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); //
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:
```

Ternary operators (cont)

> 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);
consol e.log(x ** y);
//// Increment and Decrement
Operators
// 10
consol e.l og(x);
// 11+1 (operation applied
first)
consol e.l og( ++x);
// 11+1 (operation applied
later)
consol e.l og( 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
consol e.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	console.log(-person);
	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

Reference types (cont)

	Because JavaScript is a dynamic language, variables can be set, added, deleted at runtime or any time. And they can be of any type	selectedColors[2] = 'yellow';
		selectedColors[3] = 8;
	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",lastName);

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.
	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.

Control flow (cont)

- 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.
	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){}

Control flow (cont)

	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)
	In this type of loop, the property's value is selected instead of the whole 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.

Functions

```
// Functions
// Performing a task:
function greet (name, lastName){
        con sol e.l og( 'Hello '
+ name + ' ' + lastName + '!!!')
}
greet( " Jua na");
```

Functions (cont)

```
> let lastName = "la Loca"
greet("Juana",lastName);
// Calculating a value:
function square (Number) {
   return Number * Number;
}
let n = square(2);
console.log(n);
//4
console.log(4/2);
```

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';

Basic concepts (cont)

	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;)
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