Data types

Primitive types

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

Object

Function

Array

Date

RegExp

Error

Comments

// or /* */

String

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

String (cont)

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

Boolean

false, 0, empty strings (""), NaN, null, and undefined	interpreted as *false*, the rest as *true*
Boolean(expr)	Explicit conversion to Boolean

Array

e.g const *array* = ["a", 10, "c"]	Create an array
array.length	**Sets** 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

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

Array (cont)

array.pop()	Removes the last element of an array, and returns that element
array.push(item1, item2, ..., itemX)	Adds new elements to the end of an array, and returns the new length
array.sort()	Sorts the elements of an array
array.toString	Converts an array to a string, and returns the result
array.valueOf()	Returns the primitive value of an array
array.filter(function(currentValue, index, arr), thisValue)	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

Date

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

Date (cont)

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

⚠ 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 {
        con str uct or( name) {
                thi s.name = name;
       }
        say Hello() {
        retuHello, I'm ${this.name}!;
}
}
const p = new Person ("Ma ria ");
consol e.l og( p.s ayH ell o());
```

`this` refers to the person object

Arithmetic operators (similar to C/Java)

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

Same operator precedence as usual

Assignment operators (similar to C/Java)

x = y	Sets x to the value of y
x *op*= y	Same as x = x *op* y
x:*number*	size.x = *number*

Comparison operators (similar to C/Java)

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

⚠ Strings are compared alphabetically

Bitwise operators (similar to C/Java)

x & y	x AND y
x | y	x OR y
~x	NOT x
x ^ y	x XOR y
x << y	x left shift of y
x >> y	x right shift of y
x >>> y	x unsigned right shift of y

Type operators

typeof	Returns the type of a variable
instanceof	Returns true if an object is an instance of an object type

Number

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

Symbol

Symbol-(expr)	Create a unique Symbol with key expr
Symbol.fo-r(expr)	Return the Symbol with the key expr, if it doesn't exists a new Symbol is created and returned.
Symbol.ke-yFor(s-ymbol)	Returns the string key corresponding to symbol
Many functions similar to string also exists, 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, re-
ord ered, mutated, or dynami cally queried at any
time. Objects keys are always strings or symbols.
const obj = {
        name: " Car rot ",
       for: " Max ",
        det ails: {
```

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 *var*	Declare a block scope variable whose value is never intended to change It cannot be reassigned, therefore it must be initialized

Variables (cont)

var *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

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 Attributes	inside <body>		
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

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/

Not published yet.
Last updated 5th November, 2023.