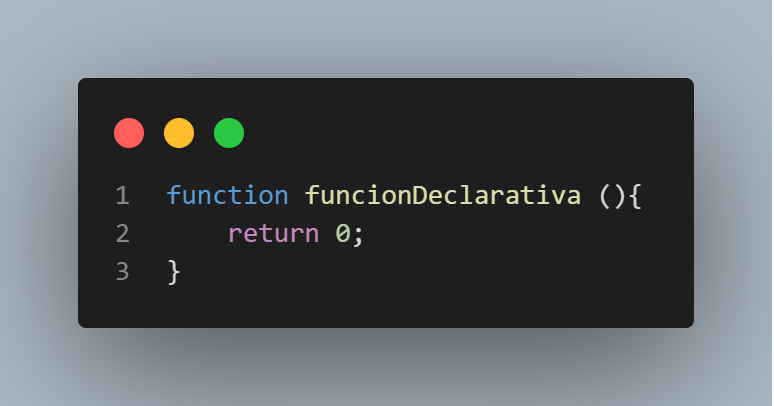
FUNCIONES

Pueden ser declarativas o expresivas:

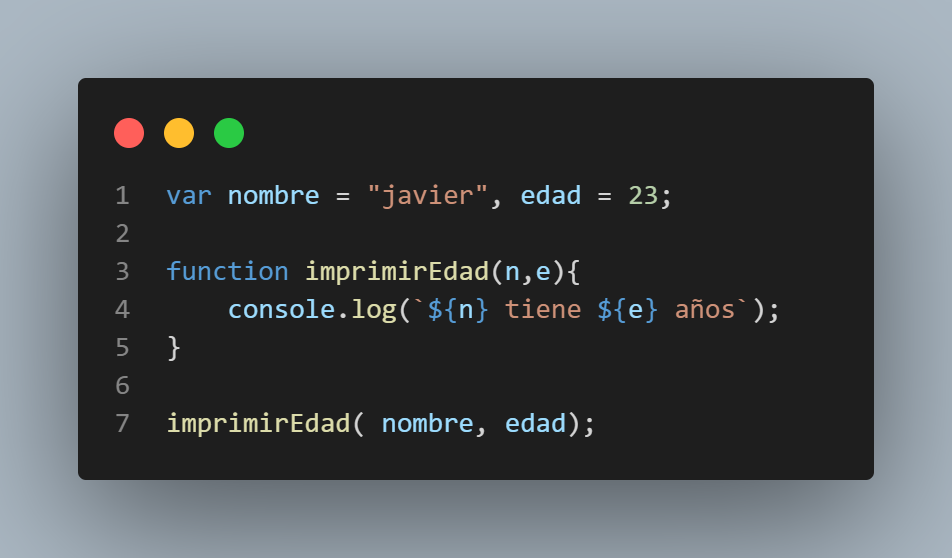
**DECLARATIVAS:**

****

**EXPRESIVAS:**



Las funciones dentro de los paréntesis reciben parámetros, estos parámetros con las variables con las que ejecutara la función.



**n** y **e** son los parámetros con los que trabajara la función, estos serán reemplazados por las variables que uno desee.

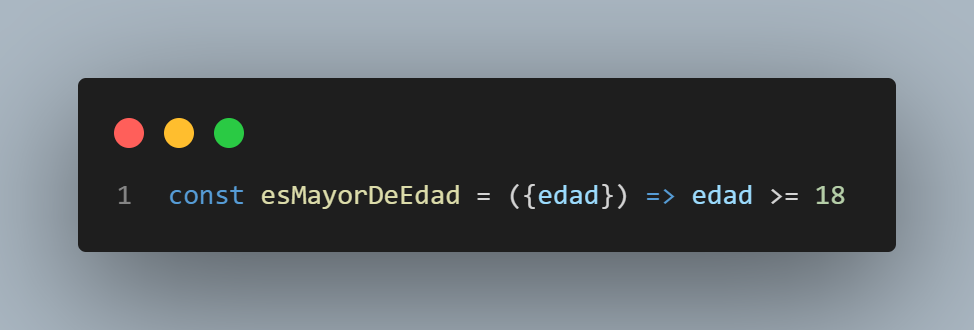
Las funciones también pueden tomar como parámetros objetos o incluso claves de algún objeto.



ARROW FUNCTIONS:

Es una nomenclatura más corta y fácil de escribir funciones.

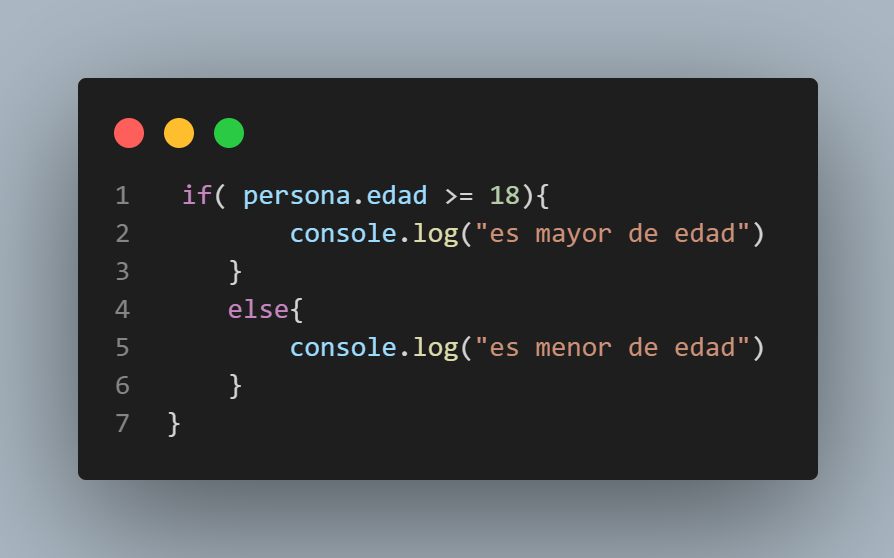
Primero se declara una variable (preferiblemente con **const**) que llevara por nombre con el que se llamara la función debido a que la función será una anónima y no será necesario escribir la palabra **function**; si la función solo lleva un parámetro se obvian los paréntesis y no es necesario colocarlos; luego va el operador que indicará que es una función (**=>**) luego se colocan el proceso que hará la función.



CONDICIONALES:

Son estructuras de control que permiten decidir si un bloque de código se ejecuta o no, mediante una condición que se cumple o no.

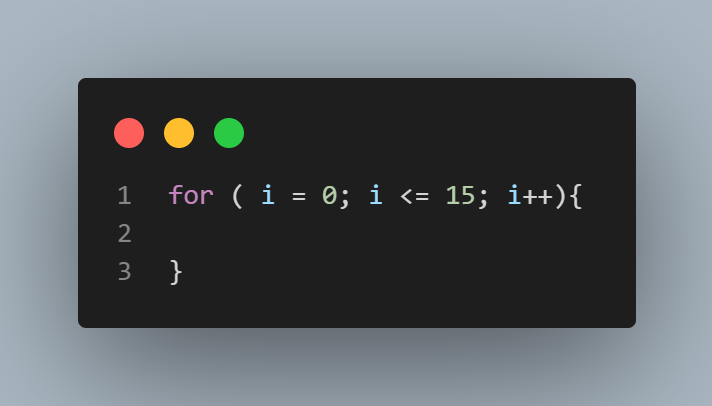
**If** es la condicional más usada junto con su negación **else.**



LOOPS:

Son procesos repetitivos que se pueden generar.

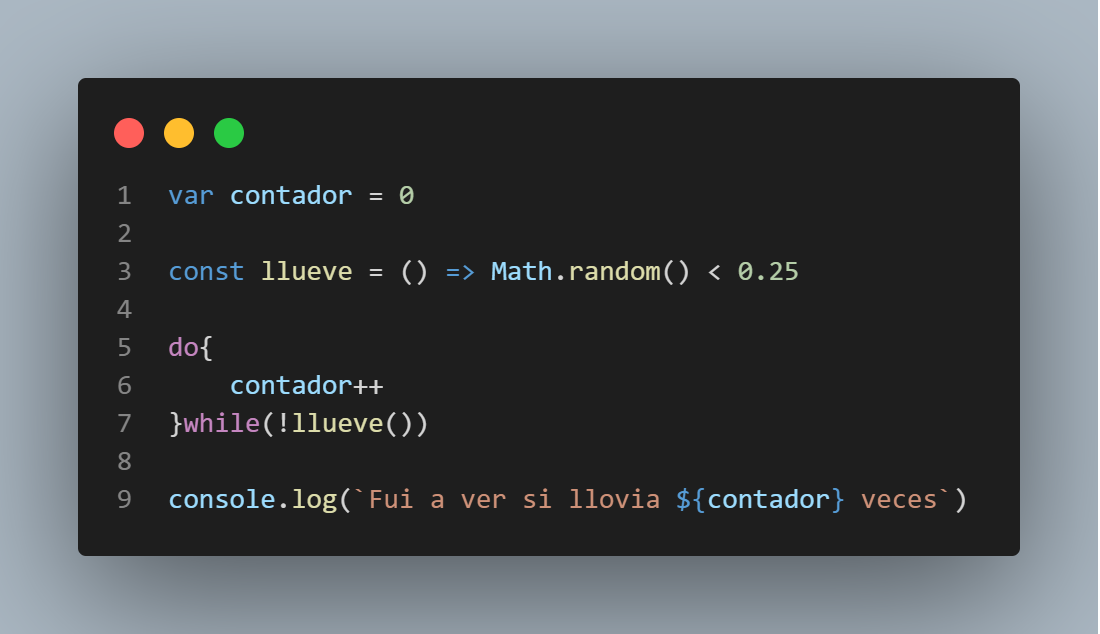
**For:** lleva por lo general una variable ***i*** que se inicia en **0** luego se establece hasta que valor llegara que será el punto final del loop y finalmente se le indica de cuanto en cuanto aumentara.



**While:** ejecutara el código hasta que la condición se rompa.



**Do While:** ejecuta el código una vez y allí valida la condición, si se cumple se volverá a ejecutar.

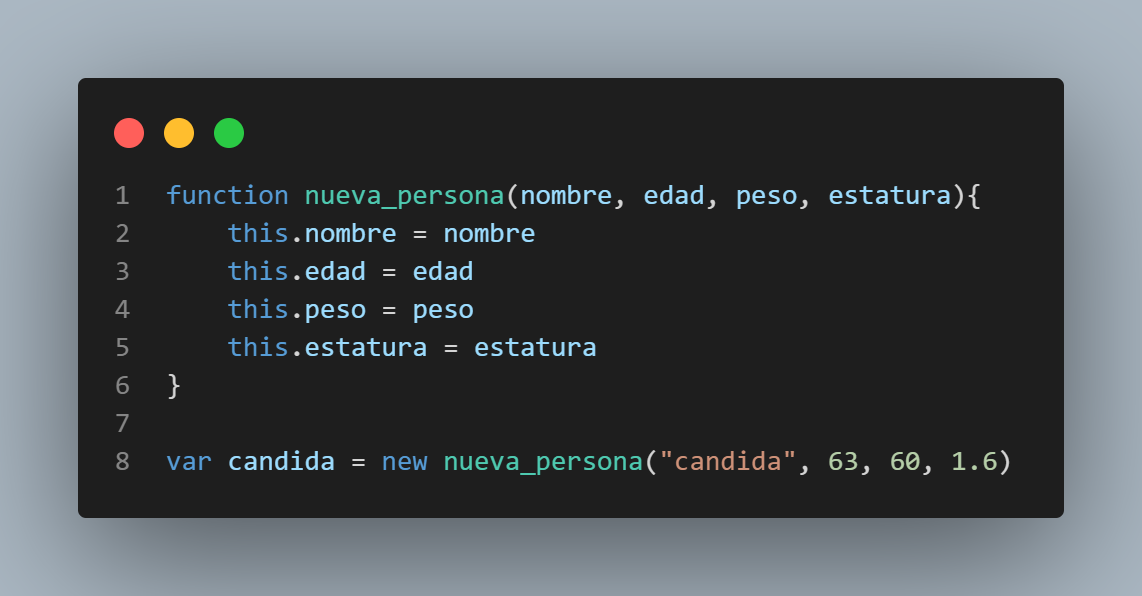


**Switch:** ejecuta un bloque de código según sea el caso, para ello emplea múltiples opciones.



OBJETOS:

Las clases en JavaScript se conocen como **prototipos,** estos prototipos son funciones que entre otras cosas ayudan a crear nuevos objetos.



En los prototipos se emplean nuevos elementos como el **this,** que hace referencia al atributo nombrado en el objeto recién creado; y el **new**, que le dice al lenguaje que debe generar un nuevo objeto.

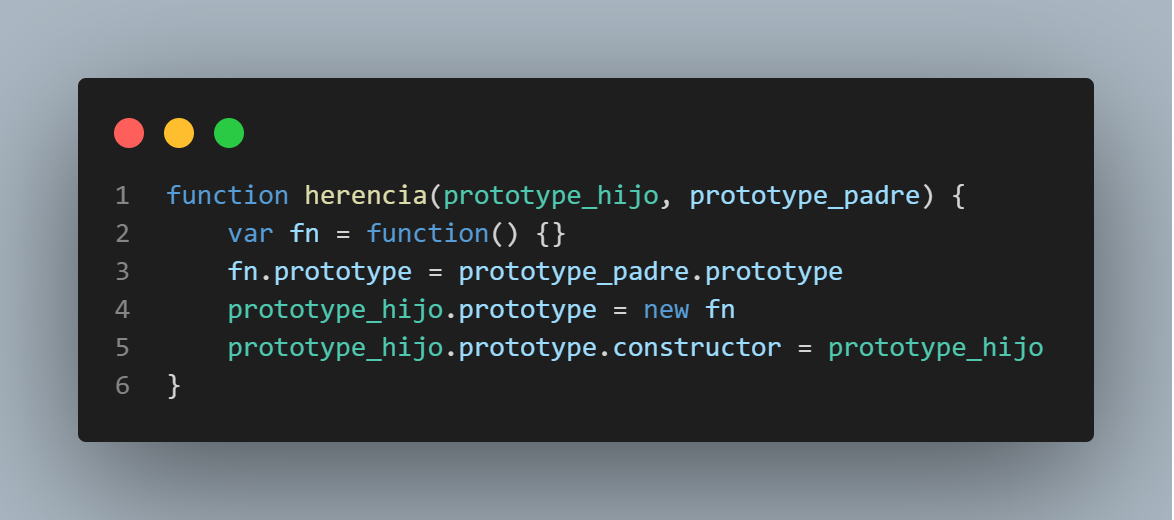
Las funciones dentro de los prototipos tienen una declaración diferente y tienen una conexión directa con el prototipo bajo el cual se crean, un objeto externo del prototipo por ende no puede usarlas.



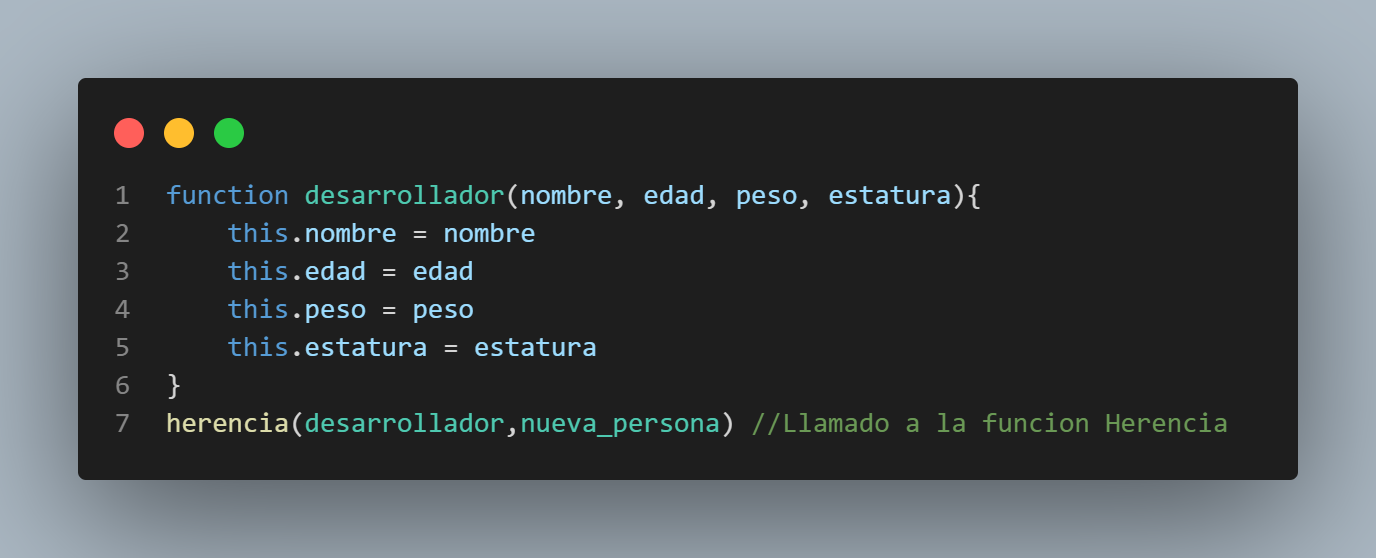
**Clases y Herencia en JavaScript:**

La herencia como tal en JavaScript no existe, pero permite hacer que un prototipo herede de otro.

Para eso se crea una función que será la encargada de decirle al lenguaje que un prototipo heredará de otro.

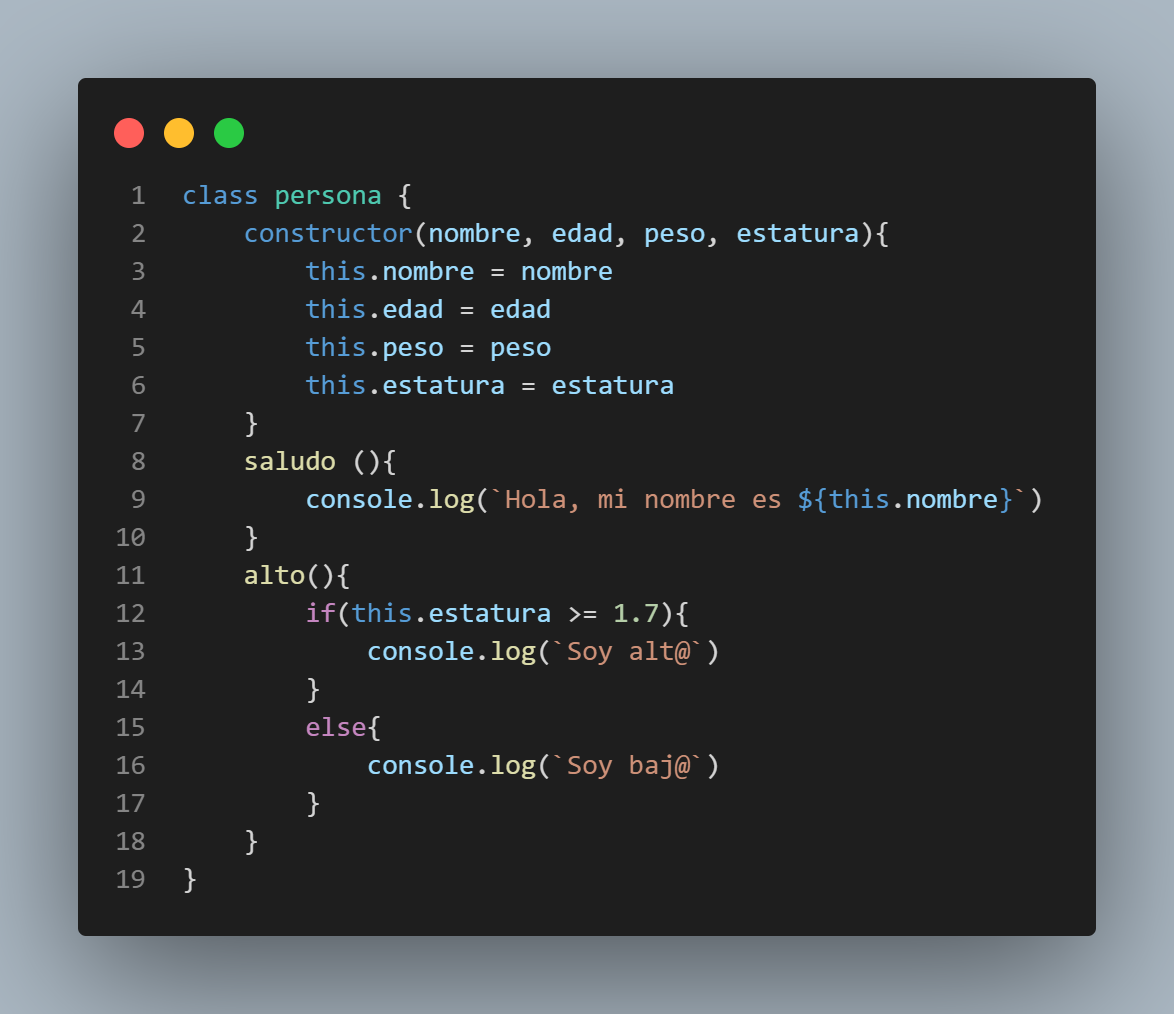
****

Los prototipos que se creen como hijos heredaran todos los métodos que tienen los prototipos padre, además podrán reescribir métodos también. Sin embargo para que esto ocurra se debe llamar la función herencia después de que se crea la función constructora del prototipo hijo.



El prototipo hijo podrá invocar cualquier método creado bajo el prototipo padre.

Con ECMAScript 2015 se creó una nueva forma más fácil de trabajar con estos prototipos, llamándolos clases, a pesar de que sigan siendo básicamente prototipos. Para ello se declara el prototipo como una clase y dentro se escriben los métodos; con esto ya no es necesario escribir los métodos como funciones, sino solo el nombre de éstos.



Con esta nueva forma la herencia cambia también. Ya no es necesario el crear una función que la genere, solo basta con invocar el atributo **Extends.**

class desarrollador extends persona {

    constructor (nombre, edad, peso, estatura){

        super (nombre, edad, peso, estatura)

    }

    Saludo () {

        console.log (`Hola, mi nombre es ${this.nombre}`)

        console.log ("Y soy desarrollador/a")

    }

}

Los métodos se escriben normal pero para invocar el método **constructor** no se puede escribir el atributo this, para ello se coloca el atributo **super** y a él se le agregan los parámetros que vayan.

**Funciones como parámetros:** las funciones también pueden ser parámetro.

saludo(fn){

        console.log(`Hola, mi nombre es ${this.nombre}`)

        console.log("Y soy desarrollador/a")

        if (fn){

            fn (true) // false en el caso que no sea Dev

        }

    }

function respuestaSaludo(esDev){

    console.log(`Buenos días`)

    if(esDev){

        console.log(`Ah, no sabía que eras Dev`)

    }

}

juana.saludo (respuestaSaludo) //Llamado de la función como parámetro