javascript

JS

By
Abdul Yesdani

Javascript introduction

JavaScript is the world's most popular programming language.

JavaScript is the programming language of the Web. JavaScript is easy to learn.

Alongside HTML and CSS, JavaScript is one of the core technologies of the World Wide Web. Over 97% of websites use it client-side for web page behavior.

Most web browsers have a dedicated JavaScript engine or JavaScript virtual machine to execute the code.

Different engines have different names For example: V8 – in Chrome and Opera. SpiderMonkey – in Firefox.

How it Works?

The engine (embedded if it’s a browser) reads (“parses”) the script.

Then it converts (“compiles”) the script to the machine language.

And then the machine code runs, pretty fast.

Content change with javascript in HTML element

javascript can change HTML element content:

```
document.getElementById("demo").innerHTML = "Hello JavaScript";
```

Can change HTML style:

```
document.getElementById("demo").style.fontSize = "35px";
```

Can hide/ show a HTML element:

```
document.getElementById("demo").style.display = "none";

document.getElementById("demo").style.display = "block";
```

Javascript intro

in-browser JavaScript is able to:

- Add new HTML to the page, change the existing content, modify styles.
- React to user actions, run on mouse clicks, pointer movements, key presses.
- Send requests over the network to remote servers, download and upload files (AJAX technology).
- Get and set cookies, ask questions to the visitor, show messages.
- Remember the data on the client-side (“local storage”).

Where is it inserted?

In HTML, JavaScript code is inserted between <script> and </script> tags.

```
<script>

document.getElementById("demo").innerHTML = "My First JavaScript";

</script>
```

Javascript functions and events

A JavaScript `function` is a block of JavaScript code, that can be executed when "called" for.

For example, a function can be called when an event occurs, like when the user clicks a button.

Scripts can be placed in the <body>, or in the <head> section of an HTML page, or in both.

Placing scripts at the bottom of the <body> element improves the display speed, because script interpretation slows down the display.

Scripts can also be placed in external files:

Advantages of external javascript files

Placing scripts in external files has some advantages:

- It separates HTML and code
- It makes HTML and JavaScript easier to read and maintain
- Cached JavaScript files can speed up page loads

To add several script files to one page - use several script tags:

```
<script src="myScript1.js"></script>

<script src="myScript2.js"></script>
```

Referencing external script

An external script can be referenced in 3 different ways:

- With a full URL (a full web address)
- With a file path (like /js/)
- Without any path
- `<script src="https://www.google.com/js/myScript.js"></script>`
- `<script src="/js/myScript.js"></script>`
- `<script src="myScript.js"></script>`

Displaying data

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

```
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML =”Hello! World”;
</script>
<button onclick="window.print()">Print this page</button>
```

Identifiers and naming convention

In JavaScript, the first character must be a letter, or an underscore (_), or a dollar sign ($).

Subsequent characters may be letters, digits, underscores, or dollar signs.

All JavaScript identifiers are case sensitive.

Examples:

Let student_name;

Var $courseTytle;

variables

```
// How to create variables:

var x;

let y;

Const z=100;

x = 5;              // How to use variables:

y = 6;

let z = x + y;  //expression assigned to variable
```

variables

```
<script>

var carName = "Volvo";

Var price= 7890000

document.getElementById("demo").innerHTML = carName;

document.getElementById("de1").innerHTML = price;

</script>

<h2 id=”demo”> </h2>

<div id=”de1”> </div>
```

Let keyword

The `let` keyword was introduced in ES6 (2015).

Variables defined with `let` cannot be Redeclared.

Variables defined with `let` must be Declared before use.

Variables defined with `let` have Block Scope.

```
let x = "John Doe";

let x = 0;

// SyntaxError: 'x' has already been declared
```

Const keyword

Variables defined with `const` cannot be Redeclared.

Variables defined with `const` cannot be Reassigned.

Variables defined with `const` have Block Scope.

```
const PI = 3.141592653589793;

PI = 3.14;        // This will give an error

PI = PI + 10;     // This will also give an error
```

When constant ?

Use const when you declare:
A new Array
A new Object
A new Function
A new RegExp
It does not define a constant value. It defines a constant reference to a value.
you can NOT:
Reassign a constant value,
Reassign a constant array,
Reassign a constant object
But you CAN:Change the elements of constant array
Change the properties of constant object

Dynamic type

JavaScript has dynamic types. This means that the same variable can be used to hold different data types:

```
let x;              // Now x is undefined

x = 5;              // Now x is a Number

x = "John";         // Now x is a String
```

Var keyword

When JavaScript was first created, this was the only way to declare variables.

The design of var is confusing and error-prone. So let was created in modern versions of JavaScript, a new keyword for creating variables that works somewhat differently to var, fixing its issues in the process.

So now using var is irrelevant.

strings

A string (or a text string) is a series of characters like "John Doe".

Strings are written with quotes. You can use single or double quotes:

```
let carName1 = "Volvo XC60";   // Using double quotes

let carName2 = 'Volvo XC60';   // Using single quotes
```

numbers

Numbers can be written with, or without decimals:

```
let x1 = 34.00;      // Written with decimals

let x2 = 34;         // Written without decimals
```

Extra large or extra small numbers can be written with scientific (exponential) notation:

```
let y = 123e5;       // 12300000

let z = 123e-5;      // 0.00123
```

booleans

Booleans can only have two values: `true` or `false`.

```
let x = 5;

let y = 5;

let z = 6;

(x == y)        // Returns true

(x == z)        // Returns false
```

Arrays

JavaScript arrays are written with square brackets.

Array items are separated by commas.

The following code declares (creates) an array called `cars`, containing three items (car names):

```
const cars = ["Saab", "Volvo", "BMW"];

Const numbers=[789,6754,455546,7634];
```

Objects

JavaScript objects are written with curly braces `{ }`.

Object properties are written as name:value pairs, separated by commas.

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

Typeof operator

JavaScript `typeof` operator to find the type of a JavaScript variable.

The `typeof` operator returns the type of a variable or an expression:

```
typeof 314              // Returns "number"

typeof 3.14             // Returns "number"

typeof (3 + 4)             // Returns "number

typeof "John"           // Returns "string"
```

Calling function targeting a HTML element by Id

```
function toCelsius(fahrenheit) {

  return (5/9) * (fahrenheit-32); }

document.getElementById("demo").innerHTML = toCelsius(77);

 let x = toCelsius(77);
 let text = "The temperature is " + x + " Celsius";
```

You can use the function directly, as a variable value:

```
 let text = "The temperature is " + toCelsius(77) + "
 Celsius";
```

Objects

Object contains element with name:value pairs.

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

You can access object properties in two ways:

objectName.propertyName Or

objectName["propertyName"]

Example:

```
person.lastName    or      person[lastName]
```

Object Methods

Objects can also have methods.

Methods are actions that can be performed on objects.

Methods are stored in properties as function definitions.

```
const person = {
  firstName: "John",   lastName : "Doe",   id    : 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  } };
```

functions

A JavaScript function is a block of code designed to perform a particular task.

A JavaScript function is executed when "something" invokes it (calls it).

When an event occurs (when a user clicks a button)

When it is invoked (called) from JavaScript code

Automatically (self invoked)

```
function myFunction(p1, p2)
{

  return p1 * p2;   // The
function returns the product
of p1 and p2 }
```

This keyword

In a function definition, `this` refers to the "owner" of the function.

In the example above, `this` is the person object that "owns" the `fullName` function.

In other words, `this.firstName` means the `firstName` property of this object.

Accessing object methods

You access an object method with the following syntax:

```
objectName.methodName()
```

example:

```
name = person.fullName();
```

Calling the function

```
let x = myFunction(4, 3);    // Function is called,
return value will end up in x


function myFunction(a, b) {

  return a * b;

     // Function returns the product of a and b

}
```

Events

An HTML event can be something the browser does, or something a user does.

Here are some examples of HTML events:

- An HTML web page has finished loading
- An HTML input field was changed
- An HTML button was clicked

JavaScript lets you execute code when events are detected. HTML allows event handler attributes, with JavaScript code, to be added to HTML elements.

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

Events

```
<button onclick="displayDate()">The time is?</button>
<script>
function displayDate() {
  document.getElementById("demo").innerHTML = Date();
}
</script>
<p id="demo"></p>
```

Common events

Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

String

A JavaScript string is zero or more characters written inside quotes.

```
let answer1 = "It's alright";

let answer2 = "He is called 'Johnny'";

let answer3 = 'He is called "Jock"';

let text = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

text.length;     // Will return 26
```

Escape character

```
let text = "We are the so-called \"Vikings\" from the north.";
let text= 'It\'s alright.';
 let text = "The character \\ is called backslash.";
```

Other escape sequence characters:

\n : for new line

\t : for tab

\v : vertical tab

String methods

There are 3 methods for extracting a part of a string:

- `slice(start, end)`
- `substring(start, end)`
- `substr(start, length)`

```
let str = "Apple, Banana, Kiwi";

str.slice(7, 13)      // Returns Banana

substring(7, 13)      // Returns Banana

str.substr(7, 6)     // Returns Banana

str.substr(7)     // Returns Banana,Kiwi

str.substr(-4)     // Returns Kiwi
```

String methods

The `replace()` method replaces a specified value with another value in a string:

```
let text = "Please visit Microsoft!";

let newText = text.replace("Microsoft", "Google");
```

The `replace()` method does not change the string it is called on. It returns a new string.

By default, the `replace()` method replaces only the first match:

String methods

`concat()` joins two or more string

```
let text1 = "Hello";

let text2 = "World";

let text3 = text1.concat(" ", text2);
```

The `concat()` method can be used instead of the plus operator. These two lines do the same.

```
text = "Hello" + " " + "World!";

text = "Hello".concat(" ", "World!");
```

String methods

The `trim()` method removes whitespace from both sides of a string:

```
let text = "     Hello World!     ";

text.trim()     // Returns "Hello World!"

let text = "5";

text.padStart(4,0)     // Returns 0005

let text = "5";

text.padEnd(4,0)      // Returns 5000
```

Extracting characters

```
let text = "HELLO WORLD";

text.charAt(0)
```

The `charCodeAt()` method returns the unicode of the character at a specified index in a string:

The method returns a UTF-16 code (an integer between 0 and 65535).

```
let text = "HELLO WORLD";

text.charCodeAt(0)        // Returns 72

text[0]                    // returns H  with property access
```

String methods

A string can be converted to an array with the `split()` method:

```
text.split(",")        // Split on commas
text.split(" ")           // Split on spaces
text.split("|")           // Split on pipe
text.split("")          // Split in single characters
```

Dont do

```
x = new String();          // Declares x as a String object

y = new Number();          // Declares y as a Number object

z = new Boolean();         // Declares z as a Boolean object
```

Avoid `String`, `Number`, and `Boolean` objects. They complicate your code and slow down execution speed.

String methods

```
let text1 = "Hello World!";          // String
let text2 = text1.toUpperCase();  // text2 is
text1 converted to upper

 let text1 = "Hello World!";          // String

 let text2 = text1.toLowerCase();  // text2 is
 text1 converted to lower
```

Searching string

The match() method searches a string for a match against a regular expression, and returns the matches, as an Array object.

Search a string for "ain":

```
let text = "The rain in SPAIN stays mainly in the plain";

text.match(/ain/g)     // Returns an array [ain,ain,ain]
```

The `includes()` method returns true if a string contains a specified valu

```
let text = "Hello world, welcome to the universe.";

text.includes("world")     // Returns true
```

Searching string

The `startsWith()` method returns `true` if a string begins with a specified value, otherwise `false`:

```
let text = "Hello world, welcome to the universe.";

text.startsWith("Hello")    // Returns true


text.startsWith("world")     // Returns false

text.startsWith("world", 5)     // Returns false

text.startsWith("world", 6)     // Returns true
```

String searching

The `endsWith()` method returns `true` if a string ends with a specified value, otherwise `false`:

```
var text = "John Doe";

 text.endsWith("Doe")     // Returns true

 let text = "Hello world, welcome to the
 universe.";

 text.endsWith("world", 11)     // Returns true
```

Searching strings

- `let str = "Please locate where 'locate' occurs!";`
- `str.indexOf("locate")  // Returns 7`
- `let str = "Please locate where 'locate' occurs!";`
- `str.lastIndexOf("locate")  // Returns 21  last occurrence`
- `str.indexOf("locate", 15)  // Returns 21`
- `str.lastIndexOf("locate", 15)  // Returns 7`
- `str.search("locate")  // Returns 7`

Template literals

- Template Literals
- Template Strings
- String Templates
- Back-Tics Syntax

All are similar words.

Template Literals use back-ticks (``) rather than the quotes ("") to define a string:

```
let text = `Hello World!`;
```

With template literals, you can use both single and double quotes inside a string

```
let text = `He's often called "Johnny"`;
```

Template literals

Template literals allows multiline strings:

```
let text =

`The quick

brown fox

jumps over

the lazy dog`;
```

interpolation

Template literals provide an easy way to interpolate variables and expressions into strings.

This method is called string interpolation.

```
let firstName = "John"; let lastName = "Doe";

let text = `Welcome ${firstName}, ${lastName}!`;

let price = 10; let VAT = 0.25;

let total = `Total: ${(price * (1 + VAT)).toFixed(2)}`;
```

Array methods

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.toString();
document.getElementById("demo").innerHTML = fruits.join(" * ");
fruits.pop();  // Removes "Mango" from fruits   or
let x = fruits.pop();  // x = "Mango"
fruits.push("Kiwi");   // Adds "Kiwi" to fruits
let x = fruits.push("Kiwi");   //  x = 4
```

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

```
fruits.shift();   // Removes "Banana" from fruits
```

Array methods

`unshift()` method adds a new element to an array (at the beginning), and "unshifts" older elements:

```
fruits.unshift("Lemon");    // Adds "Lemon" to fruits

fruits[0] = "Kiwi";        // Changes the first element of fruits to "Kiwi"

fruits[fruits.length] ="Grapes";          // Appends "Grapes" to fruits

delete fruits[0];  // Changes the first element in fruits to undefined

document.getElementById("demo2").innerHTML ="The first fruit is: " + fruits[0];
```

Array methods

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

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];

fruits.splice(2, 0, "Lemon", "Kiwi"); // add lemon , kiwi at index 2

The splice() method adds new elements to an array, and returns an array with the deleted
elements.

let removed = fruits.splice(2, 2, "Lemon", "Kiwi")

fruits.splice(0, 1);   // Removes the first element
```

Array methods

```
const myGirls = ["Cecilie", "Lone"];// Concatenate (join) myGirls and
myBoys

const myBoys = ["Emil", "Tobias", "Linus"];

const myChildren = myGirls.concat(myBoys);

fruits.sort();          // Sorts the elements of fruits

fruits.reverse();       // Then reverse the order of the elements

const points = [40, 100, 1, 5, 25, 10];   // sorting numbers

points.sort(function(a, b){return a - b});

points.sort(function(a, b){return b - a});   // descending sort
```

Array methods- iteration

The `forEach()` method calls a function (a callback function) once for each array element

```
const numbers = [45, 4, 9, 16, 25];
let txt = "";
numbers.forEach(myFunction);
function myFunction(value, index, array) {
  txt += value + "<br>"; }
document.getElementById("demo").innerHTML = txt;
```

Array map()

The `map()` method creates a new array by performing a function on each array element.

```
const numbers1 = [45, 4, 9, 16, 25];
const numbers2 = numbers1.map(myFunction);
document.getElementById("demo").innerHTML = numbers2;
function myFunction(value, index, array) {
  return value * 2;
}
```

Creating Dates

```
<script>
const d = new Date();
const dt = new Date(2021, 11, 24);
document.getElementById("demo").innerHTML = d;
//previous century date:
const d = new Date(99, 0, 24);
Sun Jan 24 1999 00:00:00 GMT+0530 (India Standard Time)
</script>
```

Date formats

```
const d = new Date("2021-03-25T12:00:00Z"); //ISO

const d = new Date("03/25/2021");   // short date

const d = new Date("Mar 25 2015");  // long date

const d = new Date("Mar 25 2015");

const dt = new Date();

dt.setFullYear(2020, 11, 3);    // set the date.
```

Get date methods

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)
Date.now()	Get the time. ECMAScript 5.

Math methods

```
Math.round(4.7);     // returns 5

Math.round(4.4);     // returns 4

Math.ceil(4.2);      // returns 5

Math.floor(4.9);     // returns 4

Math.trunc(4.9);     // returns 4

Math.pow(8, 2);       // returns 64

Math.sqrt(64);       // returns 8

Math.floor(Math.random() * 10);  // random no 0 -10
```

Decision making

```
<input id="age" value="18" />

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>

function myFunction() {

  let age = document.getElementById("age").value;

  let voteable = (age < 18) ? "Too young":"Old enough";

  document.getElementById("demo").innerHTML = voteable + " to vote.";     </script>
```

conditions

```
if (time < 10) {
  greeting = "Good morning";
} else if (time < 20) {
  greeting = "Good day";
} else {
  greeting = "Good evening";
}
```

Switch case

```
switch (new Date().getDay()) {
  case 0:
    day = "Sunday";
    break;
  case 1:
    day = "Monday";
    break;
  case 2:
     day = "Tuesday";
    break;
  case 3:
    day = "Wednesday";
    break;
  case 4:
    day = "Thursday";
    break;
  case 5:
    day = "Friday";
    break;
  case 6:     day = "Saturday"; }
```

For loop

```
const cars = ["BMW", "Volvo", "Saab", "Ford", "Fiat", "Audi"];
let text = "";
for (let i = 0; i < cars.length; i++) {
  text += cars[i] + "<br>";
}
document.getElementById("demo").innerHTML = text;
```

For in loop

```
const person = {fname:"John", lname:"Doe", age:25};

let text = "";

for (let x in person) {

  text += person[x];

}
```

Call ( )

call() used to invoke (call) a method with an owner object as an argument.

```
const person = {
  fullName: function() {
    return this.firstName + " " + this.lastName;
  }  }
const person1 = {
  firstName:"John",
  lastName: "Doe" }
document.getElementById("demo").innerHTML = person.fullName.call(person1);
```

apply( )

`apply()` method, you can write a method that can be used on different objects.

```
const person = {
  fullName: function() {
    return this.firstName + " " + this.lastName;
  } }
const person1 = {
  firstName: "Mary",
  lastName: "Doe" }
person.fullName.apply(person1);
```

class

```
class Car {
  constructor(name, year) {
    this.name = name;
    this.year = year;
  }
  age(x) {
    return x - this.year;
  } }
```

```
let date = new Date();
let year =
date.getFullYear();
let myCar = new Car("Ford",
2014);
document.getElementById("dem
o").innerHTML=
"My car is " +
myCar.age(year)+"years old";
```

Inheritance

```
class Car {
  constructor(brand) {
    this.carname =
brand;
  }
  present() {
    return 'I have a '
+ this.carname;
  }
}
```

```
class Model extends Car {
  constructor(brand, mod)
{
    super(brand);
    this.model = mod;
  }
  show() {
    return this.present()
+ ', it is a ' +
this.model;
  }}
```

inheritance

```
let myCar = new Model("Ford", "Mustang");

document.getElementById("demo").innerHTML =
myCar.show();
```

The `super()` method refers to the parent class.

By calling the `super()` method in the constructor method, we call the parent's constructor method and gets access to the parent's properties and methods.

Setter and getter

```
class Car {
  constructor(brand) {
    this.carname = brand;
  }
  get cnam() {
    return this.carname;
  }
set cnam(x) {
    this.carname = x;
  }
}
let myCar = new Car("Ford");

document.getElementById("demo").innerHTML = myCar.cnam;
```

Function sequence

JavaScript functions are executed in the sequence they are called. Not in the sequence they are defined.

```
function myFirst() {
  myDisplayer("Hello"); }
function mySecond() {
  myDisplayer("Goodbye"); }
mySecond();
myFirst();
```

Asynchronous

Functions running in parallel with other functions are called asynchronous.

using the JavaScript function `setTimeout()`, you can specify a callback function to be executed on time-out:

```
setTimeout(myFunction, 3000);

function myFunction() {
document.getElementById("demo
").innerHTML = "Hello!
World”;}
```

myFunction is passed to `setTimeout()` as an argument.

3000 is the number of milliseconds before time-out, so `myFunction()` will be called after 3 seconds.

Promise

A Promise is a JavaScript object that links producing code and consuming code.

```
const myPromise = new Promise(function(myResolve, myReject) {
  setTimeout(function(){ myResolve("I love You !!"); }, 3000);
});
myPromise.then(function(value) {
  document.getElementById("demo").innerHTML = value;
});
```

Async and await

async and await make promises easier to write. async makes a function return a Promise. await makes a function wait for a Promise.

The keyword `async` before a function makes the function return a promise.

```
async function myFunction()
{    return "Hello";

}
```

```
function myDisplayer(some) {
document.getElementById("demo").innerHTML = some;

}

async function myFunction() {

return "Hello";}

myFunction().then(

  function(value) {myDisplayer(value);},

  function(error) {myDisplayer(error);}
```

Async - await

```
async function myDisplay() {
  let myPromise = new Promise(function(resolve) {
    setTimeout(function() {resolve("Hello! World!!");}, 3000);
  });
  document.getElementById("demo").innerHTML = await myPromise;
}
myDisplay();
```

Validation for form data

```
<script>
function validateForm() {
  let x =
document.forms["myForm"]["fname"].value;
  if (x == "") {
    alert("Name must be filled out");
    return false;
  }
}
</script>
```

```
<form name="myForm"
action="/action_page.php"
onsubmit="return validateForm()"
method="post">
  Name: <input type="text"
name="fname">
  <input type="submit" value="Submit">
</form>
```

Arrow functions

Arrow functions allow us to write shorter function syntax.

```
hello = () => "Hello World!";

hello = (val) => "Hello " + val;

document.getElementById("demo").innerHTML =
hello("Universe!");
```

End

Email Id:

yasdanis@gmail.com

Blog:

https://askyedu.blogspot.com/