6th Mar ‘20

<https://www.youtube.com/watch?v=PkZNo7MFNFg> freeCodecamp.org

code editors: sublime, Code visual studio, atom.

Online editors: codepen, scrimba. Codepen.com [okstamps@gmail.com](mailto:okstamps@gmail.com) 1@678582

Js ignore comments. Inline comments: // comment

Multi line comment /\* text \*/

Data types and variables: 7 data types: undefined, string, number, Boolean, null, symbol, object

Symbol is an immutable primitive value that is unique. Object can store key:value pairs

Declare variable is 3 ways: var <varName> = value (global scope). let <varName> = value (local scope)

const <varName> = value (cannot change the value) // var a; every js statement ends in ‘;’ semi colon

console.log allows to see things in the console.

Variable names and function names in Js is **case sensitive**

myVar++ , compound assignment var += can be used. If a number divided by 2 is 0, then the number is even.

Backtick ‘`’ can be used in include single and double quotation marks inside a string.

Concatenate strings with ‘+’. A variable can be considered as a ‘box’.

Strings are immutable.

var myVar

function formSentence(noun, proNoun, adjective, adVerb) {

myVar = "The " + noun + proNoun + adjective + adVerb // variable has function scope

return myVar

}

var text = formSentence("quick", "brown", "fox", "jumped")

console.log(text)

console.log(myVar)

output: "The quickbrownfoxjumped"

"The quickbrownfoxjumped"

--

Array can store any datatype. Arrays are mutable. Array.push()

var myArray = [ ["Jaison", "Jacob", 10], ["Sonia", "Jaison"], ["Esther", "Sarah"]]

myArray.push("Hello", "world")

console.log(myArray)

array.pop() pops the last element. // array.shift() removes the first element from the array.

Array.unshift() adds element to the beginning of the array. Typeof – used to find the type of a variable.

**If a variable is defined without ‘var’ or other specifiers, it becomes ‘global’** automatically.

Variables defined inside a function and parameters has a ‘local’ scope.

‘undefined’ is the default return value of a function.

var myArray = [1,2,3,4,0,5];

console.log(myArray);

myArray.push(6,7);

console.log(myArray);

myArray.pop();

console.log(myArray);

myArray.shift();

console.log(myArray);

var js = JSON.**stringify**(myArray);

console.log(js);

Always use ‘paratheses’ for an ‘if’ condition. true / false is lowercase.

Strict equality check ‘===’ uses no type conversion before the comparison it does with like ‘==’

function checkVal(a) {

if (a === 10) {

return 'a == 10'

} else if (a > 10) {

return 'a > 10'

} else

return 'a < 10'

}

console.log(checkVal(10))

function switchCase(val) {

var answer = " ";

switch (val) {

case "a":

answer = "Apple";

break;

case "b":

answer = "Banana";

break;

case "c":

answer = "Carrot";

break;

case "d":

anwer = "Dango";

break;

default:

answer = "Happy"

break;

}

return answer;

}

var res = switchCase("a");

console.log(res)

objects use properties to access data.

Properties can be accessed using object.propertyname or object[property name]

Delete object.property can be used to delete the property.

randomRagne(), parseInt(string, base). Ternary: <condition> ? ‘return if true’ : ‘return if false’

function checkSign(num) {

return num > 0 ? "positive" : num < 0 ? "negative" : "zero"; // see the usage like 'else if'

}

console.log(checkSign(0));

‘let’ will not allow a variable to be declared twice in the same scope. let maps only to its ‘scope’

“use strict” at the top of the programs to catch coding errors.

Can mutate an array declared with ‘const’

Object.freeze(objectconstVARIABLE) immutates the variable.

// anonymous function

Const var just = function () {

return new Date();

};

// another form

var just = () {

return new Date();

};

// another form

var just = () => new Date();

console.log(just);

--

Rest operator is 3 dots’ …’

It has curly-bracket syntax, dynamic typing, prototype-based object-orientation, and first-class functions. In this path you will learn the basics of JavaScript as well as more advanced topics such as promises, asynchronous programming, proxies and reflection.

17th March ‘20

Modern JavasScript from the beinning. [**Brad Traversy**](https://www.udemy.com/user/brad-traversy/) (Udemy)

**JQuery** is a JavaScript library, a framework that helps you use JavaScript to simplify common web tasks. **Ajax** is a technique using JavaScript to construct an XMLHttpRequest.

Node.js is an open-source, cross-platform, JavaScript runtime environment that executes JavaScript code outside of a browser.

Ajax is a set of web development techniques using many web technologies on the client side to create asynchronous web applications. With Ajax, web applications can send and retrieve data from a server asynchronously without interfering with the display and behavior of the existing page

Es2015 // es5 prototypes es6 classes, (react, angular are JS web frameworks) // arrow functions // Materialize CSS, Skelton CSS, Webpack & Babel to compile code to older es5 js.

Visual studio code – live server – local host port#5500 //live server // bracket pair colorizer // Atom live server

127.0.0.1 is loopback address

document.querySelector("h1").style.color = "red" // use f12 to access chrome browser console.

console.table({a:1,b:2,c:3})

console.error(“This is an error”) // console.clear(); // console.warn(“This is a warning”) // console.time(); console.timeEnd();

variable definitions using var, (let, const introduced in es6 2015)

variables defined with var, let can be reassigned

variable names can include only letters, numbers, \_, $. Cannot start with a number

private variables in js uses \_variablename notation. Multi name variables use camelCase notation

php\_case, PascalCase.

Let is identical to var in ‘global scope’, but has advantage when it comes to ‘block scope’

Select text, ‘ctrl+//’ to comment out the text

Cannot reassign variables defined with ‘const’ modifier. Const variables has to be initialized upon declaration.

Data inside a ‘const object’ can be reassigned. But cannot redeclare the object itself.

Const data ={name:”Jaison”, age:44} // data.name=”Jacob” is valid

Array.push(value) – to add an element to an array.

Data types: Primitive and reference types.

primitive– stored where the variable directly accesses – on the stack (string, number, Boolean, null, undefined, symbos(es6)

Reference – objects that are stored on the heap (dynamically allocated memory) – a pointer to a location in memory (arrays, object literals, functions, dates, anything else)

Object wrappers for strings and numbers.

JS is a dynamically typed language – means datatypes are associated with the values, not with the variables.

Java, C# are statically typed – means datatype is associated with the variable, not with the value

TypeScript, Flow are superset of JS, which allows additional features such as static typing.

Console.log(typeof variable) // Boolean ‘true / false’ are lower case. // typeof null is an object, which is a bug. // const sym = Symbol() // ‘length’ is a property of string datatype

Programmer doesType conversion and JS does type coercion. String(value), object.toString()

Object.toFixed() – works only on numbers; can give decimals. parseInt(‘100’), parseFloat(‘100.30’)

The null value is zero // JavaScript engine.

String.concat(,) string.toUpperCase(); string.toLowerCase(); indexOf(<char of a string>); lastIndexOf(char); charAt(2); string.substring(0,4); slice(0,4); str.split(<char>); str.replace(arg1, arg2);

Str.includes(‘string’) – returns true/false

Slice is mostly used with arrays to pull things out, also can be used with strings.

Document.body.innerHTML

Template strings(es6): uses backtick and ${} notation.

Constructors use ‘new’ always.

Array.push(val) – adds val to the end. Array.unshift(val) – adds val to the front.

Array.pop() – removes from end. Array.shift() – removes from front. Array.splice(fromindex, toindex) – removes elements // Array.reverse() // val = Array.concat(array)

Array sorting:

var a= [10,20,30,15,25,35,50,3]

//sorting takes a function

**val = a.sort(function(x,y){return x - y})**

console.log(val)

output: [ 3, 10, 15, 20, 25, 30, 35, 50 ]

-------------

Array.find(val)

var a= [10,20,30,15,25,35,50,3]

function over50(num){

return num >28

}

**x = a.find(over50)**

console.log(x)

output:30

To reference a property inside an object, use ‘this’. In JS, dictionaries are called ‘object literals’

Month January starts as 0 for year in Date object. So December will be 11.

Today.getMonth(), today.getFullYear(), today.getHours(), today.getMilliseconds(), today.getTime() – returns seconds past 1st jan 1970.

Date.setMonth(val), date.setDate(val)

To check the value and type in a comparison, use triple equals. ‘===’.

If else if else // logical ‘and’ is represented as && and ‘or’ as ||.

Ternary is - <condition>? True : False // 10 > 5? ’10 is greater’: ’10 is lesser’

Curly braces are optional in if statement.

Switch is a different way of using ‘if’.

**Functions** are block of code, defined and called at a later time.

Function decorators, expressions, IIFE’s – immediately invokable function expressions, property methods.

Functions defined without names are called anonymous functions.

When a function is assigned to a variable, its called function expression. const exp = function(){ return 2 \* 2}; // hoisting, closures

IIFE’s (function(){ console.log(“Hello”);})() //

(function(firstName){

console.log("Hello " + firstName);

})("Jaisons")

Module pattern based on IIFE’s

When a function is put inside an object, its called a method.

const todo = { add : function(){

console.log("Add..");

},

edit : function(num){

return num;

}

}

// adding another property to the object literal

todo.delete = function(){

console.log("delete...")

}

todo.add()

let val = todo.edit(22)

console.log(val)

todo.delete()

General popular loop types: for-loop, while-loop, do-while-loop

‘continue’ means, keep going with the loop, next iteration.

--

//array.forEach(arrayobject, index, array)

const cars = ["maruthi","suzuki","honda","ford"]

cars.forEach(function(car){

console.log(car);

})

Output: maruthi, Suzuki, honda, ford

--

//MAP works with object literal arrays

const users = [{id:01, name:"Jaison"},{id:02, name:"Jacob"},{id:03, name:"Esther"}]

let output = users.map(function(user){

return user.id;

})

console.log(output)

output: Array(3) [ 1, 2, 3 ]

const users = {id:01, name:"Jaison",lname:"Jacob"}

for(let a in users){

console.log(`${a} : ${users[a]}`);

}

Output: id : 1

name : Jaison

lname : Jacob

In client-side JS, the browser window is the invironment. For Node.js, the computer is the environment. JSEngine is ‘VA’

Window -> local storage holds the api’s and the window object.

Alert(msg); x = prompt(); confirm(“Are you sure?”);

Window.location // window.history.go(-2) // window.history.length // window.navigator;

**Scope: global, function, block**

‘var’ creates confusion when it is declared n global and in block scope; block overrides global.

Let and const have block-level scope and var has function-level scope.

Browser creates the tree/node. JS can be used to read, write, manipulate the objects. Elements have attribute and text.

Document.all; // document.all[0]; // Array.from(object) // attr.getAttribute()

Jquery is not suggested for DOM manipulation // document.getElementById()

Document.querySelector(<>) – any css selector can be given

DOMSelectors for multiple elements: getElementsByClassName()

getElemensByTagName() // document.querySelectorAll(‘li:nth-child(odd))

document.createElement(<>);

document.querySelector(<>).addEventListener(‘click’,function);

---

const txtNode = document.createElement('h1');

txtNode.id = 'task-title';

txtNode.appendChild(document.createTextNode("Hello world"));

console.log(txtNode);

--

Mouse events: click, dblclick, mousedown, mouseup, mouseenter, mouseleave, mouseover, mousemove, mouseout.

Btn.addEventListener(‘mouseenter’, <function>)

Keyboard events: submit, keydown, keyup, keypress, keyrelease, focus, blur, cut, paste

Input, change

**Event bubbling, event delegation**

Bubbling goes up from lower elements to higher. Delegation goes down from Upper element to lower.

Local and session storage, cookies. Json.stringify() – to convert array into string. Json.parse() – to pull it out.

LocalStorage.setItem(‘name’:’John’); // sessionStorage.setItem(‘name’:’Beth);

OOP in JS

Es5

//constructor function

function Person(name,dob){

this.name = name;

this.birthDate = new Date(dob);

//method defenition

this.calculateAge = function(){

const diff = Date.now() - this.birthDate.getTime();

const ageDate = new Date(diff);

return Math.abs(ageDate.getUTCFullYear() - 1970)

}

}

emp1 = new Person("John", "10/02/1975");

name = emp1.name;

dob = emp1.birthDate;

age = emp1.calculateAge();

console.log(name, dob, age)

output: John

Date Thu Oct 02 1975 00:00:00 GMT+0530 (India Standard Time)

44

const func = new Function('x','y', 'return x + y');

console.log(func(1,2));

output: 3

--

const ob = {name:"john"};

//note the Object type

const obj = new Object({name : "jacob"});

console.log(ob)

console.log(obj)

output: Object { name: "john" } // Object { name: "jacob" }

--

//regular expression

const rgexp = /\w+/

const reg1 = new RegExp('\\w+')

console.log(rgexp)

console.log(reg1)

output: /\w+/ // /\w+/

function Person(name,age){

this.name = name;

this.age = age;

}

//prototype - adding a property to the object

Person.prototype.lastName = function(lName){

return `${lName}`;

}

p1 = new Person("Emil",8)

p2 = p1.lastName("Jaison");

console.log(p2);

-----

Inheritance: use previous function.call(parameters) to invoke the constructor of base function.

Customer.prototype = Object.create(Person.prototype) –to access the prototype methods defined with Person.

Customer.prototype.constructor = Customer – makes customer prototype return Custoemer()

**ES6:**

class Person{

//see the constuctor method

constructor(fName,lName){

this.firsstName = fName;

this.lastName = lName;

}

}

p = new Person("Jaison", "Jacob")

console.log(p.firsstName)

class Person{

//see the constuctor method

constructor(fName,lName){

this.firsstName = fName;

this.lastName = lName;

}

//methods dont use any qualifiers

greets(){

console.log(`Hi ${this.firsstName}`);

}

static add(x, y){

return x + y

}

}

p = new Person("Jaison", "Jacob")

console.log(p.firsstName)

p.greets();

When a method doesn’t have to refer anything of a class, it can be defined as ‘static’.

class Person{

//see the constuctor method

constructor(fName,lName){

this.firsstName = fName;

this.lastName = lName;

}

//methods dont use any qualifiers

greets(){

console.log(`Hi ${this.firsstName}`);

return null;

}

static add(x, y){

return x + y

}

}

// extends

class Customer extends Person{

constructor(fname, lname, age, membership){

// super

super(fname, lname);

this.age = age;

this.membership = membership;

}

}

p = new Person("Jaison", "Jacob");

console.log(p);

c = new Customer("Esther", "Sarah", 11, "Yes");

console.log(c);

console.log(c.greets());

Synchronous code is blocking code – loadposts example.

Asychronous programs are faster.

There are a few ways to work with async code: callback, async/await, promises.

Asynchronous JavaScript and XML (AJAX).

XMLHttpRequest(XHR) – core technology in AJAX.

Other libs and methods to make HTTP requests: Fetch API’s, Axios, Superagent, jQuery, NodeHTTP

Fetch API is part of Vanilla JS. XHTTpRequest.open, onload, onprogress, onerror, send, readyState

In JSON, the keys and values has to be enclosed in double quotes(only the value: string).

<https://jsonlint.com/> is a JSON validator.

e.preventDefault. e is event object, given as function argument.

API is a contract provided by one piece of software to another. Structured request and response.

All API’s have their own rules and structure. API is a messenger and REST uses HTTP request to format that message.

**Callback** is a function that is passed in as a parameter to another function and that is run inside the function body. Synchronous and asynchronous callbacks.

const posts = [ {title:"post one", body:"post 1"}, {title:"post two", body:"post 2"}];

function createPost(post){

setTimeout(function(){

posts.push(post),2000

});

}

function getPosts(){

setTimeout(function(){

let output = " ";

posts.forEach(function(post){

output += `<li>${post.title}, ${post.body}</li>`;

}),1000

document.body.innerHTML = output;

})

}

createPost({title:"post 3", body:"This is post 3"});

getPosts();

//callback

function createPost(post, callback) {

setTimeout(function(){

posts.push(post);

//callback 'getPosts()

callback();

},2000);

}

function getPosts(){

setTimeout(function(){

let output = " ";

posts.forEach(function(post){

output += `<li>${post.title}, ${post.body}</li>`;

}),1000

document.body.innerHTML = output;

})

}

createPost({title:"post 3", body:"This is post 3"}, getPosts);

to test API, <https://jsonplaceholder.typicode.com/>

the ‘prototype’ of es5 actually runs under the es6 classes

**Promises** are a es6 standard and are an alternative to asynchronous callbacks.

Return new Promise(function(resolve,reject){

Statements;

If(!error)

Resolve();

Else

Reject(“something went wrong”);

}

<function>().then(<function>);

.catch(function(err){

Cosole.log(err)}

**Fetch** is the new standard to make http request. Fetch API returns a promise.

fetch('https://devcamper.io/api/v1/bootcamps/34343')

.then(res => res.json())

.then(res => {

if (!res.ok) {

throw new Error(res.error);

}

return res;

})

.catch(err => console.log(err));

function handleErrors(res) {

if (!res.ok) throw new Error(res.error);

return res;

}

fetch('https://devcamper.io/api/v1/bootcamps/34343')

.then(res => res.json())

.then(handleErrors)

.then(res => console.log(res.data))

.catch(err => console.log(err));

**Arrow functions es6:** Arrow function uses a lexical ‘this’

// one line function does not need any curly braces

const arrowFunc = () => console.log("Hello World!")

arrowFunc();

91881 30497 // ktym

--

// one line function that returns

const arrowFunc = () => "Hello"

a = arrowFunc();

console.log(a)

-------

// function that returns an object

const arrowFunc = () => ({name:"Jaison"})

a = arrowFunc();

console.log(a)

// function that takes an argument

const arrowFunc = (name) => `name is ${name}`

a = arrowFunc("Jaison");

console.log(a)

// Single parameter does not need paranthesis

const arrowFunc = name => `Name is ${name}`

a = arrowFunc("Jaison");

console.log(a)

// built-in function that takes a paameter and returns a value. its not creating a new function

const users = ["Nathan", "Rodgrigus", "Susan"]

const arrowFunc = users.map(name => name.length);

console.log(arrowFunc)

**Fetch library:**

<!-- http library home page – index.htm>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Easty Html</title>

</head>

<body>

<h1> Easy Html </h1>

<script src="easyhttp2.js"></script>

<script src="app.js"></script>

</body>

</html>

Fetch.js

//library using 'fetch'

class EasyHttp{

get(url){

fetch(url)

.then(res => res.json())

.then(data => console.log(data))

.then(err => console.log(err))

}

}

Appl.js

//running app

const http = new EasyHttp;

http.get("https://jsonplaceholder.typicode.com/users");

//library using 'fetch and promise'

class EasyHttp{

get(url){

return new Promise((resolve, reject) =>{

fetch(url)

.then(res => res.json())

.then(data => resolve(data))

.catch(err => reject(err));

})

}

}

//running app - using promise

const http = new EasyHttp;

http.get("https://jsonplaceholder.typicode.com/users")

.then(data => console.log(data))

.then(err => console.log(err));

/\*\*

\* EasyHTTP Library

\* Library for making HTTP requests

\*

\* @version 2.0.0

\* @author Brad Traversy

\* @license MIT

\*

\*\*/

class EasyHTTP {

// Make an HTTP GET Request

get(url) {

return new Promise((resolve, reject) => {

fetch(url)

.then(res => res.json())

.then(data => resolve(data))

.catch(err => reject(err));

});

}

// Make an HTTP POST Request

post(url, data) {

return new Promise((resolve, reject) => {

fetch(url, {

method: 'POST',

headers: {

'Content-type': 'application/json'

},

body: JSON.stringify(data)

})

.then(res => res.json())

.then(data => resolve(data))

.catch(err => reject(err));

});

}

// Make an HTTP PUT Request

put(url, data) {

return new Promise((resolve, reject) => {

fetch(url, {

method: 'PUT',

headers: {

'Content-type': 'application/json'

},

body: JSON.stringify(data)

})

.then(res => res.json())

.then(data => resolve(data))

.catch(err => reject(err));

});

}

// Make an HTTP DELETE Request

delete(url) {

return new Promise((resolve, reject) => {

fetch(url, {

method: 'DELETE',

headers: {

'Content-type': 'application/json'

}

})

.then(res => res.json())

.then(() => resolve('Resource Deleted...'))

.catch(err => reject(err));

});

}

}

const http = new EasyHTTP;

// Get Users

// http.get('https://jsonplaceholder.typicode.com/users')

// .then(data => console.log(data))

// .catch(err => console.log(err));

// User Data

const data = {

name: 'John Doe',

username: 'johndoe',

email: 'jdoe@gmail.com'

}

// Create User

// http.post('https://jsonplaceholder.typicode.com/users', data)

// .then(data => console.log(data))

// .catch(err => console.log(err));

// Update Post

// http.put('https://jsonplaceholder.typicode.com/users/2', data)

// .then(data => console.log(data))

// .catch(err => console.log(err));

// Delete User

http.delete('https://jsonplaceholder.typicode.com/users/2')

.then(data => console.log(data))

.catch(err => console.log(err));

**Async / await:**

Async will return a promise

// async

async function show(){

return "hello friends";

}

show()

.then(res => console.log(res));

// async / Promise await - awaits 3 seconds and prints

async function show(){

const pro = new Promise((resolve, reject) =>{

setTimeout(() => resolve("Hello Ron"),3000);

});

const res = await pro; // wait until promise is resolved

return res;

}

show()

.then(res => console.log(res));

// async / await - throws error

async function show(){

const pro = new Promise((resolve, reject) =>{

setTimeout(() => resolve("Hello Ron"),3000);

});

const err = true;

if (!err){

const res = await pro;

return res;

} else {

await Promise.reject(new Error("Something went wrong!"));

}

}

show()

.then(res => console.log(res))

.catch(err => console.log(err));

// async / await fetch

async function fetchData(){

// await response of the fetch call

const resp = await fetch("https://jsonplaceholder.typicode.com/users");

// only proceed once its resolved

const data = await resp.json();

// only proceed once sencod promise is resolved

return data;

}

fetchData().then(p => console.log(p));

try {code} catch(e){console.log(e.message, e.name, e instanceof RefrenceError/TypeError} finally{} is an elegant way of handling errors.

Throw new SyntaxError(“Incorrect syntax used”);

**Regular Exp:**

Const re = /hello/I // i = case insensitive

Re.source, re.exec(“hello world”) // re.test(“Hello world”) // str.match(re) // str.search(re) // str.replace(re,”txt”)

Patterns, metacharacters, symbols

^$.\*? – metacharacters // character sets are put inside brackets [A-Za-z0-9]. //

Quantifiers are given inside braces {2} // parenthesis () for grouping.

Shorthand character classes. /\w/ word character /\W/ non-wordcharacter

/\d/ match any digit /\D/ match any non-digit // /\s/ whitespace character, /\S/ match non white space character // /\b/ word boundary.

Assertions: x(?=y) match only if ‘x’ is followed by ‘y’. // x(?!y) match only if ‘x’ is NOT followed by ‘y’.

**Iterators and generators:** both used to iterate through something. Iterators are like advance loops that can be paused and generators are like functions that can be paused and can return multiple values.

To make js understand that the function is a generator, put an asterik \* close to the function\*.

//generator

function\* makeId(){

let id = 1;

while(true){

yield id++;

}

}

const a = makeId();

// console.log(a.next().value);

// console.log(a.next().value);

// console.log(a.next().value);

// console.log(a.next().value);

for(let i = 0; i<10; i++){

console.log(a.next().value)

}

**Symbols:** very unique, serves as a object property identifier. Symbols are not enumerable in for .. loops.

//symbols

const KEY1 = Symbol();

const KEY2 = Symbol("sym2");

const obj = {one:"One", two:"Two"};

obj[KEY1] = "Key1";

obj[KEY2] = "Key2";

obj.key3 = "Key3v";

obj.key4 = "Key4v";

for(let i in obj){

console.log(`${i}, ${obj[i]}`);

}

Output: one, One

two, Two

key3, Key3v

key4, Key4v

Symbols are ignored by JSON.stringify().

**De-structuring:**

//Destructuring assignment

let a, b;

[a,b] = [100,200];

[a,b,...rest] = [110,120,130,140,150];

({a,b} = {a:10,b:20,c:30,d:40,e:50});

//spread operator

({ a, b, ...rest } = { a: 10, b: 20, c: 30, d: 40, e: 50 });

//Array destructuring

const people = ['mike','john','susan'];

const [person1,person2,person3] = people;

function get(){

return ["first","second","third"];

}

let one,two,three;

[one,two,three] = get();

//Object destructuring

const person = {name:"Jaiosn", age:32, city:"Bengaluru", sayHello: function(){

console.log("Hello");

}

}

const {name,age,city,sayHello} = person;

sayHello();

--

Es6 is EcmaScript6

**Maps:** are key value pairs. Any type can be used as a key or value.

//Maps

const map1 = new Map();

//set keys

const key1 = "This is key1", key2 = {}, key3 = function () {};

//set map values by key

map1.set(key1, "value of key1");

map1.set(key2, "value of key2");

map1.set(key3, "value of key3");

//get values by key

console.log(map1.get(key1), map1.size);

//creating an array out of a map

const keyvalArray = Array.from(map1);

console.log(keyvalArray);

**sets:** are unique values of any type. Primitive or reference.

//set

const mySet = new Set();

mySet.add("one");

mySet.add(1);

mySet.add({one:"one"});

mySet.add([1,'one',{one:"two"}]);

const yourSet = new Set(["one",2,"Hello"]);

console.log(mySet, mySet.size, mySet.has(1));

yourSet.delete("Hello");

for(let item of mySet.entries()){

console.log(item);

}

for (let key of mySet.keys()) {

console.log(key);

}

**A pattern:** is a re-usable solution that can be applied to occurring problems in software design. Or programming templates.

Module pattern, Module and revealing pattern, Singleton pattern, Factory pattern, Observer pattern, Mediator pattern, state pattern.

Babel is compiler, webpack is module loader.

Common js module. Npm install

Frontend framework: Angular, React, Vue.js

Server side language: Node.js, Express, Adonisjs, loopback, swagger

JSON Server is a Node Module that you can use to create demo rest json webservice in less than a minute. All you need is a JSON file for sample data.

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

--

ECMA 262 // ECMAScript

<!DOCTYPE html>

<html>

<body>

<h2>What Can JavaScript Do?</h2>

<p>JavaScript can change HTML attribute values.</p>

<p>In this case JavaScript changes the value of the src (source) attribute of an image.</p>

<button onclick="document.getElementById('myImage').src='pic\_bulbon.gif'">Turn on the light</button>

<img id="myImage" src="pic\_bulboff.gif" style="width:100px">

<button onclick="document.getElementById('myImage').src='pic\_bulboff.gif'">Turn off the light</button>

</body>

</html>

Javascript function:

<!DOCTYPE html>

<html>

<body>

<h1>A Web Page</h1>

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

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

<script>

function myFunction() {

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

}

</script>

</body>

</html>

--

You can place an external script reference in <head> or <body> as you like.

The script will behave as if it was located exactly where the <script> tag is located.

<!DOCTYPE html>

<html>

<body>

<h2>External JavaScript</h2>

<p id="demo">A Paragraph.</p>

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

<p>(myFunction is stored in an external file called "myScript.js")</p>

<script src="myScript.js"></script>

</body>

</html>

--

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().

--

JavaScript Programs

A computer program is a list of "instructions" to be "executed" by a computer.

In a programming language, these programming instructions are called statements.

A JavaScript program is a list of programming statements.

JavaScript statements are composed of:

Values, Operators, Expressions, Keywords, and Comments.

Semicolons separate JavaScript statements. JavaScript ignores multiple spaces.

--

function myFunction() {

document.getElementById("demo1").innerHTML = "Hello Dolly!";

document.getElementById("demo2").innerHTML = "How are you?";

}

Keyword Description

break Terminates a switch or a loop

continue Jumps out of a loop and starts at the top

debugger Stops the execution of JavaScript, and calls (if available) the debugging function

do ... while Executes a block of statements, and repeats the block, while a condition is true

for Marks a block of statements to be executed, as long as a condition is true

function Declares a function

if ... else Marks a block of statements to be executed, depending on a condition

return Exits a function

switch Marks a block of statements to be executed, depending on different cases

try ... catch Implements error handling to a block of statements

var Declares a variable

Fixed values are called literals. Variable values are called variables.

All JavaScript identifiers are case sensitive.

JavaScript uses the Unicode character set.

JS datatypes:

var length = 16; // Number

var lastName = "Johnson"; // String

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

typeof "" // Returns "string"

typeof "John" // Returns "string"

typeof "John Doe" // Returns "string"

The typeof operator can return one of these primitive types:

string

number

boolean

undefined

toCelsius refers to the function object, and toCelsius() refers to the function result

<element event='some JavaScript'>

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

There are 4 ways to create a new date object:

new Date()

new Date(year, month, day, hours, minutes, seconds, milliseconds)

new Date(milliseconds)

new Date(date string)

There are generally 3 types of JavaScript date input formats:

Type Example

ISO Date "2015-03-25" (The International Standard)

Short Date "03/25/2015"

Long Date "Mar 25 2015" or "25 Mar 2015"

--

var d = new Date("2015-03-25T12:00:00Z");

Date and time is separated with a capital T.

UTC time is defined with a capital letter Z.

If you want to modify the time relative to UTC, remove the Z and add +HH:MM or -HH:MM instead:

var d = new Date("2015-03-25T12:00:00-06:30");

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.

--

Set Date Methods

Set Date methods are used for setting a part of a date:

Method Description

setDate() Set the day as a number (1-31)

setFullYear() Set the year (optionally month and day)

setHours() Set the hour (0-23)

setMilliseconds() Set the milliseconds (0-999)

setMinutes() Set the minutes (0-59)

setMonth() Set the month (0-11)

setSeconds() Set the seconds (0-59)

setTime() Set the time (milliseconds since January 1, 1970)

--

age = Number(age);

if (isNaN(age)) {

voteable = "Input is not a number";

} else {

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

}

Different Kinds of Loops

JavaScript supports different kinds of loops:

for - loops through a block of code a number of times

for/in - loops through the properties of an object

for/of - loops through the values of an iterable object

while - loops through a block of code while a specified condition is true

do/while - also loops through a block of code while a specified condition is true

--

for (i = 0, len = cars.length, text = ""; i < len; i++) {

text += cars[i] + "<br>";

}

---

var i = 2;

var len = cars.length;

var text = "";

for (; i < len; i++) {

text += cars[i] + "<br>";

}

---

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

var text = "";

var x;

for (x in person) {

text += person[x];

}

--

var cars = ['BMW', 'Volvo', 'Mini'];

var x;

for (x of cars) {

document.write(x + "<br >");

}

--

var txt = 'JavaScript';

var x;

for (x of txt) {

document.write(x + "<br >");

}

The continue statement (with or without a label reference) can only be used to skip one loop iteration.

The break statement, without a label reference, can only be used to jump out of a loop or a switch.

With a label reference, the break statement can be used to jump out of any code block:

--

for (i = 0; i < 10; i++) {

if (i === 3) { break; }

text += "The number is " + i + "<br>";

}

--

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

list: {

text += cars[0] + "<br>";

text += cars[1] + "<br>";

break list;

text += cars[2] + "<br>";

text += cars[3] + "<br>";

}

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

--

The constructor property returns the constructor function for all JavaScript variables.

String(), Number(), toString() conversion functions. Returns NaN

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.

Regular expressions can be used to perform all types of text search and text replace operations.

Syntax

/pattern/modifiers;

Search(), replace()

var str = "Visit W3Schools";

var n = str.search(/w3schools/i);

--

JS 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.

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

<script>

try {

adddlert("Welcome guest!");

}

catch(err) {

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

}

</script>

The throw statement allows you to create a custom error.

If you use throw together with try and catch, you can control program flow and generate custom error messages.

<!DOCTYPE html>

<html>

<body>

<p>Please input a number between 5 and 10:</p>

<input id="demo" type="text">

<button type="button" onclick="myFunction()">Test Input</button>

<p id="p01"></p>

<script>

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;

}

}

</script>

</body>

</html>

--

HTML Validation:

<input id="demo" type="number" min="5" max="10" step="1">

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

try {

Block of code to try

}

catch(err) {

Block of code to handle errors

}

finally {

Block of code to be executed regardless of the try / catch result

}

Automatically Global

If you assign a value to a variable that has not been declared, it will automatically become a GLOBAL variable.

This code example will declare a global variable carName, even if the value is assigned inside a function.

--

"use strict"; Defines that JavaScript code should be executed in "strict mode".

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

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

Variables declared with the let keyword can have Block Scope.

Variables declared with var and let are quite similar when declared inside a function.

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

With Arrow Function:

hello = () => {

return "Hello World!";

}

--

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

--

//Regular Function:

hello = function() {

document.getElementById("demo").innerHTML += this;

}

//The window object calls the function:

window.addEventListener("load", hello);

//A button object calls the function:

document.getElementById("btn").addEventListener("click", hello);

Class:

class Car {

constructor(brand) {

this.carname = brand;

}

present() {

return "I have a " + this.carname;

}

}

mycar = new Car("Ford");

document.getElementById("demo").innerHTML = mycar.present();

--

Static Methods

Static methods are defined on the class itself, and not on the prototype.

That means you cannot call a static method on the object (mycar), but on the class (Car):

Example

Create a static method and call it on the class:

class Car {

constructor(brand) {

this.carname = brand;

}

static hello() {

return "Hello!!";

}

}

mycar = new Car("Ford");

//Call 'hello()' on the class Car:

document.getElementById("demo").innerHTML = Car.hello();

//and NOT on the 'mycar' object:

//document.getElementById("demo").innerHTML = mycar.hello();

//this would raise an error.

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;

}

}

mycar = new Model("Ford", "Mustang");

document.getElementById("demo").innerHTML = mycar.show();

JavaScript Form Validation

HTML form validation can be done by JavaScript.

If a form field (fname) is empty, this function alerts a message, and returns false, to prevent the form from being submitted:

JavaScript Example

function validateForm() {

var x = document.forms["myForm"]["fname"].value;

if (x == "") {

alert("Name must be filled out");

return false;

}

}

The function can be called when the form is submitted:

HTML Form Example

<form name="myForm" action="/action\_page.php" onsubmit="return validateForm()" method="post">

Name: <input type="text" name="fname">

<input type="submit" value="Submit">

</form>

<form action="/action\_page.php" method="post">

<input type="text" name="fname" required>

<input type="submit" value="Submit">

</form>

--

Constraint Validation CSS Pseudo Selectors

Selector Description

:disabled Selects input elements with the "disabled" attribute specified

:invalid Selects input elements with invalid values

:optional Selects input elements with no "required" attribute specified

:required Selects input elements with the "required" attribute specified

:valid Selects input elements with valid values

HTML Constraint Validation

HTML5 introduced a new HTML validation concept called constraint validation.

HTML constraint validation is based on:

Constraint validation HTML Input Attributes

Constraint validation CSS Pseudo Selectors

Constraint validation DOM Properties and Methods

Constraint Validation HTML Input Attributes

Attribute Description

disabled Specifies that the input element should be disabled

max Specifies the maximum value of an input element

min Specifies the minimum value of an input element

pattern Specifies the value pattern of an input element

required Specifies that the input field requires an element

type Specifies the type of an input element

Constraint Validation DOM Methods

Property Description

checkValidity() Returns true if an input element contains valid data.

setCustomValidity() Sets the validationMessage property of an input element.

If an input field contains invalid data, display a message:

The checkValidity() Method

<input id="id1" type="number" min="100" max="300" required>

<button onclick="myFunction()">OK</button>

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

<script>

function myFunction() {

var inpObj = document.getElementById("id1");

if (!inpObj.checkValidity()) {

document.getElementById("demo").innerHTML = inpObj.validationMessage;

}

}

</script>

--

### **The HTML DOM Tree of Objects**



What is the HTML DOM?

The HTML DOM is a standard object model and programming interface for HTML. It defines:

The HTML elements as objects

The properties of all HTML elements

The methods to access all HTML elements

The events for all HTML elements

Finding HTML Elements

Method Description

document.getElementById(id) Find an element by element id

document.getElementsByTagName(name) Find elements by tag name

document.getElementsByClassName(name) Find elements by class name

Adding and Deleting Elements

Method Description

document.createElement(element) Create an HTML element

document.removeChild(element) Remove an HTML element

document.appendChild(element) Add an HTML element

document.replaceChild(new, old) Replace an HTML element

document.write(text) Write into the HTML output stream

Adding Event handlers

document.getElementById(id).onclick = function(){code}

Finding HTML Objects

Property Description DOM

document.anchors Returns all <a> elements that have a name attribute 1

document.applets Returns all <applet> elements (Deprecated in HTML5) 1

document.baseURI Returns the absolute base URI of the document 3

document.body Returns the <body> element 1

document.cookie Returns the document's cookie 1

document.doctype Returns the document's doctype 3

document.documentElement Returns the <html> element 3

document.documentMode Returns the mode used by the browser 3

document.documentURI Returns the URI of the document 3

document.domain Returns the domain name of the document server 1

document.domConfig Obsolete. Returns the DOM configuration 3

document.embeds Returns all <embed> elements 3

document.forms Returns all <form> elements 1

document.head Returns the <head> element 3

document.images Returns all <img> elements 1

document.implementation Returns the DOM implementation 3

document.inputEncoding Returns the document's encoding (character set) 3

document.lastModified Returns the date and time the document was updated 3

document.links Returns all <area> and <a> elements that have a href attribute 1

document.readyState Returns the (loading) status of the document 3

document.referrer Returns the URI of the referrer (the linking document) 1

document.scripts Returns all <script> elements 3

document.strictErrorChecking Returns if error checking is enforced 3

document.title Returns the <title> element 1

document.URL Returns the complete URL of the document 1

Examples of HTML events:

When a user clicks the mouse

When a web page has loaded

When an image has been loaded

When the mouse moves over an element

When an input field is changed

When an HTML form is submitted

When a user strokes a key

--

document.getElementById("myP").addEventListener("click", myFunction, true);

Nodes: parentNode, childNode<html>

<body>

<h1 id="id01">My First Page</h1>

<p id="id02">Hello!</p>

<script>

document.getElementById("id02").innerHTML = document.getElementById("id01").childNodes[0].nodeValue;

</script>

</body>

</html>

Node Type Example

ELEMENT\_NODE 1 <h1 class="heading">W3Schools</h1>

ATTRIBUTE\_NODE 2 class = "heading" (deprecated)

TEXT\_NODE 3 W3Schools

COMMENT\_NODE 8 <!-- This is a comment -->

DOCUMENT\_NODE 9 The HTML document itself (the parent of <html>)

DOCUMENT\_TYPE\_NODE 10 <!Doctype html>

Block level elements in HTML:

<address><article><aside><blockquote><canvas><dd><div><dl><dt><fieldset><figcaption><figure><footer><form><h1>-<h6> <header><hr><li><main><nav><noscript><ol><p><pre><section><table><tfoot><ul><video>

Inline elements:

Inline elements does not start on a new line and takesup only as much memory as it can.

Inline elements in HTML:

<a><abbr><acronym><b><bdo><big><br><button><cite><code><dfn><em><i><img><input><kbd><label><map><object><output><q><samp><script><select><small><span><strong><sub><sup><textarea><time><tt><var>

The <div> Element

The <div> element is often used as a container for other HTML elements.

The <div> element has no required attributes, but style, class and id are common.

When used together with CSS, the <div> element can be used to style blocks of content:

<span> element is often used as a container of some text: style, class, id are common.

The HTML class attribute is used to define equal styles for elements with the same class name.

Difference Between Class and ID

An HTML element can only have one unique id that belongs to that single element, while a class name can be used by multiple elements:

Example

<style>

/\* Style the element with the id "myHeader" \*/

#myHeader {

background-color: lightblue;

color: black;

padding: 40px;

text-align: center;

}

/\* Style all elements with the class name "city" \*/

.city {

background-color: tomato;

color: white;

padding: 10px;

}

</style>

<!-- A unique element -->

<h1 id="myHeader">My Cities</h1>

<!-- Multiple similar elements -->

<h2 class="city">London</h2>

<p>London is the capital of England.</p>

<h2 class="city">Paris</h2>

<p>Paris is the capital of France.</p>

<h2 class="city">Tokyo</h2>

<p>Tokyo is the capital of Japan.</p>

An iframe is used to display a web page inside another. Src=

The HTML <head> Element

The <head> element is a container for metadata (data about data) and is placed between the <html> tag and the <body> tag.

HTML metadata is data about the HTML document. Metadata is not displayed.

Metadata typically define the document title, character set, styles, scripts, and other meta information.

The following tags describe metadata: <title>, <style>, <meta>, <link>, <script>, and <base>.

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<meta name="description" content="Free Web tutorials">

<meta name="keywords" content="HTML,CSS,XML,JavaScript">

<meta name="author" content="John Doe">

<meta http-equiv = "refresh" content ="5">// refresh the document every 5 second

</head>

<body>

<p>All meta information goes before the body.</p>

</body>

</html>

Setting The Viewport

HTML5 introduced a method to let web designers take control over the viewport, through the <meta> tag.

The viewport is the user's visible area of a web page. It varies with the device, and will be smaller on a mobile phone than on a computer screen.

You should include the following <meta> viewport element in all your web pages:

<meta name="viewport" content="width=device-width, initial-scale=1.0">

A <meta> viewport element gives the browser instructions on how to control the page's dimensions and scaling.

Omitting <html>, <head> and <body>?

According to the HTML5 standard; the <html>, the <body>, and the <head> tag can be omitted.

The following code will validate as HTML5:

Example

<!DOCTYPE html>

<title>Page Title</title>

<h1>This is a heading</h1>

<p>This is a paragraph.</p>

--

HTML Layout elements:

<header> - Defines a header for a document or a section

<nav> - Defines a container for navigation links

<section> - Defines a section in a document

<article> - Defines an independent self-contained article

<aside> - Defines content aside from the content (like a sidebar)

<footer> - Defines a footer for a document or a section

<details> - Defines additional details

<summary> - Defines a heading for the <details> element

HTML Layout Techniques

There are five different ways to create multicolumn layouts. Each way has its pros and cons:

HTML tables (not recommended)

CSS float property

CSS flexbox

CSS framework

CSS grid

Responsive web design:

<!DOCTYPE html>

<html>

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<body>

<h2>Show Different Images Depending on Browser Width</h2>

<p>Resize the browser width and the image will change at 600px and 1500px.</p>

<picture>

<source srcset="img\_smallflower.jpg" media="(max-width: 600px)">

<source srcset="img\_flowers.jpg" media="(max-width: 1500px)">

<source srcset="flowers.jpg">

<img src="img\_flowers.jpg" alt="Flowers" style="width:auto;">

</picture>

</body>

</html>

--

HTML Computer Code Elements

Tag Description

<code> Defines programming code

<kbd> Defines keyboard input

<samp> Defines computer output

<var> Defines a variable

<pre> Defines preformatted text

--

A character entity looks like this:

&entity\_name;

OR

&#entity\_number;

To display a less than sign (<) we must write: &lt; or &#60;

--

Some Other Useful HTML Character Entities

Result Description Entity Name Entity Number

non-breaking space &nbsp; &#160;

< less than &lt; &#60;

> greater than &gt; &#62;

& ampersand &amp; &#38;

" double quotation mark &quot; &#34;

' single quotation mark (apostrophe) &apos; &#39;

¢ cent &cent; &#162;

£ pound &pound; &#163;

¥ yen &yen; &#165;

€ euro &euro; &#8364;

© copyright &copy; &#169;

® registered trademark &reg; &#174;

--

HTML Symbol Entities

HTML entities were described in the previous chapter.

Many mathematical, technical, and currency symbols, are not present on a normal keyboard.

To add such symbols to an HTML page, you can use an HTML entity name.

If no entity name exists, you can use an entity number, a decimal, or hexadecimal reference.

Example

<p>I will display &euro;</p>

<p>I will display &#8364;</p>

<p>I will display &#x20AC;</p>

Will display as:

I will display €

I will display €

I will display €

--

URL:

scheme://prefix.domain:port/path/filename

Explanation:

scheme - defines the type of Internet service (most common is http or https)

prefix - defines a domain prefix (default for http is www)

domain - defines the Internet domain name (like w3schools.com)

port - defines the port number at the host (default for http is 80)

path - defines a path at the server (If omitted: the root directory of the site)

filename - defines the name of a document or resource

Common URL Schemes

The table below lists some common schemes:

Scheme Short for Used for

http HyperText Transfer Protocol Common web pages. Not encrypted

https Secure HyperText Transfer Protocol Secure web pages. Encrypted

ftp File Transfer Protocol Downloading or uploading files

file A file on your computer

--

The default character-set in HTML5 is UTF-8.

The HTML <form> element defines a form that is used to collect user input:

The <input> Element

The <input> element is the most important form element.

The <input> element can be displayed in several ways, depending on the type attribute.

Here are some examples:

Type Description

<input type="text"> Defines a one-line text input field

<input type="radio"> Defines a radio button (for selecting one of many choices)

<input type="submit"> Defines a submit button (for submitting the form)

<form>

<input type="radio" name="gender" value="male" checked> Male<br>

<input type="radio" name="gender" value="female"> Female<br>

<input type="radio" name="gender" value="other"> Other

</form>

Here is the list of all <form> attributes:

Attribute Description

accept-charset Specifies the charset used in the submitted form (default: the page charset).

action Specifies an address (url) where to submit the form (default: the submitting page).

autocomplete Specifies if the browser should autocomplete the form (default: on).

enctype Specifies the encoding of the submitted data (default: is url-encoded).

method Specifies the HTTP method used when submitting the form (default: GET).

name Specifies a name used to identify the form (for DOM usage: document.forms.name).

novalidate Specifies that the browser should not validate the form.

target Specifies the target of the address in the action attribute (default: \_self).

<form action="/action\_page.php" method="post">

<fieldset>

<legend>Personal information:</legend>

First name:<br>

<input type="text" name="firstname" value="Mickey"><br>

Last name:<br>

<input type="text" name="lastname" value="Mouse"><br><br>

<input type="submit" value="Submit">

</fieldset>

</form>

--

HTML Form Elements

Tag Description

<form> Defines an HTML form for user input

<input> Defines an input control

<textarea> Defines a multiline input control (text area)

<label> Defines a label for an <input> element

<fieldset> Groups related elements in a form

<legend> Defines a caption for a <fieldset> element

<select> Defines a drop-down list

<optgroup> Defines a group of related options in a drop-down list

<option> Defines an option in a drop-down list

<button> Defines a clickable button

<datalist> Specifies a list of pre-defined options for input controls

<output> Defines the result of a calculation

<form action="/action\_page.php">

<input list="browsers">

<datalist id="browsers">

<option value="Internet Explorer">

<option value="Firefox">

<option value="Chrome">

<option value="Opera">

<option value="Safari">

</datalist>

</form>

--

HTML Input Types

Here are the different input types you can use in HTML:

<input type="button">

<input type="checkbox">

<input type="color">

<input type="date">

<input type="datetime-local">

<input type="email">

<input type="file">

<input type="hidden">

<input type="image">

<input type="month">

<input type="number">

<input type="password">

<input type="radio">

<input type="range">

<input type="reset">

<input type="search">

<input type="submit">

<input type="tel">

<input type="text">

<input type="time">

<input type="url">

<input type="week">

--

<form>

User name:<br>

<input type="text" name="username"><br>

User password:<br>

<input type="password" name="psw">

</form>

--

<form>

<input type="radio" name="gender" value="male" checked> Male<br>

<input type="radio" name="gender" value="female"> Female<br>

<input type="radio" name="gender" value="other"> Other

</form>

--

HTML5 Input Types

HTML5 added several new input types:

color

date

datetime-local

email

month

number

range

search

tel

time

url

week

<!DOCTYPE html>

<html>

<body>

<h2>Color Picker</h2>

<p>The <strong>input type="color"</strong> is used for input fields that should contain a color.</p>

<p>Depending on browser support:<br>A color picker can pop-up when you enter the input field.</p>

<form action="/action\_page.php">

Select your favorite color:

<input type="color" name="favcolor" value="#ff0000">

<input type="submit">

</form>

<p><b>Note:</b> type="color" is not supported in Internet Explorer 11 and earlier versions or Safari 9.1 and earlier versions.</p>

</body>

</html>

--

<form action="/action\_page.php">

Birthday:

<input type="date" name="bday">

<input type="submit">

</form>

--

Input Restrictions

Here is a list of some common input restrictions:

Attribute Description

checked Specifies that an input field should be pre-selected when the page loads (for type="checkbox" or type="radio")

disabled Specifies that an input field should be disabled

max Specifies the maximum value for an input field

maxlength Specifies the maximum number of character for an input field

min Specifies the minimum value for an input field

pattern Specifies a regular expression to check the input value against

readonly Specifies that an input field is read only (cannot be changed)

required Specifies that an input field is required (must be filled out)

size Specifies the width (in characters) of an input field

step Specifies the legal number intervals for an input field

value Specifies the default value for an input field

--

HTML5 Attributes

HTML5 added the following attributes for <input>:

autocomplete

autofocus

form

formaction

formenctype

formmethod

formnovalidate

formtarget

height and width

list

min and max

multiple

pattern (regexp)

placeholder

required

step

and the following attributes for <form>:

autocomplete

novalidate

--

The novalidate Attribute

The novalidate attribute is a <form> attribute.

When present, novalidate specifies that the form data should not be validated when submitted.

The autofocus Attribute

The autofocus attribute specifies that the input field should automatically get focus when the page loads.

New HTML5 Elements

The most interesting new HTML5 elements are:

New semantic elements like <header>, <footer>, <article>, and <section>.

New attributes of form elements like number, date, time, calendar, and range.

New graphic elements: <svg> and <canvas>.

New multimedia elements: <audio>

New Semantic/Structural Elements

HTML5 offers new elements for better document structure:

Tag Description

<article> Defines an article in a document

<aside> Defines content aside from the page content

<bdi> Isolates a part of text that might be formatted in a different direction from other text outside it

<details> Defines additional details that the user can view or hide

<dialog> Defines a dialog box or window

<figcaption> Defines a caption for a <figure> element

<figure> Defines self-contained content

<footer> Defines a footer for a document or section

<header> Defines a header for a document or section

<main> Defines the main content of a document

<mark> Defines marked/highlighted text

<meter> Defines a scalar measurement within a known range (a gauge)

<nav> Defines navigation links

<progress> Represents the progress of a task

<rp> Defines what to show in browsers that do not support ruby annotations

<rt> Defines an explanation/pronunciation of characters (for East Asian typography)

<ruby> Defines a ruby annotation (for East Asian typography)

<section> Defines a section in a document

<summary> Defines a visible heading for a <details> element

<time> Defines a date/time

<wbr> Defines a possible line-break and <video>.

New Form Elements

Tag Description

<datalist> Specifies a list of pre-defined options for input controls

<output> Defines the result of a calculation

--

New Form Elements

Tag Description

<datalist> Specifies a list of pre-defined options for input controls

<output> Defines the result of a calculation

--

New Input Types

New Input Types New Input Attributes

color

date

datetime

datetime-local

email

month

number

range

search

tel

time

url

week

autocomplete

autofocus

form

formaction

formenctype

formmethod

formnovalidate

formtarget

height and width

list

min and max

multiple

pattern (regexp)

placeholder

required

step

--

HTML5 - New Attribute Syntax

HTML5 allows four different syntaxes for attributes.

This example demonstrates the different syntaxes used in an <input> tag:

Type Example

Empty <input type="text" value="John" disabled>

Unquoted <input type="text" value=John>

Double-quoted <input type="text" value="John Doe">

Single-quoted <input type="text" value='John Doe'>

--

HTML5 Graphics

Tag Description

<canvas> Draw graphics, on the fly, via scripting (usually JavaScript)

<picture> Defines a container for multiple image resources

<svg> Draw scalable vector graphics

--

New Media Elements

Tag Description

<audio> Defines sound content

<embed> Defines a container for an external (non-HTML) application

<picture> Defines a container for multiple image resources

<source> Defines multiple media resources for media elements (<video> and <audio>)

<track> Defines text tracks for media elements (<video> and <audio>)

<video> Defines video or movie

--

New Semantic Elements in HTML5

Many web sites contain HTML code like: <div id="nav"> <div class="header"> <div id="footer">

to indicate navigation, header, and footer.

HTML5 offers new semantic elements to define different parts of a web page:

<article>

<aside>

<details>

<figcaption>

<figure>

<footer>

<header>

<main>

<mark>

<nav>

<section>

<summary>

<time>

--

Differences Between SVG and Canvas

SVG is a language for describing 2D graphics in XML.

Canvas draws 2D graphics, on the fly (with a JavaScript).

SVG is XML based, which means that every element is available within the SVG DOM. You can attach JavaScript event handlers for an element.

In SVG, each drawn shape is remembered as an object. If attributes of an SVG object are changed, the browser can automatically re-render the shape.

Canvas is rendered pixel by pixel. In canvas, once the graphic is drawn, it is forgotten by the browser. If its position should be changed, the entire scene needs to be redrawn, including any objects that might have been covered by the graphic.

Comparison of Canvas and SVG

The table below shows some important differences between Canvas and SVG:

Canvas SVG

Resolution dependent

No support for event handlers

Poor text rendering capabilities

You can save the resulting image as .png or .jpg

Well suited for graphic-intensive games

Resolution independent

Support for event handlers

Best suited for applications with large rendering areas (Google Maps)

Slow rendering if complex (anything that uses the DOM a lot will be slow)

Not suited for game applications

--

Multimedia formats:

Format File Description

MPEG .mpg

.mpeg MPEG. Developed by the Moving Pictures Expert Group. The first popular video format on the web. Used to be supported by all browsers, but it is not supported in HTML5 (See MP4).

AVI .avi AVI (Audio Video Interleave). Developed by Microsoft. Commonly used in video cameras and TV hardware. Plays well on Windows computers, but not in web browsers.

WMV .wmv WMV (Windows Media Video). Developed by Microsoft. Commonly used in video cameras and TV hardware. Plays well on Windows computers, but not in web browsers.

QuickTime .mov QuickTime. Developed by Apple. Commonly used in video cameras and TV hardware. Plays well on Apple computers, but not in web browsers. (See MP4)

RealVideo .rm

.ram RealVideo. Developed by Real Media to allow video streaming with low bandwidths. It is still used for online video and Internet TV, but does not play in web browsers.

Flash .swf

.flv Flash. Developed by Macromedia. Often requires an extra component (plug-in) to play in web browsers.

Ogg .ogg Theora Ogg. Developed by the Xiph.Org Foundation. Supported by HTML5.

WebM .webm WebM. Developed by the web giants, Mozilla, Opera, Adobe, and Google. Supported by HTML5.

MPEG-4

or MP4 .mp4 MP4. Developed by the Moving Pictures Expert Group. Based on QuickTime. Commonly used in newer video cameras and TV hardware. Supported by all HTML5 browsers. Recommended by YouTube.

Audio Formats

MP3 is the newest format for compressed recorded music. The term MP3 has become synonymous with digital music.

If your website is about recorded music, MP3 is the choice.

Format File Description

MIDI .mid

.midi MIDI (Musical Instrument Digital Interface). Main format for all electronic music devices like synthesizers and PC sound cards. MIDI files do not contain sound, but digital notes that can be played by electronics. Plays well on all computers and music hardware, but not in web browsers.

RealAudio .rm

.ram RealAudio. Developed by Real Media to allow streaming of audio with low bandwidths. Does not play in web browsers.

WMA .wma WMA (Windows Media Audio). Developed by Microsoft. Commonly used in music players. Plays well on Windows computers, but not in web browsers.

AAC .aac AAC (Advanced Audio Coding). Developed by Apple as the default format for iTunes. Plays well on Apple computers, but not in web browsers.

WAV .wav WAV. Developed by IBM and Microsoft. Plays well on Windows, Macintosh, and Linux operating systems. Supported by HTML5.

Ogg .ogg Ogg. Developed by the Xiph.Org Foundation. Supported by HTML5.

MP3 .mp3 MP3 files are actually the sound part of MPEG files. MP3 is the most popular format for music players. Combines good compression (small files) with high quality. Supported by all browsers.

MP4 .mp4 MP4 is a video format, but can also be used for audio. MP4 video is the upcoming video format on the internet. This leads to automatic support for MP4 audio by all browsers.

--

<video width="320" height="240" controls>

<source src="movie.mp4" type="video/mp4">

<source src="movie.ogg" type="video/ogg">

Your browser does not support the video tag.

</video>

HTML5 Video Tags

Tag Description

<video> Defines a video or movie

<source> Defines multiple media resources for media elements, such as <video> and <audio>

<track> Defines text tracks in media players

--

<audio controls>

<source src="horse.ogg" type="audio/ogg">

<source src="horse.mp3" type="audio/mpeg">

Your browser does not support the audio element.

</audio>

--

Youtube video:

<!DOCTYPE html>

<html>

<body>

<iframe width="420" height="345" src="https://www.youtube.com/embed/tgbNymZ7vqY">

</iframe>

</body>

</html>

--

<!DOCTYPE html>

<html>

<body>

<object width="420" height="315"

data="https://www.youtube.com/v/tgbNymZ7vqY">

</object>

</body>

</html>

Geolocation:

<!DOCTYPE html>

<html>

<body>

<p>Click the button to get your coordinates.</p>

<button onclick="getLocation()">Try It</button>

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

<script>

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

function getLocation() {

if (navigator.geolocation) {

navigator.geolocation.getCurrentPosition(showPosition);

} else {

x.innerHTML = "Geolocation is not supported by this browser.";

}

}

function showPosition(position) {

x.innerHTML = "Latitude: " + position.coords.latitude +

"<br>Longitude: " + position.coords.longitude;

}

</script>

</body>

</html>

--

Displaying the Result in a Map

To display the result in a map, you need access to a map service, like Google Maps.

In the example below, the returned latitude and longitude is used to show the location in a Google Map (using a static image):

Example

function showPosition(position) {

var latlon = position.coords.latitude + "," + position.coords.longitude;

var img\_url = "https://maps.googleapis.com/maps/api/staticmap?center=

"+latlon+"&zoom=14&size=400x300&sensor=false&key=YOUR\_KEY";

document.getElementById("mapholder").innerHTML = "<img src='"+img\_url+"'>";

}

--

What is a Web Worker?

When executing scripts in an HTML page, the page becomes unresponsive until the script is finished.

A web worker is a JavaScript that runs in the background, independently of other scripts, without affecting the performance of the page. You can continue to do whatever you want: clicking, selecting things, etc., while the web worker runs in the background.

--

What is HTML Web Storage?

With web storage, web applications can store data locally within the user's browser.

Before HTML5, application data had to be stored in cookies, included in every server request. Web storage is more secure, and large amounts of data can be stored locally, without affecting website performance.

Unlike cookies, the storage limit is far larger (at least 5MB) and information is never transferred to the server.

Web storage is per origin (per domain and protocol). All pages, from one origin, can store and access the same data.

window.localStorage - stores data with no expiration date

window.sessionStorage - stores data for one session (data is lost when the browser tab is closed)

--

Server-Sent Events - One Way Messaging

A server-sent event is when a web page automatically gets updates from a server.

This was also possible before, but the web page would have to ask if any updates were available. With server-sent events, the updates come automatically.

Examples: Facebook/Twitter updates, stock price updates, news feeds, sport results, etc.

--

The EventSource Object

In the examples above we used the onmessage event to get messages. But other events are also available:

Events Description

onopen When a connection to the server is opened

onmessage When a message is received

onerror When an error occurs

--

<!-- This is a comment -->

Http requests: beautifulsoup4 – to parse web data // lxml parser // html5lib parser

From bs4 import BeautifulSoup. Soup.prettify()

http responses: 200 success, 300 redirects, 400 – client error, 500 – server error

httpbin.org //

xcross