Unit-2: Advanced JavaScript–II

String Object, Math Object, Date Object, Introduction to DOM, Accessing Form Elements, Event Handling

STRING OBJECT:

JavaScript draws a fine line between a string value and a string object. Both let you use the same methods on their contents, so that by and large, you do not have to create a string object with the new String() every time you want to assign a string value to a variable. A simple assignment operation is all you need to create a string value that behaves on the surface very much like a full-fledged string object. Some situations demand explicitly creating a String object.

SYNTAX:

```
var variable_name=new String(Some string or value);
```

Example:

```
var a=new String("Hello");
var b=new String(123);
alert(a);
alert(b);                                   OUTPUT: Hello    123
```

Property of String object:

1. **length:**

   This property returns the number of characters in the string. Spaces and punctuation symbols count as characters. This is a read only property.

   SYNTAX:

   ```
   object_variable.length
   ```

   Example:

   ```
   var a=new String("Hello"), b="FYBCA";
   alert(a.length);
   alert(b.length);                         OUTPUT: 5        5
   ```

Methods of String object:

1. **toLowerCase():**

   If you want to change the case of a string to lower case (to remove case sensitivity when comparing strings, etc.), this method is used. This method returns a string after converting it into lower case letters. Any non-alphabetical characters remain unchanged by this method.

   SYNTAX:

   ```
   some_var=string_var.toLowerCase();
   ```

   Example:

   ```
   var a=new String("Hello"), b="Fybca", c, d;
   c= a.toLowerCase();
   d= b.toLowerCase();
   alert(c);
   alert(d);                        OUTPUT: hello        fybca
   ```

2. **toUpperCase():**

   If you want to change the case of a string to upper case (to remove case sensitivity when comparing strings, etc.), this method is used. This method returns a string after converting it into upper case letters. Any non-alphabetical characters remain unchanged by this method.

   SYNTAX:

some_var=string_var.toLowerCase();

Example:

```
var a=new String("Hello"), b="Fybca", c, d;
c= a.toUpperCase();
d= b.toUpperCase();
alert(c);
alert(d);                    OUTPUT: HELLO    FYBCA
```

3. **charAt():**

If you want to extract a single character within a string, this method is used. It accepts one parameter which is the index position of the character you want to extract from the string. This method treats the position of the string characters as starting at 0. This function returns the character at that index position.

SYNTAX:

some_var=string_var.charAt(index_position);

Example:

```
var a=new String("Hello"), b="Fybca", c, d;
c=a.charAt(0);
d=b.charAt(b.length-1);
alert(c);
alert(d);                          OUTPUT: H        a
```

4. **indexOf():**

This method is used for searching for the occurrence of one string inside another. A string contained inside another is usually termed a substring. This function has two parameters: The string you want to find and The character position you want to start searching from (optional). The character position in the string starts at 0. If the second parameter is missing, searching starts from the beginning of the string (index 0). The return value is the character position in the string at which the substring was found. If no match was there, then -1 is returned. This function is case sensitive i.e. upper and lower case alphabets are considered different.

SYNTAX:

some_var=string_var.indexOf(index_position, start_search_position);

Example:

```
var a=new String("Hello"), b, c, d;
b=a.indexOf("e");
c=b.indexOf("e",2);
d=a.indexOf("E");
alert(b);
alert(c);
alert(d);                    OUTPUT: 1    -1    -1
```

5. **substr():**

If you want to cut out part of a string and assign that cut-out part to another variable or use it in an expression, this method is used. The first parameter indicates the start position of the first character you want included in your substring. The second parameter specifies the length of the string of characters that you want to cut out from the original string. The original string remains as it is.

SYNTAX:

some_var=string_var.substr(start_position, number_of_characters);

Example:

```
var a=new String("Hello"), b, c, d;
b=a.substr(0,2);
c=a.substr(3,2);
alert(b);
alert(c);                          OUTPUT: He    lo
```

MATH OBJECT:

This object provides a number of useful mathematical functions and number manipulation methods. The Math object is automatically created by JavaScript and does not need to be created. So there is no need to declare a variable as a Math object or define a new Math object before being able to use it. The properties of the Math object include some useful math constants like the PI property, etc. The methods of the Math object include some operations that are impossible or complex to perform using the standard mathematical operators.

Property of Math object:

1. **PI:**
   This property returns the value of Π (pi) which is approx. 3.14159. This is a read only constant property.
   SYNTAX:
   Math.PI
   Example:

```
var area_circle=0, radius_circle=2;
area_circle=Math.PI * pow(radius_circle,2);
alert(area_circle);                    OUTPUT: 12.566370614359172
```

Methods of Math object:

1. **abs():**
   This method returns the absolute value of the number passed as its parameter.Essentially, this means that it returns the positive value of the number.
   SYNTAX:
   Math.abs(some_var)
   Example:

```
var a=10, b=-5, c, d;
c= Math.abs(a);
d= Math.abs(b);
alert(c);
alert(d);                              OUTPUT: 10        5
```

2. **ceil():**
   This method rounds a number up to the next largest whole number or integer. This function rounds up. If the number is a whole number then the same number is returned. If it has a fraction, then the whole number next to it on the right side is returned.
   SYNTAX:
   Math.ceil(some_var)
   Example:

```
var a=10.5, b=15, c=-9.5, d, e, f;
d= Math.ceil(a);
e= Math.ceil(b);
f= Math.ceil(c);
alert(d);
alert(e);
alert(f);                      OUTPUT: 11        15        -9
```

3. **floor():**
   This method removes the fraction part of the number and returns a whole number. This function rounds down. If the number is a whole number then the same number is returned. If it has a fraction, then the whole number next to it on the left side is returned.
   SYNTAX:
   Math.floor(some_var)
   Example:

```
var a=10.5, b=15, c=-9.5, d, e, f;
```

```
d= Math.floorl(a);
e= Math.floor(b);
f= Math.floor(c);
alert(d);
alert(e);
alert(f);                    OUTPUT: 10    15    -10
```

4. **max():**
   This method finds the maximum of the list of parameters provided. It requires minimum two parameters. The parameters must be numbers either whole numbers or floating point numbers.

   SYNTAX:

   ```
   Math.max(some_var1, some_var2,..., some_varN)
   ```

   Example:

   ```
   var a=10.5, b=10, c=9.5, d;
   d= Math.max(a,b,c);
   alert(d);                    OUTPUT: 10.5
   ```

5. **min():**
   This method finds the minimum of the list of parameters provided. It requires minimum two parameters. The parameters must be numbers either whole numbers or floating point numbers.

   SYNTAX:

   ```
   Math.min(some_var1, some_var2,..., some_varN)
   ```

   Example:

   ```
   var a=10.5, b=10, c=9.5, d;
   d= Math.min(a,b,c);
   alert(d);                    OUTPUT: 9.5
   ```

6. **round():**
   This method return the round of a number to the nearest integer. It round up only if the decimal part is 5 or greater than 5 otherwise it return rounded down value. If the number is an integer, then it returns the same number.

   SYNTAX:

   ```
   Math.round(some_var)
   ```

   Example:

   ```
   var a=10.7, b;
   b= Math.round(a);
   alert(b);                    OUTPUT: 11
   ```

DATE OBJECT:

This object handles everything to do with date and time in JavaScript. These methods are useful in manipulating one parameter of the six date and time parameters. You can either fetch a parameter or modify a parameter using these methods. The Date object is internally stored as a number which represents the number of milliseconds from 1st January 1970 00:00:00.00 GMT. The Date object methods are divided into two groups depending on whether it returns a value or modifies a value namely get and set methods. The get methods return a parameter value whereas the set methods modify one parameter of the object itself.

SYNTAX:

```
var variable_name=new Date(Some_value or EMPTY)
```

Example:

```
var a=new Date();
var b=new Date("Feb 15 2010 11:10:45");          OUTPUT:
alert(a);                Sat Feb 12 13:42:42 UTC+0530 2011
```

```
alert(b);                    Tue Feb 15 11:10:45 UTC+0530 2011
```

Methods of Date object:

1. **getDate():**

   This method fetches the day number of the month between 1 and 31. It requires no parameter.

   SYNTAX:

   ```
   some_var.getDate()
   ```

   Example:

   ```
   var a=new Date();
   var b=new Date("Feb 15 2010 11:10:45");
   var c,d;
   c=a.getDate();
   d=b.getDate();
   alert(c);
   alert(d);                    OUTPUT:  12        15
   ```

2. **getDay():**

   This method fetches the day number of the week between 0 and 6. It requires no parameter. 0 indicates Sunday, 1 indicates Monday,... 6 indicates Saturday.

   SYNTAX:

   ```
   some_var.getDay()
   ```

   Example:

   ```
   var a=new Date();
   var b=new Date("Feb 15 2010 11:10:45");
   var c,d;
   c=a.getDay();
   d=b.getDay();
   alert(c);
   alert(d);                    OUTPUT:  6         3
   ```

3. **getFullYear():**

   This method fetches the 4-digited Year number. It requires no parameter.

   SYNTAX:

   ```
   some_var.getFullYear()
   ```

   Example:

   ```
   var a=new Date();
   var b=new Date("Feb 15 2010 11:10:45");
   var c,d;
   c=a.getFullYear();
   d=b.getFullYear();
   alert(c);
   alert(d);                OUTPUT:  2011      2010
   ```

4. **getMonth():**

   This method fetches the month number of the year between 0 and 11. It requires no parameter. 0 indicates January, 1 indicates February,... 11 indicates December.

   SYNTAX:

   ```
   some_var.getMonth()
   ```

   Example:

   ```
   var a=new Date();
   var b=new Date("Mar 15 2010 11:10:45");
   var c,d;
   c=a.getMonth();
   d=b.getMonth();
   alert(c);
   ```

```
alert(d);                    OUTPUT:  1        2
```

5. **getTime():**
   This method is used to fetch the number of milliseconds that has passed since 1st January 1970 00:00:00.0 GMT. It returns a 13-digited number representing the number of milliseconds.

   SYNTAX:

   ```
   some_var.getTime()
   ```

   Example:

   ```
   var a=new Date();
   var b=new Date("Mar 15 2010 11:10:45");
   var c,d;
   c=a.getTime();
   d=b.getTime();
   alert(c);                    OUTPUT:  1294996119703
   alert(d);                             1268631645000
   ```

6. **getHours():**
   This method fetches the hour number of the day between 0 and 23. The system hours are considered in 24 hour format.

   SYNTAX:

   ```
   some_var.getHours()
   ```

   Example:

   ```
   var a=new Date();
   var b=new Date("Mar 15 2010 11:10:45");
   var c,d;
   c=a.getHours();
   d=b.getHours();
   alert(c);
   alert(d);                    OUTPUT:  14       11
   ```

7. **getMinutes():**
   This method fetches the Number of minutes passed in the given hour between 0 and 59.

   SYNTAX:

   ```
   some_var.getMinutes()
   ```

   Example:

   ```
   var a=new Date();
   var b=new Date("Mar 15 2010 11:10:45");
   var c,d;
   c=a.getMinutes();
   d=b.getMinutes();
   alert(c);
   alert(d);                    OUTPUT:  42       10
   ```

8. **getSeconds():**
   This method fetches the Number of seconds passed in the given minute between 0 and 59.

   SYNTAX:

   ```
   some_var.getSeconds()
   ```

   Example:

   ```
   var a=new Date();
   var b=new Date("Mar 15 2010 11:10:45");
   var c,d;
   c=a.getSeconds();
   d=b.getSeconds();
   ```

```
alert(c);
alert(d);                                   OUTPUT:  21      45
```

9. setDate():

This method is used to change the date number i.e. 1 to 31 for the given date value. All the other parameters of the Date remain same except for the Day. If the Date number exceeds beyond the maximum allowable days in that month then the month rolls over to the next and appropriate date in that month is set.

SYNTAX:

```
some_var.setDate(date_number)
```

Example:

```
var a=new Date();
var b=new Date("Mar 15 2010 11:10:45");
a.setDate(15);
b.setDate(20);                          OUTPUT:
alert(a);           Tue Feb 15 14:52:41 UTC+0530 2011
alert(b);           Sat Mar 20 11:10:45 UTC+0530 2010
```

10. setFullYear():

This method is used to change the Year number for the given date value. All the other parameters of the Date remain same except for the Day.

SYNTAX:

```
some_var.setFullYear(year_number)
```

Example:

```
var a=new Date();
var b=new Date("Mar 15 2010 11:10:45");
a.setFullYear(2000);
b.setFullYear(2012);                          OUTPUT:
alert(a);           Fri Jan 14 14:58:52 UTC+0530 2000
alert(b);           Thu Mar 15 11:10:45 UTC+0530 2012
```

11. setMonth():

This method is used to change the month number i.e. 0 to 11 for the given date value. All the other parameters of the Date remain same except for the Day. If the Month number exceeds beyond the maximum allowable months in a year then the year rolls over to the next and appropriate date and month in the next year is set.

SYNTAX:

```
some_var.setMonth(month_number)
```

Example:

```
var a=new Date();
var b=new Date("Mar 15 2010 11:10:45");
a.setMonth(11);
b.setMonth(2);                                OUTPUT:
alert(a);          Mon Dec 12 15:02:15 UTC+0530  2011
alert(b);            Mon Mar 15 11:10:45 UTC+0530 2010
```

12. setTime():

This method is used to set number of milliseconds since 1st January 1970 00:00:00.0 GMT.

SYNTAX:

```
some_var.setTime (milliseconds_number)
```

Example:

```
var a=new Date();
a.setTime(1234567890123);
alert(a);           OUTPUT: Sat Feb 14 05:01:30 UTC+0530 2009
```

13. setHours():

This method is used to change the Hour number to the specified number.

SYNTAX:

```
some_var.setHours(hour_number)
```

Example:

```
var a=new Date();                OUTPUT:
alert(a);                        Mon SEP 16 15:02:15 UTC+0530  2024
a.setHours(11);
alert(a);                        Mon SEP 16 11:02:15 UTC+0530  2024
```

14. setMinutes():

This method is used to change the Minutes number to the specified number.

SYNTAX:

```
some_var.setMinutes(minute_number)
```

Example:

```
var a=new Date();                OUTPUT:
alert(a);                        Mon SEP 16 15:02:15 UTC+0530  2024
a.setMinutes(25);
alert(a);                        Mon SEP 16 15:25:15 UTC+0530  2024
```

15. setSeconds():

This method is used to change the Seconds number to the specified number.

SYNTAX:

```
some_var.setSeconds(second_number)
```

Example:

```
var a=new Date();                OUTPUT:
alert(a);                        Mon SEP 16 15:02:15 UTC+0530  2024
a.setSeconds(45);
alert(a);                        Mon SEP 16 15:02:45 UTC+0530  2024
```

INTRODUCTION TO DOM

An HTML page is rendered in a browser. The browser assembles all the elements contained in the HTML page, downloaded from the web server, in its memory. Once done, the browser then renders these objects in the browser window. Once the HTML page is rendered in the browser window, the browser can no longer recognize individual HTML elements. To create an interactive web page, it is important that the browser continues to recognize individual HTML objects even after they are rendered. This allows the browser to access the properties of these objects using the built-in methods of the object. Once the properties of an object are accessible, the functionality of the object can be controlled any time.

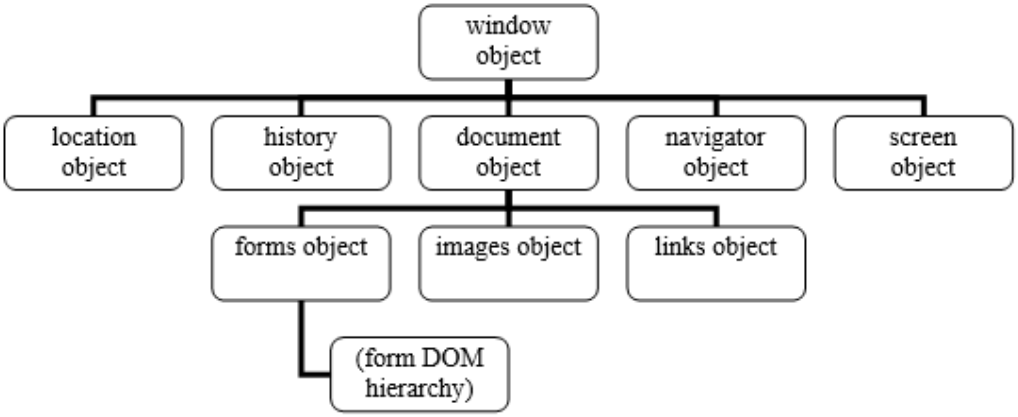
Javascript enabled browsers are capable of recognizing individual objects in an HTML page, after the page has been rendered in the browser, because the Javascript enabled browser recognizes and uses the DOM (Document Object Model). The HTML objects, which belong to the DOM, have a descending relationship with each other. The topmost object in the DOM is the **Window** itself. The next level in the DOM is the **Navigator** (browser), **History**, **Location**, **Screen**, and the **Document** displayed in the browser's window. If at all there is a HTML form coded in it, then the next level in the DOM under the Document object is that of the **Form**. The DOM hierarchy continues downward to encompass individual elements on a form, such as the textbox, radio button, etc.

When any Javascript enabled browser loads a web page, the browser automatically creates a number of Javascript objects that map to the DOM. It is the DOM which provides Javascript access to the HTML objects that are contained in the webpage. Not only is Javascript object-based, the browser is also made up of objects. Some objects provided by Javascript needs to be created but the objects of the browser are automatically created and placed in a hierarchy,

like an inverted tree structure, where you need to go through each object to access the next object below it. The collection of objects that the browser makes available to you for use with Javascript is generally called the **DOM** or **BOM.**

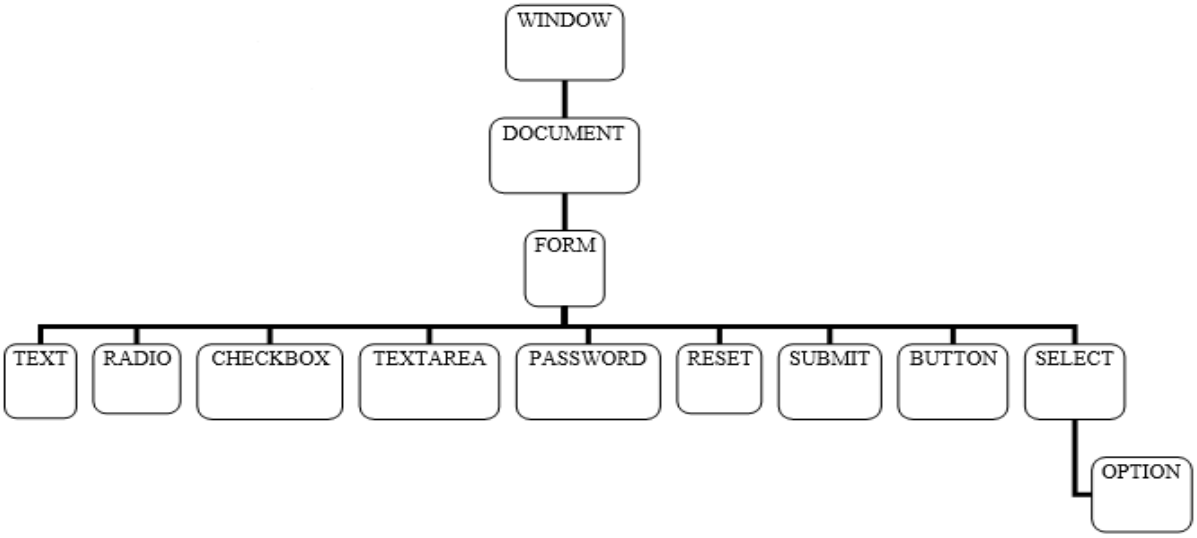
DOM HIERARCHY

The DOM is available for all scriptable browsers. The core DOM is available in all browsers, but the HTML DOM hierarchy differs from browser to browser and depends on the HTML elements that have been included in the webpage. Since it forms a tree like structure, the topmost object is called the root object. To access an object below it, you must use the root object. If there is an object placed far below in the hierarchy, you must include all the objects from root to that object. Each of the standard objects in the DOM has methods to access and manipulate that object and properties to change how the methods will be applied. The root object is the window object. All the objects below it are called the child objects. An object above a child object is called the parent object. When working with DOM it is very much necessary that you do not create variables with the same name as that of the objects or its properties, which may lead to unexpected results.



FORM DOM HIERARCHY

Any document i.e. the HTML page, can contain various HTML objects like: Images, Hyperlinks, Frames, Form with various form elements, etc. The browser creates one array in memory per HTML object in the document. These arrays hold indexed elements. The index value corresponds to where the HTML object is placed in the document. The arrays created are Images, Links, Frames, Forms, etc. All the form elements must be named and can also be accessed by their names in Javascript. Each form element can also be accessed as an object. Most of the form elements have common properties like name, value, type. **getElementById** collection is a W3C standard and so fully supported by all browsers. It is an alternative for retrieving a reference to any element in a document that has a unique identifier assigned to its **id** attribute. The following structure shows the hierarchy of the form objects in which all the controls of form are at the same level.



OBJECTS & COLLECTIONS IN DOM:

1. **window OBJECT:**

This object represents the browser's window, in which your web page is contained. This object is a global object, which means that you don't need to use its name to access its methods of properties or the objects below it. Some of the properties of this object like document, history, screen and location are themselves objects which in turn have their own properties or methods. It allows you to find out what browser is running, the page the user has visited, the size of the browser window, resolution of user screen etc.

Syntax:

window.propertyName
window.methodName([parameter])

2. **history OBJECT:**

This object keeps track of each page that the user visits. This list of pages is commonly called the history stack for the browser. It enables the user to mimic the browser's Back and Forward buttons to revisit the visited pages.

Syntax:

window.history.methodName([parameter])
window.history.propertyName

3. **location OBJECT:**

This object contains useful information about the current page's location including the URL for the page, server hosting the page, port number of the server connection, and the protocol used. It represents the address bar of the browser.

Syntax:

window.location.propertyName
window.location.methodName([parameter])

4. **screen OBJECT:**

This object contains information about the display capabilities of the client machine. It contains properties like height, width and colorDepth which indicate the screen sizes (in pixels) and the number of bits used to represent a color value respectively. This is a read-only object that lets scripts learn about the physical environment in which the browser is running.

Syntax:

window.screen.propertyName

Example: alert(window.screen.height);

5. **document OBJECT:**

Each document that gets loaded into a window becomes a document object. Through this object you can access the HTML elements, their properties and methods inside your page. This object has a number of properties associated with it, which are also array-like structures called collections. Main collections are the forms, images and links collections.

Syntax:

window.document.propertyName
window.document.methodName([parameter])

a. **IMAGES Collection**

All the <img> tags used in the HTML page are accessible using the images collection. Each <IMG> creates an **img** object, and the group of **img** objects forms the **images** collection, which is the property of the document object. The **images** collection can then be used as an **images** array with the first <img> tag at the index position 0 in the array. The individual objects can be accessed using the index position.

Syntax:

window.document.images[index].methodName([parameter]) OR
window.document.images.methodName([parameter])

Syntax:

window.document.images[index].propertyName OR
window.document.images.propertyName

b. LINKS collection

For each <a>, the browser creates an **a** object. The collection of all **a** objects in a page is contained within the **links** collection. All the <a>tags in the HTML page can be accessed using the **links** collection where the first <a> in the page will be accessible using the index value 0 in the **links** array.

Syntax:

window.document.links[index].methodName([parameter])
window.document.links[index].propertyName

c. FORMS collection

All the <form> tags used in the HTML page are accessible using the **forms** collection. Each <form> creates a **form** object, and the group of **form** objects creates the **forms** collection, which is the property of the document object. The **forms** collection can then be used as a **forms** array with the first tag at the index position 0 in the array. The individual objects can be accessed using either the index position or the name of the form. All the elements inside any form can be accessed using this object.

Syntax:

window.document.formName.propertyName

ACCESSING ELEMENTS DATA:

Traditionally, user input is captured in a **Form**. As soon as this tag is encountered by a JavaScript enabled browser, the browser creates a forms array in memory. The elements in a form must be accessed through the **forms** collection. Each element inside the form can be referenced using its name and followed by the name of the property or method or event.

Syntax:

window.document.**formName**.**elementName**.property/method/event

Example:

a=window.document.**frm**.**txt1**.value;

TEXT:

This control is accessed through the form control by using the name of the control. It is used to enter a single line of text.

Syntax:

window.document.formName.textboxname

1. **value**

   This **property** returns the string currently showing/entered in the textbox by the user. This can be used to read or write something in the text control. This is an editable property.

   Syntax:

   some_var=window.document.formName.textboxname.value;

   Example:

```
var a;
a= window.document.frm.txt1.value;
alert(a);
window.document.frm.txt1.value=”FYBCA”;
```

RADIO:

This control is accessed through the form control, as an array with the index value starting from 0. It allows for selecting one of the groups of radio buttons.

Syntax:

window.document.formName.radioGroupName[index]

1. **checked**
   This **property** returns true or false depending on whether that radio button is checked or not. All the other unchecked radio buttons have this property set to false. This is an editable property.
   Syntax:
   window.document.formName.radioGroupName[index].checked
   Example:
   ```
   if (window.document.frm.rd[0].checked)
       alert("First radio selected");
           //window.document.frm.rd[1].checked=true;
   ```
2. **value**
   This **property** returns the value of the radio button which is checked in the group. This is used as an alternative means to find out which radio button was selected. This is an editable property.
   Syntax:
   some_var=window.document.formName.radioGroupName[index].value;
   Example:
   ```
   var a;
   if (window.document.frm.rd[0].checked)
       a=window.document.frm.rd[0].value;
   else
       a=window.document.frm.rd[1].value;
   alert(a);
   ```

CHECKBOX:
This control is accessed through the form control by using the name of the control. It toggles between "on" and "off" settings. It indicates that the label text is true if it is selected, otherwise false.
Syntax:
window.document.formName.checkboxname

1. **checked**
   This **property** returns true or false depending on whether the checkbox button is checked or not. This is an editable property.
   Syntax:
   window.document.formName.checkboxname.checked
   Example:
   ```
   if (window.document.frm.chk1.checked)
       alert("First Checkbox is checked");
           //window.document.frm.chk2.checked=true;
   ```

2. **value**
   This **property** returns the value associated with a given checkbox. This property can be used as an alternative means of finding whether a checkbox is selected or not. If a checkbox is selected, it will return the value associated with that checkbox. This is an editable property.
   Syntax:
   some_var=window.document.formName.checkboxname.value;
   Example:
   ```
   var a;
   if (window.document.frm.chk1.checked)
   {
       a=window.document.frm.chk1.value;
       alert(a);
           // window.document.frm.chk1.value="a";
   }
   ```

SELECT / Dropdown:
This control is accessed using the form control, using the name of the control. It is used for efficient use of the page area, for presenting a list of choices for the user. It appears as a drop-down list or a scrollable list of selectable items.
Syntax:

window.document.formName.selectname

1. **selectedIndex**
   This **property** returns an integer, (starting from 0 for the 1st option) indicating the selected list item's index value. This is an editable property.
   Syntax:

   some_var=window.document.formName.selectname.selectedIndex;

   Example:

```
var a, b;
a=window.document.frm.sel.selectedIndex;
alert(a);
    // window.document.frm.sel.selectedIndex=2;
```

BUTTON:
This control is accessed through the form control using the name of the control. It is generally used to trigger some form processing.
Syntax:

window.document.formName.buttonname

1. **value**
   This **property** returns the label/caption displayed on the button control. This can be used to read or change the display text on the button. This is an editable property.
   Syntax:

   window.document.formName.buttonname.value

   Example:

```
var a="Changed", b;
alert(window.document.frm.btn.value);
window.document.frm.btn.value=a;
```

EVENT HANDLING:

JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page. Developers can use these events to execute JavaScript coded responses, which cause buttons to close windows, messages to be displayed to users, data to be validated, and virtually any other type of response imaginable. Events are a part of the Document Object Model (DOM). The change in the state of an object is known as an Event. 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. In html, there are various events which represents that some activity is performed by the user or by the browser. When JavaScript code is included in HTML, JavaScript react over these events and allow the execution. This process of reacting over the events is called Event Handling. Thus, JavaScript handles the HTML events via Event Handlers. For example, when a user clicks over the browser, add JavaScript code, which will execute the task to be performed on the event.

How to Handle Events in JavaScript (Event Handlers):
Binding events through tag attributes:
Here to bind an event, assign inline JavaScript code in the attribute of an element:
Syntax: <input type=……. **EventHandlerName**="**EventHandlerCode;**">

Example: <input type="button" name="btn" value="Click here" **onClick**="**abc();**">

Events in DOM:

Following are named event-handlers in JavaScript:

1. **Click** event: **onClick** event handler:
   This event fires when an object is clicked.
   It fires on **ALL** elements including Button element, Submit button element, Reset button element, Checkbox element, Radio Element, Document Object,
   Syntax: <input ... onClick="functionName();">
2. **Blur** event: **onBlur** event handler:
   This event fires when the form cursor is moved away from an object and loses the current input focus.
   It fires on Text element, Password element, Select element, Window element, Button element.
   Syntax: <input ... onBlur="functionName();">
3. **Focus** event: **onFocus** event handler:
   This event fires when the form cursor enters into an object and receives the current input focus.
   It fires on Text element, Password element, Select element, Window element, Button element Document object.
   Syntax: <input ... onFocus="functionName();">
4. **Load** event: **onLoad** event handler:
   This event fires after the web page or document is loaded in the window.
   It fires on the window object, image object.
   Syntax: <body onLoad="functionName();">
5. **MouseDown** event: **onMouseDown** event handler:
   This event is fired when any of the elements are clicked with the mouse key and the key is kept pressed and not released.
   It fires on **ALL** elements.
   Syntax: <input ... onMouseDown="functionName();">
6. **MouseUp** event: **onMouseUp** event handler:
   This event is fired when any of the elements are clicked with the mouse key and the key is released.
   It fires on **ALL** elements.
   Syntax: <input ... onMouseUp="functionName();">
7. **MouseOut** event: **onMouseOut** event handler:
   This event fires when the mouse cursor is brought over the area of any element.
   It fires on **ALL** elements.
   Syntax: <input ... onMouseOut="functionName();">
8. **MouseOver** event: **onMouseOver** event handler:
   This event fires when the mouse cursor is brought out of the area of any element.
   It fires on **ALL** elements.
   Syntax: <input ... onMouseOver="functionName();">
9. **Reset** event: **onReset** event handler:
   This event fires when the form's reset button is pressed.
   It fires on form.
   Syntax: <form ... onReset="functionName();">
10. **Submit** event: **onSubmit** event handler:
    This event fires when the form's submit button is pressed.
    It fires on form.
    Syntax: <form ... onSubmit="functionName();">