Unit 3

OBJECTS AND CLASSES IN JAVASCRIPT

DESARROLLO WEB EN ENTORNO CLIENTE 23/24

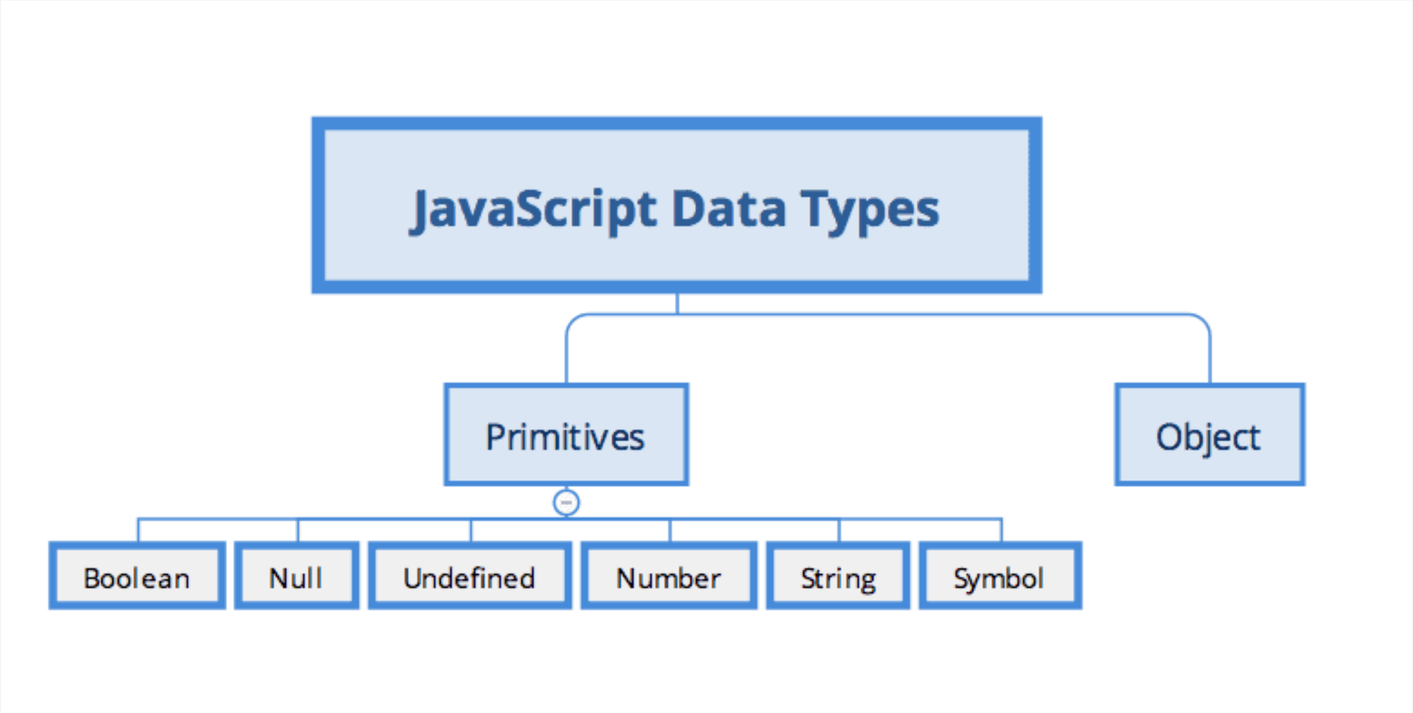
ALEJANDRO VIANA RÍOS

Unit 3

Objects and classes in JavaScript

Index

1.- JavaScript global objects



1. Number
2. String
3. Object
4. Date
5. Array
6. Date
7. JSON

1.- Number

Numbers conversion

1.- Objects > Numbers

script.js

```
1   "use strict";
2
3   let decimal=10;
4   let a=(decimal.toString()); //first way
5   let b=String(decimal);    //second way
6   console.log (typeof(a), typeof(b), a==b);
7
8   console.log (typeof(10..toString()));   //a number can be converted to string
9
10  let nulo=null;
11  console.log (String(nulo));
12  console.log (nulo.toString());  //it fails
```

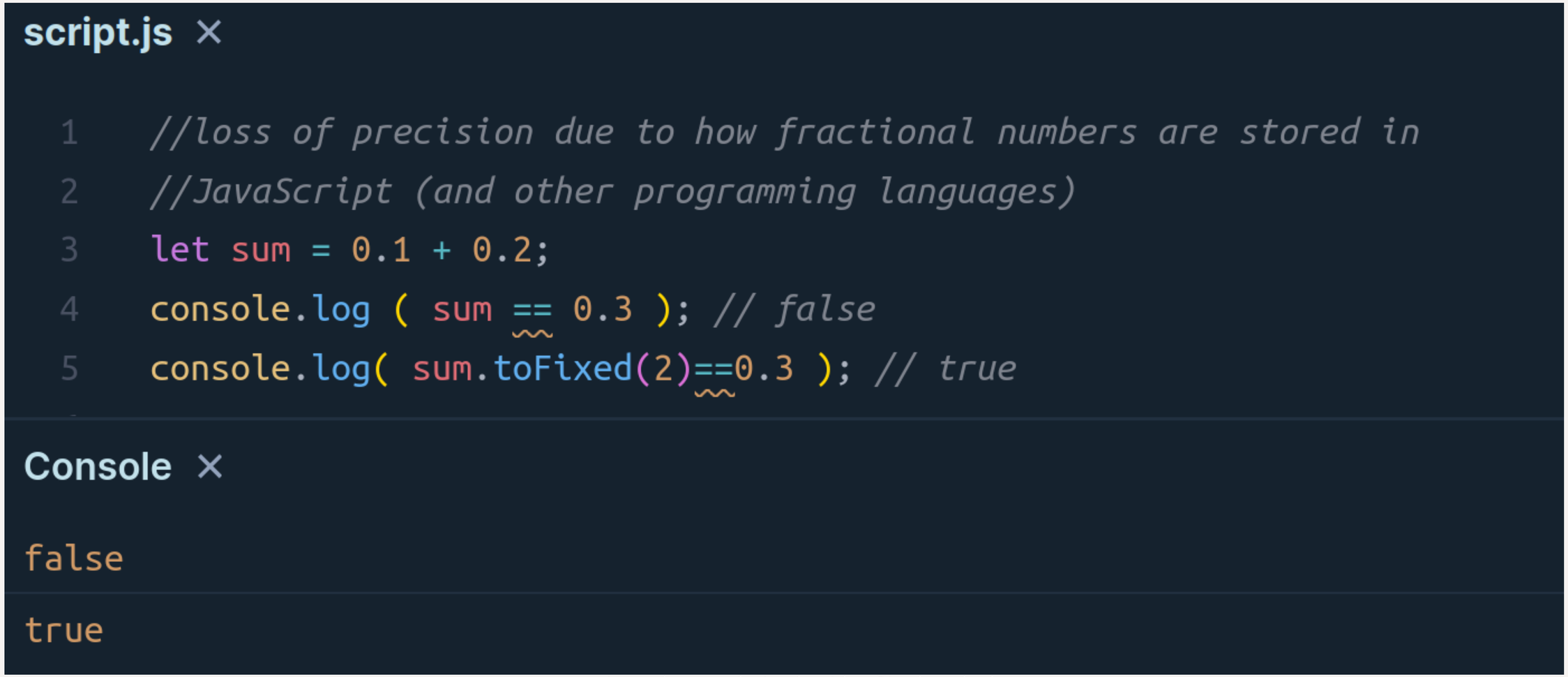
Console

```
string  string  true
string
null
▶ TypeError: nulo is null
  @https://preview-javascript.playcode.io/ line 19 > injectedScript:67:1
  mn@https://preview-javascript.playcode.io/ line 6 > injectedScript:19:5455
```

Converting numbers to strings with String function and toString method

Numbers loss of precision

1.- Objects > Numbers



getting only some decimals with toFixed method

Numbers loss of precision

1.- Objects > Numbers

script.js

```
1   //Imprecise calculations
2   //Javascript uses IEEE754 DP to storage real numbers: 1 bit for sign, 52 for number and 11 for exponent
3   //some numbers don't have enough bits while others have infinite decimals
4
5   console.log (1e500); //not enough room for storing such a big number. Returns Infinity
6
7   let sum = 0.1 + 0.2;  //0.1 and 0.2 can't be stored precisely
8   console.log ( sum == 0.3 ); // false
9   console.log( sum.toFixed(2)==0.3 ); // true
10
11  console.log(9999999999999999);  //returns 10000000000000000
12
```

Console

```
Infinity
false
true
10000000000000000
```

Some numbers are impossible of represent while others suffer of imprecision

Numbers methods in Math

1.- Objects > Numbers

script.js ×

```
1   //MATH is an object for number type, that has methods and properties related with mathematics
2   let real=3.193;
3   let real2=3.6;
4   //floor roudns down, ceil rounds up,
5   //round rounds to the next integer and trunc removes anything beyond decimal point
6   console.log (Math.floor(real), Math.ceil(real), Math.round(real), Math.trunc(real));
7   console.log (Math.floor(real2), Math.ceil(real2), Math.round(real2), Math.trunc(real2));
8
```

Console ×

```
3  4  3  3
3  4  4  3
```

Usage of methods included in Math object

Numbers isNaN function

1.- Objects > Numbers

script.js

```
"use strict";
let decimal=15;
//isNaN converts argument to a number and returns true if the argument is not a number
console.log (isNaN(decimal),
             isNaN(Infinity),
             isNaN(NaN),
             isNaN(null),
             isNaN("Pepito piscinas"));

//Math.isNaN is a more strict version of isNaN(argument).
//It checks if its argument belongs to number, but it doesn't convert to number
console.log (Number.isNaN(decimal),
             Number.isNaN(Infinity),
             Number.isNaN(NaN),
             Number.isNaN("null"),
             Number.isNaN("Pepito piscinas"));
```

Console

```
false  false  true  false  true
false  false  true  false  false
```

Usage of isNaN and Number.isNaN

Numbers isFinite function

1.- Objects > Numbers

script.js

```
"use strict";
let decimal=15;
//isFinite converts to number and returns true if it's a regular number
console.log(isFinite(decimal),
            isFinite(Infinity),
            isFinite(NaN),
            isFinite(null),
            isFinite("Pepito piscinas"));

//Math.isFinite is a more strict version of isFinite(argument).
//It checks if its argument belongs to number, but it doesn't convert to number
console.log(Number.isFinite(decimal),
            Number.isFinite(Infinity),
            Number.isFinite(NaN),
            Number.isFinite(null),
            Number.isFinite("Pepito piscinas"));
```

Console

```
true  false  false  true  false
true  false  false  false  false
```

Usage of isFinite and Number.isFinite

Numbers functions

1.- Objects > Numbers

```
//returns a number between 0 and 1 (not included)
console.log(Math.random());                                  0.04237470521314579

//Returns maximum and minimum number and a number
raised to another                                            7
console.log (Math.max(3, 7,-10, 1.2));                       0.2
console.log (Math.min(1,3,12, 0.2));                         32
console.log (Math.pow(2,5));
```

random, max, min and pow Math functions

2.- Strings

Strings operations

1.- Objects > Strings

script.js

```
let cad1="Adiós";
let cad2="Ho\nla"; // /n is new line while /t is tab
//quotes need to be protected in order to print them inside the message
//if they are the same to quote the message
let cad3="SOME CHARACTERS need to be protected in order to be printed: \" \\";

//iterate
for (let letra of cad1){
    console.log(letra);
}

//strings can't be modified
cad1[0]="a";
console.log (cad1);

//changing case
console.log(cad1.toUpperCase(), cad1.toLowerCase()
);
```

Console

```
A
d
i
ó
s
Adiós
ADIÓS  adiós
```

Acessing strings and changing case

Searching sub strings

1.- Objects > Strings

Searching for substrings

Including, starting and ending strings

script.js ×

```
let cad3="SOME CHARACTERS need to be protected in order to be printed: \" \\";

//return true if found
console.log (cad3.includes("ed"), cad3.includes("esto no está"));
//no need for comments...
console.log(cad3.startsWith("SOME"), cad3.endsWith("hola"));
```

Console ×

```
true  false
true  false
```

includes, startswith and endswith

Getting sub strings

script.js

```
let cad3="SOME CHARACTERS need to be protected in order to be printed: \" \\";

//Returns a substring. Doesn't change original
console.log (cad3.slice(2,10));     //returns substring from position 2 to 9
console.log (cad3.slice(15, cad3.length-10));    //returns substring from position 15 until 11 to the end
console.log (cad3.slice(25));  //returns substring from position 25 to the end
console.log (cad3.slice(-10,-5));  //returns substring from position 10 from the right to 5th to the right

//returns an array of substrings separated by the argument. It doesn't change original
console.log (cad3.split(" "));
```

Console

```
ME CHARA
 need to be protected in order to be pr
e protected in order to be printed: " \
inted
▶ (13) ["SOME", "CHARACTERS", "need", "to",...]
```

Getting substrings with slice and getting an array of substrings with split

3.- Objects

Creating objects

1.- Objects > Object

script.js

```
1   //Creating objects
2   let usuario={
3       id:"1",
4       nombre:"pepe",
5       edad:30
6   };
7   console.log (usuario);
8
9   //Creating objects with New and a function
10  function creaPerro(nombre, raza) {
11      this.nombre = nombre;
12      this.raza = raza;
13  }
14
15  let perro1=new creaPerro ("pirata","beagle");
16  console.log (perro1);
17
18  //creating objects with new Object
19  var miCoche = new Object();
20  miCoche.fabricante = "SEAT";
21  miCoche.modelo = "600";
22  miCoche.anyo = 1957;
23  console.log (miCoche);
```

Console

```
▶ (3) {id: "1", nombre: "pepe", edad: 30}
▶ creaPerro {nombre: "pirata", raza: "beagle"}
▶ (3) {fabricante: "SEAT", modelo: "600", ...}
```

Three ways of creating objects

Basic operations with objects

1.- Objects > Object

```
let usuario={
    id:"1", nombre:"pepe", edad:30
};



//accessing object properties
console.log(usuario.nombre);    //returns value                    'pepe'
console.log (usuario.noExiste);    //returns undefined, but no error    undefined
console.log ("hola" in usuario);     //returns false, but no error    false
console.log ("edad" in usuario);   //returns true                     true

//assigning values
usuario.esAdmin=false;                                               false
console.log(usuario);                                                {
                                                                       id: '1',
//removing properties                                                  nombre: 'pepe',
delete(usuario.edad);                                                  edad: 30,
console.log(usuario);                                                  esAdmin: false
                                                                     }
                                                                     true
                                                                     { id: '1', nombre: 'pepe', esAdmin: false }
```

Basic operations with objects

Creating and modifying objects

1.- Objects > Object

main.js

Run

```
let usuario={
    id:"1", nombre:"pepe", edad:30
};

//brackets allows to calculate in real-time the key
let llave=prompt("¿Qué quieres saber del usuario?");  //needs to be a valid key name
console.log(usuario[llave]);    //llave=edad or nombre...

//another example of brackets
let llave2=prompt("¿Qué elemento quieres?");
let cantidad={
    [llave2]: 5,
    [llave2 + "fruta"]: 8
}
console.log (cantidad);
console.log (cantidad[llave2]);
```

Output

```
node /tmp/eI3JqPL0dQ.js
¿Qué quieres saber del usuario?edad
30
¿Qué elemento quieres?fresas
{ fresas: 5, fresasfruta: 8 }
5
```

Using brackets to dinamically calculate key value

Property value shorthand

1.- Objects > Object

```
//create users using property value shorthand
function makeUser(name, age){
    return{
        name: name,
        age: age,
    };
}

let user=makeUser("paco", 40);
console.log (user);                                                    { name: 'paco', age: 40 }


//another way of doing the same
//same as before but function as arrow
//as properties have the same name as variables, they can be removed
//here they have been removed from arrow, but can be done in traditional funcion
as well
let creaUsuario=(name,age)=>(
    {name, age}
);


                                                                       { name: 'pepe', age: 30 }
user=creaUsuario("pepe", 30);                                          { name: 'pepe', age: 30 }
console.log (user);
```

You can use a shorthand to define objects

Object references

1.- Objects > Object

```
1   let aux="hola";
2   let aux2=aux;   //aux value is copied into aux2 value. Both point to a different memory location
3
4   aux2="adios";   //if I modify aux2, aux still holds its original value
5   console.log(aux, aux2);
6
7   //when copying objects (array are objects as well), they both point to the same memory location
8   let usuario={ nombre:"pepe" };
9   let usuario2=usuario;   //usuario2 and usuario point to the same memory location. they are the same object
10
11  usuario.nombre="fede";
12  console.log(usuario2.nombre);
```

```
Console ×

hola  adios

fede
```

Objects duplicated are the same object. They both point to the same memory location, just unlike variables

Object references

1.- Objects > Object

script.js

```
let objeto1=objeto2={
    nombre:"pepe",
    profesión: "fontanero"
};
objeto1.edad=33;
console.log (objeto1, objeto2);

let objeto3={
    nombre:"pepe",
    profesión: "fontanero"
};

console.log (objeto1==objeto2); //true. They are both the same object
console.log (objeto1==objeto3); //false. They are different objects (a
```

Console

```
▼ (3) {nombre: "pepe", profesión: "fontane...}
    nombre: "pepe"
    profesión: "fontanero"
    edad: 33
  ▶ [[Prototype]]: {}
▼ (3) {nombre: "pepe", profesión: "fontane...}
    nombre: "pepe"
    profesión: "fontanero"
    edad: 33
  ▶ [[Prototype]]: {}
true
false
```

Objects duplicated are the same object. They both point to the same memory location, just unlike variables

Copying objects and properties

1.- Objects > Object

script.js

```
let objeto1={
    nombre:"pepe",
    profesion: "fontanero"
};
objeto1.edad=33;

let objeto2={
    nacionalidad:"Española"
}
let objeto4=(Object.assign({}, objeto1, objeto2));
console.log(objeto4);
```

Console

```
▼ (4) {nombre: "pepe", profesion: "fontane...}
    nombre: "pepe"
    profesion: "fontanero"
    edad: 33
    nacionalidad: "Española"
  ▶ [[Prototype]]: {}
```

Assign allows to copy one or more objects into another one by value

Copying nested objects

1.- Objects > Object

script.js

```
1   //nested cloning and copying. structuredclone allows objects i
2   //should not be used, object inside object would be copied by
3   let objeto1={
4       nombre:"pepe",
5       profesion: "fontanero",
6       medidas: {
7           altura:180,
8           pecho: 100,
9           cadera: 80,
10          cintura: 100
11      }
12  };
13
14  let objeto2=Object.assign({}, objeto1);
15  objeto2.nombre="juan";
16  objeto1.medidas.altura=170;
17  console.log(objeto1, objeto2);
18  objeto2=structuredClone(objeto1);   //object 2 is now copied b
19  objeto1.medidas.altura=200;
20  console.log(objeto1.medidas.altura, objeto2.medidas.altura);
```

Console

```
▼ (3) {nombre: "pepe", profesion: "fontane...}
    nombre: "pepe"
    profesion: "fontanero"
  ▶ medidas: (4) {altura: 200, pecho: 100, cadera: 80...}
  ▶ [[Prototype]]: {}
▼ (3) {nombre: "juan", profesion: "fontane...}
    nombre: "juan"
    profesion: "fontanero"
  ▶ medidas: (4) {altura: 200, pecho: 100, cadera: 80...}
  ▶ [[Prototype]]: {}
200  170
```

structuredClone allows objects inside objects to be copied as value, not reference

Adding and removing properties

1.- Objects > Object

script.js

```
let objeto1={
    nombre:"pepe",
    profesion: "fontanero",
    medidas: {
        altura:180,
        pecho: 100,
    }
};

objeto1.edad=37;
objeto1.saluda=()=>{
  console.log("te saludo");
}
delete(objeto1.medidas);
console.log (objeto1);
objeto1.saluda();
```

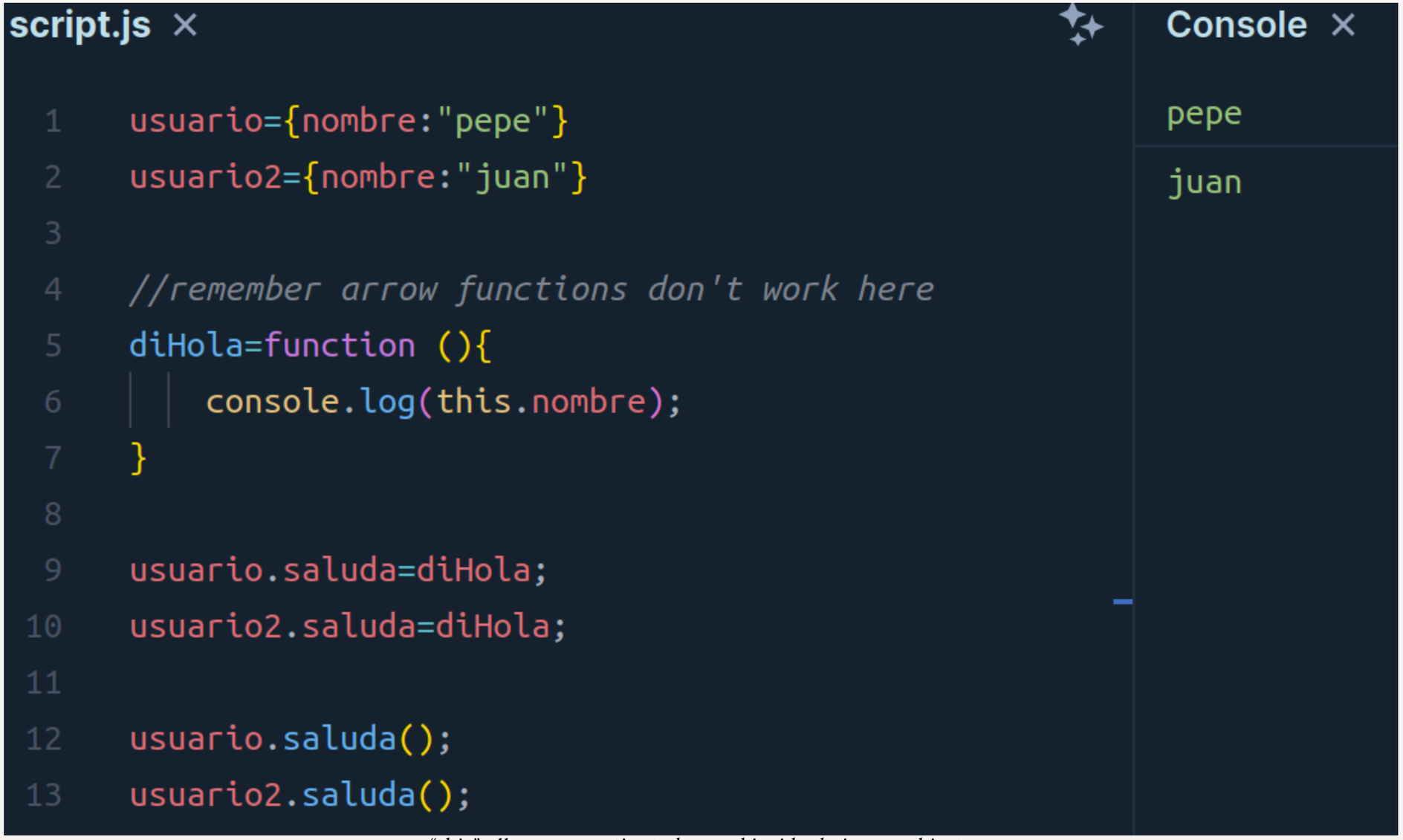
Console

```
▼ (4) {nombre: "pepe", profesion: "fontane...}
    nombre: "pepe"
    profesion: "fontanero"
    edad: 37
  ▶ saluda: ƒ ()
  ▶ [[Prototype]]: {}
te saludo
```

Delete removes object properties

Using properties inside objects (this)

1.- Objects > Object



"this" allows properties to be used inside their own objects

Using properties inside objects (this)

1.- Objects > Object

script.js

```
usuario={nombre:"pepe"}

diHola=function (){
  return(`Hola, soy ${this.nombre}`);
}
let diAdios=() => {return(`Adiós, soy ${this.nombre}`)};

//assign function to object property
usuario.saluda=diHola;
usuario.despidete=diAdios;

console.log(usuario.saluda());
console.log(usuario.despidete());
```

Console

```
Hola, soy pepe
Adiós, soy undefined
```

Arrow functions behave differnt with “this”

Using properties inside objects (this)

1.- Objects > Object

script.js

```
"use strict";
let persona={
        nombre:"pepe",
        ref1: this,
        ref2(){
          return this;
        }
}

console.log(persona.ref1.nombre);    //doesn't work. This is only for methods
console.log(persona.ref2().nombre);  //works
```

Console

```
undefined
pepe
```

“this” only works with methods

Using properties inside objects (this)

1.- Objects > Object

script.js

```
persona1={
    nombre:"pepe",
    profesion: "fontanero",
    medidas: {
        altura:180,
        pecho: 100,
    },
    buenosDias(){
        console.log (`yo, ${this.nombre}, te doy los buenos días`);
    }
};

persona1.buenasTardes=function(){
        console.log (`yo, ${this.nombre}, te doy las buenas tardes`);
}

persona1.buenasNoches=()=>{     //it doesn't work. It's an arrow function
        console.log (`yo, ${this.nombre}, te doy las buenas noches`);
}

persona1.buenosDias();
persona1.buenasTardes();
persona1.buenasNoches();
```

Console

```
yo, pepe, te doy los buenos días
yo, pepe, te doy las buenas tardes
yo, undefined, te doy las buenas noches
```

“this” allows properties to be used inside their own objects

Using properties inside objects (this)

1.- Objects > Object

script.js

```
let grupo = {
    nombre: "Los amigos",
    habitantes: ["Máximo", "Higinio", "Salustiano"],
    localidades: ["Jódar", "Guarromán", "La cabra del santo cristo"],
    muestraLista() {
        this.habitantes.forEach(
            //arrow functions have no "this"
            persona => console.log(this.nombre + ': ' + persona)
        );
    },
    muestraLocalidades() {
        this.habitantes.forEach(function(persona) {
          // Error: Cannot read property 'title' of undefined
          console.log(persona+" es de:"+ this.localidades);
        });
      }
};

grupo.muestraLista();
grupo.muestraLocalidades();
```

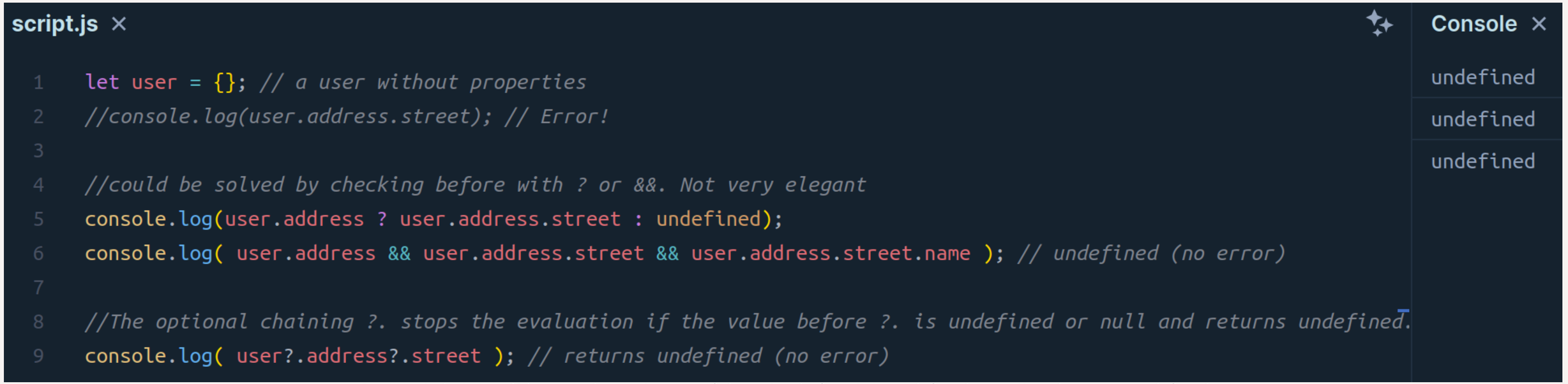
Console

```
Los amigos: Máximo
Los amigos: Higinio
Los amigos: Salustiano
Máximo es de:undefined
Higinio es de:undefined
Salustiano es de:undefined
```

Arrow functions have no “this”

Optional chaining for variables

1.- Objects > Object



Optional chaining (?.) stops the evaluation if the value before ?. is undefined or null and returns undefined

Optional chaining for methods

1.- Objects > Object

script.js

```
let userAdmin = {
  isAdmin() {
    console.log("I am admin");
  }
};

let userGuest = {};
userAdmin.isAdmin?.(); // I am admin
//userGuest.isAdmin(); // throws an error (method doesn't exist)
userGuest.isAdmin?.(); // nothing happens (although no such method)
```

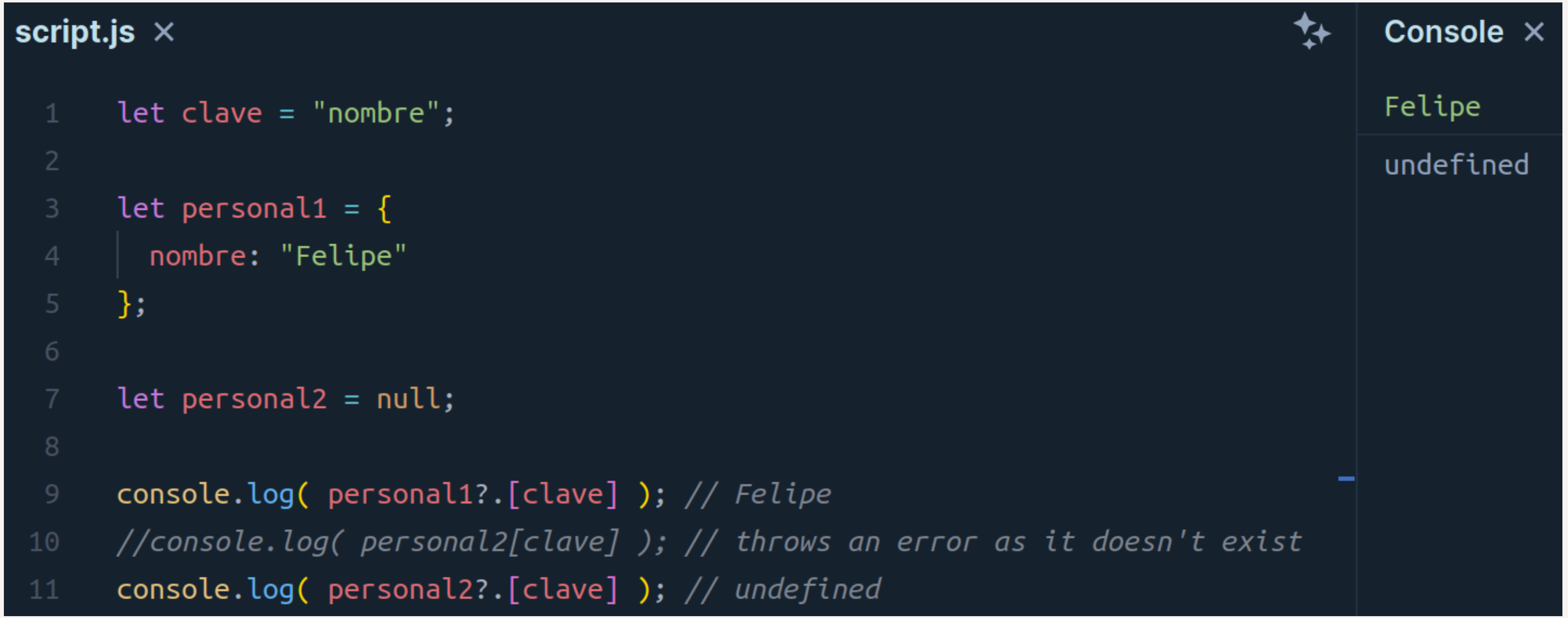
Console

```
I am admin
```

Optional chaining for functions (?.) stops the evaluation if the value before ?. is undefined or null and returns undefined

Optional chaining for functions

1.- Objects > Object



Optional chaining for functions (?.) stops the evaluation if the value before ?. is undefined or null and returns undefined

Getting object values and keys

1.- Objects > Object

script.js

```
persona1={
    nombre:"pepe",
    profesion: "fontanero",
    medidas: {
        altura:180,
        pecho: 100,
    },
};

for (let valor of Object.values(persona1)){
  console.log (valor);
}
console.log (Object.keys(persona1));
```

Console

```
pepe
fontanero
▶ (2) {altura: 180, pecho: 100}
▶ (3) ["nombre", "profesion", "medidas"]
```

Object.values and Object.keys allows to retrieve values and keys of an object

Transforming objects

1.- Objects > Object

script.js

```
let prices = {
  banana: 1,
  orange: 2,
  meat: 4,
};
let doublePrices = Object.fromEntries(
  // convert prices to array, map each key/value pair into another pair
  // and then fromEntries gives back the object
  Object.entries(prices).map(entry => [entry[0], entry[1] * 2]));
console.log(doublePrices);

const object1 = { a: 1, b: 2, c: 3 };
const object2 = Object.fromEntries(
  Object.entries(object1).map(([key, val]) => [key, val * 2]),
);
console.log(object2);
```

Console

```
▼ (3) {banana: 2, orange: 4, meat: 8}
    banana: 2
    orange: 4
    meat: 8
  ▶ [[Prototype]]: {}
▼ (3) {a: 2, b: 4, c: 6}
    a: 2
    b: 4
    c: 6
  ▶ [[Prototype]]: {}
```

fronEntries and entries allows to transform an object just like map did with arrays

Rest parameters

Rest parameteres can be used to create objects

4.- Dates

Creating date

1.- Objects > Date

script.js ×

```
1   let ahora=new Date();
2   let en1970=new Date(0);   //number of miliseconds starting at 01/01/1970. Can be negative for a date before 1970
3   let date = new Date("2018-10-20");
4   let fecha=new Date(2018, 9, 20, 14, 21, 0, 0); // (year, month (starting at 0), date, hours, minutes, seconds, ms)
5
6   console.log (ahora, en1970, date, fecha);
```

Console ×

```
Fri Oct 20 2023 14:32:43 GMT+0200 (hora de verano de Europa central)  Thu Jan 01 1970 01:00:00 GMT+0100 (hora estándar de Europa central)
Sat Oct 20 2018 02:00:00 GMT+0200 (hora de verano de Europa central)  Sat Oct 20 2018 14:21:00 GMT+0200 (hora de verano de Europa central)
```

Some ways of creating a Date object

Date getters and setters

1.- Objects > Date

script.js

```
let ahora=new Date();
let en1970=new Date(0);   //number of miliseconds starting at 01/01/1970. Can be negative for a date before 1970
let hace5anyos = new Date("2018-10-20");
let fecha=new Date(2018, 9, 20, 14, 21, 0, 0); // (year, month (starting at 0), date, hours, minutes, seconds, ms)

console.log (
    ahora.getFullYear(),
    ahora.getMonth(),
    ahora.getDay(),
    ahora.getHours(),
    ahora.getDate() //gets day of month. Not really intuitive
);

ahora.setFullYear(2022);
ahora.setMonth(ahora.getMonth()-2);
ahora.setDate(-1);  //can use negative numbers meaning one day before the last one of the previous month
ahora.setMinutes(68);   //adds minutes to current hour, beginning at 0
console.log(ahora);
```

Console

```
2023  9  5  19  20
Sat Jul 30 2022 20:08:21 GMT+0200 (hora de verano de Europa central)
```

Substracting dates

1.- Objects > Date

script.js

```
let ahora=new Date();
let hace5anyos = new Date("2018-10-20");

//substracting dates to compare
console.log (ahora-hace5anyos); //returns nº of miliseconds
console.log (ahora.getTime()-hace5anyos.getTime()); //it does the same but, apparently, quicker

```

Console

```
157826738463
157826738463
```

Substracting dates to compare

Creating dates from strings

1.- Objects > Date

script.js ×

```
1   let ms = Date.parse("2012-01-26T13:51:50.417-07:00");   //it gets miliseconds from base date
2   let date = new Date(Date.parse("2012-01-26T13:51:50.417")); //it creates a new date from a string
3   console.log(ms);
4   console.log (date);
```

Console ×

```
1327611110417
Thu Jan 26 2012 13:51:50 GMT+0100 (hora estándar de Europa central)
```

Creating date object from strings

5.- Arrays

Arrays basic operations

1.- Objects > Array

script.js

```
let matriz1=new Array();
let matrizFrutas=["chirimoya", "mango", "aguacate"];
let matrizMezcla=["hola",
        1,
        4.2,
        function(){console.log ("saludos desde dentro de la matriz");}]

//show the whole array
console.log(matrizFrutas);
//show one position
console.log (matrizMezcla[2]);
//run array function
matrizMezcla[3]();
//Access to an element and replace it
matrizFrutas[2]="sandía";
//access to the last element
console.log (matrizFrutas[matrizFrutas.length-1]);
console.log (matrizFrutas[matrizFrutas.at(-1)]);    //may not work on old browsers
```

Console

```
▶ (3) ["chirimoya", "mango", "aguacate"]
4.2
saludos desde dentro de la matriz
sandía
undefined
```

Basics array operations: create, show, access and replace

Arrays basic operations

1.- Objects > Array

script.js

```
1   //COPY arrays
2   let arr = arrCopy = [1, 2, 3];
3
4   let arrCopy2 = [...arr]; // spread the array into a list of parameters
5                            // then put the result into a new array
6   let  arrCopy3=Array.from(arr);
7
8   // do the arrays have the same contents?
9   console.log(JSON.stringify(arr) === JSON.stringify(arrCopy)); // true
10  console.log(JSON.stringify(arr) === JSON.stringify(arrCopy2)); // true
11  console.log(JSON.stringify(arr) === JSON.stringify(arrCopy3)); // true
12
13  // are the arrays equal?
14  console.log(arr === arrCopy); // true (same reference)
15  console.log(arr === arrCopy2); // false (not same reference)
16  console.log(arr === arrCopy3); // false (not same reference)
17
18  // modifying our initial array does not modify the copy:
19  arr.push(4);
20  console.log(arr); // 1, 2, 3, 4
21  console.log(arrCopy); // 1, 2, 3, 4
22  console.log(arrCopy2); // 1, 2, 3
23  console.log(arrCopy3); // 1, 2, 3
```

Console

```
true
true
true
true
false
false
▶ (4) [1, 2, 3, 4]
▶ (4) [1, 2, 3, 4]
▶ (3) [1, 2, 3]
▶ (3) [1, 2, 3]
```

Copying arrays

Arrays iteration

1.- Objects > Array

script.js

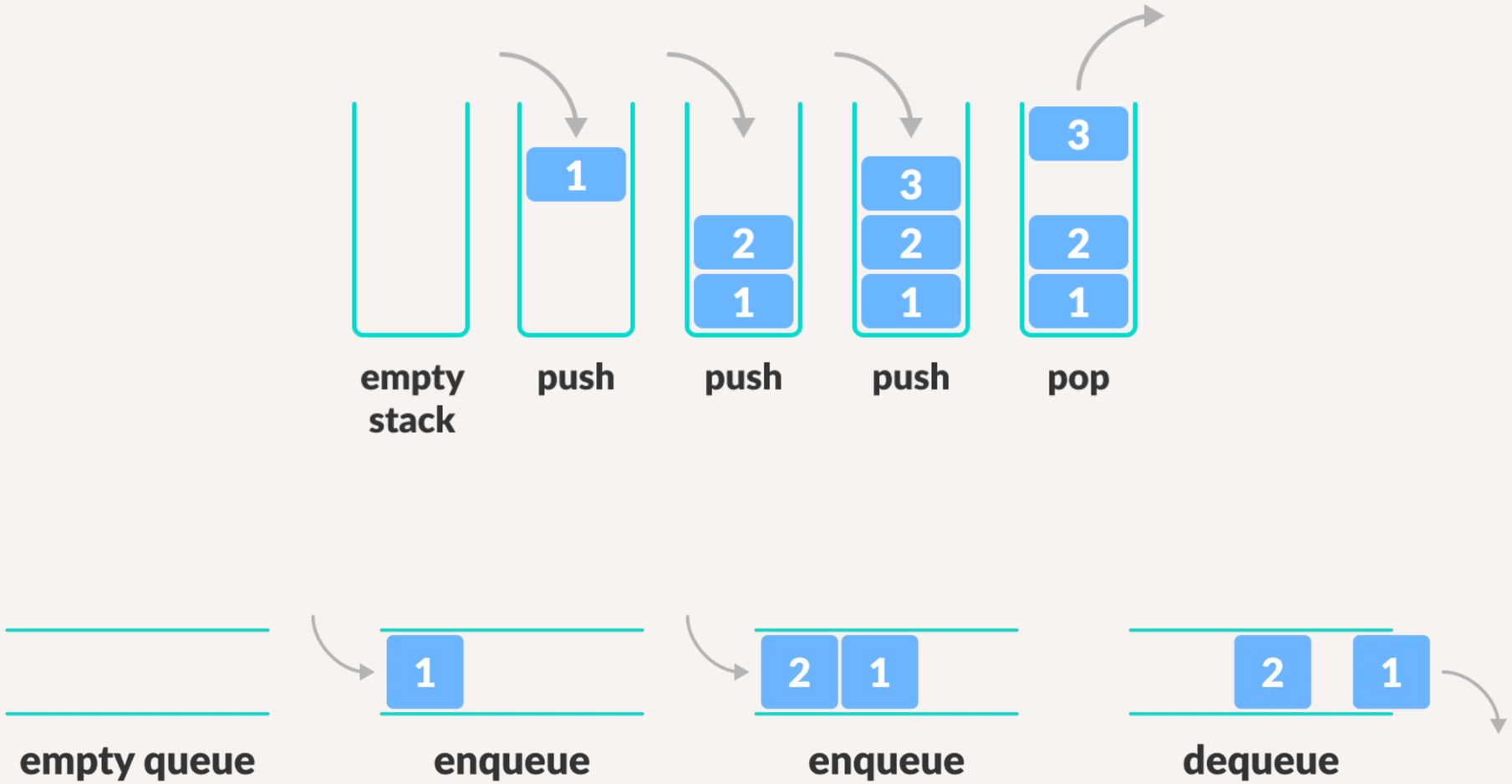
```
let matrizFrutas=["chirimoya", "mango", "aguacate"];

//option #1. Needs to define a function
let funcion = (elemento, indice) => console.log (elemento);
matrizFrutas.forEach(funcion);

//option #2
for (let fruta of matrizFrutas){
    console.log (fruta);
}

//option #3, traditional way. Needs to get array length
for (let i=0; i<matrizFrutas.length; i++){
    console.log (matrizFrutas[i]);
}
```

Console

```
chirimoya
mango
aguacate
chirimoya
mango
aguacate
chirimoya
mango
aguacate
```

Array iteration

Arrays as a stack and as a queue



Stack (pila) vs queue (cola). Arrays can act as both, besides as arrays, of course

Arrays as a queue

1.- Objects > Array

script.js ×

```
1   let matrizFrutas=["chirimoya", "mango", "aguacate"];  //most used. Can be created empty
2
3   console.log(matrizFrutas);
4   matrizFrutas.shift();   //gets an element fro the beginning
5   console.log(matrizFrutas);
6   matrizFrutas.push ("piña"); //adds "piña" to the last position
7   console.log(matrizFrutas);
```

Console ×

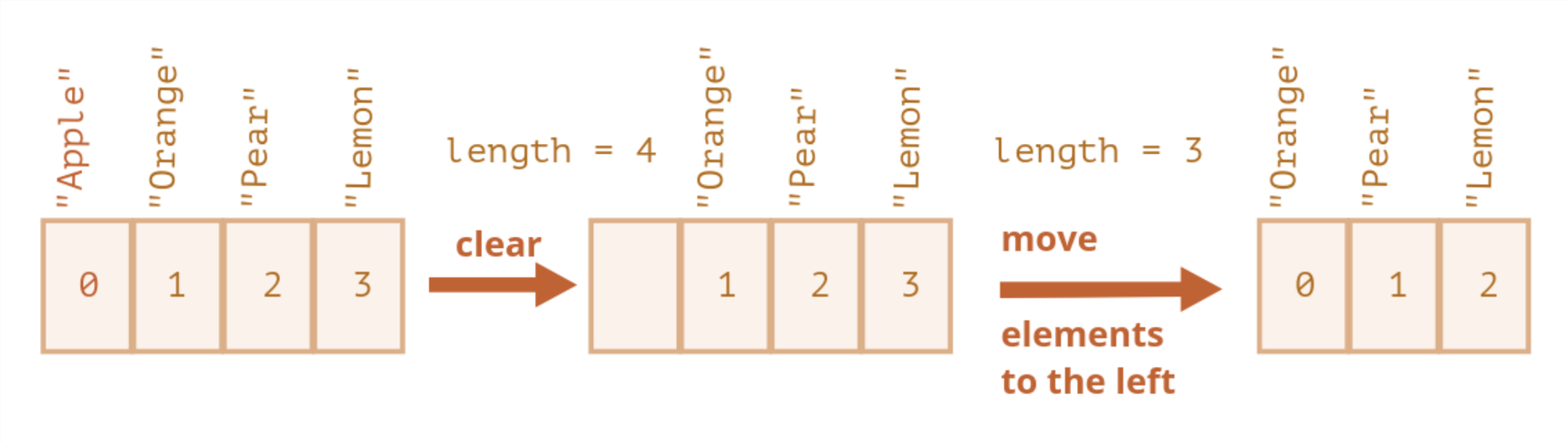
```
▶ (3) ["chirimoya", "mango", "aguacate"]
▶ (2) ["mango", "aguacate"]
▶ (3) ["mango", "aguacate", "piña"]
```

Array acting as a queue

Arrays as a queue

The shift operation must do 3 things:

- Remove the element with the index 0.
- Move all elements 1 position to the left in memory cells and renumber their index.
- Update the length property.



Working with queues is slow. When shifting, all the array elements has to be moved down. The bigger the array, the slower it it to operate as a queue

Arrays as a stack

1.- Objects > Array

script.js ×

```
1   let matrizFrutas=["chirimoya", "mango", "aguacate"];
2
3   //arrays can act as stacks, queues and, of course, arrays
4   //array as stack: pop and push
5   console.log(matrizFrutas.pop()); // remove "aguacate" and show matrizFrutas
6   console.log (matrizFrutas);
7   matrizFrutas.push ("piña");
8   console.log (matrizFrutas);
9
```

Console ×

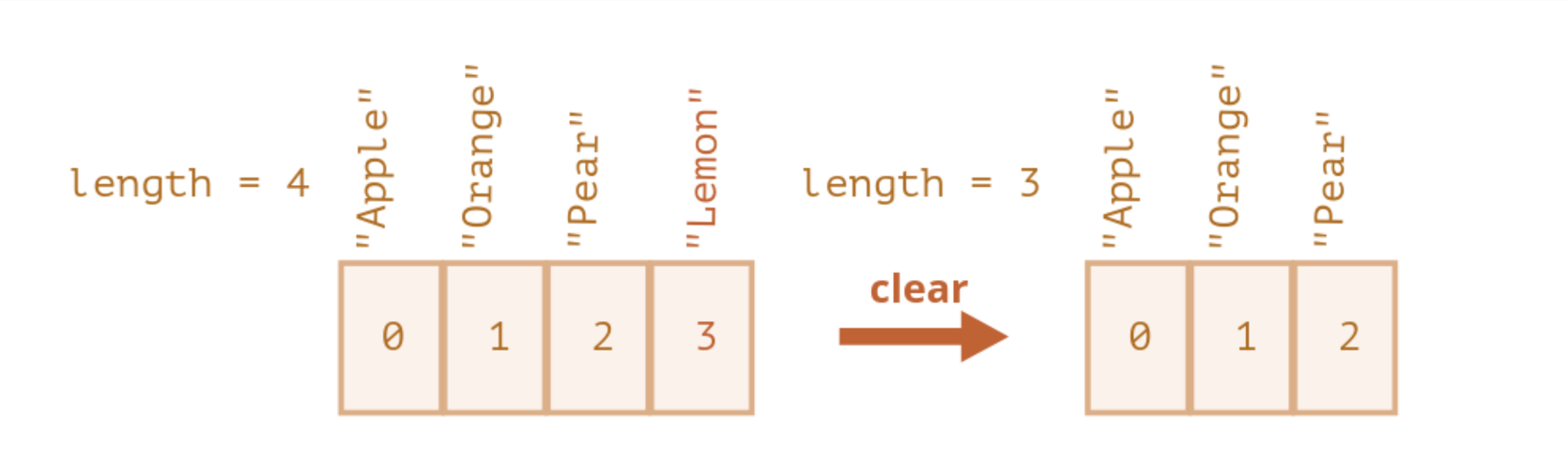
```
aguacate
▶ (2) ["chirimoya", "mango"]
▶ (3) ["chirimoya", "mango", "piña"]
```

Array acting as a stack

Arrays as a stack

Operating as a stack is fast

The pop method does not need to move anything, because other elements keep their indexes



Working with stack is fast, the rest of elements are not affected by any operation

Arrays conversion

1.- Objects > Array

script.js

```
1    let matrizFrutas=["chirimoya", "mango", "aguacate"];
2
3    let cadena=matrizFrutas.toString();
4    console.log(cadena, "---", typeof(cadena));
5
```

Console

```
chirimoya,mango,aguacate  ---  string
```

An array can be converted to String

Multidimensional arrays

1.- Objects > Array

script.js

```
let matrizComida=[
    ["chirimoya", "mango", "aguacate"],
    ["tomate", "pepino", "pimiento"],
    ["leche", "yougur", "requesón"]
];
//matrizAlimentos is a pointer to the memory space of matrizComida
let matrizAlimentos=matrizComida;

console.log (matrizComida);

for (let categoria of matrizComida){
    for (let fruta of categoria){
        console.log (fruta);
    }
}
//if matrizAlimentos is changed, so it is matrizComida.
matrizAlimentos[2][2]="puerro";
console.log (matrizComida);
```

Console

```
▼ (3) [Array(3), Array(3), Array(3)]
    ▶ 0: (3) ["chirimoya", "mango", "aguacate"]
    ▶ 1: (3) ["tomate", "pepino", "pimiento"]
    ▶ 2: (3) ["leche", "yougur", "puerro"]
    ▶ [[Prototype]]: []
chirimoya
mango
aguacate
tomate
pepino
pimiento
leche
yougur
requesón
▼ (3) [Array(3), Array(3), Array(3)]
    ▶ 0: (3) ["chirimoya", "mango", "aguacate"]
    ▶ 1: (3) ["tomate", "pepino", "pimiento"]
    ▶ 2: (3) ["leche", "yougur", "puerro"]
    ▶ [[Prototype]]: []
```

Multidimensional array

Comparing arrays

1.- Objects > Array

script.js

```
let matrizComida=[
    ["chirimoya", "mango", "aguacate"],
    ["tomate", "pepino", "pimiento"],
    ["leche", "yougur", "requesón"]
];

let matrizComida2=[
    ["chirimoya", "mango", "aguacate"],
    ["tomate", "pepino", "pimiento"],
    ["leche", "yougur", "requesón"]
];
//matrizAlimentos is a pointer to the memory space of matrizComida
let matrizAlimentos=matrizComida;

console.log (matrizComida==matrizAlimentos);
console.log (matrizComida==matrizComida2);
```

Console

```
true
false
```

Comparing arrays

Getting subarrays from arrays

1.- Objects > Array

script.js

```
let matrizFrutas=["chirimoya", "mango", "aguacate"];
let matrizComida=[
    ["chirimoya", "mango", "aguacate", "guayaba"],
    ["tomate", "pepino", "pimiento", "berenjena"],
    ["leche", "yougur", "requesón", "queso"]
];
//SLICE returns subarray, but without changing original array
matrizFrutas.slice(1,2);   //starting from position 1, remove 2 elements
console.log (matrizFrutas);
let a=matrizComida[1].slice(1);  //remove from the 1st until the last element
console.log (a);
console.log (matrizComida);
```

Console

```
▼ (3) ["chirimoya", "mango", "aguacate"]
    0: "chirimoya"
    1: "mango"
    2: "aguacate"
  ▶ [[Prototype]]: []
▼ (3) ["pepino", "pimiento", "berenjena"]
    0: "pepino"
    1: "pimiento"
    2: "berenjena"
  ▶ [[Prototype]]: []
▼ (3) [Array(4), Array(4), Array(4)]
  ▶ 0: (4) ["chirimoya", "mango", "aguacate", "..."]
  ▶ 1: (4) ["tomate", "pepino", "pimiento", "be..."]
  ▶ 2: (4) ["leche", "yougur", "requesón", "que..."]
  ▶ [[Prototype]]: []
```

Slice return a subarray, but without modifying original array

Removing elements from arrays

1.- Objects > Array

script.js

```
let matrizFrutas=["chirimoya", "mango", "aguacate"];
let matrizComida=[
    ["chirimoya", "mango", "aguacate", "guayaba"],
    ["tomate", "pepino", "pimiento", "berenjena"],
    ["leche", "yougur", "requesón", "queso"]
];

//SPLICE returns subarray, but removing it from original array
matrizFrutas.splice(1,2);   //starting from position 1, remove 2 elements
console.log (matrizFrutas);
let a=matrizComida[1].splice(1);  //remove from the 1st until the last elemen
console.log (a);
console.log (matrizComida);

```

Console

```
▼ (1) ["chirimoya"]
    0: "chirimoya"
  ▶ [[Prototype]]: []
▼ (3) ["pepino", "pimiento", "berenjena"]
    0: "pepino"
    1: "pimiento"
    2: "berenjena"
  ▶ [[Prototype]]: []
▼ (3) [Array(4), Array(1), Array(4)]
  ▶ 0: (4) ["chirimoya", "mango", "aguacate", "..."]
  ▶ 1: (1) ["tomate"]
  ▶ 2: (4) ["leche", "yougur", "requesón", "que..."]
  ▶ [[Prototype]]: []
```

Splice removes elements from arrays and return it as a subarray

Concatenating arrays

1.- Objects > Array

script.js ×

```
2   let matrizComida=[
3       ["chirimoya", "mango", "aguacate", "guayaba"],
4       ["tomate", "pepino", "pimiento", "berenjena"],
5       ["leche", "yogur", "requesón", "queso"]
6   ];
7
8   //CONCAT creates a new array that includes values from other arrays
9   console.log(matrizFrutas.concat(matrizComida[1],"otra fruta"));
10
11
```

Console ×

```
▼ (8) ["chirimoya", "mango", "aguacate", "..."]
    0: "chirimoya"
    1: "mango"
    2: "aguacate"
    3: "tomate"
    4: "pepino"
    5: "pimiento"
    6: "berenjena"
    7: "otra fruta"
  ▶ [[Prototype]]: []
```

Concat concatenates arrays and elements

Iterating arrays

1.- Objects > Array

script.js

```
let matrizFrutas=["chirimoya", "mango", "aguacate"];
let matrizComida=[
    ["chirimoya", "mango", "aguacate", "guayaba"],
    ["tomate", "pepino", "pimiento", "berenjena"],
    ["leche", "yogur", "requesón", "queso"]
];

//FOREACH method allows to run a function for every element of the array.
matrizFrutas.forEach((elemento,indice,matriz) =>{
    console.log (`El elemento ${elemento} está en la posición ${indice} de la matriz \'${matriz}\'`);
});

matrizComida[1].forEach(comida=>console.log(comida));
/*Less elegant and slower alternative
for (let i=0; i<matrizComida[1].length; i++){
    console.log (matrizComida[1][i]);
}
*/
```

Console

```
El elemento chirimoya está en la posición 0 de la matriz 'chirimoya,mango,aguacate'
El elemento mango está en la posición 1 de la matriz 'chirimoya,mango,aguacate'
El elemento aguacate está en la posición 2 de la matriz 'chirimoya,mango,aguacate'
tomate
pepino
pimiento
berenjena
```

forEach allows to iterate over an array

Searching in arrays

1.- Objects > Array

script.js ×

```
let matrizFrutas=["chirimoya", "mango", "aguacate"];

//Get position of an element in an array
console.log (matrizFrutas.indexOf("mango"), matrizFrutas.indexOf("esta no existe"));
//returns if an element exists in an array
console.log (matrizFrutas.includes("mango"), matrizFrutas.includes("esta no existe"));
//get the position of the last element (in case they are repeated)
console.log (matrizFrutas.lastIndexOf("mango"));
```

Console ×

```
1  -1
true  false
1
```

indexOf, includes and lasIndexOf allows to search inside arrays

Searching in arrays

1.- Objects > Array

```
let matrizFrutas=["chirimoya", "mango", "aguacate"];  //most used. Can be created empty
let inventario=[
    { nombre: "manzanas", cantidad: 2 },
    { nombre: "plátanos", cantidad: 0 },
    { nombre: "cerezas", cantidad: 5 },
  ];
//returns the index of the firts element that matches, -1 otherwise
console.log (matrizFrutas.findIndex(elemento=>elemento.nombre=="aguacate"));
//returns the index of the last element that matches, -1 otherwise
console.log (matrizFrutas.findLastIndex(elemento=>elemento.nombre=="aguacate"));
//returns the first element that matches, undefined otherwise
console.log (matrizFrutas.find(elemento=>elemento.nombre=="aguacate"));
//find object and get one of its properties
let elemento=inventario.find(elemento=>elemento.nombre=="manzanas");
(elemento!==undefined) ?
    console.log (`he encontrado ${elemento.cantidad} unidades`):
    console.log ("no hay");
//Alternative to previous line: define function outside
let esFruta=(fruta)=> fruta.nombre=="cerezas"; //arrow function
console.log (inventario.find(esFruta));
```

```
-1
-1
undefined
he encontrado 2 unidades
▶ (2) {nombre: "cerezas", cantidad: 5}
```

find, findIndex and findLastIndex allows to iterate over an array

Filtering arrays

1.- Objects > Array

```
let matrizFrutas=["chirimoya", "mango", "aguacate"];
let inventario=[
    { nombre: "manzanas", cantidad: 2 },
    { nombre: "plátanos", cantidad: 0 },
    { nombre: "cerezas", cantidad: 5 },
  ];

//Filter returns an array of all matching elements
let elementos = inventario.filter(item => item.cantidad < 3);
console.log (elementos);

//Iterate over results
let listado = (elemento) =>
    console.log (`El elemento ${elemento.nombre} tiene ${elemento.cantidad} unidades`);
elementos.forEach(listado);
```

```
[
  { nombre: 'manzanas', cantidad: 2 },
  { nombre: 'plátanos', cantidad: 0 }
]
'El elemento manzanas tiene 2 unidades'
'El elemento plátanos tiene 0 unidades'
```

Filtering results in arrays

Modifying arrays

1.- Objects > Array

```
let numeros=[7,13,2,5];

//MAP creates a new array as the result to apply some function to an existing one
numeros.map(x=>x*2);                                                    [ 14, 26, 4, 10 ]

//REVERSE do as its name suggests in an array. it modifies the array
numeros.reverse();                                                      [ 5, 2, 13, 7 ]
console.log (numeros);                                                  [ 5, 2, 13, 7 ]

//JOIN does the opposite to string.split. It returns a string made of concatenating the elements of an
array
let nombres=['Purificación', 'Procopio', 'Patrocinio', 'Apolinar'];
let cad=nombres.join();                                                 'Purificación,Procopio,Patrocinio,Apolinar'
console.log (cad,typeof(cad));                                          'string'

//REDUCE
//When we need to iterate over an array – we can use forEach, for or for..of.
//When we need to iterate and return the data for each element – we can use map.
//When we need to iterate and return a single value calculated by using the whole array, we can use reduce
//if numeros is empty add a zero, => sum + current, 0);
let result = numeros.reduce((sum, current) => sum + current);
console.log( result );                                                  27
```

Modifying arrays with map, reverse, join and reduce

Ordering arrays

1.- Objects > Array

```
let numeros=[7,13,2,5];
let numeros2=Array.from(numeros);    //copy numeros values into numeros2. If we use =, a reference is created
//SORT converts to strings before sorting, so 15<2 (as 1<2)
//it changes the original array and returns it
numeros2.sort();                                                                    [ 13, 2, 5, 7 ]
console.log (numeros2)                                                              [ 13, 2, 5, 7 ]
numeros2=Array.from(numeros);                                                       [ 7, 13, 2, 5 ]

//In order to sort numbers properly, a function has to be provided
//numbers a,b are sorted depending on what is returned by this function:
//   <0, then a is ordered first
//   0, then there is no change between a and b
//   >0 ,then b is ordered first
let ordenaNumeros=(a,b)=>{
    //it can't be expressed with ?, as it does not support return statement
    if (a>b){
      return 1;
    }else if (a==b){
      return 0;
    }return -1;
}
console.log(numeros2.sort(ordenaNumeros));                                          [ 2, 5, 7, 13 ]
console.log (numeros2, numeros);                                                    [ 2, 5, 7, 13 ] [ 7, 13, 2, 5 ]
numeros2=Array.from(numeros);                                                       [ 7, 13, 2, 5 ]

//a shorter and much more elegant version
numeros2.sort((a,b)=>a-b);                                                          [ 2, 5, 7, 13 ]
```

Ordering arrays with sort

Reversing arrays

1.- Objects > Array

```
let numeros=[7,13,2,5];

//reverse do as its name suggests in an array
numeros.sort((a,b)=>a-b).reverse();
```

```
[ 13, 7, 5, 2 ]
```

Reversing arrays with reverse

Spread operator

1.- Objects > Array

script.js

```
"use strict";

//Spread operator (operador de expansión)
//transform an array into a list of arguments
let arr = [3, 5, 1];
let arr2 = [-1, 0, 7];
let arr3= [2, 8, ...arr, -21, ...arr2];  //spread can be used to combine several arrays

console.log( Math.max(...arr));  //Math.max needs a list of arguments, not an array
console.log( Math.max(1, ...arr2, 21, ...arr) );  //several arrays can be combined
console.log (Math.max(...arr3));

let cad="hola";
console.log(...cad);    //returns every letter separately
console.log([...cad]);  //returns an array of letters
```

Console

```
5
21
8
h o l a
▶ (4) ["h", "o", "l", "a"]
```

Spread operator transform an array into a list of arguments. It can also be used to combine several arrays

6.- JSON

JSON.stringify

1.- Objects > JSON

script.js

```
let student = {
    name: 'John',
    age: 30,
    isAdmin: false,
    courses: ['html', 'css', 'js'],
};

let json = JSON.stringify(student);
console.log(typeof json, json);
```

Console

```
string  {"name":"John","age":30,"isAdmin":false,"courses":["html","css","js"]}
```

stringify method transforms an object into a string so it can be sent over the network

JSON.parse

1.- Objects > JSON

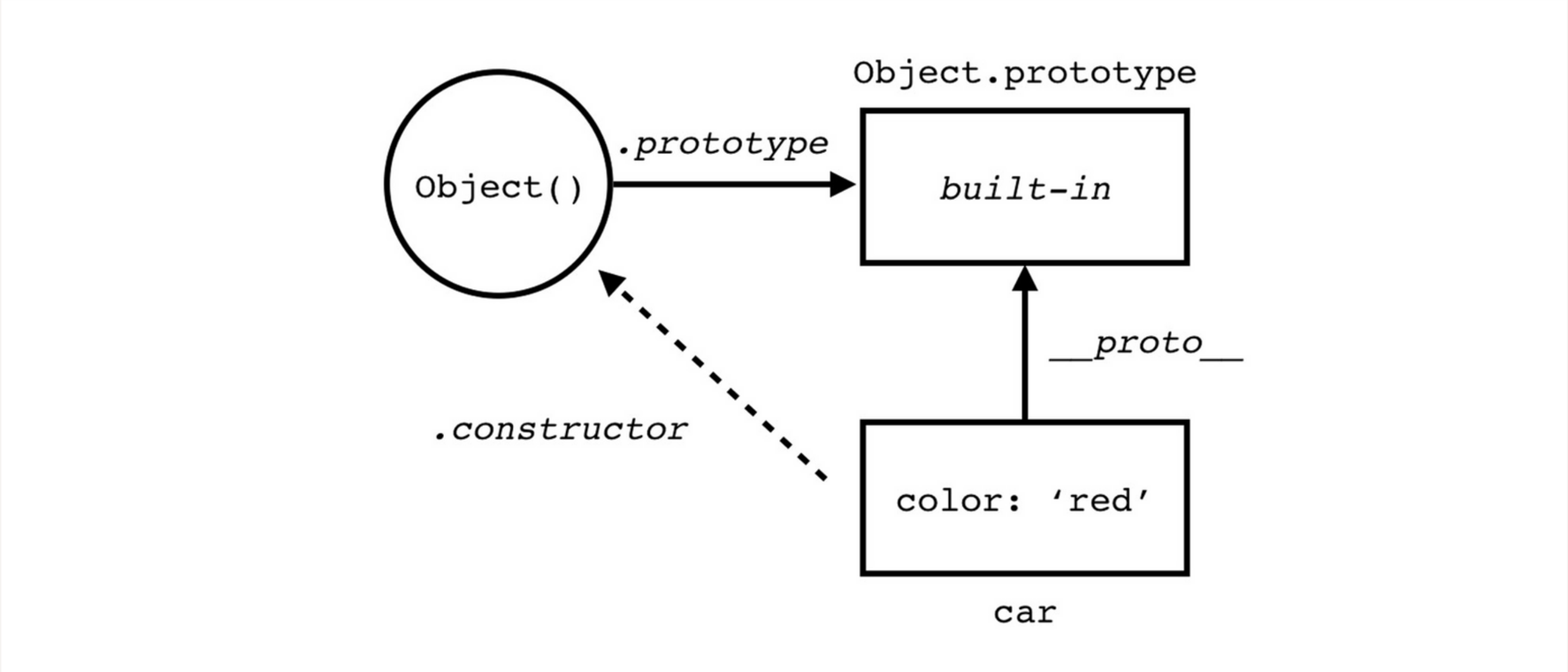
parse transforms an string into an object so it's easier to work with

2.- OO programming

Inheritance

2.- OO programming

- Objects have a hidden [[Prototype]] property
- When property is not found from in an object, then JavaScript takes it from prototype
- I can define methods or properties in a parent object and make them inherited to its children



prototype

Inheritance

2.- OO programming

script.js

```
let animal={
    come: true
};

let conejo={
    jumps: true,
};

Object.setPrototypeOf(conejo, animal);
//conejo.__proto__=animal;    shorter, but outdated.
console.log(conejo.come);
```

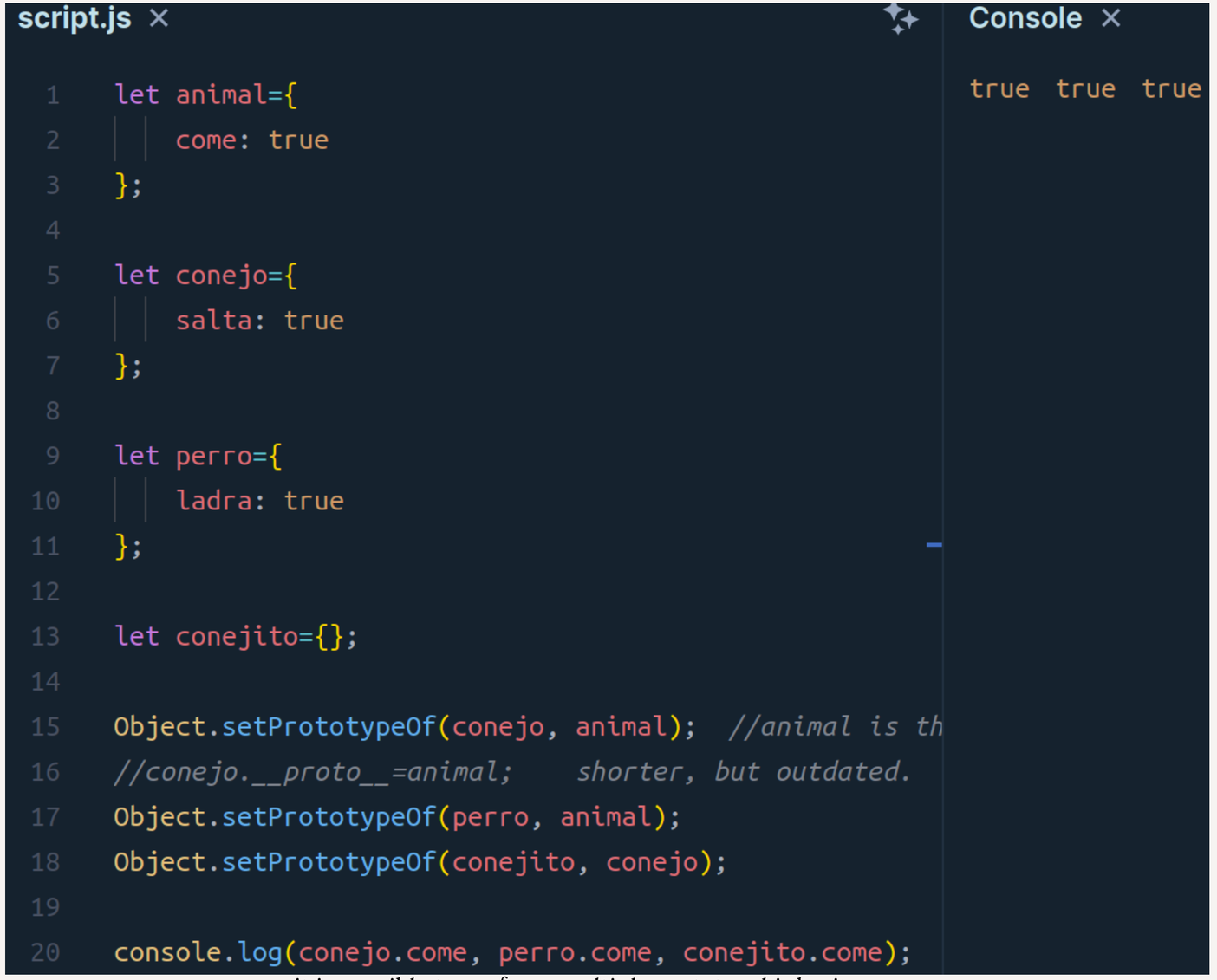
Console

```
true
```

conejo prototypically inherits from animal and animal is the prototype of conejo

Inheritance

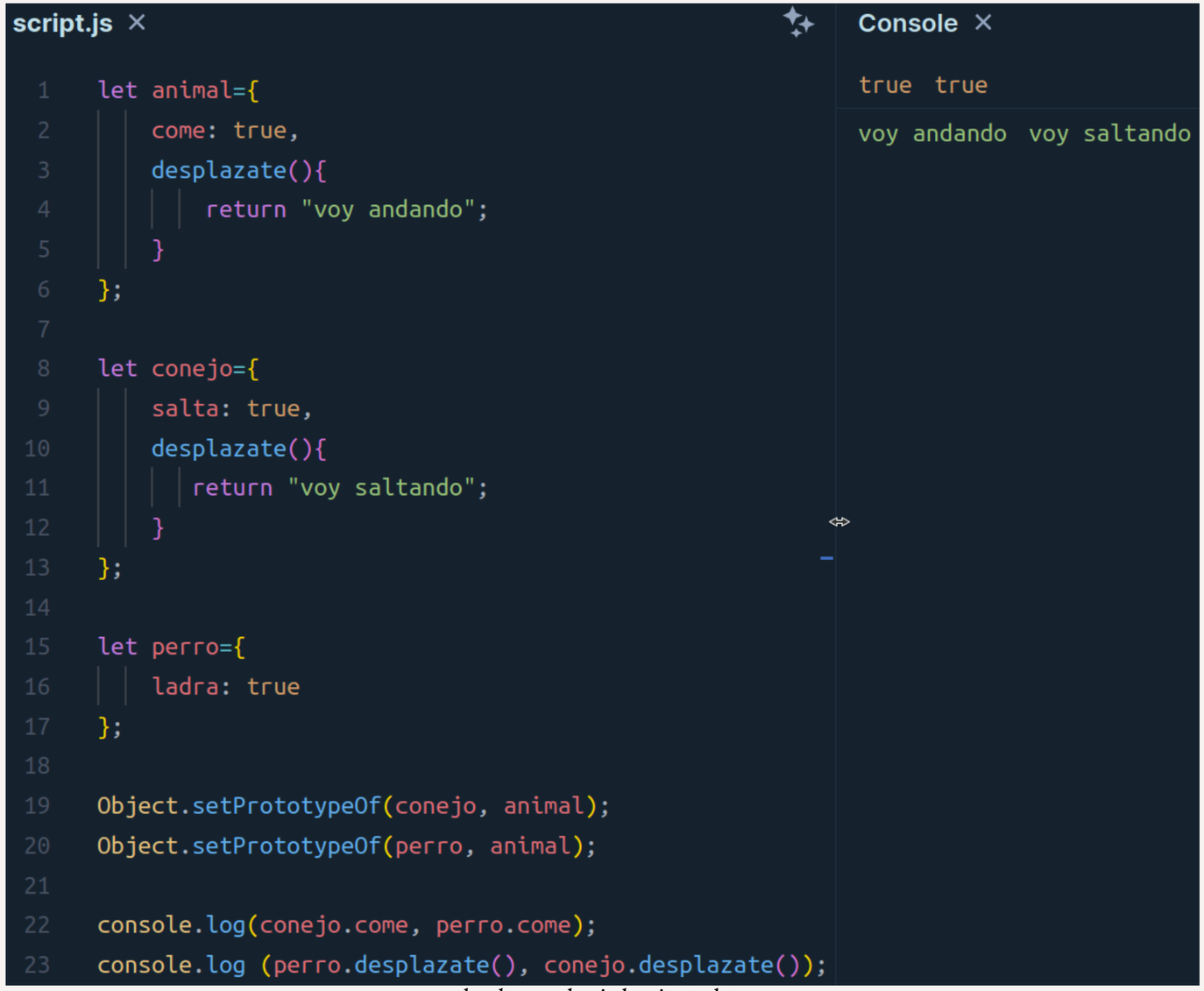
2.- OO programming



it is possible to perform multiple prototypal inheritance

Inheritance

2.- OO programming



methods can be inheritated too

Inheritance

2.- OO programming

script.js

```
"use strict";
let vehiculo={
    marca:"",
    modelo:"",
    circula(valor){
        this.circulando=valor;
    }
}

let coche={
    //circula:"",
    set marcayModelo(aux){
        [this.marca, this.modelo]=aux.split(' ');
    },

    get marcayModelo(){
        return `${this.marca} ${this.modelo}`;
    }
}

Object.setPrototypeOf(coche, vehiculo);
coche.marcayModelo="Volkswagen Golf";   //launches setter
coche.circula(true);
console.log(vehiculo, coche);
```

Console

```
▼ (3) {marca: "", modelo: "", circula: cir...}
    marca: ""
    modelo: ""
  ▶ circula: ƒ circula()
  ▶ [[Prototype]]: {}
▼ Object {marcayModelo: "Volkswagen Golf", ma...}
    marcayModelo: "Volkswagen Golf"
    marca: "Volkswagen"
    modelo: "Golf"
    circulando: true
  ▶ circula: ƒ circula()
  ▶ [[Prototype]]: (3) {marca: "", modelo: "", circula: cir...}
```

“this” refers to properties of the object which is making the call

Inheritance

2.- OO programming

script.js

```
"use strict";
let vehiculo = {
  marca: "",
  modelo: "",
  circulando: "",
  set circula(value) {
    this.circulando = value;
  },
  set marcaModelo(aux){
      [this.marca, this.modelo]=aux.split(' ');
  },
  get marcaModelo(){
      return `${this.marca} ${this.modelo}`;
  }
};

let coche = { };

Object.setPrototypeOf(coche, vehiculo);
coche.circulando="true";
coche.marcaModelo="ford fiesta";
console.log(vehiculo, coche);
```

Console

```
▼ (5) {marca: "", modelo: "", circulando: ...}
    marca: ""
    modelo: ""
    circulando: ""
    circula: undefined
    marcaModelo: " "
  ▶ [[Prototype]]: {}
▼ Object {circulando: "true", marca: "ford", ...}
    circulando: "true"
    marca: "ford"
    modelo: "fiesta"
    circula: undefined
    marcaModelo: "ford fiesta"
  ▶ [[Prototype]]: (5) {marca: "", modelo: "", circulando: ...}
```

“this” refers to properties of the object which is making the call

Inheritance

2.- OO programming

script.js

```
let persona={
    nombre:"",
    apellidos:""
}

let administrador={
    administrador:"true"
}

Object.setPrototypeOf(administrador,persona);
administrador.nombre="perico";
for (let propiedad in administrador){
    let propio=administrador.hasOwnProperty(propiedad);
    if (propio){
      console.log (propiedad+":"+administrador[propiedad]);
    }else
      console.log (propiedad+" es heredado");
}
```

Console

```
administrador:true
nombre:perico
apellidos es heredado
```

A distinction between local and inherited properties can be made BEFORE assigning a value to the former

Creating classes

script.js

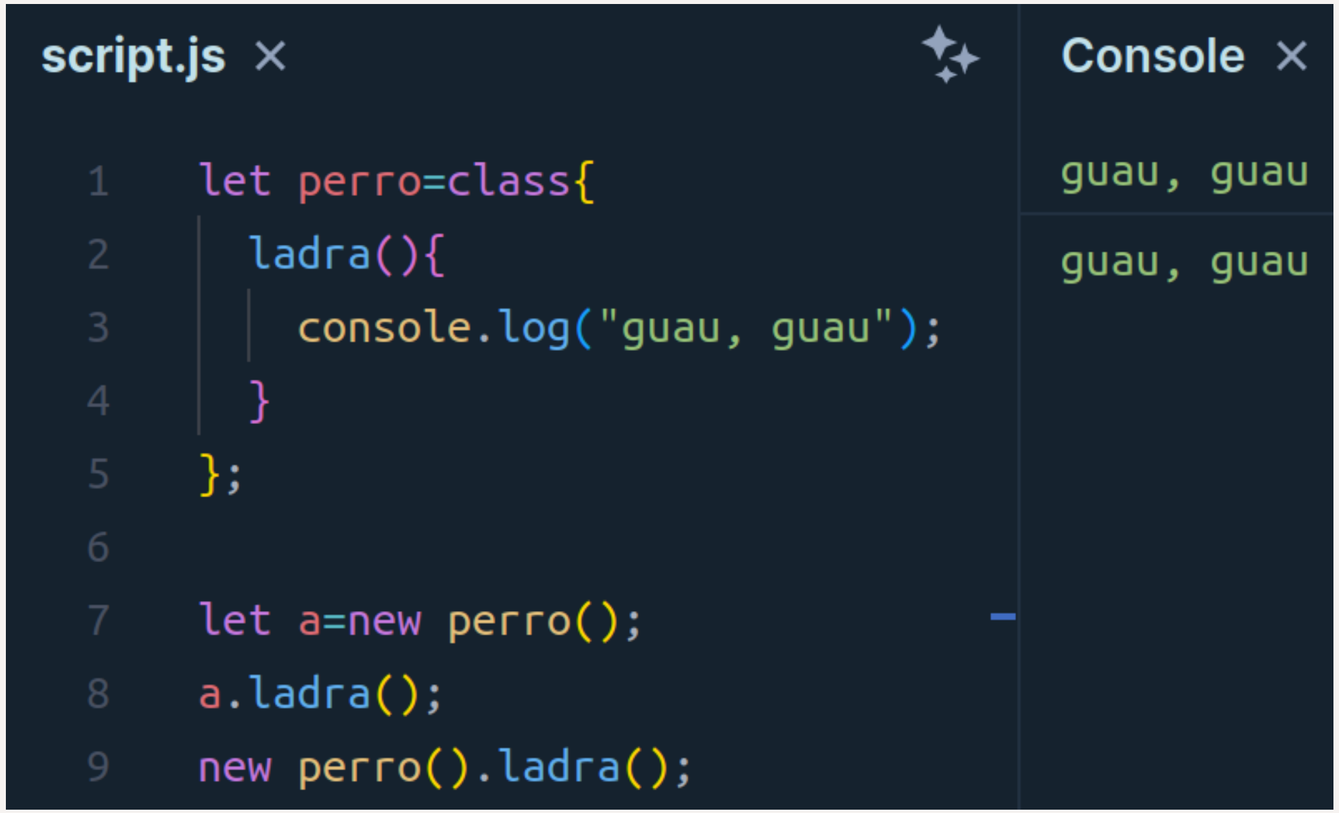
```
1   class User {
2       //automatically called by new
3       constructor(name) {
4         this.name = name;
5       }
6       //no comma to separate methods
7       sayHi() {
8         console.log(this.name);
9       }
10  }
11
12  let user = new User("Apolinar");
13  user.sayHi();
```

Console

```
Apolinar
```

creating class with new

Creating classes



class as an expression

Getters and setters


```
let lechoncillo=class lechon{

  constructor(nombre, edad){
    this.nombre=nombre;
    this.edad=edad;
  }

  //do not use the same of a property as a getter or setter
  get edad(){
    return this.anyos;
  }

  set edad(anyos){
    this.anyos=anyos;
  }

  //alternative getter and setter
  getNombreMarrano(){
    return this.nombre;
  }

  setNombreMarrano(nombre){
    this.nombre=nombre;
  }
}

let marranillo=new lechoncillo("pipas", 5);
console.log (marranillo.edad);  //edad is a getter. Look at the syntax
marranillo.edad=2;  //Edad is a setter. with this syntax, they are called with equal i

//with this alternative setter and getter, parenthesis must be used
marranillo.setNombreMarrano("puerquín");
console.log (marranillo.getNombreMarrano());
```

```
5
puerquín
```

getters and setters

This context

script.js

```
1   class Button {
2     constructor(value) {
3       this.value = value;
4     }
5     click() {
6       console.log(this.value);
7     }
8     clack = () => { //fixes this losing problem
9       console.log(this.value);
10    }
11  }
12
13  let button = new Button("hello");
14  setTimeout(button.click, 1000); // undefined.
15  setTimeout(button.clack, 1000);
```

Console

```
undefined
hello
```

losing “this” problem

Class extension

2.- OO programming > Classes

script.js

```
class Animal {
  constructor(nombre) {
    this.velocidad = 0;
    this.nombre = nombre;
  }
  corre(velocidad) {
    this.velocidad = velocidad;
    console.log(`${this.nombre} corre a una velocidad de ${this.velocidad}.`);
  }
  para() {
    this.velocidad = 0;
    console.log(`${this.nombre} se queda quieto.`);
  }
}

class Conejo extends Animal{
  salta(distancia){
    this.distancia = distancia;
    console.log(`${this.nombre} salta a una distancia de ${this.distancia}.`);
  }
}

let conejo= new Conejo("Fufo");
conejo.para();
conejo.salta(5);
```

Console

```
Fufo se queda quieto.
Fufo salta a una distancia de 5.
```

class extension

Class extension

2.- OO programming > Classes

script.js

```
function creaClase() {
  return class {
    constructor(nombre) {
      this.velocidad = 0;
      this.nombre = nombre;
    }
    corre(velocidad) {
      this.velocidad = velocidad;
      console.log(`${this.nombre} corre a una velocidad de ${this.velocidad}.`);
    }
  };
}

class animal extends creaClase() {}
new animal("pelusa").corre(4);
a=new animal("mancha");
a.corre(3);
```

Console

```
pelusa corre a una velocidad de 4.
mancha corre a una velocidad de 3.
```

using a function to create a class and extending it

Method overriding

2.- OO programming > Classes

script.js

```
class Animal {
  constructor(nombre) {
    this.velocidad = 0;
    this.nombre = nombre;
  }
  corre(velocidad) {
    this.velocidad = velocidad;
    console.log(`${this.nombre} corre a una velocidad de ${this.velocidad}.`);
  }
  para() {
    this.velocidad = 0;
    console.log(`${this.nombre} se queda quieto.`);
  }
}

class Conejo extends Animal{
  para(){ //will override parent's method "para"
    super.para(); //references parent's method "para"
    this.escondete();
  }
  escondete(){
    console.log(`${this.nombre} se esconde en su madriguera`);
  }
  salta(distancia){
    this.distancia = distancia;
    console.log(`${this.nombre} salta a una distancia de ${this.distancia}.`);
  }
}

let conejo= new Conejo("Fufo");
conejo.para();  //method from conejo, not animal
conejo.corre(6);
conejo.salta(5);
```

Console

```
Fufo se queda quieto.
Fufo se esconde en su madriguera
Fufo corre a una velocidad de 6.
Fufo salta a una distancia de 5.
```

Method overriding

Method overriding

script.js

```
class Animal {
  constructor(nombre) {
    this.velocidad = 0;
    this.nombre = nombre;
  }
  corre(velocidad) {
    this.velocidad = velocidad;
    console.log(`${this.nombre} corre a una velocidad de ${this.velocidad}.`);
  }
  para() {
    this.velocidad = 0;
    console.log(`${this.nombre} se queda quieto.`);
  }
}

class Conejo extends Animal{
  para(){ //will override parent's method "para"
    super.para(); //references parent's method "para"
    this.escondete();
  }
   escondete(msg=2000){
    //it doesn't work. When using functions, super only works with arrow functions
    //setTimeout(function() { super.stop() }, msg);
    setTimeout(() => {
        super.para();
        console.log(`${this.nombre} se esconde en su madriguera durante ${msg/1000} segundos`)
     },msg); // call parent "para" after 1sec
   }
  salta(distancia){
    this.distancia = distancia;
    console.log(`${this.nombre} salta a una distancia de ${this.distancia}.`);
  }
}

let conejo= new Conejo("Fufo");
conejo.para();  //method from conejo, not animal
conejo.corre(6);
conejo.salta(5);
```

Console

```
Fufo se queda quieto.
Fufo corre a una velocidad de 6.
Fufo salta a una distancia de 5.
Fufo se queda quieto.
Fufo se esconde en su madriguera durante 2 s
```

When using functions, super only works with arrow functions

Method overriding

2.- OO programming > Classes

script.js ×

```
function creaClase(nombre) {
  return class {
    constructor(nombre) {
      this.velocidad = 0;
      this.nombre = nombre;
    }
    corre(velocidad) {
      this.velocidad = velocidad;
      console.log(`El ${this.nombre} corre a una velocidad de ${this.velocidad}.`);
    }
  };
}

class animal extends creaClase() {}
let perro=new animal("galgo"); // Hola
perro.corre(4);
```

Console ×

```
El galgo corre a una velocidad de 4.
```

extending form a class created with a function

Method overriding

2.- OO programming > Classes

script.js

```
class Animal {
  constructor(nombre) {
    this.velocidad = 0;
    this.nombre = nombre;
  }
}

class Conejo extends Animal{
  constructor(nombre, tamanyoOrejas){
    //super has to be called this way in order for the parent to create an object for "this"
    super(nombre);
    this.tamanyoOrejas=tamanyoOrejas;
  }
}

let conejo = new Conejo("Conejito pequeño", 5);
console.log(conejo.nombre);
console.log(conejo.tamanyoOrejas);
```

Console

```
Conejito pequeño
5
```

When overriding constructor, super has to be called for an object for “this” to be created. Otherwise, it throws an error

Static methods

2.- OO programming > Classes

script.js

```
class Animal {
  constructor(nombre, velocidad=0) {
    this.velocidad = velocidad;
    this.nombre = nombre;
  }
  corre(velocidad) {
    this.velocidad = velocidad;
    console.log(`${this.nombre} corre a una velocidad de ${this.velocidad}.`);
  }
  static comparaVelocidad(animal1, animal2){
    if (animal1.velocidad-animal2.velocidad<0)
      return "corre más "+animal2.nombre;
    else
      return "corre más "+animal1.nombre;
  }
}

class conejo extends Animal{};
class perro extends Animal{};


let conejito=new conejo("achuchao",7);
let perrito=new perro("cibeles",3);
console.log(Animal.comparaVelocidad(conejito, perrito));
```

Console

```
corre más achuchao
```

Static methods are assigned to a class rather than to an object. They are summoned with no need to instantiate the class

Static methods

2.- OO programming > Classes

script.js

```
class Animal {
  constructor(nombre, maxVelocidad) {
    this.maxVelocidad = maxVelocidad;
    this.nombre = nombre;
  }

  corre(velocidad) {
    this.velocidad = velocidad;
    console.log(`${this.nombre} corre a una velocidad de ${this.velocidad}.`);
  }

  static comparaMaxVelocidad(animal1, animal2){
   return animal1.maxVelocidad-animal2.maxVelocidad;
  }
}

class conejo extends Animal{};
class perro extends Animal{};

let animales=[
  new conejo("achuchao", 7),
  new perro("roque", 2),
  new perro("cibeles", 8)
];

animales.sort(Animal.comparaMaxVelocidad);
for (let animal of animales){
  console.log(animal.nombre);
}
```

Console

```
roque
achuchao
cibeles
```

Static methods are assigned to a class rather than to an object. They are summoned with no need to instantiate the class

Static properties

2.- OO programming > Classes

script.js

```
1  class Animal {
2    static fechaCompra="1/1/2023";
3    constructor(nombre, maxVelocidad) {
4      this.maxVelocidad = maxVelocidad;
5      this.nombre = nombre;
6    }
7
8    corre(velocidad) {
9      this.velocidad = velocidad;
10     console.log(`${this.nombre} corre a una velocidad de ${this.velocidad}.`);
11   }
12
13   static comparaMaxVelocidad(animal1, animal2){
14    return animal1.maxVelocidad-animal2.maxVelocidad;
15   }
16 }
17
18 class conejo extends Animal{};
19 class perro extends Animal{};
20 let a=new conejo("achuchao", 7);
21 let b=new perro("cibeles",3);
22
23 //Animal.corre(4);  //Error, not an static method. Need to be summoned by an object
24 console.log (Animal.comparaMaxVelocidad(a,b));
25 Animal.fechaCompra="2/2/2022";  //it works, static property
26 console.log(Animal.fechaCompra, a.fechaCompra);
27 Animal.nombre="papa"; //don't work. Not a static property
28 console.log (a, b)
29 //a.comparaMaxVelocidad(a,b); //Error, function doesn't exist. It's a static method
```

Console

```
4
2/2/2022  undefined
▸ conejo {maxVelocidad: 7, nombre: "achuchao"...}
▸ perro {maxVelocidad: 3, nombre: "cibeles"}
```

Static methods are assigned to a class rather than to an object. They are summoned with no need to instantiate the class

Private properties

script.js

```
class cafetera{
  constructor (potencia){
    this._potencia= potencia;
  }
  get potencia(){
    return this._potencia;
  }
}

let cafetera1=new cafetera(100);
//underscore is a convention for private properties.
//Nothing prevents user from changing its value
cafetera1._potencia=225;
console.log (cafetera1.potencia);
```

Console

```
225
```

Private properties are declared using underscore. They are meant to be accessible only within the class, but it is a convention. Nothing prevents user from accesing then from outside

Protected properties and methods

script.js

```
class tostadora{
  #tamanyoDelPan;
  constructor (potencia, tamanyoPan){
    this._potencia=potencia;
    this.#tamanyoDelPan=10;
  }

  //an alternative way of expressing a getter
  getPotencia(){
    return this._potencia;
  }

  //an alternative way of expressing a setter
  #setPotencia(potencia){
    this._potencia=potencia;
  }

  set tamanyoPan(tamanyo){
    this.#tamanyoDelPan=tamanyo;
  }

  get tamanyoPan(){
    return this.#tamanyoDelPan;
  }
}
```

script.js

```
let tostadora1=new tostadora(100, 8);
//underscore is a convention for expressing a property to be protected, meaning
//not accessible outside the class but nothing prevents user from changing their value
tostadora1._potencia=225;
console.log (tostadora1._potencia);

//Sharp makes a property private, meaning no access outside the class
//tostadora1.#tamanyoDelPan=10;  //Error (it's private)
//traditional way of using setters and getters (with equal and no parenthesis)
tostadora1.tamanyoPan=10;
console.log (tostadora1.tamanyoPan);

//alternative way of using getters and setters (with parenthesis)
//tostadora1.setPotencia(10);  //Error, no such a function (it's private)
tostadora1.getPotencia();
```

Console

```
225
10
```

Protected properties and methods are declared using sharp. They are accessible only within the class

Protected properties and methods inheritance

script.js

```
class electrodomestico{
  #potencia
  constructor (potencia){
    this.#potencia=potencia;
  }

  getPotencia(){
    return this.#potencia;
  }

  #setPotencia(potencia){
    this.#potencia=potencia;
    //not working. Private properties do not allow brackets
    //this[#potencia]=potencia;
  }
}
```

script.js

```
class batidora extends electrodomestico{
  constructor(nombre, modelo, potencia){
    super(potencia);
    this.nombre=nombre;
    this.modelo=modelo;
  }

  getPotencia(){
    //return this.#potencia;  //error: private name #potencia not defined
    //return super.#potencia;  //error: private fields can't be accessed on super
    return super.getPotencia();
  }

  setPotencia(potencia){
    //super.#setPotencia(250);  //error: private fields can't be accessed on super
  }
}

let b=new batidora("ufesa", "ax23", 500);
console.log (b.getPotencia());
```

Console

```
500
```

Protected properties and methods can’t be inherited

Check object instance of class

script.js

```
1   //allows to check if an object is an instance of a class
2   class Electrodomestico{};
3   let electro=new Electrodomestico();
4
5   // ¿Es un objeto de la clase Electrodoméstico?
6   console.log( electro instanceof Electrodomestico ); // verdadero
7
```

Console

```
true
```

instanceof allows to check if an object is an instance of a class

Check object instance of class

script.js

```
1   //allows to check if an object is an instance of a class
2   class Electrodomestico{};
3   let electro=new Electrodomestico();
4
5   // ¿Es un objeto de la clase Electrodoméstico?
6   console.log( electro instanceof Electrodomestico ); // verdadero
7
```

Console

```
true
```

instanceof allows to check if an object is an instance of a class

Mixins

2.- OO programming > Classes

script.js

```
class Base1 {
  constructor() {
    this.contador = 0;
  }
  aumenta() {
    this.contador++;
    return this;
  }
  muestra(){
    console.log(this.contador);
    return this;
  }
}

let miMixin = (claseBase) => class extends claseBase {
  constructor(nombre) {
    super();
    this.name = nombre;
  }
  resta() {
    this.contador--;
    return this;
  }
}
```

script.js

```
class Extendida extends miMixin(Base1) {
  reinicia() {
    this.contador = 0;
    return this;
  }
}

let miObjeto = new Extendida("perico");
//miObjeto.saludar();
miObjeto.aumenta().aumenta().muestra().reinicia().muestra();
```

Console

```
2
0
```

Mixins are the only way for a JavaScript class to inherit from two objects