Javascript Fundamentals Cheat Sheet

by raposinha via cheatography.com/197915/cs/42598/

Introduction

Javascript was designed to run only in browsers so every browser uses a Javascript Engine. Node combines C++ and JS so JS can run outside of browsers.

ECMASCRIPT, Specification, defines JS standards.

The Javascript Console can be found in Chrome > Inspect > Console.

Just like browsers, Node includes the v8 JavaScript engine, so it can read and execute JavaScript scripts

Operator's precedence

The precedence is as follows: multiplication *, sum +

Bitwise operators

A little less practical.

1 = 00000001, 2 = 00000010

Bitwise are similar to Logical operators, but they operate on the singular bits of a number: each bit/8 is compared.

Bitwise OR	consol-e.log(1 | 2); //3
With OR, each individual bit is compared, if any of them is 1, the result is zero, like:	00000001 //1
	00000010 //2
	00000011 //(1 | 2)
Bitwise AND	consol-e.log(1 & 2); //

Bitwise operators (cont)

With AND, each individual bit is compared, if both bits are 1, the result is one, otherwise 0:	00000011 //(1 & 2)

Logical operators with non-booleans

If the operand/'condition' is not 'true' or 'false'(boolean) JS will try to interpret it as 'truey' or 'falsey'.

"Falsey" values:	undefined, null, 0, false, '', "", NaN
"Truthy" values:	anything else - Strings, natural numbers

Logical operators

Logical AND (&&)	Returns 'true' if both operands or conditions are 'true'	true && true => true; true && false => false
Logical OR (||)	Returns 'true' if one of the operands/-conditions are 'true'	true || false => true; true || true => true; false || true => true; false || false => false
Logical NOT (!)	Will turn the operand /condition into false if true, true if false	let happy = !sad

Ternary operators

```
// Ternary operators
// If a costumer has over 10
points they're a GOLD costumer,
otherwise they're silver.
let points = 110;
// Condition (produces boolean),
if true, set to 'gold',
otherwise, 'silver'
let custom erType = points > 100
? 'gold' : 'silver';
consol e.l og( cus tom erT ype);
There's a better way to shorten
this if the condit ion's result
is true or false:
return width > height;
instead of : return width >
height ? true : false;
```

These conditions use booleans to return a value depending on the boolean type.

Operators

Operators are used alongside variables to create expressions. With these we can create logic and algorithms.

In JavaScript we have Arithmetic, Assigment, Comparison, Bitwise and Logical Operators.

Arithmetic

Assignment

Arithmetic Operators

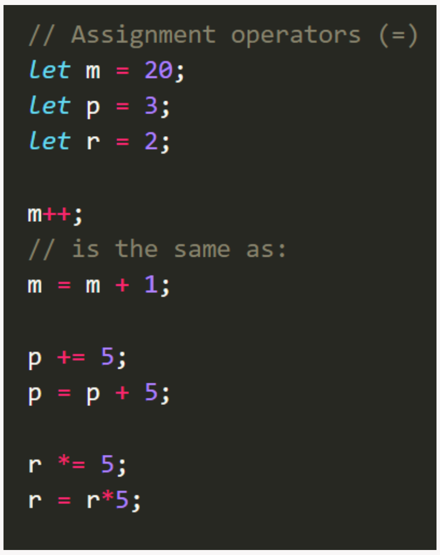
```
let x = 10;
let y = 3;
consol e.log(x + y);
consol e.log(x - y);
consol e.log(x * y);
consol e.log(x / y);
consol e.log(x % y);
```

Arithmetic Operators (cont)

```
> console.log(x ** y);
//// Increment and Decrement Operators
// 10
console.log(x);
// 11+1 (operation applied first)
console.log(++x);
// 11+1 (operation applied later)
console.log(x++);
```

Used for performing calculations, like mathematics. Usually variables with numeric values are used (operands) to produce new values (expression - something that produces a value.
For increment and decrement operators, if applied before the variable, the operation will be performed before the action. If applied after, after the action is executed.

Assignment operators



```
// Assignment operators (=)
let m = 20;
let p = 3;
let r = 2;

m++;
// is the same as:
m = m + 1;

p += 5;
p = p + 5;

r *= 5;
r = r*5;
```

Comparison operators

```
// Relational operators
let xx = 1;
consol e.l og(xx > 0);
// true, 1 is bigger than 0
consol e.l og(xx >= 1);
// true, 1 is equal or bigger
than 1
consol e.l og(xx < 1);
// false, 1 is no less than 1
consol e.l og(xx <= 1);
// true, 1 is equal or smaller
to 1
// Equality operators
consol e.l og(xx === 1);
// true, x is the same value and
type as 1
consol e.l og(xx !== 1);
// false, x is no different to 1
```

We use them to compare the value of a variable with something else.
The result of an expression that includes a comparison operator is a boolean (true or false).

Equality operators

```
// Equality operators
consol e.l og(xx === 1);
// true, x is the same value and
type as 1
consol e.l og(xx !== 1);
// false, x is no different to 1
//// Lose equality operators
consol e.log( xx == y);
//// Strict equality operators
consol e.log( xx === y);
// true
consol e.log( '1' == 1 );
// false
```

Equality operators (cont)

```
> console.log( '1' === 1 );
```

Lose equality operators ensure that two variables share value, Strict equality operators ensure that two variables share value and type. Type such as number, string, etc.

Lose equality will take the first variable's type and convert the second to that type automatically when compared.

Boilerplate project

To start off, create an HTML document. Set a <script> tag on the head or body, but best practice is at the end of the <body> element because the browser will parse the content the DOM first.

// This is a comment.

console.log("This is a sequence. It logs this message from the console.")

<script src="index.js"/>

From the terminal, launch "node index.js" to run the JavaScript script

From VSCode, run View > Terminal to run the JavaScript script

Reference types

Objects	A type that holds properties - when multiple properties are related we can fit them inside an Object.	let person = {}
	Inside an object tehre's value and keys:	{ name: 'Mosh', age: 27 }
	Objects can also be printed	consol e.log(- per- son);

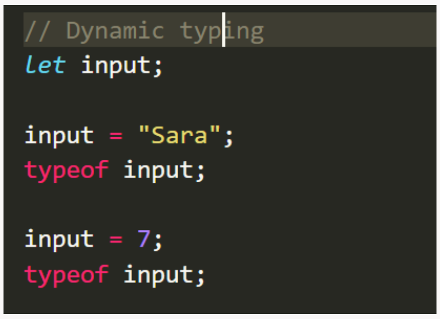
Reference types (cont)

	Object properties can be changed. (Dynamic typing, remember?)	person.name = 'Sara'
		person['n-ame'] = "-Mary"
Arrays	A type used to store other types in a list-like manner. Techni-cally an Object.	let select-edColors = [];
		let select-edColors = ['red', 'blue'];
	Array elements each have an index, in this case: red is 0, blue is 1. To access them	selectedC-olors[0]; // red
	Because JavaScript is a dynamic language, variables can be set, added, deleted at runtime or any time. And they can be of any type	selectedC-olors[2] = 'yellow';
		selectedC-olors[3] = 8;

Reference types (cont)

	Because Arrays are Objects, they have their own inherited properties like indexOf, length...	
Functions	A set of statements that perform a task or calculates a value	
	The variable we parse into the function is an 'argument'.	greet('María');
	If we don't parse a second variable, it will print undefined.	greet("-Juana",las-tName);
	All functions in JavaScript are objects, so they have properties and methods that we can access using the dot notation (I.e.: Object.keys	

JavaScript is a Dynamic Typing language

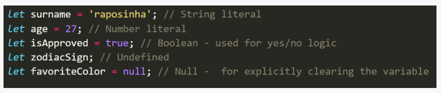


```
// Dynamic typing
let input;

input = "Sara";
typeof input;

input = 7;
typeof input;
```

Primitive variable types



```
let surname = 'raposinha'; // String literal
let age = 27; // Number literal
let isApproved = true; // Boolean - used for yes/no logic
let zodiacSign; // Undefined
let favoriteColor = null; // Null -  for explicitly clearing the variable
```

To check a primitive variable type **typeof** is used:
typeof n !== 'number'

Control flow

- If ... Else	
- Switch ... Case	switch(case) {
	case 'guest':
	console.log('Guest');
	break;
	case 'moderator':
	console.log('Moderator');
	default:
	console.log('Unknown');
	}
	Note 1: If break is not added, the condition doesn't skip and case doesn't work, it just executes the next statement within the first case read.

Control flow (cont)

	Note 2: An expression is any valid unit of code that resolves to a VALUE. Case is an expression, whether it is 2, 'a', or true. When case matches the variables, wether with a given variable or a set expression like 'true', code will execute, check the condition and if matching, execute and break.
- For ...	'for' includes 3 statements: Initial expression, where a variable is initialized, it's usually set like 'i', short for Index. Condition, where we usually compare the value of the Index to something else; the loop will continue unless this condition is false. If we want the loop to go on 5 times, we make it likeso: 1 < 5 and add the next expression. IncrementExpression will be next, so for each time the statements under for are executed it will sum one to the initial expression , check for the condition, and when i is no longer less than 5 it will stop.

Control flow (cont)

	for (let i = 0; i < 5; i++;)
	for (let i = 5; i >= 1; i--;)
- While ...	while(condition){statement}
- Do ... while	Do-whiles are always executed once even if the condition is not true.
	do { sentence } while (condition)
Infinite loops	You can create them accidentally, causing a system break. Check for them on the console
- For ... in	for(let key in person){}
	For each iteration the key variable will hold the name of one of the properties of the oobject.
	To access object's values: person.name, person["name"] or person[key] if we don't know the properties name beforehand and we need to calculate it at runtime. Here, 'key' inside the brackets is the throwaway name for the properties' value. 'key' on its own will print the property name (name, age...)
- For ... of	for (let color of colors)

Control flow (cont)

	In this type of loop, the property's value is selected instead of the whole object
	Objects are not iterable, only Arrays and Maps. To force an Object into an array, use Object.keys(object) like For ... in or Object.entries(object)
Break and continue	They can be used in any kind of loop. 'break;' interrupts the code, 'continue' jumps to the beginning of the loop on its breakpoint and the next execution happens.

Arrays

Adding elements

Even on const declarations, arrays can be written onto.	const numbers = [3,4]

Javascript Fundamentals Cheat Sheet

by raposinha via cheatography.com/197915/cs/42598/

Arrays (cont)

But not like `numbers = [3,4,5];` Because Arrays are Objects, we can use their function properties, like: push (adding to the end), .unshift (pushes existing elements to the right, adds new elements to beginning), .splice (access a specific position and add or remove elements - args: start, amount of numbers to delete, items to add)	numbers.push(5,6) numbers.unshift(1,2) numbers.splice(2,0,2.5,'b') console.log(numbers) -> 1, 2, 2.5, 'b', 3, 4, 5, 6
Finding elements	
- Primitives:	numbers = [1,2,3,4,1]
.indexOf() -> numbers.indexOf(1) -> 0	Looks for a given input inside the array and, if existing, will return the index number of said array. If not, it will return -1.

Arrays (cont)

.lastIndexOf() -> numbers.lastIndexOf(1) -> 4	Looks for a given input inside the array and, if existing, will return the index number of said array. If not, it will return -1.
.includes() -> numbers.includes(3) -> true	Checks for a given element existing in the array. Returns true or false.
'fromIndex': A second argument available for all last 3 methods. It starts the search from the given index number.	console.log(apples.indexOf(1)) //0 console.log(apples.indexOf(1,1)) //4 console.log(apples.lastIndexOf(1)) //4 console.log(apples.lastIndexOf(1,1)) //0
- Reference types:	const courses = [{id: 1, name: 'a'}, {id: 2, name: 'b'}]
.find() ->courses.find(function(course){ return course.name === 'a' }) -> true	Here because the reference is allocated in another memory slot, '.includes()' can't be used.
Arrow functions	Used to call functions, pre-existant or not

Arrays (cont)

const course = courses.find(course =>{ return course.name === 'a'; })
const courseB = courses.find(course => course.name === 'a');
const courseC = courses.find((course) =>{ return course.name === 'a'; })

Functions

```
// Functions
// Performing a task:
function greet (name, lastName){
        con sol e.l og( 'Hello ' + name + ' ' + lastName + '!!!')
}
greet( " Jua na");
let lastName = "la Loca"
greet( " Jua na", las tName);
// Calcul ating a value:
function square (Number) {
        return Number * Number;
}
let n = square(2);
consol e.l og(n);
//4
consol e.l og( 4/2);
```

Javascript Fundamentals Cheat Sheet

by raposinha via cheatography.com/197915/cs/42598/

Basic concepts

Variables	Variables are data stored somewhere in memory temporarly. When adressed, the variable adress will be accesed by the variable's name. Like a box. The name will describe its content, the contents will be stored in the box.
Declaring/initializing variables (as of ES6)	**let** name = 'raposa';
	Variables cannot be reserved keywords. They should be concise and meaningful, meaning they give us a clue of the contents. They cannot start with a number. They can't contain spaces or hyphens. Camel notation should be used (firstName)). They're case sensitive. They can be declared in the same line (let name, firstName, lastName;)

Basic concepts (cont)

Constant variables	They are used when we don't want the values to ever change. If you don't want to redefine constant should be the default.
Types	There are primitive and reference types.
Primitive types:	String, number, boolean, undefined, null

Objects

Declaring an object

```
const = circle {
  radius:1,
  location : {
  x: 1,
  y : 1
  }
  draw: functi on(){ consol e.l og( 'dra
w'}
  }
```

Factory functions	Functions that create objects in order to not repeat code everytime you need a new one

Objects (cont)

```
function create Cir cle (
){
return {
        radius: radius,
        location: {
            x: x,
            y: y
        },
        draw() {
 console.log('draw')
 }
};
 }
```

Constructor functions	Written in Pascal Notation. These to generate objects.
	function Object() {} is an example function. Whenever we create an Object literal syntax, a call is made constructor function
	The keyword 'this' is used instead reference to the object executing t

Objects (cont)

	When using the 'new' operator a new empty object is created, then the properties used with 'this' are set dinamically, then the object is returned.
	function Address (street, city, zipcode) { this.street = street; this.city = city; this.zipcode = zipcode; this.showAddress = function showAddress() { for (let key in Address){ console.log(key, Address[key]) } } }
Dynamic	Objects in JavaScript are dynamic, which means that once created you can always add new properties or methods, or remove existing ones.
Functions are objects, they have constructors	Circle.constructor -> f

Objects (cont)

Circle.**call**({},1) and const circle7 = new Circle(1) are the same, **.call** is a function prebuilt metjod. `{}` stands for the first argument, an empty object - then `this` will reference the new empty object instead of the base object, window. The rest of the arguments will be passed explicitly (like -> this.radius = radius; Circle(radius);circle7({},5). Which is to mean that if the 'new' keyword isn't used, 'this' will point to window object.

The **apply** method can also be used the same as "call", but the explicit argument are parsed through an array, like Circle.apply({}, [1,2])

In JavaScript, `radius: radius`, and 'radius,` is the same when defining an object.
camelCaseNotation, PascalNotation

Objects

Cloning	for (let key in circle) another1[key] = circle[key]
	Object.assign(another2, circle)
	const another3 = Object.assign({ color: 'yellow' }, circle)
	const another4 = {...circle}

Garbage collection

In low level languages when creating an object we have to allocate memory for it then deallocate it, not with JS. This is where the Garbage Collector comes in. It finds the variables and constants that are not used and deallocate the memory

Math Object

It's a built-in Object.

The Math namespace object contains static properties and methods for mathematical constants and functions.

It's designed for mathematical calculations and so are its Properties and Functions (Math.PI, Math.floor(), ...)

Math.random(), Math.round(), Math.max(-1,2,3), Math.min(1,2,3) (...)

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math

String Objects

String is a primitive type, primitive types don't have properties and methods, only objects. But a String Object also exists for JavaScript.

const message = new String('hi');

It's typeof will be 'object'

However, the internal JavaScript engine will automatically convert a String primitive type onto a String Object if we use the dot notation

String.length, String[3], String.includes('my'), String.startsWith('a'), String.indexOf('my')

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String

Strings

\n adds a new line within a String

Template literals	With these, ``, the text formats prints as it's written

Date Object (Built-in)

const now = new Date()	Creates the current date and time when object is created

Has get, set methods

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date -> to check formats, methods