

### Data types

#### Primitive types

Number	Number value (integers and floating point) except big integer
BigInt	Large integer Notation: number literal and an n suffix <i>e.g.</i> -3000000n
String	String of characters Notation: between single or double quotes
Boolean	Boolean - <i>true</i> or <i>false</i>
Symbol	Unique identifier
Undefined	No value assigned
Null	Deliberate non-value

#### Object

Function
Array
Date
RegExp
Error

### Comments

// or /\* \*/

### String

<code>string.length</code>	Returns the length of a string
<code>string.charAt(index)</code>	Returns the character at the index
<code>string.concat(string1, string2, ..., stringX)</code>	Concatenates two or more strings
<code>string.match(regex)</code>	Returns an array of all matches, or null if no match
<code>string.replace(old, new)</code>	Returns a new string where <i>old</i> has been replaced by <i>new</i>
<code>string.search(regex)</code>	Returns the position of the first match, or -1 if no match
<code>string.substring(start, end)</code>	Returns a string containing the characters from start to end (not included)
<code>string.toLowerCase()</code>	Returns the string converted to lowercase

### String (cont)

<code>string.toUpperCase()</code>	Returns the string converted to uppercase
<code>"age:" + age + " years" ↔ `age: \${age} years`</code>	Template literal: allows you to write strings with embedded expressions more succinctly
<b>Escape character :</b> \	' , \" , \\ , (\b , \f , \n , \r , \t , \v)

### Boolean

<code>false</code> , 0, empty strings (""), NaN, null, and undefined	interpreted as <i>false</i> , the rest as <i>true</i>
<code>Boolean(expr)</code>	Explicit conversion to Boolean

### Array

<i>e.g</i> <code>const array = ["a", 10, "c"]</code>	Create an array
<code>array.length</code>	<b>Sets</b> or returns the number of elements in an array

Can assign any properties to array, like indices. The only "magic" is that length will be automatically updated when you set a particular index.

This can create sparse array

<code>array[index] ↔ array.at(index)</code>	Returns the array at the index
<code>array.concat(string1, string2, ..., stringX)</code>	Concatenates two or more arrays
<code>array.every(fct, value)</code>	Returns true if every element pass the test, otherwise false
<code>array.flat(depth)</code>	Concatenates sub-array elements
<code>array.forEach(fct, value)</code>	Calls a function for each array element
<code>array.indexOf()</code>	Search the array for an element and returns its position
<code>array.lastIndexOf()</code>	Search the array for an element, starting at the end, and returns its position



By **Manon Gerrd**  
(manongerard)

Not published yet.  
Last updated 5th November, 2023.  
Page 1 of 6.

Sponsored by **ApolloPad.com**  
Everyone has a novel in them. Finish Yours!

### Array (cont)

<code>array.pop()</code>	Removes the last element of an array, and returns that element
<code>array.push(item1, item2, ..., itemX)</code>	Adds new elements to the end of an array, and returns the new length
<code>array.sort()</code>	Sorts the elements of an array
<code>array.toString</code>	Converts an array to a string, and returns the result
<code>array.valueOf()</code>	Returns the primitive value of an array
<code>array.filter(function(currentValue, index, arr), thisValue)</code>	Creates a new array filled with elements that pass a test provided by a function.

Index starts at 0 like in Java or C  
Arrays are not limited in type of elements

### RegExp

[https://www.w3schools.com/js/js\\_regexp.asp](https://www.w3schools.com/js/js_regexp.asp)

### Date

<code>new Date()</code>	Create the current date and time
<code>new Date(date string   year, month, [day, hours, minutes, seconds, ms]   milliseconds)</code>	Create a specific date and time
<code>Date.now()</code>	Returns the number of milliseconds since January 1, 1970
<code>date.parse()</code>	Parses a date string and returns the number of milliseconds since <b>January 1, 1970</b>
<code>date.get[UTC][Date   FullYear   Month   Day   Hours   Minutes   Seconds   Milliseconds   Time]()</code>	Getters

### Date (cont)

`date.set[UTC][Date | FullYear | Month | Hours | Minutes | Seconds | Milliseconds | Time]()` Setters

⚠ JavaScript counts months from 0 to 11 and days from 0 to 6

*date string* format :

- ISO: `YYYY[-MM-DDTHH:MM:SSZ]`

- short: `MM/DD/YYYY`

- long: `MMM DD YYYY`

### Class (similar to Java)

JavaScript offers the class syntax that's very similar to languages like Java.

```
class Person {
    constructor( name) {
        this.name = name;
    }
    say Hello() {
        return Hello, I'm ${this.name}!;
    }
}
```

```
const p = new Person ("Ma ria ");
console.log( p.s ayH ell o());
```

*this* refers to the person object

### Arithmetic operators (similar to C/Java)

<code>x + y</code> (numeric)	Adds x and y together
<code>x</code> (numeric or string) <code>+ y</code> (string)	Concatinates x and y together
<code>x - y</code>	Subtracts y from x
<code>x * y</code>	Multiplies x and y together
<code>x / y</code>	Divides x by y
<code>x % y</code>	x modulo y: the remainder when x is divided by y
<code>x ** y</code>	x to the power of y
<code>x++</code> , <code>++x</code>	Increment x (postfix and prefix)
<code>x--</code> , <code>--x</code>	Decrement x (postfix and prefix)

Same operator precedence as usual



By **Manon Gerrd**  
(manongerard)

[cheatography.com/manongerard/](https://cheatography.com/manongerard/)

Not published yet.

Last updated 5th November, 2023.

Page 2 of 6.

Sponsored by **ApolloPad.com**

Everyone has a novel in them. Finish Yours!

<https://apollopadd.com>

### Assignment operators (similar to C/Java)

<code>x = y</code>	Sets x to the value of y
<code>x op= y</code>	Same as <code>x = x op y</code>
<code>x: number</code>	<code>size.x = number</code>

### Comparison operators (similar to C/Java)

<code>x &gt; y</code>	Returns true if x is greater than y
<code>x &gt;= y</code>	Returns true if x is greater than or equal to y
<code>x &lt; y</code>	Returns true if x is less than y
<code>x &lt;= y</code>	Returns true if x is less than or equal to y
<code>x == y</code>	Returns true if x and y are equal Note: performs type coercion
<code>x != y</code>	Returns true if x and y are not equal Note: performs type coercion
<code>x === y</code>	Returns true if x and y are equal in value and type
<code>x !== y</code>	Returns true if x and y are not equal in value or type
<code>condition ? exprIfTrue : exprIfFalse</code>	Returns <code>exprIfTrue</code> if <code>condition</code> is true, <code>exprIfFalse</code> otherwise

⚠ Strings are compared alphabetically

### Bitwise operators (similar to C/Java)

<code>x &amp; y</code>	x AND y
<code>x   y</code>	x OR y
<code>~x</code>	NOT x
<code>x ^ y</code>	x XOR y
<code>x &lt;&lt; y</code>	x left shift of y
<code>x &gt;&gt; y</code>	x right shift of y
<code>x &gt;&gt;&gt; y</code>	x unsigned right shift of y

### Type operators

<code>typeof</code>	Returns the type of a variable
<code>instanceof</code>	Returns true if an object is an instance of an object type

### Number

<code>parseInt(string, radix)</code>	Parses the string into an integer
<code>parseFloat(string, radix)</code>	Parses the string into a floating-point number
<code>Number(value)</code>	Parses the value into a number if possible
<code>NaN</code>	"Not a number" value
<code>Infinity</code>	Positive or negative infinity value
<code>Math.function()</code>	Math library has a lot of math functions <i>e.g.</i> <code>abs</code> , <code>cbrt</code> , <code>ceil</code> , <code>cos</code> , <code>exp</code> , <code>floor</code> , <code>log</code> , <code>log10</code> , <code>max</code> , <code>min</code> , <code>pow</code> , <code>random</code> , <code>round</code> , <code>sign</code> , <code>sin</code> , <code>sqrt</code> , <code>tan</code> ,...
<code>Math.constant</code>	Math library has a lot of math constants <i>e.g.</i> <code>E</code> , <code>PI</code> , ...
<i>e.g.</i> <code>0b1010</code> , <code>0o12</code> , <code>10</code> , <code>0xA</code> , <code>0.1e2</code>	Prefixes to indicate the base or an exponent suffix

### Symbol

<code>Symbol(expr)</code>	Create a unique Symbol with key <code>expr</code>
<code>Symbol.for(expr)</code>	Return the Symbol with the key <code>expr</code> , if it doesn't exist a new Symbol is created and returned.
<code>Symbol.keyFor(symbol)</code>	Returns the string key corresponding to symbol

Many functions similar to string also exist, they call string's function

### Object

JavaScript objects can be thought of as collections of key-value pairs. They are hashes. They do not have fixed shapes - properties can be added, deleted, reordered, mutated, or dynamically queried at any time. Objects keys are always **strings** or **symbols**.

```
const obj = {
  name: "Carrot",
  for: "Max",
  details: {
```



By **Manon Gerrd**  
(manongerard)

[cheatography.com/manongerard/](https://cheatography.com/manongerard/)

Not published yet.  
Last updated 5th November, 2023.  
Page 3 of 6.

Sponsored by **ApolloPad.com**  
Everyone has a novel in them. Finish Yours!  
<https://apollopad.com>

### Object (cont)

```
>   color: "orange",
    size: 12,
  },
  fullDescr: function() {
    return this.name + " for " + this.for;
  }
};
```

Object properties can be accessed using dot ( . ) or square brackets ( [] ). *e.g.* obj.name or obj["name"]

A method is a function stored as a property.

### Function

#### Function Declaration

```
function fctName(args) { code }
```

Like in Python, it is possible to assign a default value to the arguments

#### Anonymous functions

```
`const var = (function (args) {
  code
});`
```

Use `var() {}`

`function` is not mandatory

#### 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)

#### Arrow functions

```
product = (a, b) => a * b;
```

### Variables

`let var` Declare a block scope local variable

`const` Declare a block scope variable whose value is never intended to change

`var` It cannot be reassigned, therefore it must be initialized

### Variables (cont)

`var` Declare a variable, not block-scoped -> discouraged in modern JavaScript

⚠ different that in C/Java: `let` and `const` declared variables occupy the entire scope they are defined in, and are in a region known as the temporal dead zone before the actual line of declaration. JavaScript is dynamically typed (like Python), for `let` can change the type through reassignment

### Control structures (similar to C)

#### - if and else

```
{{bb = 1}}
if (cdt) {
  code
} else if (cdt) {
  code
} else {
  code
}
```

#### - while

```
while (cdt) {
  code
}
```

#### - do ... while

```
do {
  code
} while (cdt);
```

#### - for loop

```
for (init; condition; update) {
  code
}
```

#### - for ... of

It iterates over iterables, most notably arrays,

```
for (const value of array) {
  code
}
```

#### - for ... in

It visits all enumerable properties of an object.

```
for (const property in object) {
  code
}
```

#### - switch



By **Manon Gerrd**  
(manongerard)

Not published yet.  
Last updated 5th November, 2023.  
Page 4 of 6.

Sponsored by **ApolloPad.com**  
Everyone has a novel in them. Finish Yours!

### Control structures (similar to C) (cont)

```
> switch(variable/expression) {
  case value1:
    // body of case 1
    break;
  case value2:
    // body of case 2
    break;
  case valueN:
    // body of case N
    break;
  default:
    // body of default
}
```

break works like in C/Java. If you don't add a break statement, execution will "fall through" to the next level.

Comparison using ===, cases can be any expression

#### - throw

Errors can be thrown using the throw statement.

```
throw new Error(string);
```

#### - try ... catch

```
try {
  code;
} catch (e) {
  console.error(string, e);
}
```

There's no conditional catch in JavaScript — if you only want to handle one type of error, you need to catch everything, identify the type of error using instanceof, and then rethrow the other cases.

```
try {
  code;
} catch (e) {
  if (e instanceof RangeError) {
    code
  } else {
    throw e;
  }
}
```

#### - return

#### - continue

### Logical operators

x && y	Returns true if both x and y are true
x    y	Returns true if x or y is true
!x	Returns true if x is false

### HTML events

```
<element event="some JavaScript">
```

**Window Event** inside <body>

#### Attributes

onafterprint	onbeforeprint	onbeforeu- nload	onerror
onhashchange	onmessage	onoffline	ononline
onpagehide	onpageshow	onpopstate	onresize
onstorage	onunload		

#### Form Events

in form elements

onblur	onchange	oncont- extmenu	onfocus
oninput	oninvalid	onreset	onsearch
onselect	onsubmit		

#### Keyboard Events

onkeydown	onkeypress	onkeyup
-----------	------------	---------

#### Mouse Events

onclick	ondblclick	onmous- edown	onmous- emove
onmouseout	onmouseover	onmouseup	onwheel

#### Drag Events

ondrag	ondragend	ondragenter	ondrag- leave
ondragover	ondragstart	ondrop	onscroll

#### Clipboard Events

oncopy	oncut	onpaste
--------	-------	---------

#### Media Events

Inside <audio>, <embed>, <img>, <object> and <video>

onabort	oncanplay	oncanplay- through	oncuec- hange
ondurationchange	onemptied	onended	onerror
onloadeddata	onloadedm- etadata	onloadstart	onpause



By **Manon Gerrd**  
(manongerard)

[cheatography.com/manongerard/](https://cheatography.com/manongerard/)

Not published yet.

Last updated 5th November, 2023.

Page 5 of 6.

Sponsored by **ApolloPad.com**

Everyone has a novel in them. Finish Yours!

<https://apollopad.com>

### HTML events (cont)

onplay	onplaying	onprogress	onratechange
onseeked	onseeking	onstalled	onsuspend
ontime-update	onvolumechange	onwaiting	

### Misc Events

ontoggle	Fires when the user opens or closes the <details> element
----------	---



By **Manon Gerrd**  
(manongerard)

[cheatography.com/manongerard/](https://cheatography.com/manongerard/)

Not published yet.

Last updated 5th November, 2023.

Page 6 of 6.

Sponsored by **ApolloPad.com**

Everyone has a novel in them. Finish

Yours!

<https://apollopad.com>