Unit 2

BASIC JAVASCRIPT SYNTAX AND STRUCTURES

DESARROLLO WEB EN ENTORNO CLIENTE 24/25

ALEJANDRO VIANA RÍOS

Unit 2

Basic Javascript syntax and structures

Index

1.- Code quality

Code quality

1.- Code quality

- JavaScript is really flexible but for the sake of debuggin some tips should be followed
- Some code-style makes easier to understand, modify and debug a program

```
1   //Bad coding style. Confusing for newers. Error-prone
2
3   i=5
4   j=0
5   i != j ? i > 0 ? console.log(Math.max(i, j)) : console.log(i) : console.log(0);
6
7   //Same code than above. Brackets could be removed, but for the sake of clarity, it's better to leave them
8   let i=5;
9   let j=0;
10  if ( i!=j ){
11      if ( i>0 ){
12          console.log( Math.max( i,j ));
13      }else{
14          console.log( i );
15      }
16  }else{
17      console.log( 0 );
18  }
```

Lines 3-5 do the same than 8-18, but the latter are much easier to understand and debug

Variables, sentences and comments

1.- Code quality

- Variables
  - Use descriptive and differente names. Caution with date and data
  - Declare them with let and initialize at creation, if possible.
  - Do not change them type during their life
  - Do not reuse variables. Declare new ones
  - Use const when a variable is not changing its value ever
- Sentences
  - Do not split one sentence into several lines
  - Place one sentence per line. Do not place several sentences in the same line (although using semicolons)
  - Use semicolons to finish each sentence although JavaScript automatically adds it
- Comments: A good code should be almost self-explanatory, huge comments explaning everything are a symtom of bad coding

Functions

1.- Code quality

- If, by reading its name, it is supposed not to have any side-effect, it should not! (isPresent, checkPermission, etc.)
- Avoid returning non standard values like objects in functions liks checkPermission (should return T/F)
- They should do what is adviced on their name. Nothing less, nothing more. One function, one action
- Try to replace code with functions

```
function showPrimes(n) {
  nextPrime:
  for (let i = 2; i < n; i++) {

    // check if i is a prime number
    for (let j = 2; j < i; j++) {
      if (i % j == 0) continue nextPrime;
    }

    alert(i);
  }
}
```

```
function showPrimes(n) {

  for (let i = 2; i < n; i++) {
    if (!isPrime(i)) continue;

    alert(i);
  }
}

function isPrime(n) {
  for (let i = 2; i < n; i++) {
    if (n % i == 0) return false;
  }

  return true;
}
```

Replacing code with function makes it easier to understand and mantain

Functions

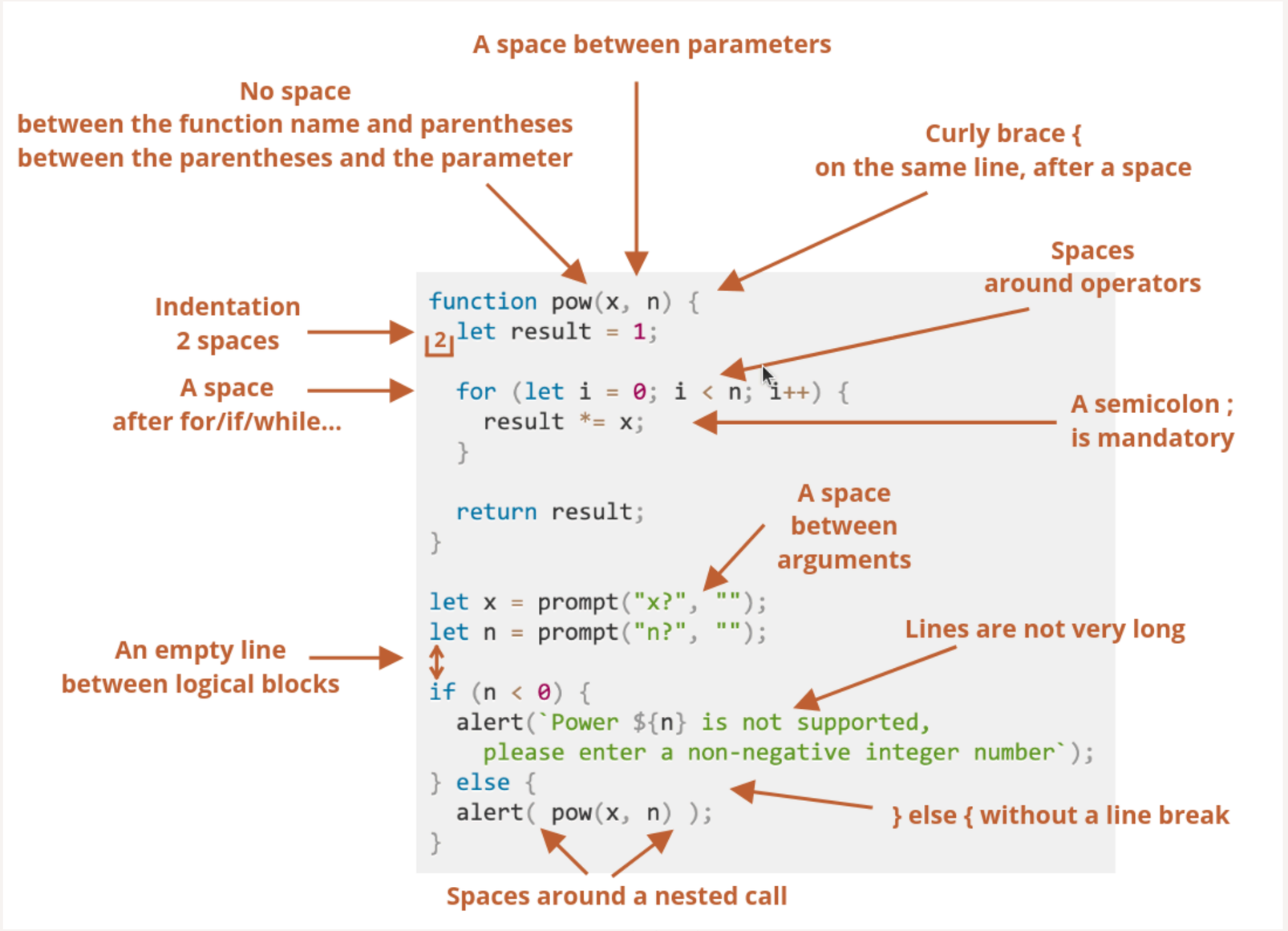
1.- Code quality

- Try to abstract code to aviod creating similar ones like, for instance, printScreen and printPrinter
- Describe what they do, their inputs and outputs. Some tools can automatically generate documentation (like jsdoc)
- Describe why the problem was solved this way and if there are more ways for solving it

```
/**
 * Returns x raised to the n-th power.
 *
 * @param {number} x The number to raise.
 * @param {number} n The power, must be a natural number.
 * @return {number} x raised to the n-th power.
 */
function pow(x, n) {
  ...
}
```

Code quality

1.- Code quality



Recommended coding way

Notation

1.- Code quality

- Keep consistency when using notation
  - Camel case: contarPalabrasImpares
  - Pascal case: ContarPalabrasImpares
  - Snake case: contar_palabras_impares
  - Kebab case: contar-palabras-impares

2.- Interacting with users

Interacting with user

2.- Interacting with users

script.js

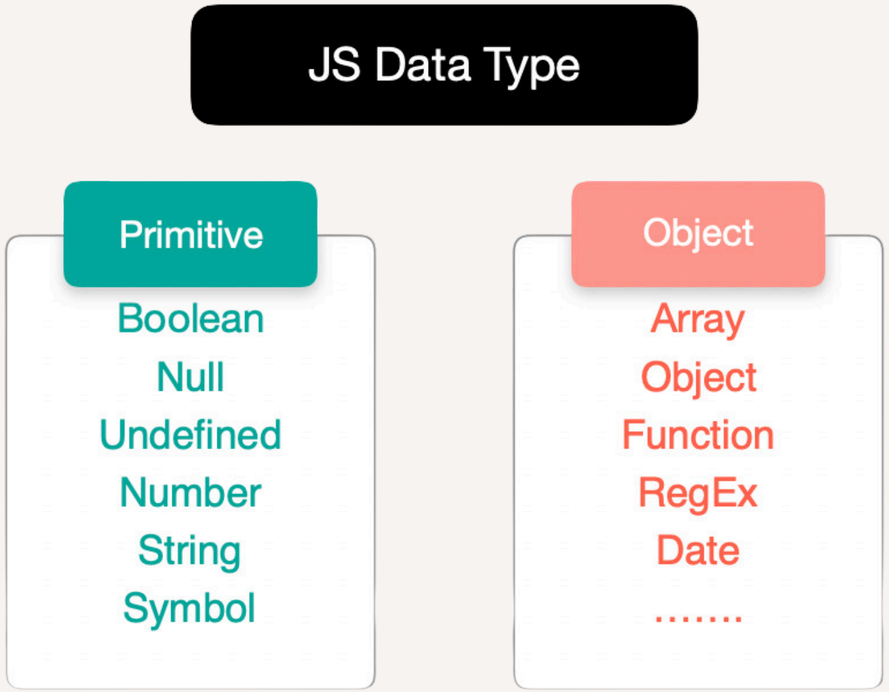
```
1   //shows up a modal window with the message and waits for the user to press ok
2   alert ("esto es un mensaje");
3
4   //shows up a modal window asking the user for a value
5   let numero=prompt("dame un número", "numero");
6
7   //shows a message at the console
8   console.log("esto es un texto");
9
10  //shows up a modal window asking the user for confirmation
11  let respuesta=confirm("¿acepta?");
```

Console

```
esto es un texto
```

4 ways of interacting with users: alert, console, confirm and prompt

3.- Data types



1. Number
2. Bigint
3. String
4. Boolean
5. Null
6. Undefined
7. Object

Primitive data types

3.- Data types

Number

- Integer or floating point
- Special values: Infinity, -Infinity, NaN (not a number)
- erroneous operations can be done and JavaScript will never stop (you will get NaN)

BigInt

- Allows to represent integers larger than (2^53)-1
- Is created by appending a “n” at the end of an integer number

String

- Chain of symbols (letters, numers, special symbols), treated as one
- Must be surrounded by double, simple quotes or backticks
- Double or simple quotes are the same. Backticks are extended functionality, allowing to embed variables or expresions by wrapping them in ${var}

Primitive data types

3.- Data types

Boolean

- Logic value that can be true or false

Null

- Intentional lack of value
- Developer assigns to a recently created variable

Undefined

- It is also a lack of value, but unintentionally
- Developer just creates variable and assigns no value

Non-primitive data types: Objects

3.- Data types

- They are a category of complex data types
- They can contain data collections (pair key-value) and functionality (methods)
- There are some kinds of objects:
  - simple ones as {}
  - complex ones as arrays, classes or functions

Objects

3.- Data types > Non-primitive data types

Array

- It’s a subtype of an object
- Ordered list of values of any type enclosed into brackets
- [1, 2, 3, “pepe”, “jose”, true]

Function

- An object representing an executable chunk of code that can be called anywhere
- function sumar (a,b){ return a+b;}

Date

- An object to work with dates and time
- new Date();

RegExp

- An object to work with regular expressions
- /\d+/

Iterables and non-iterables

3.- Data types

Iterables

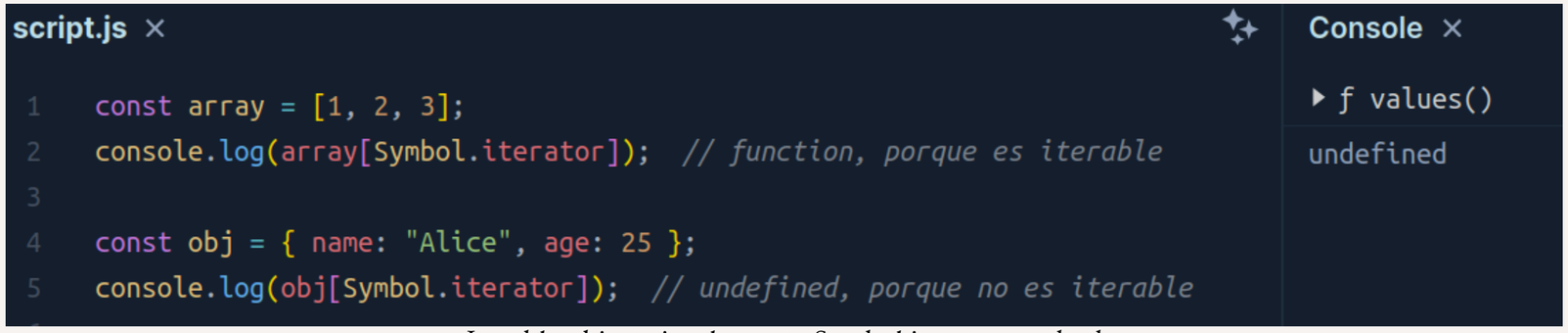
- Their elements can be secuentially accessed by using iterative structures like for...of, spread operators (...) or methods like Array.from()
- Implements Symbol.iterator method
- Examples: Arrays, Strings, Maps, Sets, NodeLists (document.querySelectorAll())

Non iterables

- Their elements can be accessed by using for...in or Object.keys()
- Don’t implement Symbol.iterator method
- Examples: Objects, numbers, booleans or Errors

Iterables

3.- Data types



Iterable objects implements Symbol.iterator method

4.- Variables

Recommendations

4.- Variables

- JavaScript is a weakly typed language: User does not need to assign any type when creating the variable and it can change (not recommended)
- It is recommended not to change variable type
- It is recommended to initialize a variable when creating
- It is only allowed to use number, letters, _ and $ for naming

Creation

4.- Variables

```
let numero1=5, numero2=10, cadena="hola don pepito";
let numero3=1,
    numero4=2,
    numero5=3;
let numero6=4;
let numero7=5;
```

Creating variables

Creation

4.- Variables

```
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);
```

Console ×

```
hola  adios
```

When duplicating variables, they point to different memory locations. They are different variables. It won't happen the same with objects

Variables

4.- Variables

- Use human-readable names like userName or shoppingCart.
- Stay away from abbreviations or short names like a, b, c, unless you really know what you’re doing.
- Make names maximally descriptive and concise. Examples of bad names are data and value.
- Be consistent. Should a site visitor is called a “user”, related variables should be named currentUser instead of currentVisitor
- Allways use lowercase except with constants whose value is known prior to execution

Creation

4.- Variables

```
let numero1=5;
let numero2=10;
let cadena="hola don pepito";
let asignaturas=["Desarrollo web en entorno cliente", "Diseño de interfaces web"];
let persona= {nombre:"procopio", apellido: "Maximus"};
let constante="esto no debería cambiar";

console.log ( numero1 + cadena );
console.log ( `La suma de ${numero1} y ${numero2} da ${numero1+numero2}` );
console.log ( numero1+numero2 );
console.log( asignaturas[1] );
console.log( persona.nombre )
console.log ( typeof(numero1), typeof(cadena), typeof(asignaturas), typeof(persona), typeof( constante ) );

```

Console ×

```
5hola don pepito
La suma de 5 y 10 da 15
15
Diseño de interfaces web
procopio
```

Some ways of declaring strings

Strings declaration

script.js

```
let single = 'single-quoted';
let double = "double-quoted";
let backticks = `backticks`;

let suma=1+2;
console.log (`suma vale ${suma}`);

let lista= `Lista de compra:
    *Pepinillos
    *Tomates
    *Calabacines`;  //allowed

/*
let lista2= "Lista de compra:
    *Pepinillos
    *Tomates
    *Calabacines";  //not allowed*/
```

Console

```
suma vale 3
```

Some ways of declaring strings

5.- Data type conversion

String conversion

5.- Data type conversions

script.js

```
1   //Mal por llamar cadena a un booleano
2   let cadena = true;
3   console.log (typeof cadena); // boolean
4
5   cadena = String(cadena) // now value is a string "true"
6   console.log (typeof cadena); // string
```

Console

```
boolean
string
```

Typeof checks variable data type and String converts to a String

Number conversion

5.- Data type conversions

script.js

```
1   console.log ( "4"/"2" );
2   let cadena2="123";
3
4   console.log (typeof(cadena2));
5   let num=Number(cadena2);
6   console.log (typeof(num));
```

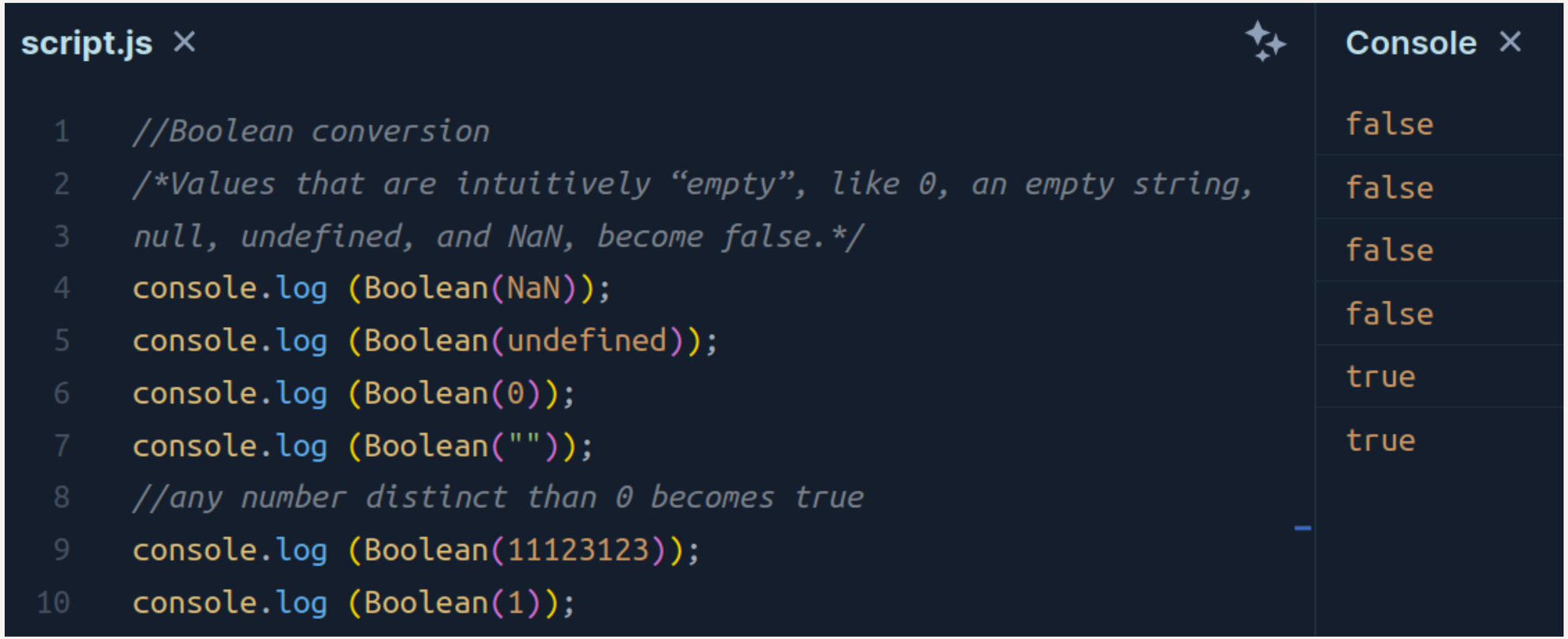
Console

```
2
string
number
```

Integer conversion

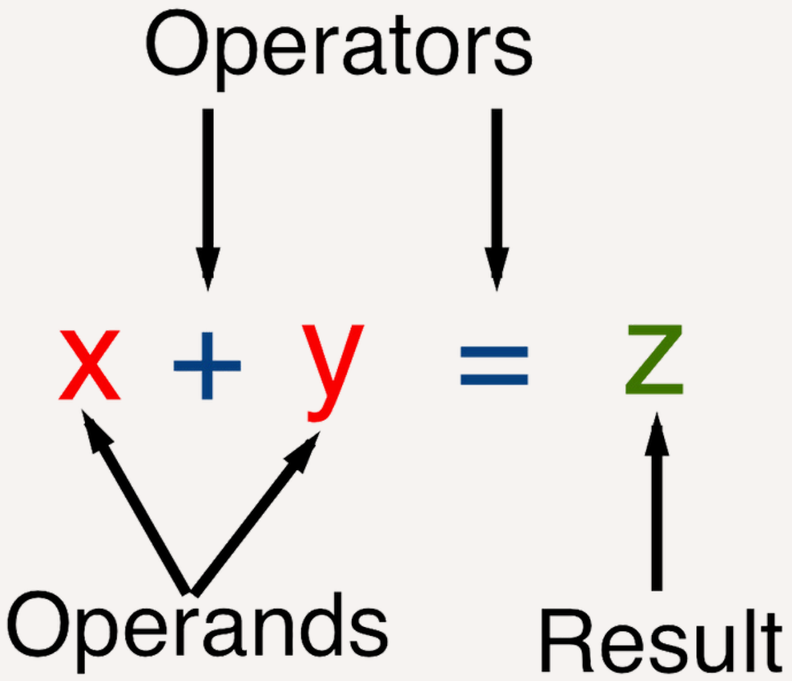
Boolean conversion

5.- Data type conversions



Boolean conversion

6.- Operators



1. String
2. Number
3. Assignment
4. Modify-in-place
5. Comparison
6. Logical

String operators

6.- Operators

script.js

```
1   //string contatenation
2   let cadena="hola",
3       cadena2="amados",
4       cadena3="alumnos",
5       cadena4='7',
6       cadena5='10';
7   console.log(cadena+" "+cadena2+" "+cadena3);
8   console.log(5+5+'4');
9
```

Console

```
hola amados alumnos
104
```

String concatenation

Number operators

6.- Operators

script.js

```
1   //Math operators: +, -, *, /, % (remainer, resto), ** (exponentiation, exponente)
2   let cadena4='7',
3       cadena5='10';
4   let numero=5,
5       numero2=12;
6   console.log (numero2%numero);
7   console.log (numero**2);
8   console.log(+cadena4 + -cadena5);   //cadenas are converted into numbers thanks to the +/- operator
9   let numero_negativo=-numero;
10  console.log (numero_negativo);
11
12  console.log(numero2);
13  numero2-=numero;
14  console.log(numero2);
15
16  //Be careful with operators precedence. It may change the result of the operation
17  let x=2*2+1;
18  console.log(x);
19  console.log(3+10*2, 3+(10*2), (3+10)*2);
```

Console

```
2
25
-3
-5
12
7
5
23  23  26
```

Numeric operators

Assignment operator

6.- Operators

script.js

```
2   //Assignment
3   //= is also an operator, returning a value. Expressions are evaluated from right to left, taking into account operator precedence
4   let a=4;
5   let b=5;
6   let c=4-(a=b+1);    //confusing syntax. Not recommended
7   let d=a=b=4+3;
8   console.log (c, d);
```

Console

```
-2  7
```

Assignment operator

Modify-in-place operators

6.- Operators

script.js

```
let n=5;
let c;
c=n--;  //first n value is returned and then n value is decremented
console.log(c);
c=--n;  //first n value is decremented and then n value is returned
console.log(c);

console.log(n=n+5, n+=5);
console.log(n);
console.log(n*=2);
console.log(n++, ++n);
console.log(n);
console.log(n--, --n);
```

Console

```
5
3
8  13
13
26
26  28
28
28  26
```

Modify-in-place operator

Comparison

6.- Operators

script.js

```
let num1=3,
    num2=num3=5;

let cad1="hola",
    cad2="Hola",
    cad3="holas";

//unicode is used to determine the order of the letters
console.log( num1>num2 );
console.log( num2==num3 );
console.log( num1!=num2 );
console.log("----------------------------");
console.log( cad1>cad2 );
console.log( cad1==cad2 );
console.log( cad1=cad2 );
console.log("----------------------------");
console.log( "4">num1 );

/*Equality cannot differentiate 0 or empty string from false
due to the fact that empty string, and false, are converted to zero*/
console.log("----------------------------");
console.log( 0==false );
console.log( ""==false );

/*therefore, if differentiate is needed, strict equality must be used
strict equality do not convert any data type so, if they are not the same
type, it returns false*/
console.log("----------------------------");
console.log( 0===false );
console.log( ""===false );

//be carefull comparing with a variable that could take null or undefined value
console.log("----------------------------");
console.log( a>4 );
let a=3;
console.log( a>4 );
```

Console

```
false
true
true
----------------------------
true
false
Hola
----------------------------
true
----------------------------
true
true
----------------------------
false
false
----------------------------
false
false
```

Logical operators

6.- Operators

script.js

```
1  ///////logical operators///////
2  let a = (d = true),
3    b = (c = false); //remember that 0, empty string, Null or NaN are false, 1 or higher are true
4
5  //or operator ||
6  console.log(a || b, b || c);
7  console.log(a || b || c); //should any operand be true, evaluation is stopped and true is returned
8  if (a || b) {
9    console.log('alguno de los dos es verdadero');
10 } else {
11   console.log('los dos son falsos');
12 }
13 console.log('--------------------------');
14
15 //or processes values until first true valor is found, returning the argument and leaving the unexplored
16 console.log(a || b || c || 'ninguno');
17 c || console.log('texto'); //should c be false, texto would shown up
18 console.log('--------------------------');
19
20 //and operator (&&)
21 console.log(a && b, b && c);
22 console.log(a && b && c); //should any operand be false, evaluation is stopped and false is returned
23 a && console.log('texto'); //if a exists, texto is shown
24 a > 0 && console.log('a es mayor que cero'); //if a is greater than zero, a text is shwon. No recomendable: poco legible
25 console.log('--------------------------');
26
27 //not operator (!)
28 console.log(!(c || d) && (a || b));
```

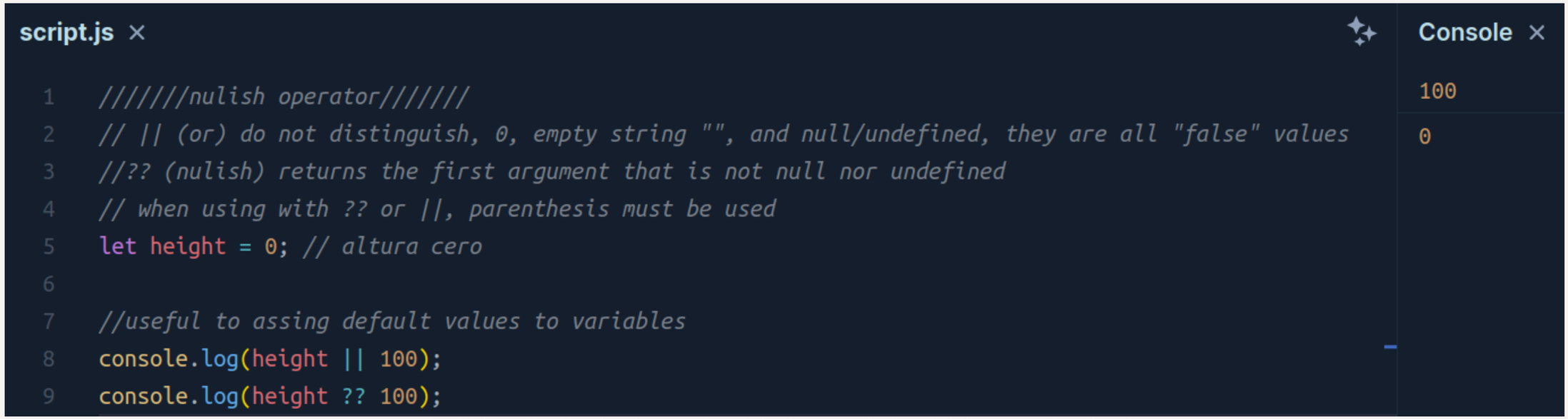
Console

```
true  false
true
alguno de los dos es verdadero
--------------------------
true
texto
--------------------------
false  false
false
texto
a es mayor que cero
--------------------------
false
```

Logical operators: ||, && and !

Logical operators

6.- Operators



Nulish coalescing (coalescencia nula) allows to distinguish between 0 and null

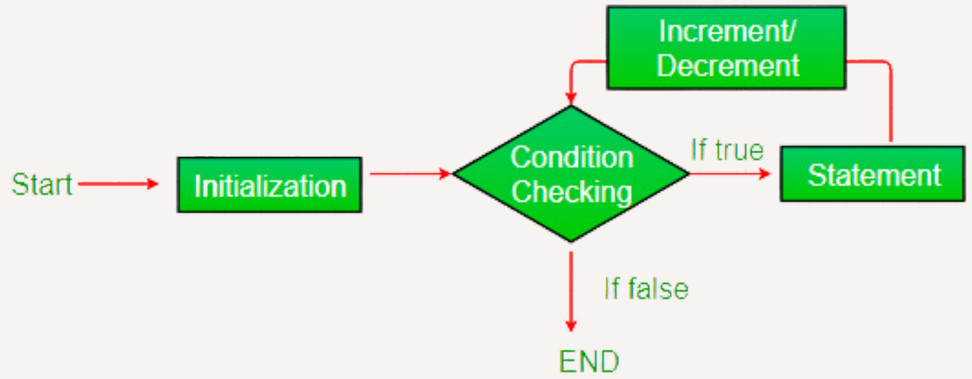
7.- Conditional execution

Condition is true

```
let number = 2;
if (number > 0) {
    // code
}
else {
    // code
}

// code after if
```

Condition is false

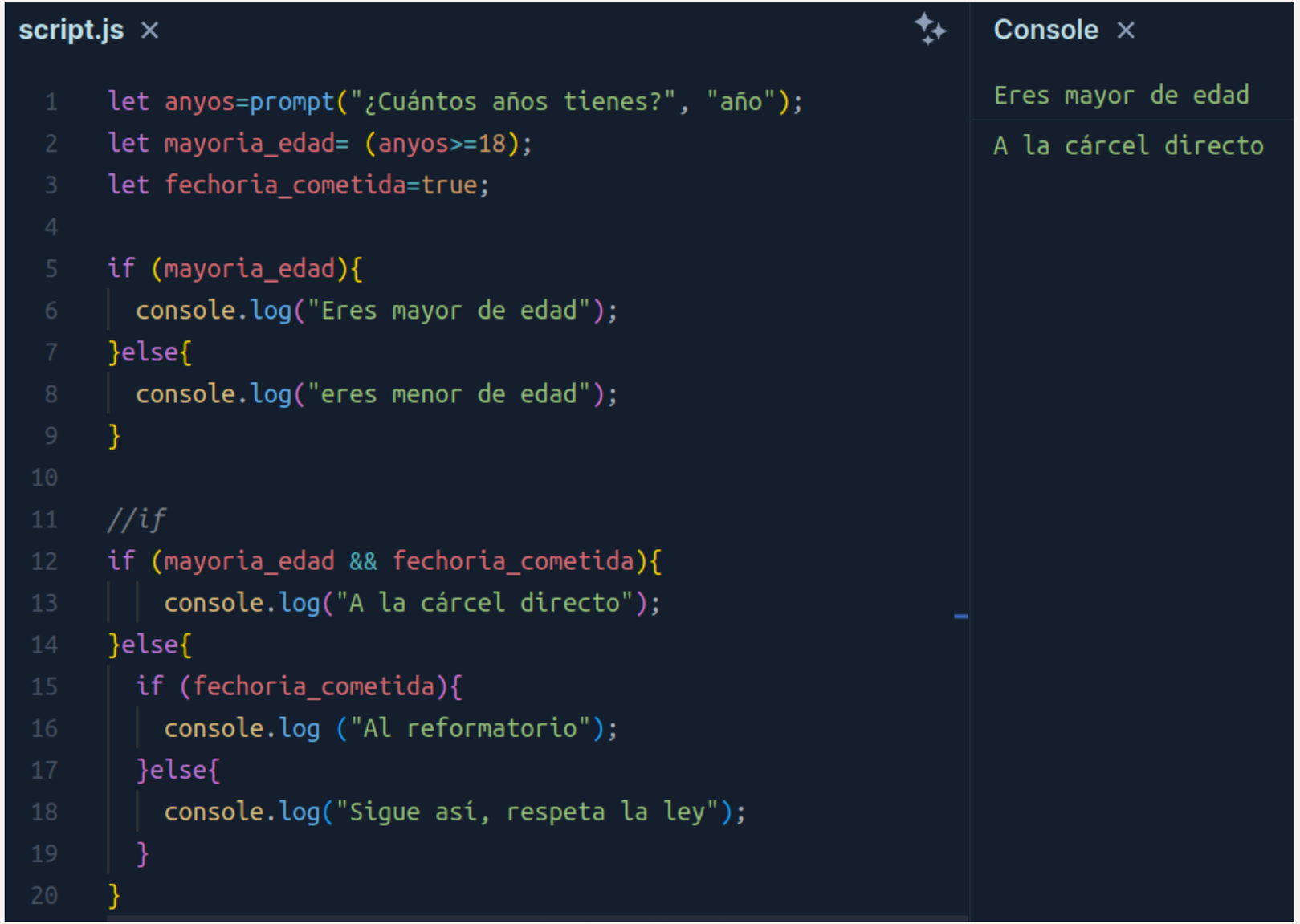
```
let number = -2;
if (number > 0) {
    // code
}
else {
    // code
}

// code after if
```



1. if..else
2. Ternary operator
3. switch

if...else

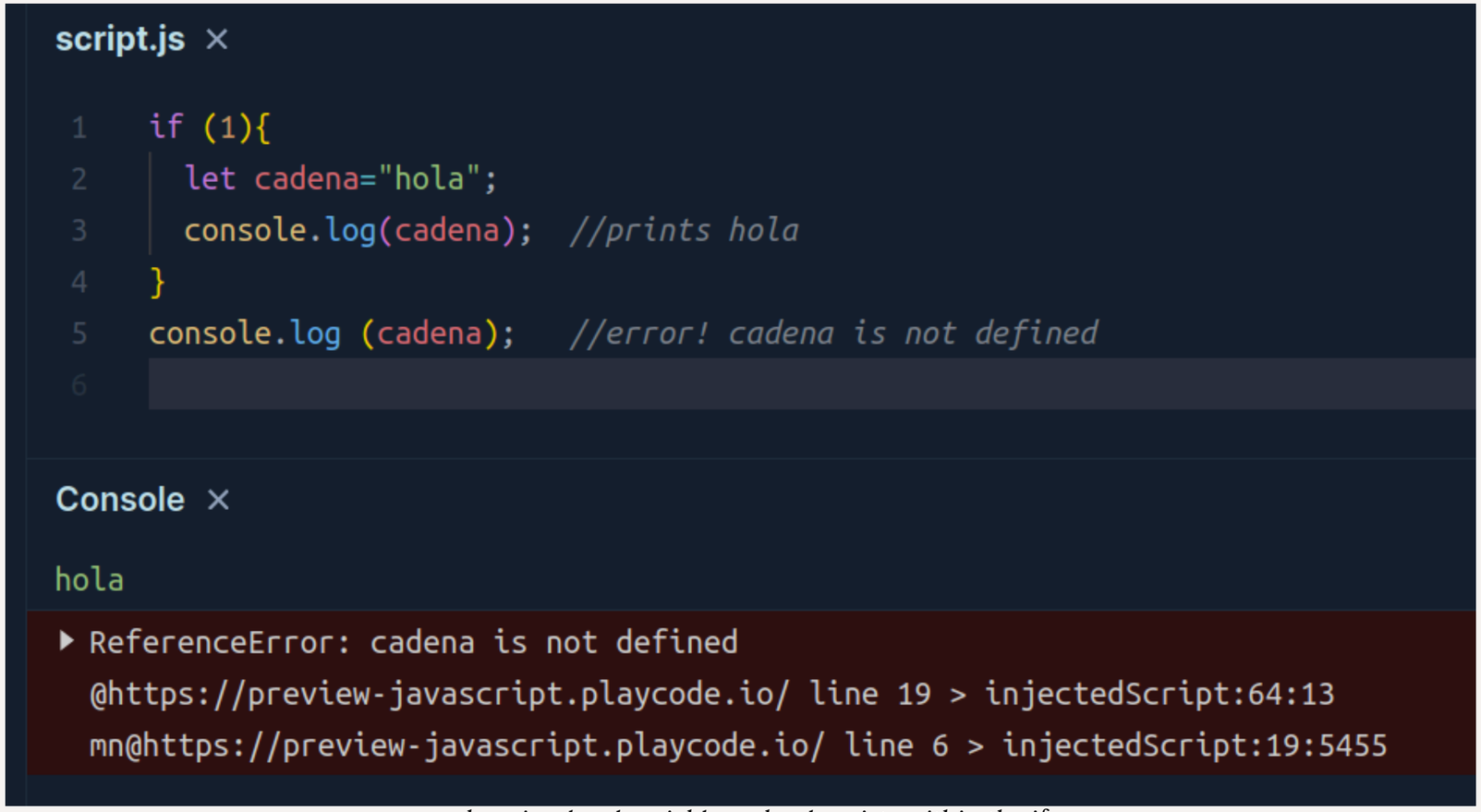
7.- Conditional execution



Conditional branching

Variable scope

7.- Conditional execution



cadena is a local variable and only exists within the if

Ternary operator

7.- Conditional execution

script.js

```
let anyos=prompt("¿Cuántos años tienes?", "año");
let mayoria_edad= (anyos>=18);
let fechoria_cometida=true;

if (mayoria_edad){
  console.log("Eres mayor de edad");
}else{
  console.log("eres menor de edad");
}

//if
if (mayoria_edad && fechoria_cometida){
    console.log("A la cárcel directo");
}else{
  if (fechoria_cometida){
    console.log ("Al reformatorio");
  }else{
    console.log("Sigue así, respeta la ley");
  }
}

/*? operator works as an if. It is shorter and appealing, but less readable.
 Be carefull, it's not very intuitive when nesting*/

//Equivalent to first if
( mayoria_edad ) ? console.log("Eres mayor de edad") : console.log("Eres menor de edad");

//Equivalent to second if
(mayoria_edad && fechoria_cometida) ? console.log("A la cárcel directo") :
    fechoria_cometida ? console.log("Al reformatorio") :
    console.log("Sigue así, respeta la ley");
```

Console

```
Eres mayor de edad
A la cárcel directo
Eres mayor de edad
A la cárcel directo
```

Conditional branching

switch

7.- Conditional execution

script.js

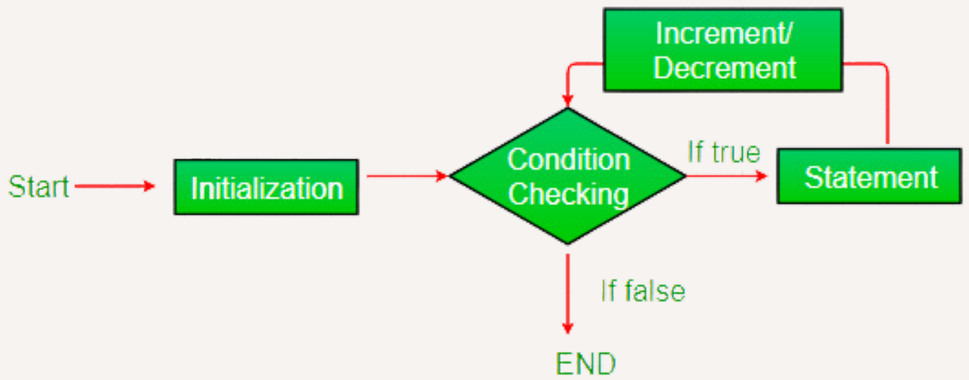
```
1   //switch
2   //this is the only structure that needs break statement
3   let nota = Number(prompt("¿Qué nota sacaste?",0));
4   switch (nota) {
5     case 0:
6     case 1:
7     case 2:
8     case 3:
9     case 4:
10      console.log( 'Tienes que estudiar más' );
11      break;
12    case 5:
13      console.log( 'Por los pelos' );
14      break;
15    case 6:
16    case 7:
17    case 8:
18      console.log( 'Bien' );
19      break;
20    default:
21      console.log( "¡Excelente!" );
22  }
```

Console

```
Tienes que estudiar más
```

switch statement

8.- Loops



1. while
2. for
3. Continue
4. Break
5. Break labels

do...while

8.- Loops

```
//do...while
//Code inside the loop is executed, at least, 1 time before checking condition
let l = 0;
do {
  console.log( l++ );
} while (l <= 3);

do{
  //something
} while (1);  //infinite loop
```

do...while

While

8.- Loops

```
//code inside the loop is executed only if the condition is true
let i = 0;
while (i < 3) {
  console.log( i++ ); //it outputs i value and then increment it
}

let j = 0;
while (j < 3) {
  console.log( ++j ); //it increments j value and then shows it
}

let k = 5;
while (k) { //elegant way to write k>0
  console.log( k-- );
}

while (1){ //infinite loop
  //something
}
```

while

for

8.- Loops

script.js

```
1   //nested loops
2   for (let m = 0; m < 3; m++) { // shows 0, then 1, then 2
3     for (let n = 0; n < 3; n++) { // shows 0, then 1, then 2
4       console.log(m,n);
5     }
6   }
7
8   //endless for
9   for (;;){
10    console.log("this loop will never end, unless you press ctrl+c");
11  }
12
```

Console

```
0 0
0 1
0 2
1 0
1 1
1 2
2 0
2 1
2 2
this loop will never end, unless you press ctrl+c
this loop will never end, unless you press ctrl+c
this loop will never end, unless you press ctrl+c
this loop will never end, unless you press ctrl+c
this loop will never end, unless you press ctrl+c
this loop will never end, unless you press ctrl+c
```

nested and endless loop

for

8.- Loops

script.js

```
const matriz= [1, 2, 3];
const obj = { name: "Alice", age: 25 };

//an array
for (let i=0; i<matriz.length; i++)
  console.log(matriz[i]);

//an object. It needs to use Object.keys(objeto), a predefined method. We'll talk about methods of predefined objects in a later chapter
//Object.keys returns an array with the keys of the object
let numero_llaves=Object.keys(obj).length
for (let i=0; i<numero_llaves; i++){
  let llave=Object.keys(obj)[i];
```

Console

```
1
2
3
Alice
25
```

iterating trough objects using regular for

for

8.- Loops

iterating trough objects using iterable or non iterable version of for

Continue

8.- Loops

```
//continue will finish the actual iteration and move on to the next one
for (i = 0; i < 10; i++) {
  // if true, skip the remaining part of the body
  if (i % 2 == 0) continue;
  console.log(i);
}
//same effect as before, but better coding
for (i = 0; i < 10; i++) {
  if (i % 2) {
    console.log( i );
  }
}

//break/continue can't be used with ?
i=0;
(i<10) ? i++ : continue;
```

Break

8.- Loops

```
//you can use break to exit the loop, while it is not recommended.
//The best way to leave it is the condition
let suma=0;
//bad coding
while (1){  //will repeat forever,
  suma++;
  if (suma==5) break;
}

//better
suma=0;
while (suma<5){
  suma++;
}

//much better
for (suma=0;suma<5;suma++);
```

break

Break labels

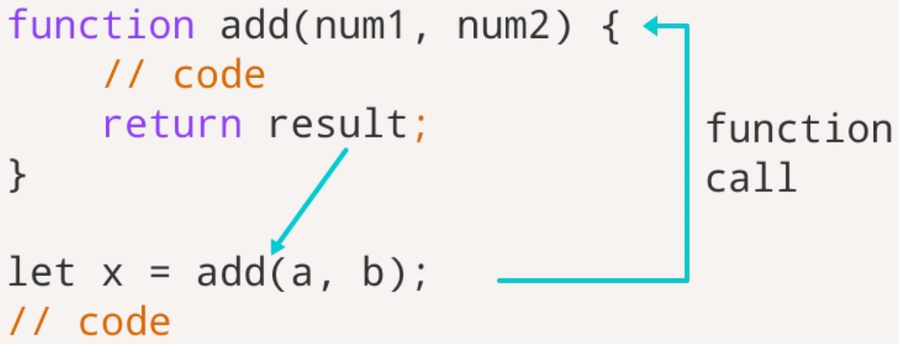
8.- Loops

```
//break only stops one loop so, in order to stop two or more, labels have to be used
salida: for (let i = 0; i < 3; i++) {
  for (let j = 0; j < 3; j++) {

    let input = prompt(`Value at coords (${i},${j})`, '');

    // if an empty string or canceled, then break out of both loops
    if (!input) break salida;

    console.log(`you have chosen ${i}${j}`);
  }
}
```

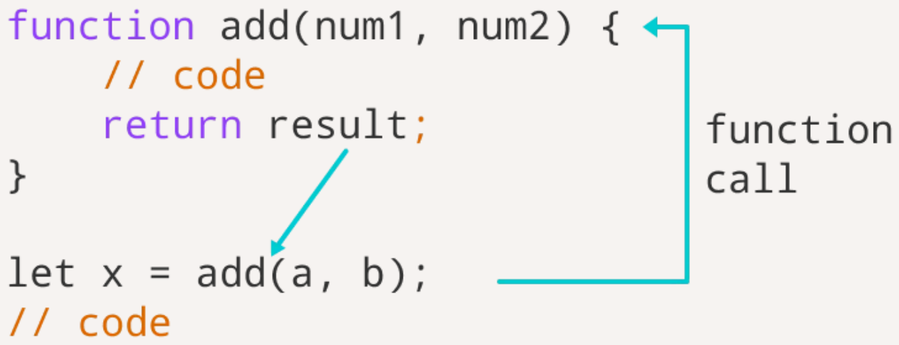
break labels

9.- Functions

1. Basic aspects
2. Advanced aspects



9.1.- Basic aspects of functions

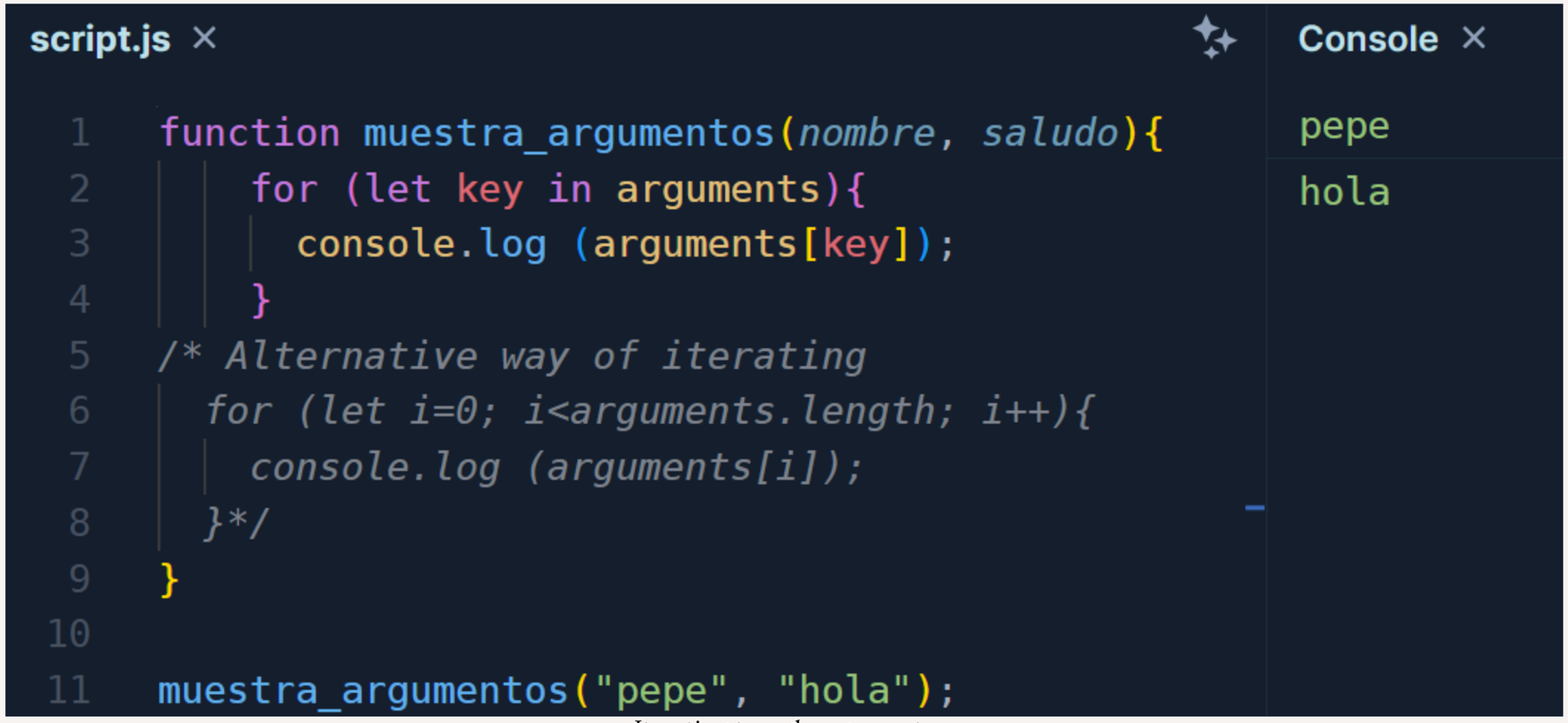


1. Creating functions
2. Local and global variables
3. Arguments
4. Return values
5. Functions as expressions
6. Arrow functions
7. Callbacks

Variable scope

local scope has preference over global one

Arguments



Iterating trough arguments

Arguments

script.js

```
1   /*how to create a default value when an argument is not provided. Is a feature of modern JavaScript*/
2   function muestraCliente2(nombre, apellido="sin apellido") { //default value for arguments
3       nombre="el inconfundible "+nombre;
4       console.log(nombre+" "+apellido);
5   }
6
7   let nombres=["procopio","patrocinio"];
8
9   for (let i=0; i<nombres.length; i++)
10      muestraCliente2(nombres[i]);    //when missing an argument, its default value is used (defined in the function)
11
12
```

Console

```
el inconfundible procopio sin apellido
el inconfundible patrocinio sin apellido
```

Default arguments in a function

Arguments as a value

script.js

```js
//an argument is a copy of a global variable. Modifying it don't change global variable
function muestraCliente(nombre, apellido) {
  nombre="el inconfundible "+nombre;
  console.log(nombre+" "+apellido);
}

let nombre="Perico";
muestraCliente(nombre,"Pérez");
console.log(nombre);  //nombre variable won't be modified

```

Console

```
el inconfundible Perico Pérez
Perico
```

Primitive arguments are passed as value

Arguments as a reference

script.js

```js
function modificarObjeto(obj) {
  obj.propiedad = 'nuevo valor'; // Modifica el objeto original
}

let miObjeto = { propiedad: 'valor original' };
modificarObjeto(miObjeto);
console.log(miObjeto.propiedad); // 'nuevo valor' (cambia)


function reasignarObjeto(obj) {
  obj = { propiedad: 'otro valor' }; // Reasigna a un nuevo objeto, pero no afecta el original
}

let otroObjeto = { propiedad: 'valor original' };
reasignarObjeto(otroObjeto);
console.log(otroObjeto.propiedad); // 'valor original' (no cambia)
```

Console

```
nuevo valor
valor original
```

Objects are passed as reference

Returning values

script.js

```
function muestraCliente2(nombre, apellido="sin apellido") {
  nombre="el inconfundible "+nombre;
  console.log(nombre+" "+apellido);
}

let nombres=["procopio","patrocinio"];

for (let i=0; i<nombres.length; i++)
muestraCliente2(nombres[i]);    //when missing an argument, its default value is used (defined in the function)

```

Console

```
el inconfundible procopio sin apellido
el inconfundible patrocinio sin apellido
```

A default argument in a function

Returning values


```
function muestraCliente3(nombre, apellido=buscaApellido(nombre)) {
//default value for arguments
    nombre="el inconfundible "+nombre;
    console.log(nombre+" "+apellido);
}

function buscaApellido(nombreBuscado){
    let nombres=[{nombre:"procopio",apellido:"máximo"},
                 {nombre:"patrocinio",apellido:"Sánchez"}];
    for (let i=0; i<nombres.length; i++){
        if (nombreBuscado==nombres[i].nombre){
            return nombres[i].apellido;
        }
    }
}

muestraCliente3("procopio");
```

```
'el inconfundible procopio máximo'
```


A function as a default argument in a function

Returning values


```
"use strict";
let usuario={nombre:"pepe",edad:"17"};
function compruebaMayoriaEdad(edad) {
    if (edad >= 18) {
      return true;
    } else {
      return false;
    }

    //alternative #1 (BEST)
    //return (usuario.edad>=18);

    //alternative #2
    //(usuario.edad>=18) ? return true : return false;
}

function garantizaAcceso(usuario){
  if (compruebaMayoriaEdad(usuario)){
   return(`acceso garantizado a ${usuario.nombre}`);
 }else{
   return(`acceso denegado a ${usuario.nombre}`);
 }

 //alternative
 /*let accion;
 compruebaMayoriaEdad(usuario) ? accion="garantizado" :
accion="denegado";
 return (`Acceso a ${usuario.nombre} ${accion}`);*/
}

'acceso denegado a pepe'

console.log(garantizaAcceso(usuario));
```

a JavaScript function can return a value

Function as an expression


```
"use strict";
////////////FUNCTIONS AS EXPRESSIONS/////////////////
//as a function can be assigned to a variable, it can be copied
let diHola=function () {   //no need to specify function name
  console.log( "Hola" );                                    'Hola'
};  //semicolon at the end. It's an expression assigned to a variable   'Hola'
let func=diHola;  //no parenthesis when assigning, they are variables
func();
diHola();
```


as JavaScript functions return a value, they can be seen as an expression and can be assigned to a variable or copied

Function as an expression


```
//Another example
// program to find the square of a number
// function is declared inside the variable
let x = function (num) { return num * num };
console.log(x(4));

// function can be used as variable value for other variables
let y = x(3);
console.log(y);
```

```
16

9
```


Function can be used as variable value fot other variables

Function as an expression

Function declared can be called before creating them while function as an expression can't

Function as an expression


```
"use strict"
if (1){
  function saluda(nombre){
    console.log("hola "+nombre);
  }
}else{
  function despidete(nombre){
    console.log("adiós "+nombre);
  }
}
saluda("pepe");
```

```
ReferenceError: saluda is not defined
```

```
"use strict";
//this is the key. I define the variable outside the scope and, assign to the function inside
let saluda, despidete;
if (1){
  saluda=function(nombre){
    console.log("hola "+nombre);
  }
}else{
  despidete=function(nombre){
    console.log("adiós "+nombre);
  }
}

saluda("pepe");
```

```
'hola pepe'
ƒ saluda()
```

Declared functions are valid within scope they were declared at, while functions as an expression can be utilized outside if declared

Arrow functions

script.js

```
1   //similar to function expressions, but specifying the returning value
2   let suma=(a,b) => a+b;
3   let saluda=(nombre)=> console.log("yo te saludo "+nombre);
4   let despidete=()=>console.log("adiós");
5
6   let edad=prompt("¿cuál es tu edad?");
7   let vasPreso=(edad>=18) ?
8     () => console.log("vas preso") :
9     () => console.log("al correccional");
10
11  saluda();
12  vasPreso();
```

Console

```
yo te saludo undefined
al correccional
```

? : syntax can be used

Arrow functions

script.js ×

```
1   //for multiline functions, curly brace and return statement must be used
2   let resta=(a,b)=>{
3     let resultado=a-b;
4     return (resultado); //could've been done in a single line, but for demonstration purposes
5   }
6
7   let suma=(a,b)=>{
8     return (a+b);
9   }
10
11  console.log (resta(4,2));
12  console.log(suma(4,2));
13
```

Console ×

```
2
6
```

Multi line arrow functions must use curly braces and return

Callbacks


```
function ask(question, yes, no) {
  if (confirm(question)) yes()  //confirm shows up a window asking accept or cancel
  else no();
}

function showOk() {
  console.log( "You agreed." );
}

function showCancel() {
  console.log( "You canceled the execution." );                    'You canceled the execution.'
}

// usage: functions showOk, showCancel are passed as arguments to ask
ask("Do you agree?", showOk, showCancel);
```

Functions can be passed as an argument to another function

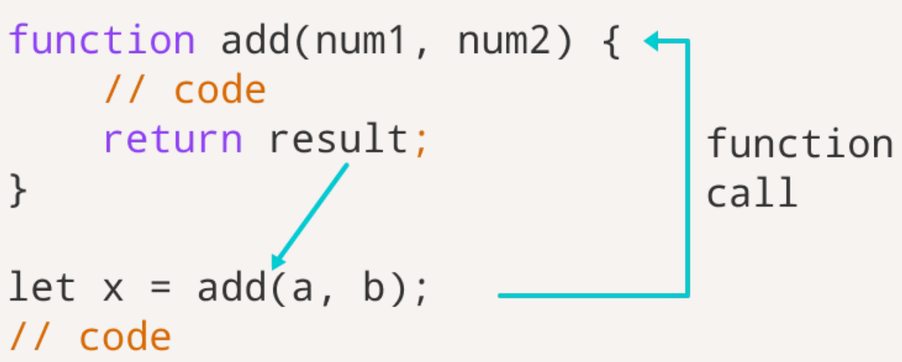
Callbacks


```
//Another way to express the last code (not recommended tough)
function pregunta(question, yes, no) {
  if (confirm(question)) yes()  //confirm shows up a window asking accept or cancel
  else no();
}

pregunta(
  "Agree?",
  function() {console.log("estás de acuerdo");},
  function () { console.log ("no estás de acuerdo");}
);
```

```
'no estás de acuerdo'
```

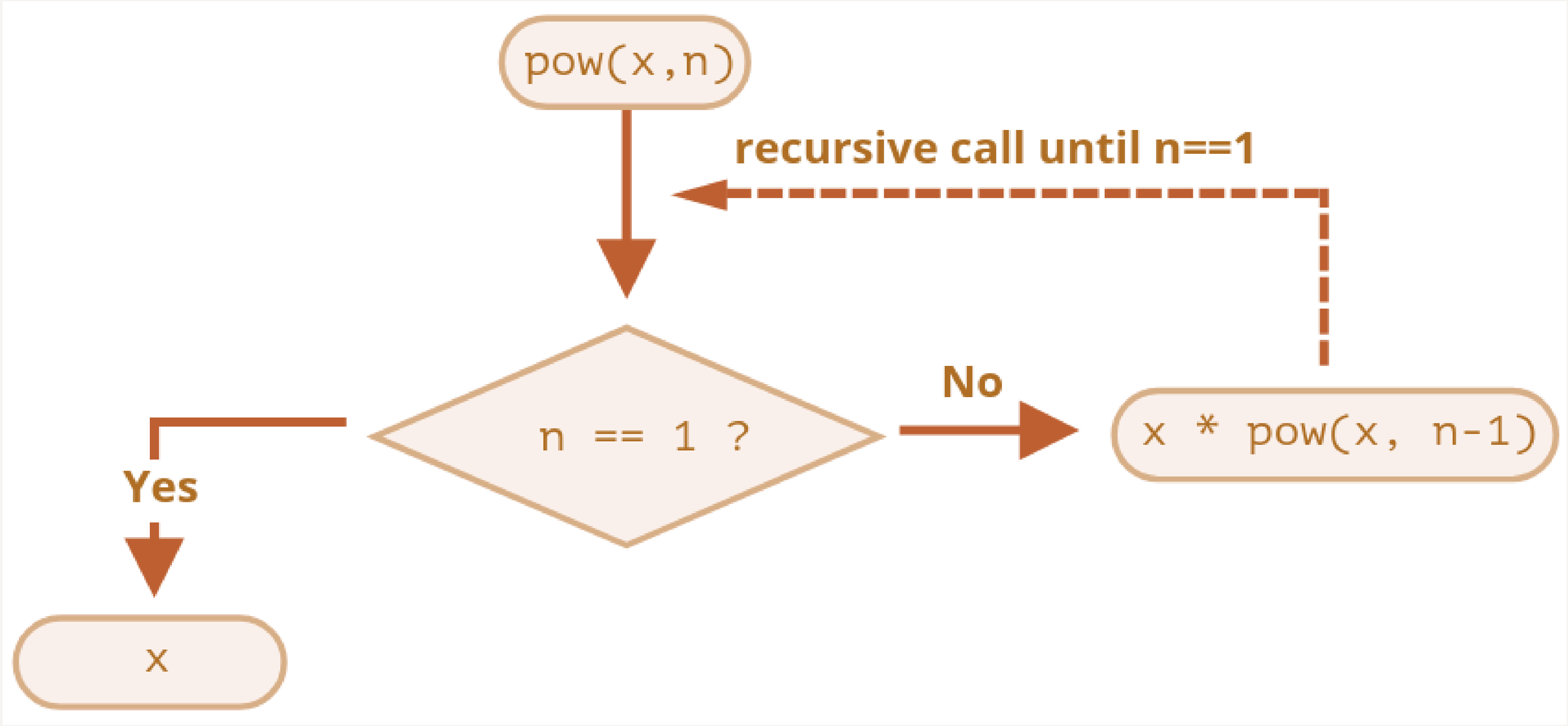
Another way to express functions callbacks

9.2.- Advanced aspects of functions



1. Recursivity
2. Rest parameters
3. Spread syntax
4. Autoexecutable functions
5. Nested functions
6. Scope
7. Closure

Recursivity



for a algorithm to be expressed as recursive, it is needed a base case and a problem that can be decomposed in subproblems

Recursivity


```
function pow(x, n) {
  let result = 1;

  // multiply result by x n times in the loop
  for (let i = 0; i < n; i++) {
    result *= x;
  }

  return result;
}

alert( pow(2, 3) ); // 8
```

```
function pow(x, n) {
  if (n == 1) {
    return x;
  } else {
    return x * pow(x, n - 1);
  }
}

alert( pow(2, 3) ); // 8
```

Iterative vs recursive pow algorithm

REST parameters

script.js

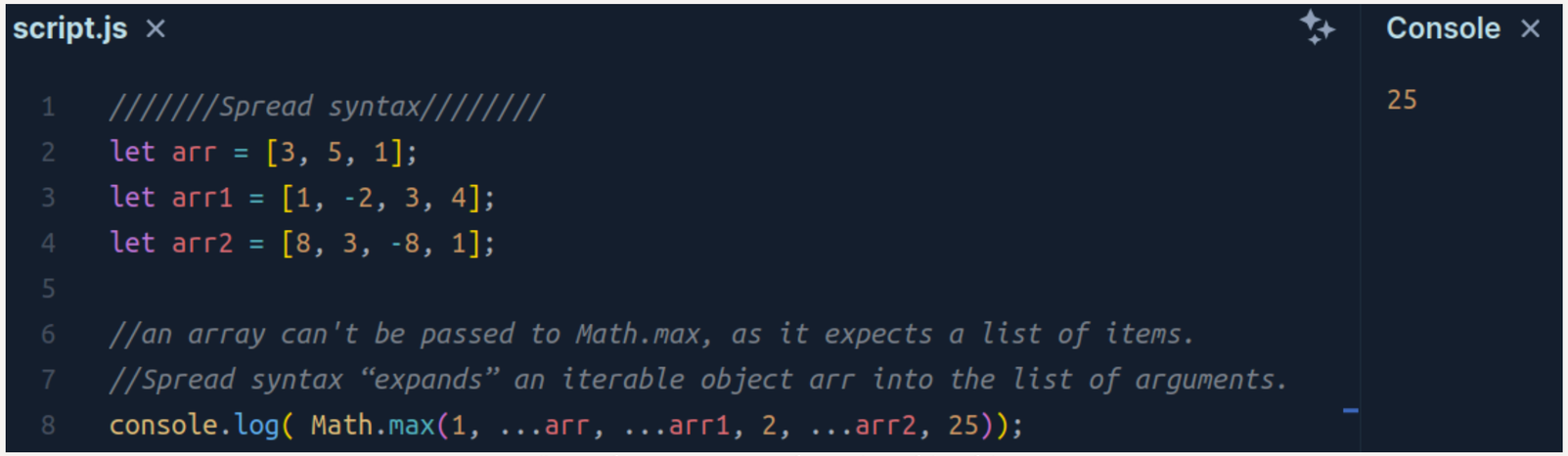
```
"use strict";

//Rest parameters
//allows to pass an undetermined number or parameters as an array. Must be placed at the end
function sumaTodo(num1, num2, ...numeros){
  let acumulado=num1+num2;
  for (let num of numeros){
    acumulado+=num;
  }
  return acumulado;
}

//palabras is an array containing all elements passed as arguments
let concatena2 = (...palabras) => {
  let resultado="";
  for (let palabra of palabras){
    resultado+=palabra;
  }
  return resultado;
}

console.log (sumaTodo(1,2,3,4,5,6,7,8));
console.log (concatena2("a", "b", "c", "d"));
```

Console

```
36
abcd
```

REST parameters allow to pass an indetermined number of arguments

Spread syntax



Spread syntax expands an iterable object

Autoexecutable functions

script.js

```
1   (function() { console.log("hola mundo") }) ();
2
3   ( function(quien){
4       console.log("hola " + quien);
5   })("mundo");
```

Console

```
hola mundo
```

```
hola mundo
```

Autoexecutable functions. Called once, lost forever

Nested functions

script.js

```
function externa() {
  console.log("Soy la función externa");

  function interna() {
    console.log("Soy la función interna");
  }

  interna(); // Llamamos a la función interna desde dentro de la función externa
}
//interna();  //error: unknown function
externa(); // Llamada a la función externa
```

Console

```
Soy la función externa
Soy la función interna
```

Nested functions. Inner function is invisible outside

Nested functions

script.js

```
1   //a function created within another function
2   //inner function is invisible outside and can use outer variables
3   function saludador(quien){
4     return function(){ //creación de la funcion anónima a retornar. No hace falta nombre
5       console.log("hola " + quien);
6     }
7   }
8   var saluda = saludador("mundo");
9   saluda(); //hola mundo
```

Console

```
hola mundo
```

It is possible to return a nested function. Thus, it is summoned when calling the external one

Scope

script.js ×

```
let mensaje1="hola don pepito",
    mensaje2="hola don josé";

function muestraMensaje() {
  let mensaje1 = "Saludos desde dentro de la función!"; // If a local variable with the same name is defined, global variable can't be used
  if (1){
    let mensaje1="mensaje 1 dentro del if";
  }
  console.log (mensaje1, mensaje2);
  mensaje2="cambio el mensaje dentro de la función";  //a global variable can be accessed and modified inside a function
}

console.log ("muestro variables globales antes de llamar a la función:", mensaje1, mensaje2); //showing global message variable
muestraMensaje();
console.log ("muestro variables globales después de llamar a la función:", mensaje1, mensaje2); //showing global message variable
```

Console ×

```
muestro variables globales antes de llamar a la función:  hola don pepito  hola don josé
Saludos desde dentro de la función!  hola don josé
muestro variables globales después de llamar a la función:  hola don pepito  cambio el mensaje dentro de la función
```

Use of global, local and block scope in a function

Scope

script.js

```
let globalVar = 'Global';
function exampleFunction() {
    let localFuncVar = 'Local a la función';
    console.log(globalVar);
    console.log(localFuncVar);
    //console.log(localIfVar);  //Error. It's not defined
    if (1){
      let localIfVar="Local al if";
      console.log(globalVar);
      console.log(localFuncVar);
      console.log(localIfVar);
      }
}

exampleFunction();
console.log(globalVar);
//console.log(localFuncVar);    //Error. It's not defined
//console.log(localIfVar);    //Error. It's not defined
```

Console

```
Global
Local a la función
Global
Local a la función
Local al if
Global
```

Global, function and block scope

Scope

script.js

```
1   let x = 1;
2
3   function func() {
4     console.log(x); // Error: No se puede acceder a 'x' antes de inicializarla
5     let x = 2;
6   }
7   func();
8
```

Console

```
▶ ReferenceError: can't access lexical declaration 'x2' before initialization
  func@https://preview-javascript.playcode.io/ line 19 > injectedScript:68:15
  @https://preview-javascript.playcode.io/ line 19 > injectedScript:72:1
  mn@https://preview-javascript.playcode.io/ line 6 > injectedScript:19:5455
```

Error accessing a local variable not defined yet

Scope

script.js

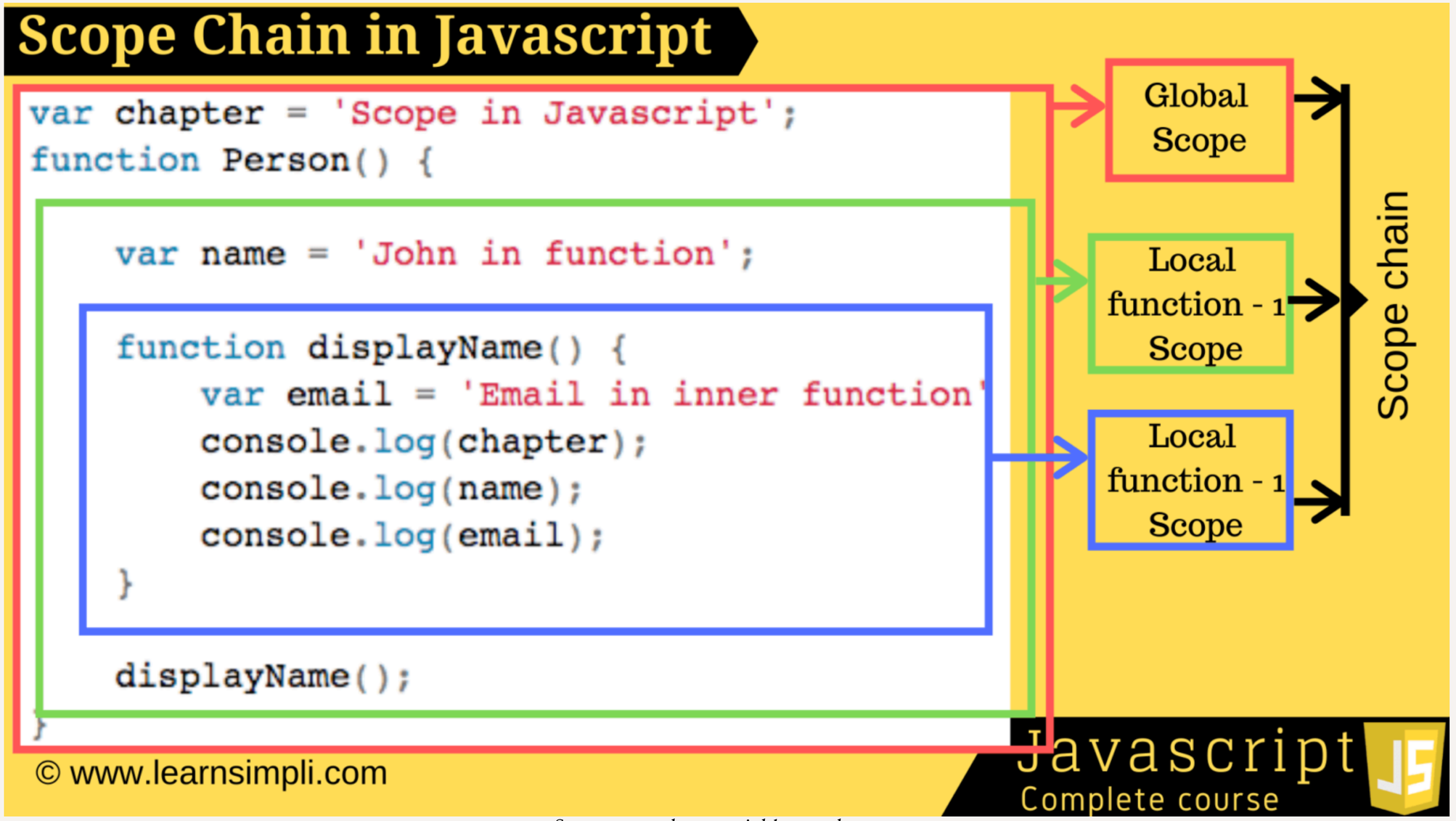
```
1   "use strict";
2
3   let frase = "hola";
4
5   if (1) {
6     let usuario = "Patrocinio";
7     function saluda() {
8       console.log(`${frase}, ${usuario}`);
9     }
10  }
11
12  saluda(); //Saluda is unknown outside the "if" block
```

Console

```
▶ TypeError: saluda is not a fun
  ction
  @https://preview-javascript.pl
  aycode.io/ line 19 > injectedS
  cript:74:1
  mn@https://preview-javascript.
  playcode.io/ line 6 > injected
  Script:19:5455
```

Functions, as variables, can only be used within their scope

Scope and closure



Scopes are where variables are known

Scope and closure

Closure: Nested functions remember their lexical environment when they was created. It allows functions to remember their state

Scope and closure

script.js

```
1   function persona(aux){
2     let nombre=aux;
3     return function(){
4       return nombre;
5     }
6   }
7
8   let persona1=persona("pepe");
9   let persona2=persona("manolo");
10  //same variable, different value
11  console.log(persona1());
12  console.log(persona2());
```

Console

```
pepe
manolo
```

Same variable name, different value due to scope