CSS selectors, Properties Box Model

Attribute Selectors

- It is possible to style HTML elements that have specific attributes or attribute values and the [attribute] selector is used to select elements with a specified attribute.
- Eg – To select all <a> elements with a target attribute:

```
a[target] {
  background-color: yellow;
}
```

CSS selectors, Properties Box Model

Attribute Selectors

```
<html>
<head>
<style>
      a[target] {
        background-color: yellow;
      }
</style>
</head>
<body>

<p>The links with a target attribute gets a yellow background:</p>

<a href="https://www.google.com">google.com</a>
<a href="http://www.wikipedia.org" target="_top">wikipedia.org</a>
</body>
</html>
```

Attribute Selectors

CSS [attribute="value"] Selector

The [attribute="value"] selector is used to select elements with a specified attribute and value.

CSS [attribute~="value"] Selector

The [attribute~="value"] selector is used to select elements with an attribute value containing a specified word.

CSS [attribute|="value"] Selector

The [attribute|="value"] selector is used to select elements with the specified attribute starting with the specified value.

Attribute Selectors

CSS [attribute^="value"] Selector

The [attribute^="value"] selector is used to select elements whose attribute value begins with a specified value.

CSS [attribute$="value"] Selector

The [attribute$="value"] selector is used to select elements whose attribute value ends with a specified value.

CSS [attribute*="value"] Selector

The [attribute*="value"] selector is used to select elements whose attribute value contains a specified value.

CSS selectors, Properties Box Model

Attribute Selectors with examples

Selector	Example	Example description
[*attribute*]	[target]	Selects all elements with a target attribute
[*attribute=value*]	[target=_blank]	Selects all elements with target="_blank"
[*attribute~=value*]	[title~=flower]	Selects all elements with a title attribute containing the word "flower"
[*attribute|=value*]	[lang|=en]	Selects all elements with a lang attribute value starting with "en"
[*attribute^=value*]	a[href^="https"]	Selects every <a> element whose href attribute value begins with "https"
[*attribute$=value*]	a[href$=".pdf"]	Selects every <a> element whose href attribute value ends with ".pdf"
[*attribute*=value*]	a[href*="w3schools"]	Selects every <a> element whose href attribute value contains the substring "w3schools"

Contextual Selectors/descendant selector

- Used to select elements that are descendants of another element in the document tree.
- Eg-

```
<body>
<p>web<em>technology</em></p>
<ul>
<li>html</li>
<li>sgml</li>
<li><em>xhtml</em></li>
</ul>
</body>
```

p em{color:blue} this rule will only select elements that are descendants of <p> elements. If this rule is applied, the element within the <li> will not be colored blue

CSS selectors, Properties Box Model

Child selector, pseudo class

- The **child selector** selects all elements that are the children of a specified element.
- Eg – selects all <p> elements that are children of a element:

```
div > p {   background-color: yellow; }
```

Pseudo-classes

- A pseudo-class is used to define a special state of an element.

For example, it can be used to:

- Style an element when a user mouses over it
- Style visited and unvisited links differently
- Style an element when it gets focus

CSS selectors, Properties Box Model

Child selector, pseudo class

- Syntax

```
selector:pseudo-class {
  property: value;
}
```

We can style links in different ways in each of the four states

- a:link - a normal, unvisited link
- a:visited - a link the user has visited
- a:hover - a link when the user mouses over it
- a:active - a link the moment it is clicked

CSS selectors, Properties Box Model

pseudo elements

- CSS pseudo-elements are used to add special effects to some selectors

Syntax

selector:pseudo-element {property: value}

selector.class:pseudo-element {property: value}

Sr.No.	Value & Description
1	**:first-line** Use this element to add special styles to the first line of the text in a selector.
2	**:first-letter** Use this element to add special style to the first letter of the text in a selector.
3	**:before** Use this element to insert some content before an element.
4	**:after** Use this element to insert some content after an element.

CSS selectors, Properties Box Model

pseudo elements

```
<html>
  <head>
    <style type = "text/css">
      p:first-line { text-decoration: underline; }
      p.noline:first-line { text-decoration: none; }
    </style>
  </head>
  <body>
    <p class = "noline">
      This line would not have any underline because this belongs to noline class.
    </p>

    <p>
      The first line of this paragraph will be underlined as defined in the
      CSS rule above. Rest of the lines in this paragraph will remain normal. .
    </p>
  </body>
</html>
```

CSS selectors, Properties Box Model

Group selectors ,universal selector

- The **grouping selector** selects all the HTML elements with the same style definitions thereby minimizing the code
- Eg –

```
h1, h2, p {
  text-align: center;
  color: red;
}
```

Universal Selector

- The universal selector (*) selects all HTML elements on the page.
- Eg – will affect every HTML element on the page:

```
* {
  text-align: center;
  color: blue;
}
```

CSS selectors, Properties Box Model

Group selectors ,universal selector

- The **grouping selector** selects all the HTML elements with the same style definitions thereby minimizing the code
- Eg –

```
h1, h2, p {
  text-align: center;
  color: red;
}
```

Universal Selector

- The universal selector (*) selects all HTML elements on the page.
- Eg – will affect every HTML element on the page:

```
* {
  text-align: center;
  color: blue;
}
```

CSS selectors, Properties Box Model

Style properties

- Number Values- integers(whole numbers and real fractional numbers
- Percentage values- p {font-size:200%}
- Length Values- inches, centimeters or points.
- Color values using color keywords-
  - h1 {color : DarkGreen;}
- Color values using RGB values
  - #RRGGBB / #RGB / rgb (R,G,B) / rgb (R%,G%,B%)

Style properties

Font Family

- The font family of a text is set with the font-family property.
- The font-family property should hold several font names as a "fallback" system. If the browser does not support the first font, it tries the next font.
- Eg - p{font-family: "Times New Roman", Times, serif;}

- Two types of names are used to categorize fonts
  - Family-names- Arial ,Times new roman, Tahoma
  - Generic families- serif ,sans-serif, monospace

CSS selectors, Properties Box Model

Style properties

- Font-Sizes
  - Specifies the size of the font(length/percentage)
    - Values are- xx-large, x-large, large, medium, small, x-small and xx-small
- Font variant
  - Specifies whether the font will be displayed in small caps
    - p em {font-variant: small-caps;}
- Font Styles
  - Affects the posture of the text
    - (normal/italic/oblique)
- Font Weights
  - Specifies the thickness of the font

Style properties

- **Font Shorthand**
  - Covers all the different font properties in one single property. The value of font property is a list of all the font properties
  - Order of values

    font-style | font-variant | font-weight | font-size | font-family

- Eg -

```
p {font: Italic bold 30px  Arial, sans-serif; }
```

Style properties

Text Properties

- Text-indent(pt / % / em)
- Text-align-left, right, center, justified
- Text-decoration- underline, overline, line-through, blink
- Text-transform- capitalize(first letter in the word with capital letter), uppercase, lowercase, none

List properties

List style type – property to select the type of marker that appears before each item

General Syntax

Values: none | disc | circle |square |decimal-leading-zero | lower-alpha |lower- latin |upper-roman | lower- greek

Style properties

Text Properties

- Text-indent(pt / % / em)
- Text-align-left, right, center, justified
- Text-decoration- underline, overline, line-through, blink
- Text-transform- capitalize(first letter in the word with capital letter), uppercase, lowercase, none

List properties

List style type – property to select the type of marker that appears before each item

General Syntax

Values: none | disc | circle |square |decimal-leading-zero | lower-alpha |lower- latin |upper-roman | lower- greek

Element Positioning

- The position of any element is dictated by the three style properties: **position, left, and top**.
- The four possible values of position are

  absolute, relative, static and fixed

position: static;

- HTML elements are positioned static by default. Static positioned elements are not affected by the top, bottom, left, and right properties.

- An element with position: static; is not positioned in any special way; it is always positioned according to the normal flow of the page:

Element Positioning

position: relative;

- An element with position: relative; is positioned relative to its normal position.

- Setting the top, right, bottom, and left properties of a relatively-positioned element will cause it to be adjusted away from its normal position. Other content will not be adjusted to fit into any gap left by the element.

Eg –

```
div.relative {
        position: relative;
        left: 30px;
        border: 3px solid #73AD21;
        }
```

position: fixed;

- An element with position: fixed; is positioned relative to the viewport, which means it always stays in the same place even if the page is scrolled. The top, right, bottom, and left properties are used to position the element.
- A fixed element does not leave a gap in the page where it would normally have been located.
- Notice the fixed element in the lower-right corner of the page.

Eg –

```
div.fixed {
    position: fixed;
    bottom: 0;
    right: 0;
    width: 300px;
    border: 3px solid #73AD21;
}
```

position: absolute;

- An element with position: absolute; is positioned relative to the nearest positioned ancestor (instead of positioned relative to the viewport, like fixed).
- However; if an absolute positioned element has no positioned ancestors, it uses the document body, and moves along with page scrolling.
- A "positioned" element is one whose position is anything except static.

Eg –

```
div.absolute {
        position: absolute;
        top: 80px;
        right: 0;
        width: 200px;
        height: 100px;
        border: 3px solid #73AD21;
}
```

Stacking of elements

- The ***z-index*** property sets or returns the stack order of a positioned element.
- An element with greater stack order (1) is always in front of another element with lower stack order (0).
- A positioned element is an element with the position property set to: relative, absolute, or fixed.
- This property is useful if you want to create overlapping elements.

Syntax

z-index: auto|*number*|initial|inherit;

Stacking of elements

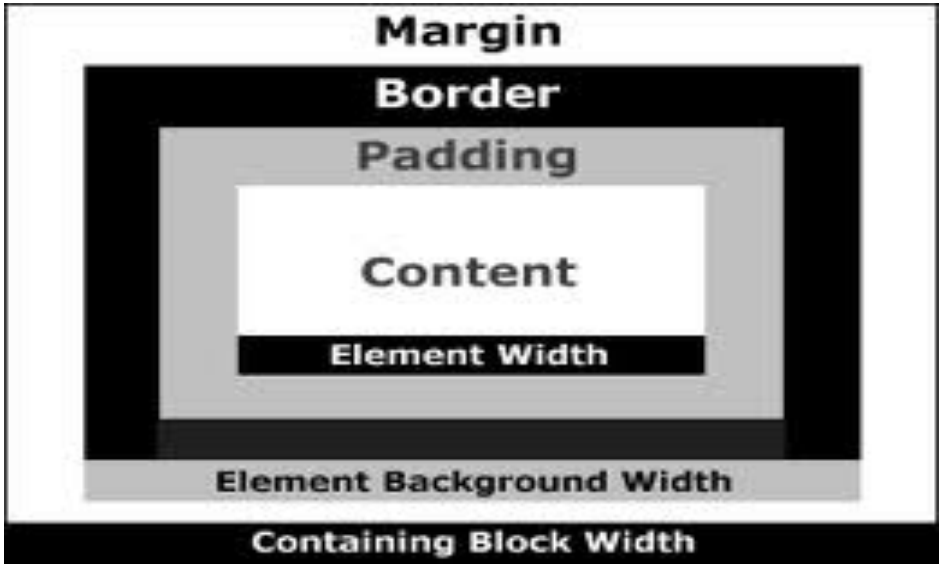
Value	Description
Auto	Sets the stack order equal to its parents. This is default
Number	Sets the stack order of the element. Negative numbers are allowed
initial	Sets this property to its default value
inherit	Inherits this property from its parent element.

CSS Box Model

Box model is about CSS-based layouts and designs.

The box model is a term used when referring to the rectangular boxes placed around every element in the web page.

Each XTML element (like letters, images, form controls, tables,etc) has an "invisible box" around it. **By default its invisible.**



CSS selectors, Properties Box Model

Box Model

- **Content Area-** its the innermost part of the box i.e xhtml element or the content. The CSS width and height property defines the width and the height of this element
- **Padding**- its the first layer going outwards from the actual element . It defines the space b/w the content area of the box and the border

  p{padding: 30px 20px 30px 20px;}
- **Border** – Middle layer in the box model is the element's border. The space used by the border in the box model is the thickness of the border. The border outlines the visible portion of the element

  p{border: 5px solid red;}
- **Margin** – the space just outside the border .The margin is completely invisible , no background color and will not contain any elements behind it

Style Properties

Background Images

- The background-image property is used to place an image in the background of an element

  Eg-

  body{background-image: url(“image.jpg”);}

  p{background-image: url(“image.jpg”);}

- Background-repeat
  - Specifies whether a background image repeats itself. The default is repeat(image repeats in x- and y- directions)
- Eg-

  Body{background-image: url(“image.jpg”);

  background-repeat: no-repeat/repeat-x/repeat-y/repeat;}

Conflict Resolution

Two or more conflicting CSS rules are sometimes applied to the same element.
What are the rules in CSS that resolve the question of which style rule will actually be used when a page is rendered by a browser?

Inheritance

Some properties are passed from parent to child. For example, this rule in a style sheet would be inherited by all child elements of the body and make every font on the page display as Georgia.

```
body {font-family: Georgia;}
```

Conflict Resolution

The Cascade

Within the cascade, more than one factor can figure into determining which one of several conflicting CSS rules will actually be applied. These factors are source order, specificity and importance. Location is part of the cascade, too.

Source order means the order in which rules appear in the style sheet. A rule that appears later in the source order will generally overrule an earlier rule.

Eg –

```
body {font-family: Georgia;}
h1, h2, h3 {font-family: Arial;}
```

PES UNIVERSITY ONLINE

Specificity

Specificity is determined by a mathematical formula, but common sense can help you understand it.

Eg -

```
p {font-family: Georgia;}

.feature  p {font-family: Arial;}
```

In this case, the selector .feature p is more specific than the selector p. For any paragraph assigned to the class 'feature' the font-family would be Arial. Here selecting a paragraph that belongs to a particular class is a more specific choice than selecting all paragraphs. The more specific selector overrules the less specific selector.

Conflict Resolution

!important

There are rules that are declared !important. !important rules always overrule other rules, no matter what inheritance, source order or specificity might otherwise do. A user created stylesheet can use !important to overrule the author's CSS.

Eg -

```
*{font-family: Arial !important;}
```

This rule would mean that everything (* selects everything) would be Arial no matter what other rules were used in the CSS.

Conflict Resolution

Location

Style rules can exist in a number of locations in relation to the HTML page affected. The location of a rule also plays into determining which rule actually ends up being implemented. The locations are:

- Browser style rules
- External style rules
- Internal style (in the document head) rules
- Inline style rules
- Individual user style rules

JavaScript is a lightweight, interpreted **programming (scripting)** language. It is designed for creating network-centric applications. **JavaScript** is very easy to implement because it is integrated with HTML. It is open and cross-platform.

Applications of JavaScript Programming

- Client side validation
- Manipulating HTML Pages
- User Notifications
- Back-end Data Loading


The process with client-side scripting
Client Side
Server
Request for file.css
File.css
Script Executed
File out put
Web browser
File displayed on your computer
Learn best software engineering practices @ www.baabtra.com
baabtra.com
Mentoring Partner

JavaScript- Pros & Cons

The merits of using JavaScript are –

- **Less server interaction** – You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.
- **Immediate feedback to the visitors** – They don't have to wait for a page reload to see if they have forgotten to enter something.
- **Increased interactivity** – You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
- **Richer interfaces** – You can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

Demerits

- Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.

- JavaScript cannot be used for networking applications because there is no such support available.

- JavaScript doesn't have any multi-threading or multiprocessor capabilities

JavaScript syntax

- JavaScript can be implemented using JavaScript statements that are placed within the <script>... </script> HTML tags in a web page.
- We can place the <script> tags, containing your JavaScript, anywhere within your web page, but it is normally recommended that you should keep it within the <head> tags.

Syntax

```
<script type = "text/javascript">
   JavaScript code
 </script>
```


```
<html>
  <body>
    <script  type = "text/javascript">
            document.write ("Hello World!")
</script>
  </body>
</html>
```

JavaScript - Placement in HTML File

- 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.**
- **Script in an external file and then include in <head>...</head> section.**

JavaScript Basics

JavaScript datatypes

JavaScript allows you to work with three primitive data types –

- **Numbers,** eg. 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.**

JavaScript Basics

JavaScript comments

- Comments are important because they help other people understand what is going on in the code or remind if forgot something. In JavaScript there are have two different options:
  - *Single-line comments* — To include a comment that is limited to a single line, precede it with //
  - *Multi-line comments* — In case you want to write longer comments between several lines, wrap it in /* and */ to avoid it from being executed

JavaScript variables

JavaScript has variables. Variables can be thought of as named containers.

Variables are declared with the **var** keyword as follows.

Eg –

```
<script type = "text/javascript">
   var money;
   money = 2000.50;
   var name, age;
   name =“Ali”;
</script>
```

JavaScript variables

JavaScript is untyped language. This means that a JavaScript variable can hold a value of any data type. Unlike many other languages, you don't have to tell JavaScript during variable declaration what type of value the variable will hold. The value type of a variable can change during the execution of a program and JavaScript takes care of it automatically.

JavaScript variable Scope

JavaScript variables have only two scopes.

- **Global Variables** – A global variable has global scope which means it can be defined anywhere in your JavaScript code.
- **Local Variables** – A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.

JavaScript variable Scope


```
<html>
  <body onload = checkscope();>
    <script type = "text/javascript">
            var myVar = "global";     // Declare a global variable
        function checkscope( ) {
          var myVar = "local";    // Declare a local variable
          document.write(myVar);
        }
    </script>
  </body>
</html>
```

Hoisting of var

Hoisting is a JavaScript mechanism where variables and function declarations are moved to the top of their scope before code execution.

eg -

```
console.log (greeter);
var greeter = "say hello“
```

is interpreted as

```
var greeter;
console.log(greeter); // greeter is undefined
greeter = "say hello"
```

JavaScript var issues

```
var greeter = "hey hi";
  var times = 4;
  if (times > 3) {
     var greeter = "say Hello instead";
  }
     console.log(greeter) // "say Hello instead“
```

While this is not a problem if you knowingly want greeter to be redefined, it becomes a problem when you do not realize that a variable greeter has already been defined before.

JavaScript variable Scope

- **let**

A block is a chunk of code bounded by {}. A block lives in curly braces.

So a variable declared in a block with let is only available for use within that block.

Eg -

```
let greeting = "say Hi";
  let times = 4;
  if (times > 3) {
      let hello = "say Hello instead";
      console.log(hello);// "say Hello instead"
   }
  console.log(hello) // hello is not defined
```

JavaScript variable Scope

Just like var, a variable declared with let can be updated within its scope.
Unlike var, a let variable cannot be re-declared within its scope.

Eg - this works

```
let greeting = "say Hi";
greeting = "say Hello instead";
```

Whereas,

```
let greeting = "say Hi";
let greeting = "say Hello instead"; // error: Identifier 'greeting' has already been declared
```

- **Const**

Variables declared with the const maintain constant values. const declarations share some similarities with let declarations like const declarations are block scoped but const cannot be updated or re-declared

Eg -

```
const greeting = "say Hi";
greeting = "say Hello instead";    // error: Assignment to constant variable.
const greeting = "say Hello instead";   // error: Identifier 'greeting' has already been declared
```

JavaScript Basics

Var Vs let Vs const

- var declarations are globally scoped or function scoped while let and const are block scoped.
- var variables can be updated and re-declared within its scope; let variables can be updated but not re-declared; const variables can neither be updated nor re-declared.
- They are all hoisted to the top of their scope. But while var variables are initialized with undefined, let and const variables are not initialized.
- While var and let can be declared without being initialized, const must be initialized during declaration.

JavaScript operators

JavaScript supports the following types of operators.

- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

JavaScript operators

1. + Addition
2. **— Subtraction**
3. * Multiplication
4. / Division
5. (...) — Grouping operator, operations within brackets are executed earlier than those outside
6. % Modulus (remainder)
7. ++ Increment numbers
8. -- Decrement numbers
9.

JavaScript Basics

JavaScript operators

Sr.No	Comparison Operator & Description
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 condition becomes true. **Ex:** (A <= B) is true.

JavaScript operators

Sr.No	Logical Operator & 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.

Conditional (Ternary) Operator

JavaScript also contains a conditional operator that assigns a value to a variable based on some condition.

Syntax

```
variablename = (condition) ? value1:value2
```

- The **typeof** operator is a unary operator that is placed before its single operand, which can be of any type. Its value is a string indicating the data type of the operand.

Type	String Returned by typeof
Number	"number"
String	"string"
Boolean	"boolean"
Object	"object"
Function	"function"
Undefined	"undefined"
Null	"object"

JavaScript Basics

JavaScript Functions

- A JavaScript function is a block of code designed to perform a particular task.
- A JavaScript function is defined with the function keyword, followed by a name, followed by parentheses ().
- Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).The parentheses may include parameter names separated by commas:
- (parameter1, parameter2, ...)
- The code to be executed, by the function, is placed inside curly brackets: {}

JavaScript Functions

Syntax

```
function name(parameter1, parameter2, parameter3)
{
    // code to be executed
}
```

Eg –

```
function sayHello()
{
  alert(  "Hello World")
}
```

```
</head>
<script type = "text/javascript">
        function sayHello() {
          document.write ("Hello there!");
        }
    </script>
</head>

  <body>
<form>
        <input type = "button" onclick = "sayHello()" value = "Say Hello">
    </form>
    <p>Use different text in write method and then try...</p>
  </body>
```

JavaScript Functions with parameters


```
<head>
    <script type = "text/javascript">
      function sayHello(name, age) {
        document.write (name + " is " + age + " years old.");
      }
    </script>
  </head>

  <body>
<form>
      <input type = "button" onclick = "sayHello('Zara', 7)" value = "Say Hello">
    </form>
</body>
```

JavaScript Arrow function

Arrow functions allow us to write shorter function syntax:
Eg – normal function

```
hello = function() {
  return "Hello World!";
}
```

Can be rewritten as

```
hello = () => {   return "Hello World!"; }
```

If the function has only one statement, and the statement returns a value, you can remove the brackets and the return keyword:

JavaScript Screen output

JavaScript can "display" data in different ways:

- Writing into an HTML element, using innerHTML.
- Writing into the HTML output using document.write().
- Writing into an alert box, using window.alert().

Using innerHTML

To access an HTML element, JavaScript can use the **document.getElementById(id)** method.

The **id** attribute defines the HTML element. The **innerHTML** property defines the HTML content:

JavaScript Object

- You define (and create) a JavaScript object with an object literal:
- JavaScript objects are containers for **named values** called properties or methods.
- The values are written as **name:value** pairs (name and value separated by a colon).

Object Definition

You define (and create) a JavaScript object with an object literal:

```
var person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};
```

JavaScript Object

The **name:values** pairs in JavaScript objects are called **properties**

Property	Property Value
firstName	John
lastName	Doe
age	50
eyeColor	blue

Accessing Object Properties

You can access object properties in two ways:

objectName.propertyName

objectName["propertyName"]

JavaScript Object

```
<script>
var person = {
  firstName: "John",
  lastName : "Doe",
  id     :  5566  };
document.getElementById("demo").innerHTML    =person.firstName    +    "    "    +
person.lastName;
</script>
```

or

```
// Display some data from the object:
document.getElementById("demo").innerHTML    =person["firstName"]    +    "    "    +
person["lastName"];
```

JavaScript Object

Date objects are created with the new Date() constructor.

There are 4 ways to create a new date object:

- new Date() - creates a new date object with the current date and time
- new Date(year, month, day, hours, minutes, seconds, milliseconds)
- new Date(milliseconds)
- new Date(date string)

Method	Description
getFullYear()	Get the **year** as a four digit number (yyyy)
getMonth()	Get the **month** as a number (0-11)
getDate()	Get the **day** as a number (1-31)
getHours()	Get the **hour** (0-23)
getMinutes()	Get the **minute** (0-59)
getSeconds()	Get the **second** (0-59)
getMilliseconds()	Get the **millisecond** (0-999)
getTime()	Get the time (milliseconds since January 1, 1970)
getDay()	Get the weekday as a number (0-6)

JavaScript Object

The JavaScript Math object allows you to perform mathematical tasks on numbers.

```
<body>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML =
"<p><b>Math.E:</b> " + Math.E + "</p>" +
"<p><b>Math.PI:</b> " + Math.PI + "</p>" +
"<p><b>Math.SQRT2:</b> " + Math.SQRT2 + "</p>"
</script>
</body>
```

JavaScript Arrays

JavaScript arrays are used to store multiple values in a single variable. An array can hold many values under a single name, and you can access the values by referring to an index number.

Syntax

var *array_name* = [*item1*, *item2*, ...];

Arrays can be created by **new** keyword

```
var cars = new Array("Saab", "Volvo", "BMW");
```

Access the Elements of an Array

You access an array element by referring to the **index number**.

```
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars[0];
```

JavaScript Arrays

```
<html>
<body>
<h2>JavaScript Arrays</h2>
<p id="demo"></p>
<script>
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
</script>
</body>
</html>
```

Changing an Array Element

```
var cars = ["Saab", "Volvo", "BMW"];
cars[0] = "Opel";
document.getElementById("demo").innerHTML = cars[0];
```

Array Elements Can Be Objects

JavaScript variables can be objects. Arrays are special kinds of objects. We can have objects in an Array. We can have functions in an Array. We can have arrays in an Array.

```
myArray[0] = Date.now;
myArray[1] = myFunction;
myArray[2] = myCars;
```

Array Properties and Methods

1) The **length** property of an array returns the length of an array (the number of array elements).

Eg –

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.length;   // the length of fruits is 4
```

Accessing the First Array Element

```
<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
var first = fruits[0];
document.getElementById("demo").innerHTML = first;
</script>
```

JavaScript Arrays - looping

Eg –

```
<script>
var fruits, text, fLen, i;
fruits = ["Banana", "Orange", "Apple", "Mango"];
fLen = fruits.length;
text = "<ul>";
for (i = 0; i < fLen; i++) {
  text += "<li>" + fruits[i] + "</li>";
}
text += "</ul>";
document.getElementById("demo").innerHTML = text;
</script>
```

Adding Array Elements

Both methods adds a new element (Lemon) to fruits

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Lemon");
```

Or

```
Fruits[fruits.length] = “Lemon”
```

JavaScript Arrays method

1) Converting Arrays to Strings

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.toString();
```

2) The **join()** method also joins all array elements into a string.

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.join(" * ");
```

3) **Popping and Pushing**

The pop() method removes the last element from an array:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.pop();
```

4) **Shifting Elements**

The shift() method removes the first array element and "shifts" all other elements to a lower index.

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.shift();
```

The **unshift()** method adds a new element to an array (at the beginning), and "unshifts" older elements. The unshift() method returns the new array length.

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.unshift("Lemon");    // Adds a new element "Lemon" to fruits
```

5) **Deleting Elements**

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
delete fruits[0];          // Changes the first element in fruits to undefined
```

6) **Splicing an Array**

The splice() method can be used to add new items to an array.

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 0, "Lemon", "Kiwi");
```

- The first parameter (2) defines the position where new elements should be added (spliced in).
- The second parameter (0) defines how many elements should be removed.
- The rest of the parameters ("Lemon" , "Kiwi") define the new elements to be added.
- The splice() method returns an array with the deleted items:

Eg –

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 2, "Lemon", "Kiwi");
```

7) **Merging (Concatenating) Arrays**

The concat() method creates a new array by merging (concatenating) existing arrays

```
var arr1 = ["Cecilie", "Lone"];
var myChildren = arr1.concat(["Emil", "Tobias", "Linus"]);
```

8) **Sorting an Array**

The sort() method sorts an array alphabetically.

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.sort();        // Sorts the elements of fruits
```

9) The **reverse()** method reverses the elements in an array.

```
fruits.reverse();
```

Prototype Inheritance

All JavaScript objects inherit properties and methods from a prototype:

- Date objects inherit from Date.prototype
- Array objects inherit from Array.prototype
- Person objects inherit from Person.prototype

The Object.prototype is on the top of the prototype inheritance chain:

Date objects, Array objects, and Person objects inherit from Object.prototype.

- Sometimes you want to add new properties (or methods) to all existing objects of a given type.
- Sometimes you want to add new properties (or methods) to an object constructor.

Using the prototype Property

- The JavaScript prototype property allows you to add new properties to object constructors:
- The JavaScript prototype property also allows you to add new methods to objects constructors:

.