

P.PORTO

Introduction to Javascript

Programming in Web Environment

Index

Overview

The usefulness of Javascript

Javascript

ECMAScript6

Good Practices

Overview

Initially implemented only in Web browsers as client side

Currently also used on the server side

It is an interpreted programming language

Behaves like scripting languages, performs operations in the order in which they appear, and does not have a main-type function

Overview

JavaScript is a multi-paradigm language

Supports programming styles:

- event-oriented
- oriented to objects
- functional
- imperative

Has APIs to work with text, matrices, dates, regular expressions

Overview

JavaScript is not java!

- Java was developed at Sun Microsystems
- JavaScript was developed in Netscape

JavaScript programs run on a runtime that may be:

- in a browser (SpiderMonkey, V8, Chakra)
- installed as an application on the server or desktop (NodeJS)

Overview

A "client-side" language"

- Used to give "interactivity" to web pages
- Insert dynamic text in HTML
- React to events
- Does not include any I/O, storage, or graphics, delegating these actions to the host environment in which it is embedded
- Gets information about the user's computer

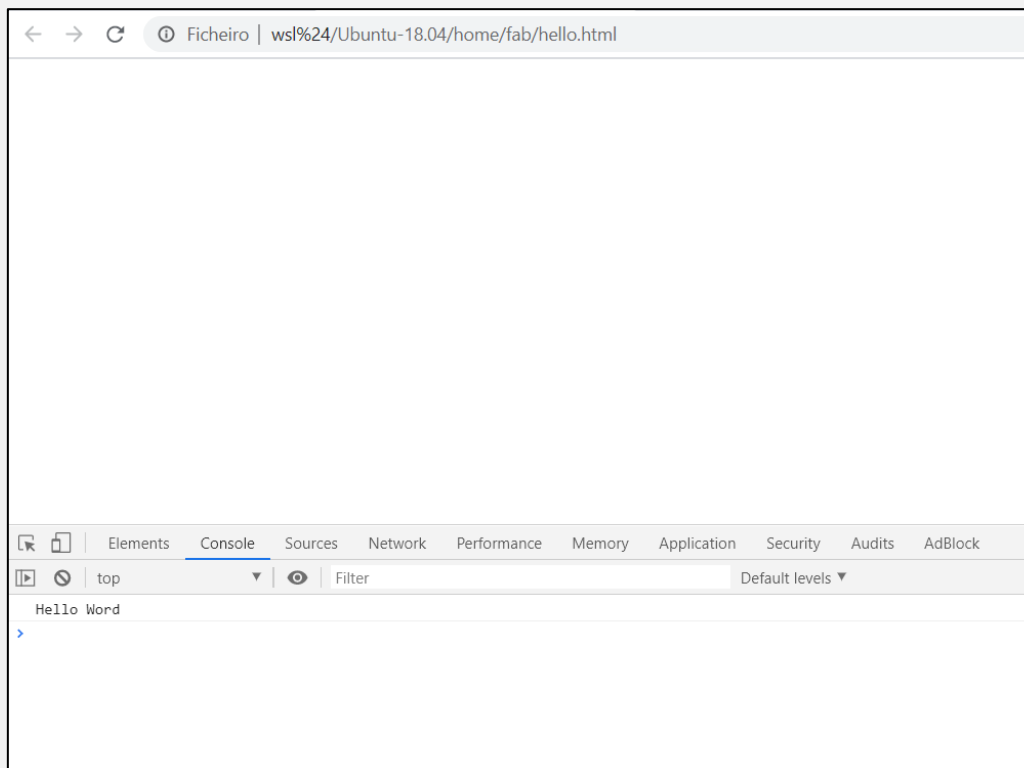
Overview

Runs inside an HTML file within the script tag

```
hello.html > ...  
1  <!DOCTYPE html>  
2  <head></head>  
3  <body>  
4      <script>  
5          console.log("Hello Word");  
6      </script>  
7  </body>  
10
```

Overview

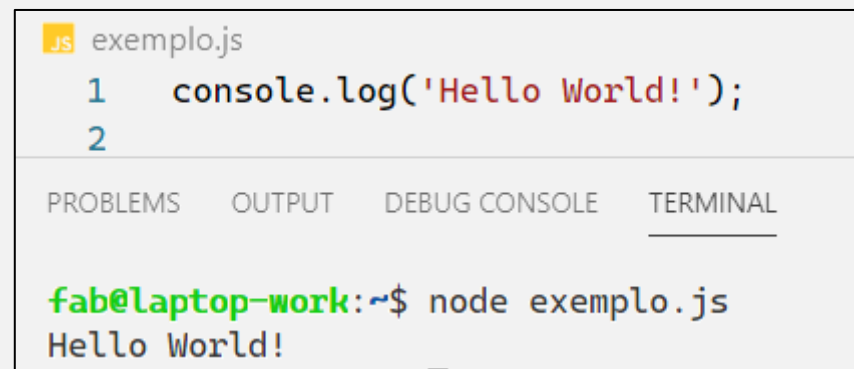
Runs inside an HTML file within the script tag



Overview

A "server-side" language

- NodeJS runtime on a server or Desktop
- do not have access to unique browser objects
- files with the extension *.js and *.jsh
- run the file on a terminal with the:
 - node exemplo.js

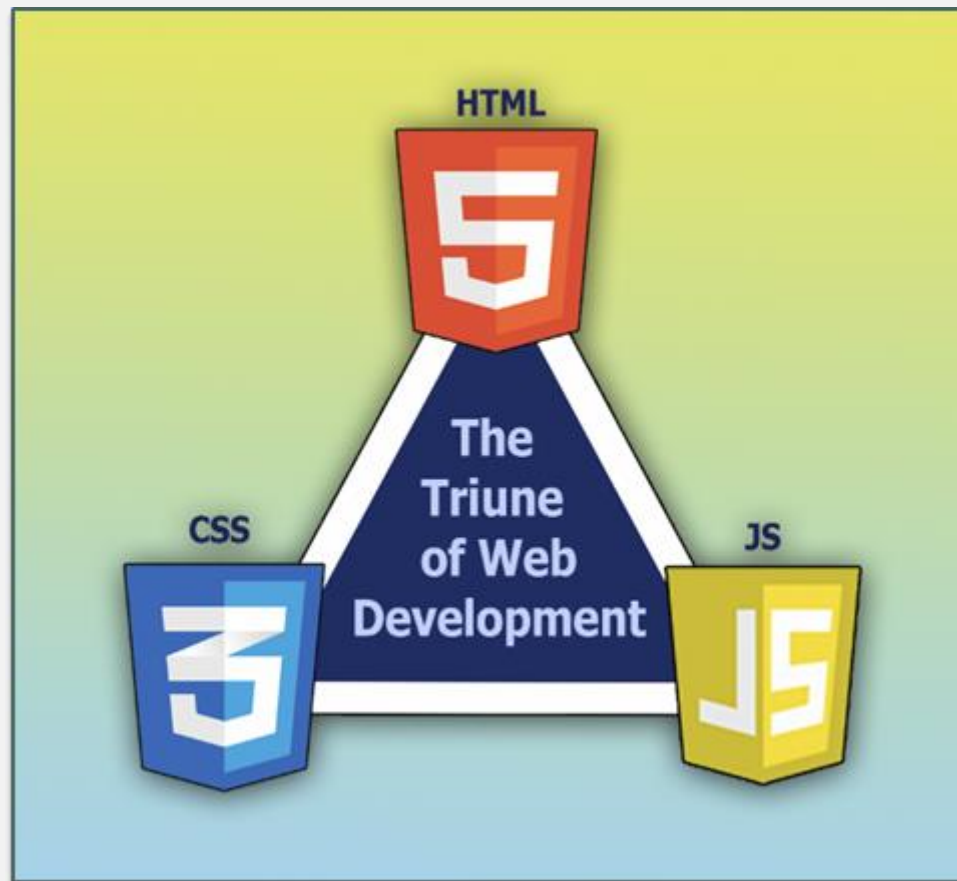


```
js exemplo.js
1 console.log('Hello World!');
2
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
fab@laptop-work:~$ node exemplo.js
Hello World!
```

The usefulness of JavaScript



The usefulness of JavaScript

Detect and react to user-initiated events

Improve navigation on a web page with helps, automatic scroll, messages and alerts, dynamic images, etc.

Control the appearance of the web page dynamically

The usefulness of JavaScript

If the user has plug-ins installed

Validate what the user typed in a form before sending it to the server

Check valid email addresses, social security numbers, credit card details, etc.

The usefulness of JavaScript

Aritmetic calculations, manipulation of dates and times, and works with arrays, strings, and objects

Manipulate animations and produce web-based applications and games

Other...

The usefulness of JavaScript



The screenshot shows the Node.js website with a dark navigation bar at the top containing the Node.js logo and links for HOME, ABOUT, DOWNLOADS, DOCS, GET INVOLVED, SECURITY, and NEWS. Below the navigation bar, a white section states that Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. A green banner highlights that new security releases are now available for all release lines. The main content area is titled 'Download for Windows (x64)' and features two green buttons: '12.16.1 LTS Recommended For Most Users' and '13.9.0 Current Latest Features'. At the bottom, there are links for 'Other Downloads', 'Changelog', and 'API Docs' for both versions, followed by a link to the 'Long Term Support (LTS) schedule'.

node

HOME | ABOUT | DOWNLOADS | DOCS | GET INVOLVED | SECURITY | NEWS

Node.js® is a JavaScript runtime built on [Chrome's V8 JavaScript engine](#).

New security releases now available for all release lines

Download for Windows (x64)

12.16.1 LTS
Recommended For Most Users

13.9.0 Current
Latest Features

[Other Downloads](#) | [Changelog](#) | [API Docs](#) [Other Downloads](#) | [Changelog](#) | [API Docs](#)

Or have a look at the [Long Term Support \(LTS\) schedule](#).

The usefulness of JavaScript

Run Scripts on Server/Desktop

Develop web applications using web frameworks

Interact with the operating system of the device where it is installed

Other...

JavaScript > Variables

Variables are created by declaring them. They are assigned values using the JavaScript assignment operator, '='

Assigning a name to variables in JavaScript must take into account the rules for naming names as well as considering the significance of the name.

To declare a variable, you must use the var or let keyword. You do not specify the type of data, it is inferred by the javascript runtime:

```
var variableName;
```


JavaScript > Variables

To declare the numberOfOranges variable:

```
var numberOfOranges;
```

In this example, there is a new variable named numberOfOranges. The semicolon ends the declaration. The numberOfOranges variable does not yet have an assigned value

JavaScript variables are case sensitive

numberOfOranges, numeroforanges, NUMBERoFoRANGES
and NumberOfOranges are four different variables

JavaScript > Variables

To assign a value to a variable, use the JavaScript assignment operator, which is the symbol equal to (=). If you want to declare a variable and assign a value to it on the same line, use this format:

```
var variableName = variávelValor;
```

For example, to name your variable `numberOfOranges` and provide the numeric value 18, use this statement:

```
var numberOfOranges = 18;
```

JavaScript > Variables

You can declare/assign multiple variables at the same time

To assign a value to a variable, use the JavaScript assignment operator '='

No assigned values:

```
var variableName1, variableName2, variableName3;
```

With assigned values:

```
var variableName1 = 10;  
var variableName2 = "doc", variableName3 = 3.14159;
```

JavaScript > Variables

Important rules to highlight:

- a variable name must begin with a letter or a character '_' or a character '\$'. The variable name cannot begin with a number or any character other than a letter (in addition to the underscore).
- Reserved words:

abstract	arguments	await	boolean
break	byte	case	catch
char	class	const	continue
debugger	default	delete	do
double	else	enum	eval
export	extends	false	final
finally	float	for	function
goto	if	implements	import
in	instanceof	int	interface
let	long	native	new
null	package	private	protected
public	return	short	static
super	switch	synchronized	this
throw	throws	transient	true
try	typeof	var	void
volatile	while	with	yield

JavaScript > Data Types

Numbers

JavaScript does not require numbers to be declared as integers, floating-point (decimal) numbers, or any other type

The number will remain of the same type until the mathematical operation forces the javascript to unless you do an operation that forces you to change the number type to consider

For example, if we use an integer in a variable, it will have no decimal places unless you perform an operation that results in a decimal number (e.g. dividing unevenly).

JavaScript > Data Types

String

Strings are variables that represent a string of text. The string can contain letters, words, spaces, numbers, symbols, etc.

Strings are defined as follows:

```
var variableName = "stringText";
```

In JavaScript, strings are defined by placing them in quotation marks

JavaScript allows you to use double quotes " " or single quotes ' ' to set the value of the string.

JavaScript > Data Types

- **String**
 - **Methods:** `charAt`, `charCodeAt`, `fromCharCode`, `indexOf`, `lastIndexOf`, `replace`, `split`, `substring`, `toLowerCase`, `toUpperCase`

```
var s = "Connie Client";  
var fName = s.substring(0, s.indexOf(" ")); // "Connie"  
var len = s.length; // 13
```

JavaScript > Data Types

Array

Arrays are used to store sets of values in a single variable

We can access the values through the index

Create an array:

Syntax

```
var array_name = [item1, item2, ...];  
array_name[0] = new_item;
```

or

```
var array_name = new Array(item1, item2, ...);  
array_name[0] = new_item;
```


JavaScript > Data Types

Array

Arrays are a special type of objects. The `typeof` operator in JavaScript returns "object" to arrays.

Arrays use numbers to access their "elements". In this example, `person[0]` returns John:

```
var person = ["John", "Doe", 46];
```

Objects use names to access their "members." In this example, `person.firstName` or `person['firstName']` return John

```
var person = {firstName:"John", lastName:"Doe", age:46};
```

JavaScript > Data Types

Array

Methods: concat, join, pop, push, reverse, shift, slice, sort, splice, toString, unshift

```
var a = ["Stef", "Jason"]; // Stef, Jason
a.push("Brian"); // Stef, Jason, Brian
a.unshift("Kelly"); // Kelly, Stef, Jason, Brian
a.pop(); // Kelly, Stef, Jason
a.shift(); // Stef, Jason
a.sort(); // Jason, Stef
```

JavaScript > Data Types

Boolean

```
var RexCodes = true;  
  
var RexIsNotCool = false;
```

A Boolean variable is one that has the value of true or false

The words true and false do not need to be quoted

JavaScript recognizes as Boolean values

We can also use the number 1 for true and the number 0 for false

JavaScript > Data Types

NULL

Null means that the variable has no value. It is not a space, nor is it a zero; it's just nothing.

To define a variable with a value null:

```
var variableName = null;
```

Like Boolean variables, you don't have to put this value in quotation marks

JavaScript > Data Types

Undefined

A variable that has not yet received a value

```
var myVar;  
console.log(myVar); // undefined  
  
myVar = 200;  
console.log(myVar); // 200
```

JavaScript > Comparisons

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
&&	logical and
	logical or
!	logical not

JavaScript > Assignments

There is a whole set of assignment operators.

Unlike the equal sign, the other assignment operators serve as shortcuts to modify the value of variables

For example, a shorter way to say `x = x + 5` is to say `x += 5`

Operator	Example	Same As
<code>=</code>	<code>x = y</code>	<code>x = y</code>
<code>+=</code>	<code>x += y</code>	<code>x = x + y</code>
<code>-=</code>	<code>x -= y</code>	<code>x = x - y</code>
<code>*=</code>	<code>x *= y</code>	<code>x = x * y</code>
<code>/=</code>	<code>x /= y</code>	<code>x = x / y</code>
<code>%=</code>	<code>x %= y</code>	<code>x = x % y</code>
<code><<=</code>	<code>x <<= y</code>	<code>x = x << y</code>
<code>>>=</code>	<code>x >>= y</code>	<code>x = x >> y</code>
<code>>>>=</code>	<code>x >>>= y</code>	<code>x = x >>> y</code>
<code>&=</code>	<code>x &= y</code>	<code>x = x & y</code>
<code>^=</code>	<code>x ^= y</code>	<code>x = x ^ y</code>
<code> =</code>	<code>x = y</code>	<code>x = x y</code>
<code>**=</code>	<code>x **= y</code>	<code>x = x ** y</code>

JavaScript > Conditionals

Conditional code: runs code if something is true

```
if (condicional) {...} else {...}
```

Another way to "affirm the same" is as follows:

```
(condicional) ? {...} : {...}
```


JavaScript > Cycles

For cycle

```
var sum = 0;  
for (var i = 0; i < 100; i++) {  
    sum = sum + i;  
}
```

```
var s1 = "hello";  
var s2 = "";  
for (var i = 0; i < s1.length; i++) {  
    s2 += s1.charAt(i) + s1.charAt(i);  
}
```

JavaScript > Cycles

While cycle

```
while (condição) {  
    statements;  
}
```

```
do {  
    statements;  
} while (condição);
```

JavaScript > Comments

As in all programming languages, comments are an important part of the programming process.

Comments on a line:

```
// This is a comment on a line  
console.log("Hello"); // comment after the code
```

Comentários em bloco (multi-linha):

```
/* block comment  
    and on multiple lines */
```

JavaScript > Spaces and Newline

White spaces and Newlines

Most white spaces, such as spaces, tabs, and empty lines, are ignored in JavaScript and generally only help with readability.

In production code, all non-essential blanks are usually removed so that javascript files are downloaded more quickly.

JavaScript > Math object

O objeto Math

Methods: `abs`, `ceil`, `cos`, `floor`, `log`, `max`, `min`, `pow`,
`random`, `round`, `sin`, `sqrt`, `tan`

Properties: `E`, `PI`

```
var rand1to10 = Math.floor(Math.random() * 10 + 1);  
var three = Math.floor(Math.PI);
```

JavaScript > Hoisting

“Hoisting” is javascript's default behavior that "moves" the declaration of variables to the top

Even if a variable is "hoisted", its initialization will not be

In JavaScript, a variable can be used before it has been declared

```
function letTest(){  
  let x = 31  
  if (true){  
    let x = 71 // different variable  
    console.log(x) // 71  
  }  
  console.log(x) // 31  
}
```

JavaScript > Hoisting

In order to avoid errors, we must declare all variables before the start of each "scope"

- The scope of a variable declared with "var" is that of the function, if declared outside any function is global
- With ECMAScript 6, the scope of a variable declared with "let" is at the "block" level within a function

```
function letTest(){  
  let x = 31  
  if (true){  
    let x = 71 // different variable  
    console.log(x) // 71  
  }  
  console.log(x) // 31  
}
```

JavaScript > Functions

A function consists of the function word followed by the function name, arguments, and function body

Example:

```
function sayWhat() {  
    alert("JavaScript is good!");  
}
```


JavaScript > Functions

When defining a function in JavaScript, you do not specify the data type of the parameters

A function in JS does not perform type checking on the arguments that are passed to it

A function in JS does not check the number of arguments received is expected

JavaScript > Functions

If a function is invoked with no arguments (less than declared), the missing values are set to "undefined"

If a function is invoked with more arguments (more than those that were declared), these arguments can be accessed by using the object of the arguments

Functions in JS are "methods of an object"

JavaScript > Functions

In JS arguments of primitive types are passed to functions by value

Objects are passed to functions by reference

There are two ways to declare a function in JS:

- By "declaration" (alert: it is "hoisted", i.e. "pulled up")

```
//Function Declaration  
function square(x){  
    return x*x;  
}
```

- By "expression"

```
//Function Expression  
const square = function (x){  
    return x*x;  
}
```

JavaScript > Functions

Alert: JavaScript is case sensitive!

- When declaring a variable or function, we have to pay attention to its upper and lower case letters. `myFunction` is not the same thing as `MyFunction` or `myFUNCTION` or `myfunction`
- We must access internal objects with the appropriate encapsulation. `Math` and `Data` start with capital letters, but not with `window` and `document` in browsers not
- Most internal methods are combined words, capitalizing on all but the first. Example: `getElementById` (usually called `camelCase`)

JavaScript > Errors

The **try** statement allows you to test a block of code for errors.

The **catch** statement allows you to handle the error.

The **throw** statement allows you to create custom errors.

The **finally** statement allows you to execute the code after you try and capture, regardless of the result.

```
try{  
    // Code  
}  
catch(err){  
    // Code  
}
```

```
try{  
    // Code  
}  
catch(err){  
    // Code  
}  
finally{  
    // Code  
}
```

JavaScript > Errors

Example

```
function myFunction(x,message) {|
  try {
    if(x == "") throw "is empty";
    if(isNaN(x)) throw "is not a number";
    x = Number(x);
    if(x > 10) throw "is too high";
    if(x < 5) throw "is too low";
  }
  catch(err) {
    message.innerHTML = "Input " + err;
  }
  finally {
    document.getElementById("demo").value = "";
  }
}
```

JavaScript > Scope

There are two types of scope :

- local scope
- global scope

Each function has its own scope

Scope determines the accessibility (visibility) of variables

Variables defined within a function are not accessible (visible) from outside the function

JavaScript > Scope

Variables declared within a JavaScript function become locations for the function.

A variable declared outside of a function becomes GLOBAL.

```
// code here can NOT use carName

function myFunction() {
  var carName = "Volvo";

  // code here CAN use carName
}
```

```
var carName = "Volvo";

// code here can use carName

function myFunction() {

  // code here can also use carName
}
```


JavaScript > Strict Mode

We can run javascript in strict mode

It aims to force the declaration and assignment of values to variables

If the restriction is not met, an error is posted

```
"use strict";  
x = 3.14; // This will cause an error because x is not declared
```

```
x = 3.14; // This will not cause an error.  
myFunction();  
  
function myFunction() {  
  "use strict";  
  y = 3.14; // This will cause an error  
}
```

JavaScript > Objects

Anyone who understands the concept of "object" understands JavaScript!

In JS, almost everything is an object

Booleans, Numbers, Strings can be objects (if defined through the reserved word "new")

Dates and Maths are always objects

Arrays and functions are always objects

All values, except primitives, are objects

A primitive value is a value that has no properties and methods

String, Number, boolean, null, undefined

Objects are variables that contain variables

JavaScript > Objects

Variables can contain simple values:

```
var person = "John Doe";
```

Objects are also variable, but contain values as key pairs: value

A method of an object is a property of the object containing a function definition:

```
let car = {  
  brand: 'Smart',  
  model: 'forTwo',  
  year: 2013,  
  engine: {  
    horsepower: 45,  
    fuel: "diesel"  
  },  
  getAge: function() {  
    return (new Date()).getFullYear() - this.year;  
  }  
}
```

JavaScript > Objects

Creating an object:

- Using a literal object

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

- Using the reserved word "new"

```
var person = new Object();  
person.firstName = "John";  
person.lastName = "Doe";  
person.age = 50;  
person.eyeColor = "blue";
```

- Objects are mutable
 - objects are treated by reference, not by value

JavaScript > Objects

Properties are the most important part of any object

Properties are the values associated with the object

A javascript object can be seen as an unordered collection of properties

Access to the properties of an object:

```
person.firstname + " is " + person.age + " years old.";
person["firstname"] + " is " + person["age"] + " years old.";
```

JavaScript > Objects

Using a "for" cycle

```
var person = {fname:"John", lname:"Doe", age:25};  
  
for (x in person) {  
    txt += person[x];  
}
```

Object constructors

```
function Person(first, last, age, eye) {  
    this.firstName = first;  
    this.lastName = last;  
    this.age = age;  
    this.eyeColor = eye;  
}
```

```
var myFather = new Person("John", "Doe", 50, "blue");  
var myMother = new Person("Sally", "Rally", 48,  
    "green");
```

JavaScript > Prototype Objects

All objects "inherit" properties and methods from a prototype

Previously, it was shown how to use a constructor of an object and we cannot add a new property to an existing constructor

Example: "Date" objects inherit from "Date.prototype", "Array" objects inherit from "Array.prototype", "Person" object inherits from "Person.prototype"

However, the javascript prototype property allows you to add new properties to a constructor of an object

```
function Person(first, last, age, eyecolor) {  
  this.firstName = first;  
  this.lastName = last;  
  this.age = age;  
  this.eyeColor = eyecolor;  
}  
Person.prototype.nationality = "English";
```

```
function Person(first, last, age, eyecolor) {  
  this.firstName = first;  
  this.lastName = last;  
  this.age = age;  
  this.eyeColor = eyecolor;  
}  
Person.prototype.name = function() {  
  return this.firstName + " " + this.lastName;  
};
```

JavaScript > Classes

Classes in javascript are essentially "syntactic sugar" on the concept of "prototype"-based inheritance

An important difference between the function declaration and the class declaration is that the function declarations are "hoisted" and those of the class are not

It is necessary to first declare the class and then access it, otherwise the code will trigger a "ReferenceError"

```
var car1 = new Car(); // ReferenceError  
  
class Car {}
```


JavaScript > Classes

As in functions, a class can also be defined by an expression

A class expression is another way to define a class that may have a name or not

The name given a "class expression" is local to the class body.

```
//or (unnamed)
var Moto = class {
  constructor(brand, model, year){
    this.brand = brand;
    this.model = model;
    this.year = year;
  }
};

//or (named)
var bicycle = class bicycle{
  constructor(brand, model, year){
    this.brand = brand;
    this.model = model;
    this.year = year;
  }
}
```

JavaScript > Classes

The constructor method is a special method for creating and initializing an object created with the class

A constructor can use the reserved word "super" to evoke the constructor of a parent class

```
class Car {  
  constructor(brand, model, year){  
    this.brand = brand;  
    this.model = model;  
    this.year = year;  
  }  
  
  //Getter  
  get age(){  
    return this.calcAge()  
  }  
  
  calcAge(){  
    return (new Date()).getFullYear() - this.year;  
  }  
}  
  
const car1 = new Car('Smart', 'forTwo', 2013);  
document.write('The car: ${car1.year} has ${car1.age} years old <br />');
```

JavaScript > Classes

The word "static" defines the static method for a class

Static methods are invoked without being instantiated by your class and cannot be invoked by an instance of your class

Static methods are commonly used to create utilitarian functions for an application

The word "extend" is used in class declarations or expressions to create a class as "child" of another class through inheritance

JavaScript > Debugger

The word debugger stops javascript execution and performs (if available) the debug function.

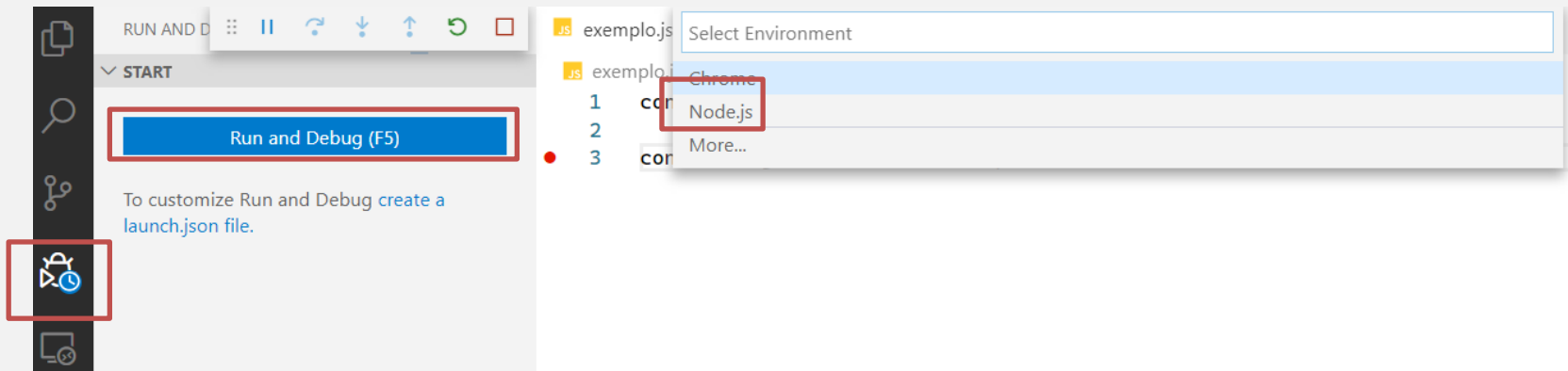
It has the same function as setting a breakpoint on the debugger.

If no breakpoint is available, the debugger statement will have no effect.

```
var x = 15 * 5;  
debugger;  
document.getElementById("demo").innerHTML = x;
```

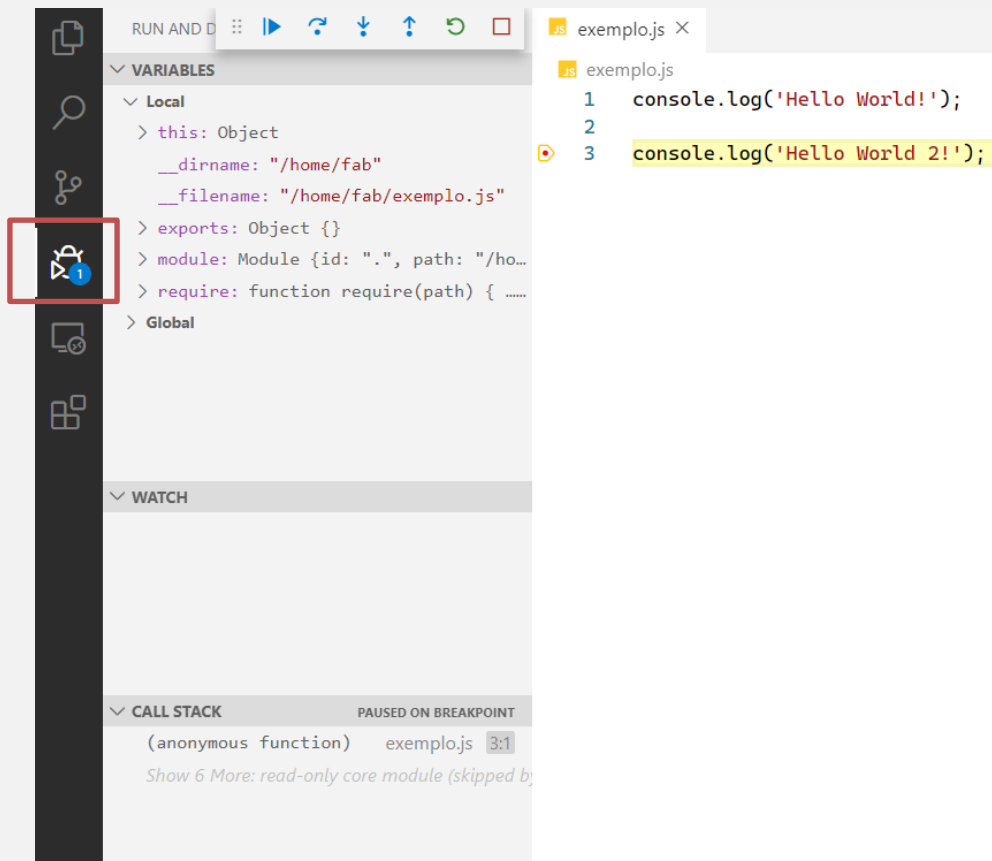
JavaScript > Debbuger

Using the NodeJS and VSCode runtime



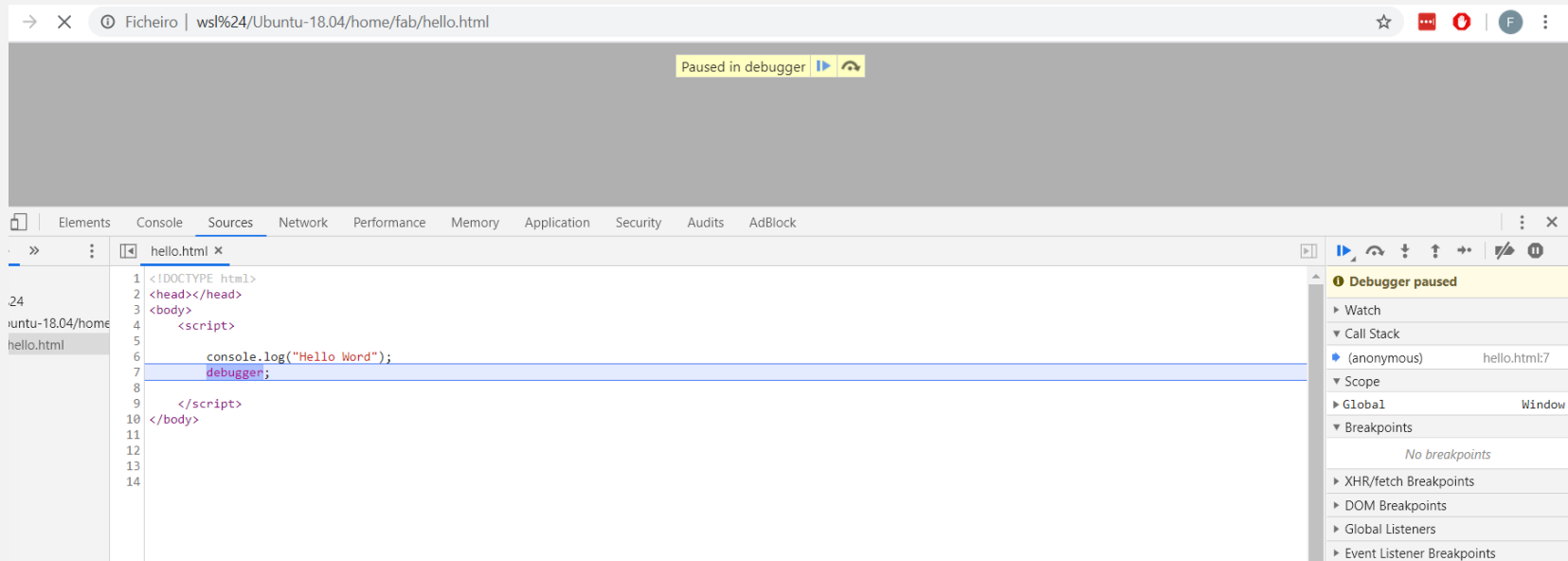
JavaScript > Debbuger

Using the NodeJS and VSCode runtime



JavaScript > Debbuger

Using the javascript runtime in the browser



ECMAScript6

The JavaScript language currently contains several incremental versions

ECMAScript 6 adds new functionality to the Javascript language, but does not break backward support

Modern browsers already contain support for JavaScript ECMAScript 6 but older versions do not

ECMAScript6

Support for constants (also known as "immutable variables")

```
const PI = 3.141593
```

Variableblock-scoped without hoisting (using "let")

```
for (let i = 0; i < a.length; i++) {  
  let x = a[i]  
  ...  
}  
for (let i = 0; i < b.length; i++) {  
  let y = b[i]  
  ...  
}
```

```
var x = 10;  
// Here x is 10  
{  
  let x = 2;  
  // Here x is 2  
}  
// Here x is 10
```

ECMAScript6

Arrow functions

```
// ES6  
const f = (x, y) => x * y;  
  
// ES5  
var f = function(x, y) {  
    return x * y;  
}
```

Default parameters in functions

```
function f (x, y = 7, z = 42) {  
    return x + y + z  
}  
f(1) === 50
```

ECMAScript6

Template Strings

```
// ES6
var customer = { name: "Foo" }
var card = { amount: 7, product: "Bar", unitprice: 42 }
var message = `Hello ${customer.name},
want to buy ${card.amount} ${card.product} for
a total of ${card.amount * card.unitprice} bucks?`

// ES5
var customer = { name: "Foo" };
var card = { amount: 7, product: "Bar", unitprice: 42 };
var message = "Hello " + customer.name + ",\n" +
"want to buy " + card.amount + " " + card.product + " for\n" +
"a total of " + (card.amount * card.unitprice) + " bucks?";
```

Improved object declaration

```
// ES6
var x = 0, y = 0
obj = { x, y }

//ES5
var x = 0, y = 0;
obj = { x: x, y: y };
```

ECMAScript6

Modules Support

```
// lib/math.js
export function sum (x, y) { return x + y }
export var pi = 3.141593

// someApp.js
import * as math from "lib/math"
console.log("2π = " + math.sum(math.pi, math.pi))

// otherApp.js
import { sum, pi } from "lib/math"
console.log("2π = " + sum(pi, pi))
```

More intuitive POO

```
class Shape {
  constructor (id, x, y) {
    this.id = id
    this.move(x, y)
  }
  move (x, y) {
    this.x = x
    this.y = y
  }
}
```

```
class Rectangle extends Shape {
  constructor (id, x, y, width, height) {
    super(id, x, y)
    this.width = width
    this.height = height
  }
}
class Circle extends Shape {
  constructor (id, x, y, radius) {
    super(id, x, y)
    this.radius = radius
  }
}
```

ECMAScript6

More intuitive POO (continued)

```
class Rectangle extends Shape {  
  ...  
  static defaultRectangle () {  
    return new Rectangle("default", 0, 0, 100, 100)  
  }  
}  
class Circle extends Shape {  
  ...  
  static defaultCircle () {  
    return new Circle("default", 0, 0, 100)  
  }  
}  
var defRectangle = Rectangle.defaultRectangle()  
var defCircle    = Circle.defaultCircle()
```

```
class Rectangle {  
  constructor (width, height) {  
    this._width  = width  
    this._height = height  
  }  
  set width  (width)  { this._width = width }  
  get width  ()       { return this._width }  
  set height (height) { this._height = height }  
  get height ()       { return this._height }  
  get area   ()       { return this._width * this._height }  
}  
var r = new Rectangle(50, 20)  
r.area === 1000
```

ECMAScript6

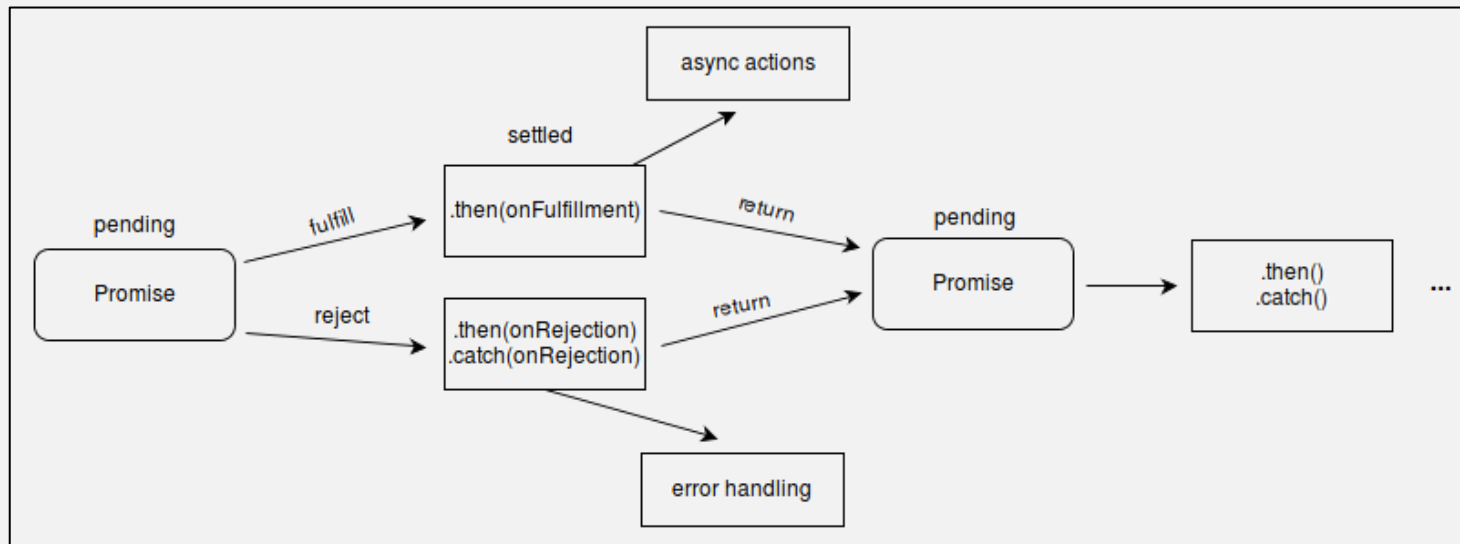
Promises

- A Promise represents a proxy for a value that is not necessarily known when the promise is created
- Allows the association of methods for handling asynchronous events in case of success and failure
- Allows asynchronous methods to return values as synchronous methods

ECMAScript6

A Promise is in one of these states:

- pending: Initial state, which has not been carried out or rejected
- fulfilled (performed): success in the operation
- rejected: operation fails
- settled :Which was held or rejected



ECMAScript6

Example

```
var promise = new Promise(function(resolve, reject) {  
  // do a thing, possibly async, then...  
  
  if (/* everything turned out fine */) {  
    resolve("Stuff worked!");  
  }  
  else {  
    reject(Error("It broke"));  
  }  
});  
promise.then(function(result) {  
  console.log(result); // "Stuff worked!"  
}, function(err) {  
  console.log(err); // Error: "It broke"  
});
```


Good Practices

Avoid global varsis

Always declare local varsis

Variable declarations at the top

Initialize variables

```
// Declare at the beginning
var firstName, lastName, price, discount, fullPrice;

// Use later
firstName = "John";
lastName = "Doe";

price = 19.90;
discount = 0.10;

fullPrice = price * 100 / discount;
```

```
// Declare and initiate at the beginning
var firstName = "",
    lastName = "",
    price = 0,
    discount = 0,
    fullPrice = 0,
    myArray = [],
    myObject = {};
```

Good Practices

Do not declare objects of type String, Boolean, Number, etc..

```
var x = "John";  
var y = new String("John");  
(x === y) // is false because x is a string and y is an object.
```

```
var x = new String("John");  
var y = new String("John");  
(x == y) // is false because you cannot compare objects.
```

Be careful with automatic conversion of variables

Good Practices

Do not use new Object()

- Use {} instead of new Object()
- Use "" instead of new String()
- Use 0 instead of new Number()
- Use false instead of new Boolean()
- Use [] instead of new Array()
- Use /(())/ instead of new RegExp()
- Use function (){} instead of new Function()

```
var x1 = {};  
var x2 = "";  
var x3 = 0;  
var x4 = false;  
var x5 = [];  
var x6 = /(())/;  
var x7 = function(){};
```

Good Practices

Use default parameters in javascript functions

```
function myFunction(x, y) {  
  if (y === undefined) {  
    y = 0;  
  }  
  return x * y;  
}  
document.getElementById("demo").innerHTML = myFunction(4);
```

References

Tutorial Javascript

- [https://developer.mozilla.org/en-US/docs/Learn/Getting started with the web/JavaScript basics](https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/JavaScript_basics)
- <https://www.codecademy.com/learn/introduction-to-javascript>
- <https://www.tutorialspoint.com/es6/index.htm>

Javascript language specification

- <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference>

P.PORTO

Introduction to Javascript

Programming in Web Environment