JavaScript cheatsheet

A quick reference guide to the most commonly used syntax and features of JavaScript, including arrays, loops, classes, modules, promises, and more.

Getting Started

Comments

```
// Single-line comment.

/* Multi-line
   comment. */
```

Strict Mode

```
"use strict";

function foo() {
  "use strict";
  bar = 1;
}
```

Variables

```
let x = 1;
const y = 2;
var z = 3;

let a, b, c;
let d = e = f = 4;
```

Data Types

```
typeof 42;          // "number"
typeof 0.5;         // "number"
typeof NaN          // "number"
typeof "text";      // "string"
typeof true;        // "boolean"
typeof undefined    // "undefined"
typeof null         // "object"
typeof Symbol()     // "symbol"
typeof 10n          // "bigint"
typeof {}           // "object"
typeof []           // "object"
typeof function     // "function"
```

Type Conversion

```
Number("42");       // 42
Number("");         // 0
Number("abc");      // NaN
Number(true);       // 1
Number(null);       // 0
Number(undefined);  // NaN

String(42);         // "42"
String(true);       // "true"
String(null);       // "null"
String(undefined);  // "undefined"

Boolean(1);         // true
Boolean(0);         // false
Boolean("");        // false
Boolean("abc");     // true
Boolean(null);      // false
Boolean(undefined);
```

Operators

```
// Arithmetic
+ - * / % **

// Comparison
==  !=  ===  !==
>  <  >=  <=

// Logical
&&  ||  !

// Bitwise
&  |  ^  ~  <<  >>  >>>

// Assignment
=  +=  -=  *=  /=  %=  **=
&&=  ||=  ??=

// Other
?.  ??  ...  ,  ?  :
```

Strings

```
"text"
'text'
`text`

`Hello, ${name}!`
`${a + b}`
`${obj.method()}`

`Line 1
Line 2`

`Tag${expr}text`
```

Regular Expressions

```
/pattern/
/pattern/flags
new RegExp("pattern")
new RegExp("pattern", "flags")

/^start/
/end$/
/[abc]/
/[a-z]/
/[^abc]/
/a|b/
/(group)/
/(?:non-capturing)/
/(?=lookahead)/
/(?!negative lookahead)/
/(?<=lookbehind)/
/(?<!negative lookbehind)/
/(?<name>capture)/

/.*/   greedy
/.*?/  lazy

/a{3}/
/a{3,}/
/a{3,5}/
/a*/
/a+/
/a?/

/\d/  /\D/
/\w/  /\W/
/\s/  /\S/
/\b/  /\B/
/\n/  /\t/  /\r/
/\0/
/\uXXXX/
/\uXXXX{Y}/
/\xXX/
/\p{Unicode property}/
/\p{Unicode property=value}/
/\k<name>/
/\1/  /\2/

/.  /g  /i  /m  /s  /u  /y  /d  /v
```

JavaScript Conditionals

If Else

```
if (condition) {
  doSomething();
} else if (other) {
  doOtherThing();
} else {
  doDefaultThing();
}
```

Ternary Operator

```
condition ? value1 : value2;

result = foo ? 1 : 0;
```

Switch

```
switch (value) {
  case 1:
    doOne();
    break;
  case 2:
    doTwo();
    break;
  case 3:
    doThree();
    break;
  default:
    doDefault();
}
```

Logical Operators

```
expr1 && expr2
expr1 || expr2
!expr

expr1 ?? expr2
```

Nullish Coalescing Operator ??

```
null ?? "default";
undefined ?? "default";
0 ?? "default";
""  ?? "default";
false ?? "default";
```

Optional Chaining Operator ?.

```
obj?.prop
obj?.[expr]
arr?.[index]
func?.(arg)

obj?.method?.()
```

Truthy and Falsy

```
// Falsy:
false
0
-0
0n
""
''
``
null
undefined
NaN

// Everything else is truthy.
```

JavaScript Functions

Function Declaration

```
function name(param1, param2) {
  return value;
}

function greet() {
  console.log("Hi");
}
```

Function Expression

```
const name = function(param1, param2) {
  return value;
};

const greet = function() {
  console.log("Hi");
};
```

Arrow Function

```
const name = (param1, param2) => {
  return value;
};

const greet = () => {
  console.log("Hi");
};

const add = (a, b) => a + b;
```

Default Parameters

```
function greet(name = "World") {
  console.log(`Hello, ${name}!`);
}

greet();
greet("Alice");
```

Rest Parameters

```
function sum(...numbers) {
  return numbers.reduce((a, b) => a + b);
}

sum(1, 2, 3, 4);
```

Spread Operator

```
const arr = [...[1, 2, 3], 4, 5];
const obj = {...{a: 1}, b: 2};
const str = [...'abc'];
const copy = [...original];

function call(...arguments) {
  console.log(arguments);
}
```

Return Statement

```
function name() {
  return value;
}

function name() {
  return;
}

function name() {
  return
  value;
}
```

Function Properties

```
function name(a, b) {}

name.length;
name.name;
name.prototype;
name.call(thisArg, a, b);
name.apply(thisArg, [a, b]);
name.bind(thisArg, a);
```

Anonymous Function

```
function() {
  doSomething();
}
```

Immediately Invoked Function Expression (IIFE)

```
(function() {
  doSomething();
})();

(function(a, b) {
  return a + b;
})(1, 2);

(() => {
  doSomething();
})();
```

Recursion

```
function factorial(n) {
  if (n <= 1) {
    return 1;
  }
  return n * factorial(n - 1);
}

factorial(5);
```

JavaScript Scope

Global Scope

```
globalVar = 1;
let globalLet = 2;
const globalConst = 3;
var globalVar = 4;

function foo() {
  globalVar = 5;
}
```

Local Scope

```
function foo() {
  let localLet = 1;
  const localConst = 2;
  var localVar = 3;
}

if (true) {
  let blockLet = 4;
  const blockConst = 5;
  var blockVar = 6;
}

console.log(blockLet);
console.log(blockConst);
console.log(blockVar);
```

Block Scope

```
let x = 1;
const y = 2;

if (true) {
  let x = 3;
  const y = 4;
  console.log(x);
  console.log(y);
}

console.log(x);
console.log(y);
```

Function Scope

```
function foo() {
  let x = 1;
  const y = 2;
  var z = 3;
}

console.log(x);
console.log(y);
console.log(z);
```

Lexical Scope

```
function outer() {
  let x = 1;
  function inner() {
    console.log(x);
  }
  return inner;
}

const fn = outer();
fn();
```

Hoisting

```
console.log(x);
console.log(y);
console.log(z);

var x = 1;
let y = 2;
const z = 3;

console.log(x);
console.log(y);
console.log(z);
```

Temporal Dead Zone

```
console.log(x);

let x = 1;

console.log(typeof x);

let y = y + 1;
```

JavaScript Arrays

Array Literal

```
[]
[1, 2, 3]
[1, 2, 3,]
[1, , 3]
[, , ,]
```

Array Constructor

```
new Array();
new Array(5);
new Array(1, 2, 3);
```

Array Properties

```
arr.length
arr[0]
arr[-1]
arr[1.5]
arr["key"]
arr[Symbol()]
```

Array Methods

```
arr.push(item);
arr.pop();
arr.shift();
arr.unshift(item);
arr.splice(start, deleteCount, item1, item2);
arr.slice(start, end);
arr.concat(arr2, arr3);
arr.indexOf(item, fromIndex);
arr.lastIndexOf(item, fromIndex);
arr.includes(item, fromIndex);
arr.at(index);
arr.join(separator);
arr.toSorted(compareFunction);
arr.toReversed();
arr.toSpliced(start, deleteCount, item1, item2);
arr.with(index, value);
arr.flat(depth);
arr.flatMap(callbackFn, thisArg);
arr.fill(value, start, end);
arr.copyWithin(target, start, end);
arr.reverse();
arr.sort(compareFunction);
arr.toString();
arr.toLocaleString();
arr.forEach(callbackFn, thisArg);
arr.map(callbackFn, thisArg);
arr.filter(callbackFn, thisArg);
arr.reduce(callbackFn, initialValue);
arr.reduceRight(callbackFn, initialValue);
arr.every(callbackFn, thisArg);
arr.some(callbackFn, thisArg);
arr.find(callbackFn, thisArg);
arr.findIndex(callbackFn, thisArg);
arr.findLast(callbackFn, thisArg);
arr.findLastIndex(callbackFn, thisArg);
arr.keys();
arr.values();
arr.entries();
arr.toJSON();
```

Array Destructuring

```
const [a, b, c] = arr;
const [a, b, c, d, e, f] = arr;
const [a, b, c, ...rest] = arr;
const [a, b, c, ...rest, z] = arr;
const [a, , c] = arr;
const [a, b, c, d, e, f, g = 7] = arr;
const [a, b, c, d, e, f, g = 7, h = 8] = arr;
const [a, [b, c], d] = arr;
const [a, b, c, d, e, f, g, h, [i, j, k, l, m, n, o, p, [q, r, s, t, u, v, w, x, [y, z]]]] = arr;
```

JavaScript Loops

While Loop

```
while (condition) {
  doSomething();
}

while (true) {
  doSomething();
  if (condition) {
    break;
  }
}
```

Do While Loop

```
do {
  doSomething();
} while (condition);

do {
  doSomething();
  if (condition) {
    break;
  }
} while (true);
```

For Loop

```
for (let i = 0; i < 10; i++) {
  console.log(i);
}

for (let i = 0, j = 10; i < j; i++, j--) {
  console.log(i, j);
}

for (; condition; ) {
  doSomething();
}

for (let i = 0; ; i++) {
  if (i > 10) {
    break;
  }
  console.log(i);
}
```

For In Loop

```
for (key in obj) {
  console.log(key);
  console.log(obj[key]);
}

for (key in arr) {
  console.log(key);
  console.log(arr[key]);
}

for (key in "abc") {
  console.log(key);
}
```

For Of Loop

```
for (item of arr) {
  console.log(item);
}

for (item of "abc") {
  console.log(item);
}

for (item of [...set]) {
  console.log(item);
}

for (item of [...map.keys()]) {
  console.log(item);
}

for (item of [...map.values()]) {
  console.log(item);
}

for (item of [...map.entries()]) {
  console.log(item);
}

for (item of [...obj.keys()]) {
  console.log(item);
}

for (item of [...obj.values()]) {
  console.log(item);
}

for (item of [...obj.entries()]) {
  console.log(item);
}

for (item of [...arguments]) {
  console.log(item);
}

for (item of [...nodeList]) {
  console.log(item);
}

for (item of [...htmlCollection]) {
  console.log(item);
}

for (item of [...iterable]) {
  console.log(item);
}
```

For Of Loop Destructuring

```
for ([a, b, c] of arr) {
  console.log(a, b, c);
}

for ([a, b, c, ...rest] of arr) {
  console.log(a, b, c, rest);
}

for ([a, b, c, ...rest, z] of arr) {
  console.log(a, b, c, rest, z);
}

for ([a, , c] of arr) {
  console.log(a, c);
}

for ([a, b, c, d, e, f, g = 7] of arr) {
  console.log(a, b, c, d, e, f, g);
}

for ([a, b, c, d, e, f, g = 7, h = 8] of arr) {
  console.log(a, b, c, d, e, f, g, h);
}

for ([a, [b, c], d] of arr) {
  console.log(a, b, c, d);
}

for ([a, b, c, d, e, f, g, h, [i, j, k, l, m, n, o, p, [q, r, s, t, u, v, w, x, [y, z]]]] of arr) {
  console.log(a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z);
}
```

For Of Loop Destructuring

```
for ({a, b, c} of arr) {
  console.log(a, b, c);
}

for ({a, b, c, ...rest} of arr) {
  console.log(a, b, c, rest);
}

for ({a, b, c, ...rest, z} of arr) {
  console.log(a, b, c, rest, z);
}

for ({a, b, c, d, e, f, g = 7} of arr) {
  console.log(a, b, c, d, e, f, g);
}

for ({a, b, c, d, e, f, g = 7, h = 8} of arr) {
  console.log(a, b, c, d, e, f, g, h);
}

for ({a, b, c, d, e, f, g, h, i: {j, k, l, m, n, o, p, q, r: {s, t, u, v, w, x, y, z}}} of arr) {
  console.log(a, b, c, d, e, f, g, h, j, k, l, m, n, o, p, q, s, t, u, v, w, x, y, z);
}
```

JavaScript Iterators

Iterable Protocol

```
[Symbol.iterator]() {
  return iterator;
}

for (item of iterable) {
  console.log(item);
}

[...iterable]
```

Iterator Protocol

```
{
  next() {
    return {value: 1, done: false};
    return {value: 2, done: false};
    return {value: 3, done: false};
    return {value: undefined, done: true};
  }
}

iterator.next();
```

Iterable Iterator

```
{
  [Symbol.iterator]() {
    return this;
  }
  next() {
    return {value: 1, done: false};
    return {value: 2, done: false};
    return {value: 3, done: false};
    return {value: undefined, done: true};
  }
}

iterable[Symbol.iterator]();
```

Generator Function

```
function* name() {
  yield 1;
  yield 2;
  yield 3;
  return 4;
}

const gen = name();
gen.next();
```

Generator Function Expression

```
const name = function* () {
  yield 1;
  yield 2;
  yield 3;
  return 4;
};

const gen = name();
gen.next();
```

Generator Method

```
{
  *name() {
    yield 1;
    yield 2;
    yield 3;
    return 4;
  }
}

const gen = obj.name();
gen.next();
```

Generator Iterator

```
function* name() {
  yield 1;
  yield 2;
  yield 3;
  return 4;
}

const gen = name();
gen.next();
gen.next();
gen.next();
gen.next();
```

Yield Expression

```
function* name() {
  yield 1;
  yield 2;
  yield 3;
  return 4;
}

const gen = name();
gen.next();
gen.next();
gen.next();
gen.next();
```

Yield Delegate

```
function* name() {
  yield* [1, 2, 3];
  yield* "abc";
  yield* [...set];
  yield* [...map.keys()];
  yield* [...map.values()];
  yield* [...map.entries()];
  yield* [...obj.keys()];
  yield* [...obj.values()];
  yield* [...obj.entries()];
  yield* [...arguments];
  yield* [...nodeList];
  yield* [...htmlCollection];
  yield* [...iterable];
  yield* name();
  yield* obj.name();
  yield* obj[Symbol()]();
  yield* obj[Symbol()]();
  yield* obj[Symbol()]
```