Module - 3( document , events , controls )

1. **Documents Object in javascript : -**

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 |

## Adding and Deleting Elements

| **Method** | **Description** |
| --- | --- |
| document.createElement(*element*) | Create an HTML element |
| document.removeChild(*element*) | Remove an HTML element |
| document.appendChild(*element*) | Add an HTML element |
| document.replaceChild(*new, old*) | Replace an HTML element |
| document.write(*text*) | Write into the HTML output stream |

## Adding Events Handlers

| **Method** | **Description** |
| --- | --- |
| document.getElementById(*id*).onclick = function(){*code*} | Adding event handler code to an onclick event |

1. **Events in javascript**

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

Often, when events happen, you may want to do something.

JavaScript lets you execute code when events are detected.

HTML allows event handler attributes, with JavaScript code, to be added to HTML elements.

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

## JavaScript Event Handlers

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 ...

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 …

Their is some events in Html using javascript.

| **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 |

## 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 |

## Window/Document events

| **Event Performed** | **Event Handler** | **Description** |
| --- | --- | --- |
| load | onload | When the browser finishes the loading of the page |
| unload | onunload | When the visitor leaves the current webpage, the browser unloads it |
| resize | onresize | When the visitor resizes the window of the browser |

1. Controls in javascript

Their is some type of controls :

* Control Loop
* Control Flow
* Control List

The control structures within JavaScript allow the program flow to change within a unit of code or function. These statements can determine whether or not given statements are executed - and provide the basis for the repeated execution of a block of code.

Most of the statements listed below are so-called conditional statements that can operate either on a statement or on a block of code enclosed with braces ({ and }). The structure provided by the use of conditional statements utilizes Booleans to determine whether or not a block gets executed. In this use of Booleans, any defined variable that is neither zero nor an empty string will be evaluated as true.

### **if**

The if statement is straightforward — if the given expression is true, the statement or statements will be executed. Otherwise, they are skipped.

**if** (a === b) {

document.body.innerHTML += "a equals b";

}

The if statement may also consist of multiple parts, incorporating else and else if sections. These keywords are part of the if statement, and identify the code blocks that are executed, if the preceding condition is false.

**if** (a === b) {

document.body.innerHTML += "a equals b";

} **else** **if** (a === c) {

document.body.innerHTML += "a equals c";

} **else** {

document.body.innerHTML += "a does not equal either b or c";

}

### **while**

The while statement executes a given statement as long as a given expression is true. For example, the code block below will increase the variable c to 10:

**while** (c < 10) {

c += 1;

*// …*

}

This control loop also recognizes the break and continue keywords. The break keyword causes the immediate termination of the loop, allowing for the loop to terminate from anywhere within the while block.

The continue keyword finishes the current iteration of the while block or statement, and checks the condition to see, if it is true. If it is true, the loop commences again.

### **do … while**

The do … while statement executes a given statement as long as a given expression is true - however, unlike the while statement, this control structure will always execute the statement or block at least once. For example, the code block below will increase the variable c to 10:

**do** {

c += 1;

} **while** (c < 10);

As with while, break and continue are both recognized and operate in the same manner. In other words, break exits the loop, and continue checks the condition before attempting to restart the loop.

### **for**

The for statement allows greater control over the condition of iteration. While it has a conditional statement, it also allows a pre-loop statement, and post-loop increment without affecting the condition. The initial expression is executed once, and the conditional is always checked at the beginning of each loop. At the end of the loop, the increment statement executes before the condition is checked once again. The syntax is:

for (<initial expression>;<condition>;<final expression>)

The for statement is usually used for integer counters:

**var** c;

**for** (c = 0; c < 10; c += 1) {

*// …*

}

While the increment statement is normally used to increase a variable by one per loop iteration, it can contain any statement, such as one that decreases the counter.

break and continue are both recognized. The continue statement will still execute the increment statement before the condition is checked.

A second version of this loop is the for .. in statement that has following form:

**for** (element **in** object) {

*// …*

}

The order of object elements accessed by this version is arbitrary. For instance, this structure can be used to loop through all the properties of an object instance. It should not be used when the object is of [Array](https://en.wikibooks.org/wiki/JavaScript/Arrays) type

## switch

The switch statement evaluates an expression, and determines flow control based on the result of the expression:

**switch**(i) {

**case** 1:

*// …*

**break**;

**case** 2:

*// …*

**break**;

**default**:

*// …*

**break**;

}