Why Study JavaScript?

JavaScript is one of the 3 languages all web developers must learn:

1. HTML to define the content of web pages

2. CSS to specify the layout of web pages

3. JavaScript to program the behavior of web pages

You don't have to get or download JavaScript.

JavaScript is already running in your browser on your computer, on your tablet, and on your smart-phone.

JavaScript is free to use for everyone.

JavaScript Introduction

JavaScript Can Change HTML Content

One of many JavaScript HTML methods is getElementById().

The example below "finds" an HTML element (with id="demo"), and changes the element content (innerHTML) to "Hello JavaScript":

<button type="button" onclick="document.getElementById('ja').innerHTML='bruh'">as</button>

JavaScript Can Change HTML Styles (CSS)

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

The <script> Tag

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

JavaScript in <head> or <body>

You can place any number of scripts in an HTML document.

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

<script>

function myFunction() {

document.getElementById("demo").innerHTML = "Paragraph changed.";

}</script>

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

External JavaScript

External scripts are practical when the same code is used in many different web pages.

JavaScript files have the file extension .js.

To use an external script, put the name of the script file in the src (source) attribute of a <script> tag

External References

External scripts can be referenced with a full URL or with a path relative to the current web page.

JavaScript Display Possibilities

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().

Writing into the browser console, using console.log()

Using document.write()

<body>

<h1>My First Web Page</h1>

<p>My first paragraph.</p>

<script>

document.write(5 + 6);

</script>

</body>

Pop up

<script>

alert(5 + 6);

</script>

Using console.log()

For debugging purposes, you can call the console.log() method in the browser to display data.

<button onclick="window.print()">Print this page</button>

JavaScript Statements

<script>

var x, y, z; // Statement 1

x = 5; // Statement 2

y = 6; // Statement 3

z = x + y; // Statement 4

document.getElementById("demo").innerHTML =

"The value of z is " + z + ".";

</script>

JavaScript variables are containers for storing data values.

In this example, x, y, and z, are variables, declared with the var keyword:

const keyword to define a variable that cannot be reassigned, and the let keyword to define a variable with restricted scope.

const keyword to define a variable that cannot be reassigned, and the let keyword to define a variable with restricted scope.

JavaScript can handle many types of data, but for now, just think of numbers and strings.

Strings are written inside double or single quotes. Numbers are written without quotes.

If you put a number in quotes, it will be treated as a text string.

You can declare many variables in one statement.

var person = "John Doe", carName = "Volvo", price = 200;

Remember that JavaScript identifiers (names) must begin with:

A letter (A-Z or a-z)

A dollar sign ($)

Or an underscore (\_)

Since JavaScript treats a dollar sign as a letter, identifiers containing $ are valid variable names:

JavaScript Underscore (\_)

Since JavaScript treats underscore as a letter, identifiers containing \_ are valid variable names:

JavaScript Operators

The assignment operator (=) assigns a value to a variable.

The addition operator (+) adds numbers:

Operator Description

+ Addition

- Subtraction

\* Multiplication

\*\* Exponentiation (ES2016)

/ Division

% Modulus (Division Remainder)

++ Increment

-- Decrement

== equal to

=== equal value and equal type

!= not equal

!== not equal value or not equal type

> greater than

< less than

>= greater than or equal to

<= less than or equal to

? ternary operator

The numbers (in an arithmetic operation) are called operands.

The operation (to be performed between the two operands) is defined by an operator.

var x = 5;

document.getElementById("demo").innerHTML = Math.pow(x,2);

JavaScript Data Types

JavaScript variables can hold many data types: numbers, strings, objects and more:

var length = 16; // Number

var lastName = "Johnson"; // String

var x = {firstName:"John", lastName:"Doe"}; // Object

When adding a number and a string, JavaScript will treat the number as a string.

var x = "Volvo" + 16;

JavaScript evaluates expressions from left to right. Different sequences can produce different results:

var x = 16 + 4 + "Volvo";

Result:

20Volvo

JavaScript Types are Dynamic

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

Example

var x; // Now x is undefined

x = 5; // Now x is a Number

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

JavaScript 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:

Example

var carName1 = "Volvo XC60"; // Using double quotes

var carName2 = 'Volvo XC60'; // Using single quotes

JavaScript Numbers

JavaScript has only one type of numbers.

Numbers can be written with, or without decimals:

Example

var x1 = 34.00; // Written with decimals

var x2 = 34; // Written without decimals

res:

34

34

JavaScript Booleans

Booleans can only have two values: true or false.

JavaScript Booleans

Booleans can only have two values: true or false.

true

false

Booleans are often used in conditional testing.

JavaScript Arrays

JavaScript arrays are written with square brackets.

var cars = ["Saab", "Volvo", "BMW"];

document.getElementById("demo").innerHTML = cars[0];

Array indexes are zero-based, which means the first item is [0], second is [1], and so on.

JavaScript Objects

JavaScript objects are written with curly braces {}.

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

The typeof Operator

You can use the JavaScript typeof operator to find the type of a JavaScript variable.

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

An empty value has nothing to do with undefined.

An empty string has both a legal value and a type.

var car = "";

Null

In JavaScript null is "nothing". It is supposed to be something that doesn't exist.

Unfortunately, in JavaScript, the data type of null is an object.

JavaScript Functions

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

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

<script>

function myFunction(p1, p2) {

return p1 \* p2;

}

document.getElementById("demo").innerHTML = myFunction(4, 3);

</script>

JavaScript Function Syntax

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: {}

Function parameters are listed inside the parentheses () in the function definition.

Function arguments are the values received by the function when it is invoked.

Function Invocation

The code inside the function will execute when "something" invokes (calls) the function:

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

When it is invoked (called) from JavaScript code

Automatically (self invoked)

Function Return

When JavaScript reaches a return statement, the function will stop executing.

If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.

Functions often compute a return value. The return value is "returned" back to the "caller":

Why Functions?

You can reuse code: Define the code once, and use it many times.

You can use the same code many times with different arguments, to produce different results.

Local Variables

Variables declared within a JavaScript function, become LOCAL to the function.

Local variables can only be accessed from within the function.

JavaScript Objects

Real Life Objects, Properties, and Methods

In real life, a car is an object.

A car has properties like weight and color, and methods like start and stop:

All cars have the same properties, but the property values differ from car to car.

All cars have the same methods, but the methods are performed at different times.

JavaScript Objects

You have already learned that JavaScript variables are containers for data values.

This code assigns a simple value (Fiat) to a variable named car:

Objects are variables too. But objects can contain many values.

This code assigns many values (Fiat, 500, white) to a variable named car:

var car = {type:"Fiat", model:"500", color:"white"};

The values are written as name:value pairs (name and value separated by a colon).

Accessing Object Properties

You can access object properties in two ways:

objectName.propertyName

objectName["propertyName"]

If you access a method without the () parentheses, it will return the function definition:

JavaScript Events

HTML events are "things" that happen to HTML elements.

When JavaScript is used in HTML pages, JavaScript can "react" on these events.

HTML 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

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

<element event="some JavaScript">

JavaScript code is often several lines long. It is more common to see event attributes calling functions:

Common HTML Events

Here is a list of some common HTML 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

What can JavaScript Do?

Event handlers can be used to handle, and verify, user input, user actions, and browser actions:

Things that should be done every time a page loads

Things that should be done when the page is closed

Action that should be performed when a user clicks a button

Content that should be verified when a user inputs data

JavaScript Strings

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

String Length

To find the length of a string, use the built-in length property:

String Length

To find the length of a string, use the built-in length property:

Strings Can be Objects

Normally, JavaScript strings are primitive values, created from literals:

var firstName = "John";

But strings can also be defined as objects with the keyword new:

var firstName = new String("John");

When using the === operator, equal strings are not equal, because the === operator expects equality in both type and value.

JavaScript String Methods

var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

var sln = txt.length;

Finding a String in a String

The indexOf() method returns the index of (the position of) the first occurrence of a specified text in a string:

var str = "Please locate where 'locate' occurs!";

var pos = str.indexOf("locate");

The lastIndexOf() method returns the index of the last occurrence of a specified text in a string:

var str = "Please locate where 'locate' occurs!";

var pos = str.lastIndexOf("locate");

The lastIndexOf() methods searches backwards (from the end to the beginning), meaning: if the second parameter is 15, the search starts at position 15, and searches to the beginning of the string.

Searching for a String in a String

The search() method searches a string for a specified value and returns the position of the match:

var str = "Please locate where 'locate' occurs!";

var pos = str.search("locate");

The slice() Method

slice() extracts a part of a string and returns the extracted part in a new string.

var str = "Apple, Banana, Kiwi";

var res = str.slice(7, 13);

The substring() Method

substring() is similar to slice().

The difference is that substring() cannot accept negative indexes

var str = "Apple, Banana, Kiwi";

var res = str.substring(7, 13);

Banana

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

str = "Please visit Microsoft!";

var n = str.replace("Microsoft", "W3Schools");

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

var text2 = text1.toUpperCase();

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

var text2 = text1.toLowerCase();

String.trim()

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

var str = " Hello World! ";

alert(str.replace(/^[\s\uFEFF\xA0]+|[\s\uFEFF\xA0]+$/g, ''));

var str = "HELLO WORLD";

str.charAt(0); // returns H

var txt = "a,b,c,d,e"; // String

txt.split(","); // Split on commas

f the separator is omitted, the returned array will contain the whole string in index [0].

JavaScript Numbers

JavaScript has only one type of number. Numbers can be written with or without decimals.

var x = 3.14;

var y = 3;

JavaScript Numbers are Always 64-bit Floating Point

Unlike many other programming languages, JavaScript does not define different types of numbers, like integers, short, long, floating-point etc.

Precision

Integers (numbers without a period or exponent notation) are accurate up to 15 digits.

Adding Numbers and Strings

JavaScript uses the + operator for both addition and concatenation.

Numbers are added. Strings are concatenated.

var x = 10;

var y = 20;

var z = x + y; // z will be 30 (a number)

If you add two strings, the result will be a string concatenation:

var x = "10";

var y = "20";

var z = x + y; // z will be 1020 (a string)

If you add a number and a string, the result will be a string concatenation:

If you add a number and a string, the result will be a string concatenation:

var x = 100 / "Apple";

isNaN(x); // returns true because x is Not a Number

Infinity

Infinity (or -Infinity) is the value JavaScript will return if you calculate a number outside the largest possible number.

var myNumber = 2;

while (myNumber != Infinity) { // Execute until Infinity

myNumber = myNumber \* myNumber;

}

Hexadecimal

JavaScript interprets numeric constants as hexadecimal if they are preceded by 0x.

var x = 0xFF; // x will be 255

Never write a number with a leading zero (like 07).

Some JavaScript versions interpret numbers as octal if they are written with a leading zero.

Numbers Can be Objects

Normally JavaScript numbers are primitive values created from literals:

var x = 123;

But numbers can also be defined as objects with the keyword new:

var y = new Number(123);

Do not create Number objects. It slows down execution speed.

The new keyword complicates the code. This can produce some unexpected results

When using the === operator, equal numbers are not equal, because the === operator expects equality in both type and value.

JavaScript Number Methods

Number Methods and Properties

Primitive values (like 3.14 or 2014), cannot have properties and methods (because they are not objects).

But with JavaScript, methods and properties are also available to primitive values, because JavaScript treats primitive values as objects when executing methods and properties.

The toString() method returns a number as a string.

The toExponential() Method

toExponential() returns a string, with a number rounded and written using exponential notation.

The toFixed() Method

toFixed() returns a string, with the number written with a specified number of decimals:

var x = 9.656;

x.toFixed(0);

Number(true); // returns 1

Number(false); // returns 0

Number("10"); // returns 10

Number(" 10"); // returns 10

Number("10 "); // returns 10

Number(" 10 "); // returns 10

Number("10.33"); // returns 10.33

parseInt("10"); // returns 10

parseInt("10.33"); // returns 10

The parseFloat() Method

parseFloat() parses a string and returns a number. Spaces are allowed. Only the first number is returned:

Number Properties

Property Description

MAX\_VALUE Returns the largest number possible in JavaScript

MIN\_VALUE Returns the smallest number possible in JavaScript

var x = Number.MAX\_VALUE;

JavaScript Arrays

var cars = ["Saab", "Volvo", "BMW"];

What is an Array?

An array is a special variable, which can hold more than one value at a time.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

Using the JavaScript Keyword new

The following example also creates an Array, and assigns values to it:

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

Access the Elements of an Array

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

This statement accesses the value of the first element in cars:

var cars = ["Saab", "Volvo", "BMW"];

document.getElementById("demo").innerHTML = cars[0];

Changing an Array Element

This statement changes the value of the first element in cars:

cars[0] = "Opel";

Arrays are Objects

Arrays are a special type of objects. The typeof operator in JavaScript returns "object" for arrays.

But, JavaScript arrays are best described as arrays.

Arrays use numbers to access its "elements". In this example, person[0] returns John:

Array Elements Can Be Objects

JavaScript variables can be objects. Arrays are special kinds of objects.

Because of this, you can have variables of different types in the same Array.

Array Properties and Methods

var x = cars.length; // The length property returns the number of elements

var y = cars.sort(); // The sort() method sorts arrays

Accessing the Last Array Element

Example

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

var last = fruits[fruits.length - 1];

New element can also be added to an array using the length property:

var fruits = ["Banana", "Orange", "Apple", "Mango"];

fruits[fruits.length] = "Lemon"; // adds a new element (Lemon) to fruits

The Difference Between Arrays and Objects

In JavaScript, arrays use numbered indexes.

In JavaScript, objects use named indexes.

Avoid new Array()

There is no need to use the JavaScript's built-in array constructor new Array().

Use [] instead.

var fruits = ["Banana", "Orange", "Apple", "Mango"];

typeof fruits; // returns object

The typeof operator returns object because a JavaScript array is an object.

Converting Arrays to Strings

The JavaScript method toString() converts an array to a string of (comma separated) array values.

Converting Arrays to Strings

Example

var fruits = ["Banana", "Orange", "Apple", "Mango"];

document.getElementById("demo").innerHTML = fruits.toString();

Result:

Banana,Orange,Apple,Mango

var fruits = ["Banana", "Orange", "Apple", "Mango"];

document.getElementById("demo").innerHTML = fruits.join(" \* ");

Result:

Banana \* Orange \* Apple \* Mango

Popping and Pushing

When you work with arrays, it is easy to remove elements and add new elements.

This is what popping and pushing is:

Popping items out of an array, or pushing items into an array.

Popping

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

Example

var fruits = ["Banana", "Orange", "Apple", "Mango"];

fruits.pop(); // Removes the last element ("Mango") from fruits

var fruits = ["Banana", "Orange", "Apple", "Mango"];

fruits.push("Kiwi");

shift()

The shift() method removes the first element of an array (and "shifts" all other elements to the left):

Banana,Orange,Apple,Mango

Orange,Apple,Mango

var fruits = ["Banana", "Orange", "Apple", "Mango"];

fruits.unshift("Lemon"); // Adds a new element "Lemon" to fruits

Deleting Elements

Since JavaScript arrays are objects, elements can be deleted by using the JavaScript operator delete:

Example

var fruits = ["Banana", "Orange", "Apple", "Mango"];

delete fruits[0];

Splicing an Array

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

Example

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:

Merging (Concatenating) Arrays

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

Example (Merging Two Arrays)

var myGirls = ["Cecilie", "Lone"];

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

var myChildren = myGirls.concat(myBoys);

Banana,Orange,Lemon,Apple,Mango

Orange,Lemon,Apple,Mango

Slicing an Array

The slice() method slices out a piece of an array into a new array.

This example slices out a part of an array starting from array element 1 ()

Sorting an Array

The sort() method sorts an array alphabetically:

var fruits = ["Banana", "Orange", "Apple", "Mango"];

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

The compare function compares all the values in the array, two values at a time (a, b).

The Compare Function

The purpose of the compare function is to define an alternative sort order.

var points = [40, 100, 1, 5, 25, 10];

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

The compare function should return a negative, zero, or positive value, depending on the arguments:

When comparing 40 and 100, the sort() method calls the compare function(40, 100).

The function calculates 40 - 100 (a - b), and since the result is negative (-60), the sort function will sort 40 as a value lower than 100

Find the Highest (or Lowest) Array Value

There are no built-in functions for finding the max or min value in an array.

However, after you have sorted an array, you can use the index to obtain the highest and lowest values.

Sorting ascending:

var points = [40, 100, 1, 5, 25, 10];

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

// now points[0] contains the lowest value

// and points[points.length-1] contains the highest value

Note that the function takes 3 arguments:

The item value

The item index

The array itself

var a = [1, 2];

a.forEach(myFunction);

function myFunction(value, index, array) {

console.log(value);

}

Array.map()

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

The map() method does not execute the function for array elements without values.

var numbers1 = [45, 4, 9, 16, 25];

var numbers2 = numbers1.map(myFunction);

function myFunction(value) {

return value \* 2;

}

90,8,18,32,50

Array.filter()

The filter() method creates a new array with array elements that passes a test.

This example creates a new array from elements with a value larger than 18:

var numbers = [45, 4, 9, 16, 25];

var over18 = numbers.filter(myFunction);

function myFunction(value, index, array) {

return value > 18;

}

Array.reduce()

The reduce() method runs a function on each array element to produce (reduce it to) a single value.

The reduce() method works from left-to-right in the array. See also reduceRight().

var numbers1 = [45, 4, 9, 16, 25];

var sum = numbers1.reduce(myFunction);

function myFunction(total, value, index, array) {

return total + value;

}

Array.every()

The every() method check if all array values pass a test.

This example check if all array values are larger than 18:

var numbers = [45, 20,50, 25];

var allOver18 = numbers.every(myFunction);

document.getElementById("demo").innerHTML = "All over 18 is " + allOver18;

function myFunction(value, index, array) {

return value > 18;

}

Array.some()

The some() method check if some array values pass a test.

var numbers = [45, 4, 9, 16, 25];

var someOver18 = numbers.some(myFunction);

function myFunction(value, index, array) {

return value > 18;

}

var fruits = ["Apple", "Orange", "Apple", "Mango"];

var a = fruits.indexOf("Apple");

Array.find()

The find() method returns the value of the first array element that passes a test function.

This example finds (returns the value of) the first element that is larger than 18:

var numbers = [4, 9, 16, 25, 29];

var first = numbers.find(myFunction);

function myFunction(value, index, array) {

return value > 18;

}

JavaScript Date Object lets us work with dates:

Wed Dec 09 2020 13:18:52 GMT+0530 (India Standard Time)

var d = new Date();

console.log(d);

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

console.log(Math.PI);

console.log(Math.pow(2, 3));

console.log(Math.sqrt(64));

Math.abs(-4.7); // returns 4.7

Math.ceil(4.4); // returns 5

Math.floor(4.7); // returns 4

Math.min(0, 150, 30, 20, -8, -200); // returns -200

Math.max(0, 150, 30, 20, -8, -200); // returns 150

console.log(Math.random()); // returns a random number

console.log(Boolean(10 > 9));

Operator Description Example

== equal to if (day == "Monday")

> greater than if (salary > 9000)

< less than if (age < 18)

Everything Without a "Value" is False

Note the difference between (x==y) and (x===y).

Comparing two JavaScript objects will always return false.

Comparison Operators

Comparison operators are used in logical statements to determine equality or difference between variables or values.

Given that x = 5, the table below explains the comparison operators:

&& and (x < 10 && y > 1) is true

|| or (x == 5 || y == 5) is false

! not !(x == y) is true

Logical Operators

Logical operators are used to determine the logic between variables or values.

Given that x = 6 and y = 3, the table below explains the logical operators:

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

Example

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

JavaScript if else and else if

Conditional statements are used to perform different actions based on different conditions.

Conditional Statements

Very often when you write code, you want to perform different actions for different decisions.

You can use conditional statements in your code to do this

Use if to specify a block of code to be executed, if a specified condition is true

Use else to specify a block of code to be executed, if the same condition is false

Use else if to specify a new condition to test, if the first condition is false

Use switch to specify many alternative blocks of code to be executed

if (condition1) {

// block of code to be executed if condition1 is true

} else if (condition2) {

// block of code to be executed if the condition1 is false and condition2 is true

} else {

// block of code to be executed if the condition1 is false and condition2 is false

}

The switch statement is used to perform different actions based on different conditions.

switch(expression) {

case x:

// code block

break;

case y:

// code block

break;

default:

// code block

}

prompt("Please enter your name", "Harry Potter");

var b = new Date().getDay();

console.log(b);

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";

}

The break Keyword

When JavaScript reaches a break keyword, it breaks out of the switch block.

This will stop the execution of inside the block.

The default Keyword

The default keyword specifies the code to run if there is no case match:

Common Code Blocks

Sometimes you will want different switch cases to use the same code.

In this example case 4 and 5 share the same code block, and 0 and 6 share another code block:

switch (new Date().getDay()) {

case 4:

case 5:

text = "Soon it is Weekend";

break;

case 0:

case 6:

text = "It is Weekend";

break;

default:

text = "Looking forward to the Weekend";

}

Strict Comparison

Switch cases use strict comparison (===).

The values must be of the same type to match.

A strict comparison can only be true if the operands are of the same type

JavaScript For Loop

JavaScript Loops

Loops are handy, if you want to run the same code over and over again, each time with a different value.

var i;

for (i = 0; i < 5; i++) {

console.log(i);

}

var ar = [1, 2, 3, 4, 5];

var i, sum = 0;

for (i = 0; i < ar.length; i++) {

sum += ar[i];

}

console.log(sum);

<script>

var txt = "";

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

var x;

for (x in person) {

txt += person[x] + " ";

}

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

</script>

The While Loop

The while loop loops through a block of code as long as a specified condition is true.

var i=0, sum = 0;

while(i<10)

{

sum+=i;

i++;

}

The Do/While Loop

The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

do {

// code block to be executed

}

while (condition);

The break statement "jumps out" of a loop.

The continue statement "jumps over" one iteration in the loop.

JavaScript Data Types

JavaScript Data Types

In JavaScript there are 5 different data types that can contain values:

string

number

boolean

object

function

There are 6 types of objects:

Object

Date

Array

String

Number

Boolean

And 2 data types that cannot contain values:

null

undefined

JavaScript Type Conversion

var sum = String(23);

console.log(typeof (sum));

String(false) // returns "false"

String(true) // returns "true"

JavaScript Bitwise Operators

Operator Name Description

& AND Sets each bit to 1 if both bits are 1

| OR Sets each bit to 1 if one of two bits is 1

^ XOR Sets each bit to 1 if only one of two bits is 1

~ NOT Inverts all the bits

<< Zero fill left shift Shifts left by pushing zeros in from the right and let the leftmost bits fall off

>> Signed right shift Shifts right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

var x = 5 & 1;

What Is a Regular Expression?

A regular expression is a sequence of characters that forms a search pattern.

When you search for data in a text, you can use this search pattern to describe what you are searching for.

A regular expression can be a single character, or a more complicated pattern.

In JavaScript, regular expressions are often used with the two string methods: search() and replace().

The search() method uses an expression to search for a match, and returns the position of the match.

The replace() method returns a modified string where the pattern is replaced.

JavaScript Errors

The try statement lets you test a block of code for errors.

The catch statement lets you handle the error.

The throw statement lets you create custom errors.

The finally statement lets you execute code, after try and catch, regardless of the result.

try {

adddlert("Welcome guest!");

}

catch(err) {

document.getElementById("demo").innerHTML = err.message;

}

The throw statement allows you to create a custom error.

function myFunction() {

var message, x;

message = document.getElementById("p01");

message.innerHTML = "";

x = document.getElementById("demo").value;

try {

if(x == "") throw "empty";

if(isNaN(x)) throw "not a number";

x = Number(x);

if(x < 5) throw "too low";

if(x > 10) throw "too high";

}

catch(err) {

message.innerHTML = "Input is " + err;

}

}

Scope determines the accessibility (visibility) of variables.

In JavaScript there are two types of scope:

Local scope

Global scope

JavaScript has function scope: Each function creates a new scope.

Scope determines the accessibility (visibility) of these variables.

Variables defined inside a function are not accessible (visible) from outside the function.

Variables declared within a JavaScript function, become LOCAL to the function.

Local variables have Function scope: They can only be accessed from within the function.

A variable declared outside a function, becomes GLOBAL.

A global variable has global scope: All scripts and functions on a web page can access it.

The lifetime of a JavaScript variable starts when it is declared.

Local variables are deleted when the function is completed.

In a web browser, global variables are deleted when you close the browser window

Hoisting is JavaScript's default behavior of moving declarations to the top.

JavaScript Declarations are Hoisted

In JavaScript, a variable can be declared after it has been used.

In other words; a variable can be used before it has been declared.

x = 5; // Assign 5 to x

elem = document.getElementById("demo"); // Find an element

elem.innerHTML = x; // Display x in the element

var x; // Declare x

The JavaScript this Keyword

What is this?

The JavaScript this keyword refers to the object it belongs to.

<script>

// Create an object:

var person = {

firstName: "John",

lastName : "Doe",

id : 5566,

fullName : function() {

return this.firstName + " " + this.lastName;

}

};

// Display data from the object:

document.getElementById("demo").innerHTML = person.fullName();

</script>

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

<script>

var x = this;

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

</script>

In this example, this refers to the window Object:

[object Window]

ES2015 introduced two important new JavaScript keywords: let and const.

These two keywords provide Block Scope variables (and constants) in JavaScript.

Global Scope

Variables declared Globally (outside any function) have Global Scope.

Local variables can only be accessed from inside the function where they are declared.

Variables declared with the var keyword cannot have Block Scope.

Variables declared inside a block {} can be accessed from outside the block.

Variables declared with the let keyword can have Block Scope.

Variables declared inside a block {} cannot be accessed from outside the block:

{

let x = 2;

}

// x can NOT be used here

var carName = "Volvo";

// code here can use window.carName

let carName = "Volvo";

// code here cannot use window.carName

Redeclaring

Redeclaring a JavaScript variable with var is allowed anywhere in a program:

var x = 2;

// Now x is 2

var x = 3;

// Now x is 3

Redeclaring a var variable with let, in the same scope, or in the same block, is not allowed:

var x = 2; // Allowed

let x = 3; // Not allowed

{

var x = 4; // Allowed

let x = 5 // Not allowed

}

Variables defined with const behave like let variables, except they cannot be reassigned:

Declaring a variable with const is similar to let when it comes to Block Scope.

The x declared in the block, in this example, is not the same as the x declared outside the block:

The keyword const is a little misleading.

It does NOT define a constant value. It defines a constant reference to a value.

Because of this, we cannot change constant primitive values, but we can change the properties of constant objects.

Correct

JavaScript const variables must be assigned a value when they are declared:

const PI = 3.14159265359;

JavaScript Class Syntax

class ClassName {

constructor() { ... }

}

class Car {

constructor(name, year) {

this.name = name;

this.year = year;

}

age() {

let date = new Date();

return date.getFullYear() - this.year;

}

}

let myCar = new Car("Ford", 2014);

document.getElementById("demo").innerHTML =

"My car is " + myCar.age() + " years old.";