClick="alert('Hello world!');">

<input type="button" value="val" onClick="asd();">

function asd(){

alert(“asd”);

}

**47. Events: mouse**

onMouseover

<img src="before-pic.jpg" onMouseover="src='after-pic.jpg'">

**48. Events: fields**

onFocus and onBlur

<input type="text" size="30" onFocus="this.style.backgroundColor='yellow';">

Here ‘this.style‘ mean is its style.

<input type="text" size="30" onFocus="makeFieldYellow();" onBlur="makeFieldWhite();">

**49. Reading field values**

document.getElementById(“id”)

document:

In JavaScript when you write document, this will represent the html file where you have imported or written in script tag this JavaScript file.

getElementById:

now you can get element by its id wit this method.

e.g. <input type="text" id="email"> you will get this with document.getElementById(“email”). Now you have the whole element whatever you will do with this, you can do with their defined methods. document.getElementById(“email”).value will give you the value of that element which has id=”email”;

you can save this value in variable like this, var asd = document.getElementById(“email”).value;

**50. Setting field values**

you can also change its value like this, document.getElementById(“email”).value = “something”;

IDs are always unique so when you getting and setting the values you should use getElementById.

**51. Reading and setting paragraph text**

As we discussed in before chapters about document.getElementById(“id”);

<p id=”content” >hello world</p>

Line 1: var whatsThere = document.getElementById("content").innerHTML;

innerHTML will get its value whatever in opening and closing tag.

Now the variable name whatsThere is equal to “hello world”;

Line 1 reads all the HTML within the element with an id of "content" and assigns it to the variable whatsThere. If the element is, for example, a paragraph whose content is " <em>Hello, Sailor! </em>" the variable whatsThere captures "<em>Hello, Sailor! </em>"

You can also set this thing as we did in chap 50,

document.getElementById("content").innerHTML = “asd”

**52. Manipulating images and text**

Now the other example as we discussed before, see 49 chap;

This method will add the additional class in your element.

document.getElementById("p1").className = "changeColor";

CSS: .changeColor {

background-color: “red”;

}

document.getElementById("p1").className += " changeColor";

Compared with the code that replaces all the existing classes with a new one, this code has two small differences.

1. It's += instead of just =
2. A space before the class name is required.

**53. Swapping images, 54. Swapping images and setting classes and 55. Setting style**

It’s only using the onMouseover event, targeting attributes and setting classes etc., nothing new, you can see the code.

document.getElementById("p1").style.fontSize = "2em";

**56 Target all elements by tag name**

As we discussed before in chap 49, now you also have the method getElementsByTagName;

You can target the element by its name. p, h1, img, ul, div etc.

var par = document.getElementsByTagName("p");

Now the variable par has an array of element p (paragraph); **all the p element of your document will save in par variable like an array**. Now you can target one by one.

par[0].innerHTML = "This SUV is too small.";

par[1].innerHTML = "This SUV is too big.";

**57 Target some elements by tag name**

Some elements by tag name, but in previous chap you getting all “p” elements of your document, because we had document reference.

But we can give the specific box (dive, p, ul, table, etc.) reference then we can get all of its elements.

e.g.

if we do as previous: var par = document.getElementsByTagName("p");

we have par array with 7 length;

<body>

<p>hello</p>

<p>hi</p>

<p>whatever</p>

<div id=”divRef”>

<p>asd</p>

<p>abc</p>

<p>xyz</p>

<p>ssda</p>

</div>

</body>

Exp 1

Line 1) var e = document.getElementByID("divRef ");

Line 2) var paragraphs = e.getElementsByTagName("p");

or

Exp 2

var paragraphs = document.getElementByID("divRef ").getElementsByTagName("p");

now we have paragraphs array with 4 length;

hope you get this, why we have 4 length’s array..

because at Line 1 we have div reference. Then line 2 we are going inside the box.