JS

JavaScript Fundamentals

By:Ahmed ElMahdy

JS

More About Numbers…

Numbers

Working with Numbers

- JavaScript supports both integer and floating-point numbers that can be represented in decimal, hexadecimal or octal notation.
- Unlike other languages, JavaScript does not treat integer and floating-point numbers differently.
- All numbers in JavaScript are represented as floating-point numbers.
- Numbers can also be represented in hexadecimal notation (base 16).
- Hexadecimal numbers are prefixed with 0x.

Tip: The biggest safe integer in JavaScript is 9007199254740991 (253-1), whereas the smallest safe integer is -9007199254740991 (-(253-1)).

Note: The for-in loop should not be used to iterate over an array where the index order is important. The for-in loop is optimized for iterating over object's properties, you should better use a for loop with a numeric index or for-of loop.

Numbers

Operating on Numbers and Strings

- The + operator is used for both addition and concatenation.
- Performing mathematical operations on numbers and strings may produce interesting results
- The operators has same precedence are evaluated from left to right so the result will not be just a simple string .
- other mathematical operations like multiplication, division, or subtraction the result will be different. JavaScript will automatically convert numeric strings (i.e. strings containing numeric values) to numbers in all numeric operations.
- multiply or divide numbers with strings that are not numeric returns NaN (Not a Number).
- Using NaN in a mathematical operation gives NaN .

Numbers

Representing Infinity:

- Infinity represents a number too big for JavaScript to handle.
- JavaScript has special keyword Infinity and -Infinity to represent positive and negative infinity respectively.

Note: Infinity is a special value that represents the mathematical Infinity ∞, which is greater than any number. The typeof operator return number for an Infinity value.

Avoiding Precision Problems

- operations on floating-point numbers produce unexpected results.
- because JavaScript and many other languages uses binary (base 2) form to represent decimal (base 10) numbers internally. Unfortunately, most decimal fractions can't be represented exactly in binary form, so small differences occur.
- JavaScript round floating-point numbers to 17 digits, which is enough precision or accuracy in most cases.
- In JavaScript integers (numbers without fractional parts or exponential notation) are accurate is up to 15 digits.

Numbers

Performing Operations on Numbers

- JavaScript provides several properties and methods to perform operations on number values.
- In JavaScript primitive data types can act like objects when you refer to them with the property access notation (i.e. dot notation).

In the following sections, we will look at the number methods that are most commonly used

Number Methods

Parsing Integers from Strings

- parseInt() Can be used to parse an integer from a string.
- This method is particularly handy in situations when you are dealing with the values like CSS units e.g. 50px, 12pt, etc. and you would like to extract the numeric value out of it.
- If the parseInt() method encounters a character that is not numeric in the specified base, it stops parsing and returns the integer value parsed up to that point.
- If the first character cannot be converted into a number, the method will return NaN (not a number). Leading and trailing spaces are allowed.

Numbers

Number Methods

Converting Numbers to Strings

- toString() can be used to convert a number to its string equivalent.
- This method optionally accepts an integer parameter in the range 2 through 36 specifying the base to use for representing numeric values.

Formatting Numbers in Exponential Notation

- toExponential() can be used to format or represent a number in exponential notation.
- This method optionally accepts an integer parameter specifying the number of digits after the decimal point.
- The returned value is a string not a number.

Note: Exponential notation is useful for representing numbers that are either very large or very small in magnitude. For example, 62500000000 can be written as 625e+8 or 6.25e+10

JS

Numbers

Number Methods

Formatting Numbers to Fixed Decimals

- toFixed() can be used to format a number with a fixed number of digits to the right of the decimal point.
- The value returned by this method is a string and it has exactly specified number of digits after the decimal point.
- If the digits parameter is not specified or omitted, it is treated as 0.

Formatting Numbers with Precision

- Use toPrecision() to get most appropriate form of a number .
- This method returns a string representing the number to the specified precision.
- If precision is large enough to include all the digits of the integer part of number, then the number is formatted using fixed-point notation. Otherwise, the number is formatted using exponential notation. The precision parameter is optional.

Numbers

Math Object

Finding the Largest and Smallest Possible Numbers

- The Number object also has several properties associated with it.
- The Number.MAX_VALUE and Number.MIN_VALUE properties of the Number object represent the largest and smallest (closest to zero, not most negative) possible positive numbers that JavaScript can handle.
- They are constants and their actual values are 1.7976931348623157e+308, and 5e-324, respectively.
- A number that falls outside of the range of possible numbers is represented by a constant Number.POSITIVE_INFINITY or Number.NEGATIVE_INFINITY.

check out the JavaScript math operations to learn about rounding numbers, generating a random number, finding maximum or minimun value from a set of numbers, etc.

JS

String...

String

What is String in JavaScript

- by enclosingA string is a sequence of letters, numbers, special characters and arithmetic values or combination of all.
- Strings can be created the string literal (i.e. string characters) either within single quotes (') or double quotes (").

JavaScript Escape Sequences

- Escape sequences are also useful for situations where you want to use characters that can't be typed using a keyboard.
- Here are some other most commonly used escape sequences.

✓ \n is replaced by the newline character
✓ \t is replaced by the tab character
✓ \r is replaced by the carriage-return character
✓ \b is replaced by the backspace character
✓ \\ is replaced by a single backslash (\)

String

Performing Operations on Strings

- JavaScript provides several properties and methods to perform operations on string values.
- Technically, only objects can have properties and methods. But in JavaScript primitive data types can act like objects when you refer to them with the property access notation (i.e. dot notation).
- JavaScript making it possible by creating a temporary wrapper object for primitive data types.
- This process is done automatically by the JavaScript interpreter in the background.

Getting the Length of a String

The length property returns the length of the string, which is the number of characters contained in the string. This includes the number of special characters as well, such as \t or \n.

Note: Since length is a property, not a function, so don't use parentheses after it like str.length(). Instead just write str.length, otherwise it will produce an error

String

Searching for a Pattern Inside a String

- You can use the search() method to search a particular piece of text or pattern inside a string.
- The search() method also returns the index of the first match, and returns -1 if no matches were found,
- Unlike indexOf() method this method can also take a regular expression as its argument to provide advanced search capabilities.

Note: The search() method does not support global searches; it ignores the g flag or modifier (i.e. /pattern/g) of its regular expression argument.

String

Extracting a Substring from a String

- You can use the slice() method to extract a part or substring from a string.
- This method takes 2 parameters:

✓ start index (index at which to begin extraction),

✓ optional end index (index before which to end extraction), like str.slice(startIndex, endIndex).

- You can also specify negative values.
- The negative value is treated as strLength + startIndex, where strLength is the length of the string (i.e. str.length), for example, if startIndex is -5 it is treated as strLength - 5.
- If startIndex is greater than or equal to the length of the string, slice() method returns an empty string.
- Also, if optional endIndex is not specified or omitted, the slice() method extracts to the end of the string.

String

Extracting a Substring from a String

- use the substring() method to extract a section of the given string based on start and end indexes, like str.substring(startIndex, endIndex). The substring() method is very similar to the slice() method, except few differences:

✓ If either argument is less than 0 or is NaN, it is treated as 0.

✓ If either argument is greater than str.length, it is treated as if it were str.length.

✓ If startIndex is greater than endIndex, then substring() will swap those two arguments; for example, str.substring(5, 0) == str.substring(0, 5)

- Extracting a Fixed Number of Characters from a String

JavaScript also provide the substr() method which is similar to slice() with a subtle difference, the second parameter specifies the number of characters to extract instead of ending index, like str.substr(startIndex, length). If length is 0 or a negative number, an empty string is returned.

JS

String

Replacing the Contents of a String

- use the replace() method to replace part of a string with another string.
- This method takes two parameters a regular expression to match or substring to be replaced and a replacement string, like str.replace(regexp|substr, newSubstr).
- This replace() method returns a new string, it doesn't affect the original string that will remain unchanged.
- By default, the replace() method replaces only the first match, and it is case-sensitive.
- To replace the substring within a string in a case-insensitive manner you can use a regular expression *(regexp)* with an i modifier.
- To replace all the occurrences of a substring within a string in a case-insensitive manner you can use the g modifier along with the i modifier

Converting a String to Uppercase or Lowercase

- use the toUpperCase() method to convert a string to uppercase.

Concatenating Two or More Strings

- You can concatenate or combine two or more strings using the + and += assignment operators.
- JavaScript also provides concat() method to combine strings, but it is not recommended.

String

Accessing Individual Characters from a String

- Use the charAt() method to access individual character from a string, like str.charAt(index).
- The index specified should be an integer between 0 and str.length - 1.
- If no index is provided the first character in the string is returned, since the default is 0.
- Since ECMAScript 5, strings can be treated like read-only arrays, and you can access individual characters from a string using square brackets ([]) instead of the charAt() method.

Note: The only difference between accessing the character from a string using the charAt() and square bracket ([]) is that if no character is found, [] returns undefined, whereas the charAt() method returns an empty string.

Splitting a String into an Array

The split() method can be used to splits a string into an array of strings, using the syntax str.split(separator, limit).
The seperator argument specifies the string at which each split should occur, whereas the limit arguments specifies the maximum length of the array.
If separator argument is omitted or not found in the specified string, the entire string is assigned to the first element of the array.
To split a string into an array of characters, specify an empty string ("") as a separator.

JS

More About Arrays…

Arrays

Adding or Removing Elements at Any Position

- The splice() method is a very versatile array method
- allows you to add or remove elements from any index, using the syntax arr.splice(startIndex, deleteCount, elem1, ..., elemN).
- This method takes three parameters:
  ✓first parameter is the index at which to start splicing the array, it is required;
  ✓second parameter is the number of elements to remove (use 0 if you don't want to remove any elements), it is optional.
  ✓third parameter is a set of replacement elements, it is also optional.
- The splice() method returns an array of the deleted elements, or an empty array if no elements were deleted
- Unlike slice() and concat() methods, the splice() method modifies the array on which it is called on.

Arrays

Creating a String from an Array

- use the join() method to create a string by joining the elements of an array.
- This method takes an optional parameter which is a separator string that is added in between each element.
- If you omit the separator, then JavaScript will use comma (,) by default.
- Convert an array to a comma-separated string by using the toString(). This method does not accept the separator parameter like join().

Extracting a Portion of an Array

- Use the slice() method to extract out a portion of an array (i.e. subarray) but keep the original array intact you can.
- This method takes 2 parameters:

✓ start index (index at which to begin extraction).

✓ optional end index (index before which to end extraction), like arr.slice(startIndex, endIndex).

- If endIndex parameter is omitted, all elements to the end of the array are extracted.
- You can also specify negative indexes or offsets —in that case the slice() method extract the elements from the end of an array, rather then the begining.

Arrays

Merging Two or More Arrays

- The concat() method can be used to merge or combine two or more arrays.
- This method does not change the existing arrays, instead it returns a new array.
- The concat() method can take any number of array arguments, so you can create an array from any number of other arrays.

Searching Through an Array

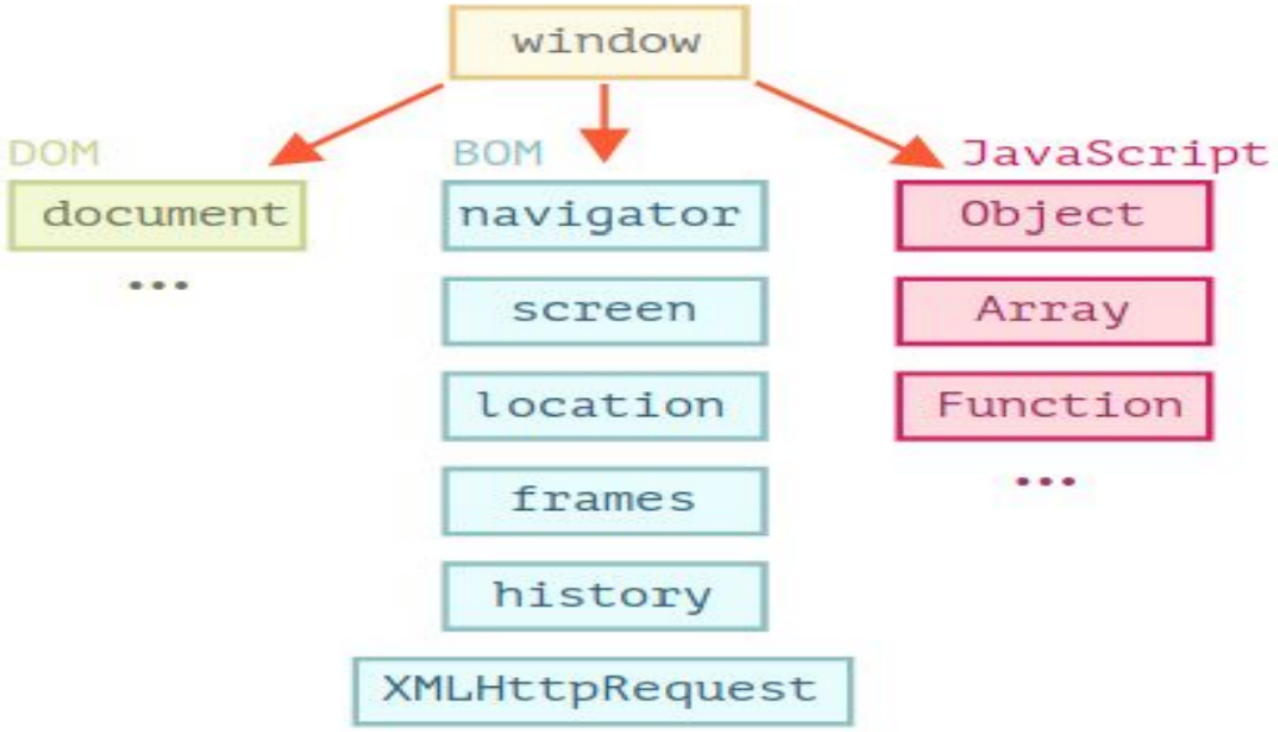
- Use the indexOf() and lastIndexOf() to search an array for a specific value.
- If the value is found, both methods return an index representing the array element.
- If the value is not found, -1 is returned.
- The indexOf() method returns the first one found, whereas the lastIndexOf() returns the last one found.
- Both methods also accept an optional integer parameter from *index* which specifies the index within the array at which to start the search.
- Use includes() method to find out whether an array includes a certain element or not.
- includes() takes the same parameters as indexOf() and lastIndexOf() methods, but it returns true or false instead of index number.

Arrays

Searching Through an Array

- Use find() method which is newly introduced in ES6 to search an array based on certain condition.
- find() returns only the value of the *first element* in the array that satisfies the provided testing function. Otherwise it return undefined.
- findIndex() method, which returns the index of a found element in the array instead of its value.
- use the filter() method to find out all the matched elements.

Browser environment



BOM

JavaScript BOM

Understanding the Browser Object Model

- It represents additional objects provided by the browser (host environment) for working with everything except th document.
- The navigator object provides background information about the browser and the operating system.
- There are many properties, but the two most widely known are:

✓ navigator.userAgent – about the current browser

✓ navigator.platform – about the platform (can help to differ between Windows/Linux/Mac etc)

- The location object allows us to read the current URL and can redirect the browser to a new one.
- Functions alert/confirm/prompt are also a part of BOM: they are directly not related to the document, but represent pure browser methods of communication with the user.

DOM

JavaScript DOM Nodes

Understanding the Document Object Model

- It is a platform and language independent model to represent the HTML or XML documents.
- It defines the logical structure of the documents and the way in which they can be accessed and manipulated by an application program.
- In the DOM, all parts of the document, such as elements, attributes, text, etc. are organized in a hierarchical tree-like structure; similar to a family tree in real life that consists of parents and children.
- In DOM terminology these individual parts of the document are known as nodes.
- The Document Object Model that represents HTML document is referred to as HTML DOM.
- The DOM that represents the XML document is referred to as XML DOM.
- With the HTML DOM, you can use JavaScript to build HTML documents, navigate their hierarchical structure, and add, modify, or delete elements and attributes or their content, and so on.

DOM

JavaScript DOM Nodes

Understanding the Document Object Model

- Almost anything found in an HTML document can be accessed, changed, deleted, or added using the JavaScript with the help of HTML DOM.

To understand this more clearly, let's consider the following simple HTML document:

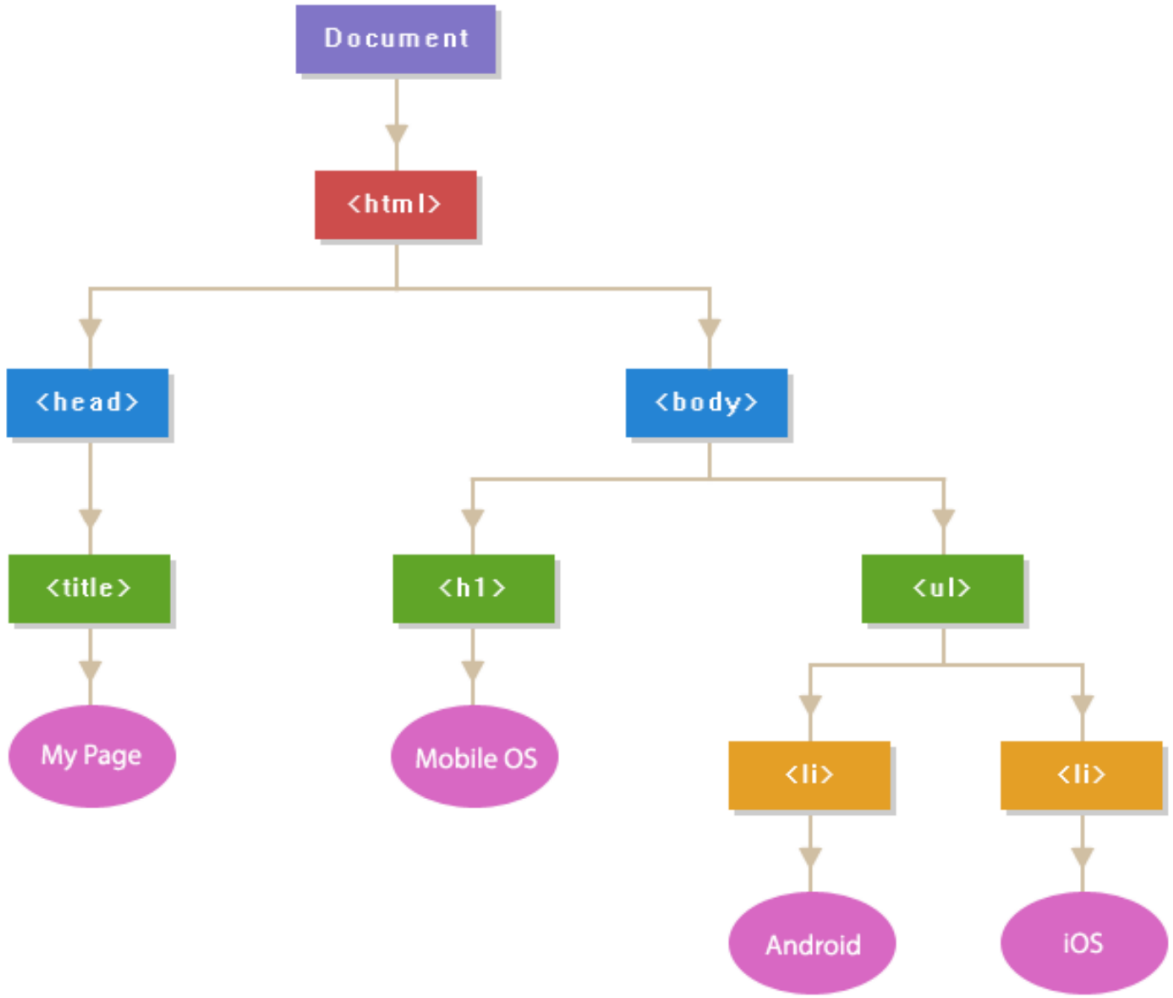
Example

```
<!DOCTYPE html>
<html>
<head>
 <title>My Page</title>
</head>
<body>
 <h1>Mobile OS</h1>
 <ul>
<li>Android</li>
<li>iOS</li>
 </ul>
</body>
</html>
```

The above HTML document can be represented by the following DOM tree

Dom

JavaScript DOM Nodes



Dom

JavaScript DOM Nodes

The above diagram demonstrates the parent/child relationships between the nodes. The topmost node i.e. the Document node is the root node of the DOM tree, which has one child, the <html> element. Whereas, the <head> and <body> elements are the child nodes of the <html> parent
node.
The <head> and <body> elements are also siblings since they are at the same level. Further, the text content inside an element is a child node of the parent element. So, for example, "Mobile OS" is considered as a child node of the <h1> that contains it, and so on.

Comments inside the HTML document are nodes in the DOM tree as well, even though it doesn't affect the visual representation of the document in any way.
Comments are useful for documenting the code, however, you will rarely need to retrieve and manipulate them.

Dom

JavaScript DOM Nodes

HTML attributes such as id, class, title, style, etc. are also considered as nodes in DOM hierarchy but they don't participate in parent/child relationships like the other nodes do.
They are accessed as properties of the element node that contains them.

Each element in an HTML document such as image, hyperlink, form, button, heading, paragraph, etc. is represented using a JavaScript object in the DOM hierarchy, and each object contains properties and methods to describe and manipulate these objects.
For example, the style property of the DOM elements can be used to get or set the inline style of an element.

Tip: The Document Object Model or DOM is, in fact, basically a representation of the various components of the browser and the current Web document (HTML or XML) that can be accessed or manipulated using a scripting language such as JavaScript.

Dom

JavaScript DOM Selectors

Selecting DOM Elements in JavaScript

JavaScript is most commonly used to get or modify the content or value of the HTML elements on the page, as well as to apply some effects like show, hide, animations etc. But, before you can perform any action you need to find or select the target HTML element.

In the following sections, you will see some of the common ways of selecting the elements on a page and do something with them using the JavaScript.

Selecting the Topmost Elements

- The topmost elements in an HTML document are available directly as document properties. For example, the *<html>* element can be accessed with document.documentElement property.
- Whereas the *<head>* element can be accessed with document.head property, and the *<body>* element can be accessed with document.body property.
- If document.body is used before the <body> tag (e.g. inside the <head>), it will return null instead of the body element. Because the point at which the script is executed, the <body> tag was not parsed by the browser, so document.body is truly null at that point.

Dom

Selecting DOM Elements in JavaScript

Selecting Elements by ID

- You can select an element based on its unique ID with the getElementById() method.
- This is the easiest way to find an HTML element in the DOM tree.
- The getElementById() method will return the element as an object if the matching element was found, or null if no matching element was found in the document.

Note: Any HTML element can have an id attribute. The value of this attribute must be unique within a page i.e. no two elements in the same page can have the same ID.

Selecting Elements by Class Name

- you can use the getElementsByClassName() method to select all the elements having specific class names.
- This method returns an array-like object of all child elements which have all of the given class names.

Selecting Elements by Tag Name

- select HTML elements by tag name using the getElementsByTagName() method.
- This method also returns an array-like object of all child elements with the given tag name.

Dom

Selecting DOM Elements in JavaScript

Selecting Elements with CSS Selectors

- use the querySelectorAll() method to select elements that matches the specified CSS selector.
- CSS selectors provide a very powerful and efficient way of selecting HTML elements in a document.
- This method returns a list of all the elements that matches the specified selectors. You can examine it just like any array.

Note: The querySelectorAll() method also supports CSS pseudoclasses like :first-child, :last-child, :hover, etc. But, for CSS pseudoelements such as ::before, ::after, ::first-line, etc. this method always returns an empty list.

Dom

JavaScript DOM Styling

Styling DOM Elements in JavaScript

- You can also apply style on HTML elements to change the visual presentation of HTML documents dynamically using JavaScript.
- You can set almost all the styles for the elements like, fonts, colors, margins, borders, background images, text alignment, width and height, position, and so on.

In the following section we'll discuss the various methods of setting styles in JavaScript.

Setting Inline Styles on Elements

- They are applied directly to the specific HTML element using the style attribute.
- In JavaScript the style property is used to get or set the inline style of an element.
- Naming Conventions of CSS Properties in JavaScript Many:

CSS properties, such as font-size, background-image, text-decoration, etc. contain hyphens (-) in their names. Since, in JavaScript hyphen is a reserved operator and it is interpreted as a minus sign, so it is not possible to write an expression, like: elem.style.font-size .

Therefore, in JavaScript, the CSS property names that contain one or more hyphens are converted to intercapitalized style word. It is done by removing the hyphens and capitalizing the letter immediately following each hyphen, thus

Dom

JavaScript DOM Styling

Styling DOM Elements in JavaScript

Getting Style Information from Elements

- You get the styles applied on the HTML elements using the style property.
- The style property isn't very useful when it comes to getting style information from the elements, because it only returns the style rules set in the element's style attribute not those that come from elsewhere, such as style rules in the embedded style sheets, or external style sheets.
- To get the values of all CSS properties that are actually used to render an element you can use the window.getComputedStyle() method.

Tip: The value 700 for the CSS property font-weight is same as the keyword bold. The color keyword red is same as rgb(255,0,0), which is the rgb notation of a color.

Events

Understanding Events and Event Handlers

- An event is something that happens when user interact with the web page, such as when he clicked a link or button, entered text into an input box or textarea, made selection in a select box, pressed key on the keyboard, moved the mouse pointer, submits a form, etc.
- In some cases, the Browser itself can trigger the events, such as the page load and unload events.
- When an event occur, you can use a JavaScript event handler (or an event listener) to detect them and perform specific task or set of tasks.
- The names for event handlers always begin with the word "on", so an event handler for the click event is called onclick, similarly an event handler for the load event is called onload, event handler for the blur event is called onblur, and so on.
- There are several ways to assign an event handler. The simplest way is to add them directly to the start tag of the HTML elements using the special event-handler attributes. For example, to assign a click handler for a button element, we can use onclick attribute.
- To keep the JavaScript separate from HTML, you can set up the event handler in an external JavaScript file or within the tags.

Note: Since HTML attributes are case-insensitive so onclick may also be written as onClick, OnClick or ONCLICK. But its *value* is case-sensitive.

Events

Mouse Events

A mouse event is triggered when the user click some element, move the mouse pointer over an element, etc. Here're some most important mouse events and their event handler:

The Click Event (onclick)

It occurs when a user clicks on an element on a web page. Often, these are form elements and links. You can handle a click event with an onclick event handler.

The Contextmenu Event (oncontextmenu)

It occurs when a user clicks the right mouse button on an element to open a context menu.
You can handle a contextmenu event with an oncontextmenu event handler.

The Mouseover Event (onmouseover)

It occurs when a user moves the mouse pointer over an element.
You can handle the mouseover event with the onmouseover event handler.

The Mouseout Event (onmouseout)

It occurs when a user moves the mouse pointer outside of an element. You can handle the mouseout event with the

Events

Mouse Events

MouseDown Event

When the mouse button is pressed.

Mouseup Event

Released over an element.

MouseMove Event

When the mouse is moved

Events

Keyboard Events

A keyboard event is fired when the user press or release a key on the keyboard. Here're some most important keyboard events and their event handler:

The Keydown Event (onkeydown)

It occurs when the user presses down a key on the keyboard.

You can handle the keydown event with the onkeydown event handler.

The Keyup Event (onkeyup)

It occurs when the user releases a key on the keyboard.

You can handle the keyup event with the onkeyup event handler.

The Keypress Event (onkeypress)

The keypress event occurs when a user presses down a key on the keyboard that has a character value associated with it. For example, keys like Ctrl, Shift, Alt, Esc, Arrow keys, etc. will not generate a keypress event, but will generate a keydown and keyup event.

You can handle the keypress event with the onkeypress event handler.