Javascript

Semana 1

Índice

Resultado de aprendizaje, contenido y criterios de evaluación

Resultado de aprendizaje
RA2. Escribe sentencias simples, aplicando la sintaxis del lenguaje y verificando su ejecución sobre navegadores web.
Contenido
1 - Variables. 2 - Tipos de datos. Conversiones. 3 - Asignaciones. 4 - Operadores 5 - Comentarios al código. 6 - Sentencias. 7 - Decisiones. 8 - Bucles. 9 - Prueba y documentación del código.
Criterios de evaluación
a) Se ha seleccionado un lenguaje de programación de clientes web en función de sus posibilidades b) Se han utilizado los distintos tipos de variables y operadores disponibles en el lenguaje. c) Se han identificado los ámbitos de utilización de las variables d) Se han reconocido y comprobado las peculiaridades del lenguaje respecto a las conversiones entre distintos tipos de datos e) Se han utilizado mecanismos de decisión en la creación de bloques de sentencias. f) Se han utilizado bucles y se ha verificado su funcionamiento

0. Introducción

JavaScript fue introducido en 1995 como una forma de agregar programas a páginas web en el navegador Netscape Navigator.

JavaScript casi no tiene nada que ver con el lenguaje de programación llamado Java. El nombre fue considerado por razones de marketin. Cuando se estaba introduciendo JavaScript, el lenguaje Java se estaba comercializando mucho y ganaba popularidad. Alguien pensó que era una buena idea intentar aprovechar este éxito.

Podemos decir que JavaScript es un lenguaje de programación que nos permite hacer que las páginas web sean dinámicas e interactivas. Ha habido varias versiones. La versión ECMAScript 3 fue la versión ampliamente soportada durante el ascenso al dominio de JavaScript, aproximadamente entre 2000 y 2010. Durante este tiempo, se estaba trabajando en una versión ambiciosa 4, la cual planeaba una serie de mejoras y extensiones radicales al lenguaje. Cambiar un lenguaje vivo y ampliamente utilizado de esa manera resultó ser políticamente difícil, y el trabajo en la versión 4 fue abandonado en 2008. Una versión 5, mucho menos ambiciosa, que solo realizaba algunas mejoras no controversiales, salió en 2009. En 2015, salió la versión 6, una actualización importante que incluía algunas de las ideas previstas para la versión 4. Desde entonces, hemos tenido nuevas actualizaciones pequeñas cada año.

Para trabajar con Javascript vamos a desplegar el entorno adecuando para ello visita sigue el “**ANEXO I. Creando el entorno**”.

Semana 1. Fundamentos de JavaScript (I)

Sesión 1

1.1. Ejemplo guiado

1.1.1. Hola mundo

Comenzamos con nuestro “Hola mundo”.

Crea un archivo nuevo en Visual Studio Code llamado **holaMundo.js** con el contenido siguiente:

```
console.log(“¡Hola mundo!”);
```

console es un objeto global que proporciona acceso a la consola de depuración del navegador o del entorno de ejecución (como Node.js). Se utiliza principalmente para imprimir mensajes, depurar código y realizar pruebas durante el desarrollo.

Los más usados:
cosole.log()
console.error()
console.warn()
console.info()
console.table()
console.time() y console.timeEnd()

Para ejecutarlo abre una terminal y sitúate donde tienes el fichero.

Ejecuta lo siguiente:

```
node  probando_stringNumber.js
```

¿Qué devuelve cada salida?

Ahora vamos a introducirnos en las variables y operaciones. Te recomiendo eches un vistazo a los Anexos AII.1. Variables y AII.2. Tipos de datos.

Crea un archivo nuevo en Visual Studio Code llamado **probando_stringNumber.js** con el contenido siguiente:

```
let x = 16 + 4 + "Volvo";
console.log(x);

let y = "Volvo" + 16 + 4;
console.log(y);
```

Para ejecutarlo abre una terminal y sitúate donde tienes el fichero.

Ejecuta lo siguiente:

```
node  probando_stringNumber.js
```

¿Qué devuelve cada salida? ¿Qué deduces?

A continuación vamos a hacer un ejemplo guiado:

1.1.2. Adivina número

Ejemplo guiado

Crear un pequeño juego en el que el usuario debe adivinar un número aleatorio entre 1 y 50 en un máximo de 5 intentos.

1. *Crea un archivo nuevo en Visual Studio Code llamado adivinaNumero.js*
2. *Necesitamos crear un número aleatorio, hacemos uso del objeto Math y su método random.*

Por defecto Math.random() devuelve un número entre 0 y 1, excluyendo el número 1. Si nosotros necesitamos entre 1 y 50 debemos multiplicar por 50, pero recordar que nunca llega a 50. Nuestro código va quedando:

```
const numeroSecreto = Math.floor(Math.random() * 50) ;
```

Hemos guardado nuestro número aleatorio en una constante, pues no lo variaremos a lo largo del programa. Te recomiendo que veas el anexo AII.1. Variables para conocer los tipos y su asignación.

Tal como está nuestro código solo muestra números del 0 al 49,999 y la actividad nos pide del 1 al 50, ¿qué hacemos? Sí, eso, efectivamente, sumar 1:

```
const numeroSecreto = Math.floor(Math.random() * 50)  + 1;
```

3. *Ahora declararemos los intentos, como constante, porque no se modificará a lo largo del código.*

```
const numeroSecreto = Math.floor(Math.random() * 50)  + 1;
let intentos = 5;
```

4. *Ahora vamos a crear una función que iniciará el juego, la llamaré jugar(). Te recomiendo mirara el anexo AIV.1. Funciones para profundizar.*

```
const numeroSecreto = Math.floor(Math.random() * 50)  + 1;
let intentos = 5;
function jugar() {
  . . . . . .
}
```

a) *Recoger el dato por consola. Usando prompt, este es un método que muestra un cuadro de diálogo solicitando información al usuario. Pero para poder usar lo con Node debemos instalar primero las librerías que utiliza prompt. Para instalarlo sigue los pasos:*

   1. *Abre la Terminal de VSC (estando situado en el proyecto). Ejecuta el comando de instalación:*

      npm install prompt-sync

   2. *En nuestro proyecto llamaremos a estas librería con un requiere:*

      require('prompt-sync')()

El código queda como:

```
const prompt = require('prompt-sync')();
const numeroSecreto = Math.floor(Math.random() * 50)  + 1;
let intentos = 5;
function jugar() {
   const respuesta = parseInt(prompt("Adivina el número entre 1 y 50:"));.
}
```

Prompt devuelve un string y nosotros queremos guardarlos como un número. Debemos convertirlo con el objeto parseInt().

b) *Comprobar si el número del usuario coincide con el aleatorio. Esto lo hacemos tomando "decisiones" con if. Mira el anexo AII.7.a. if….else para ahondar en el tema.*

```
const prompt = require('prompt-sync')();
const numeroSecreto = Math.floor(Math.random() * 50)  + 1;
let intentos = 5;
function jugar() {
   const respuesta = parseInt(prompt("Adivina el número entre 1 y 50:"));

   if (respuesta === numeroSecreto) {
       console.log("¡Felicidades! Has adivinado el número.");
       return; // Finaliza el juego
   } else if (respuesta > numeroSecreto) {
       console.log("El número secreto es menor.");
   } else {
      console.log("El número secreto es mayor.");
   }
}
```

Notar que:

- *Usamos el triple igual para incluir la comprobación del tipo.*
- *En el primer if, si se cumple, con "return" salimos de la función.*

c) *Decrementamos los intentos. Lo normal es usar intentos = intentos -1; pero una forma más concisa que nos permite usar JavaScript es: intentos--;*

```
const prompt = require('prompt-sync')();
const numeroSecreto = Math.floor(Math.random() * 50)  + 1;
let intentos = 5;
function jugar() {
  const respuesta = parseInt(prompt("Adivina el número entre 1 y 50:"));

  if (respuesta === numeroSecreto) {
      console.log("¡Felicidades! Has adivinado el número.");
      return; // Finaliza el juego
  } else if (respuesta > numeroSecreto) {
      console.log("El número secreto es menor.");
  } else {
     console.log("El número secreto es mayor.");
  }

  intentos--;
}
```

d) *Comprobamos, con sentencias if, si ha llegado al final de la partida (intentos es 0).*

```
const prompt = require('prompt-sync')();
const numeroSecreto = Math.floor(Math.random() * 50)  + 1;
let intentos = 5;
function jugar() {
  const respuesta = parseInt(prompt("Adivina el número entre 1 y 50:"));

  if (respuesta === numeroSecreto) {
      console.log("¡Felicidades! Has adivinado el número.");
      return; // Finaliza el juego
  } else if (respuesta > numeroSecreto) {
      console.log("El número secreto es menor.");
  } else {
    console.log ("El número secreto es mayor.");
  }
   intentos--;

  if (intentos === 0) {
    console.log ("Se te han acabado los intentos. El número era " + numeroSecreto);
  } else {
    jugar(); // Llamamos de nuevo a la función si quedan intentos
  }
}
```

5. *Finalmente iniciamos el juego llamando a la función jugar().*

```
const prompt = require('prompt-sync')();
const numeroSecreto = Math.floor(Math.random() * 50)  + 1;
let intentos = 5;
function jugar() { . . . . .     }
jugar();
```

Finalmente ejecuta el archivo desde una ventana de comando, con el comando:

```
node  adivinandoNumero.js
```

1.2. Ejemplo propuesto

Crea un archivo javascript donde declares tres variables que representen tu nombre, edad y curso.
Además debe mostrar un mensaje en consola que diga: "Hola, soy Juan, tengo 18 años y estudio DAW."

Sesión 2

1.3. Ejemplo guiado

1.3.1 Const y let

Ejemplo guiado

Declara constantes con tu color y comida favorita. Después cambia las preferencias de comida, por código. ¿Qué ocurre?¿Da un error?
Ahora modifica las variables usando let.
Reflexionad: ¿por qué const no permite reasignación?

Creemos un fichero llamado constlet.js.
Declaremos las constantes de color y comida favorita en el fichero.

```
// Declaración con const
const colorFavorito = "azul";
const comidaFavorita = "pizza";
console.log(“Color favorito: “ + colorFavorito);
console.log(“Comida favorita: “ + comidaFavorita);
```

Ejecútalo. No hay problema.

Imagina ahora que más adelante en el código alguien trata de darle un valor a colorFavorito, vamos a probarlo. Sigamos escribiendo:

```
// Declaración con const
const colorFavorito = "azul";
const comidaFavorita = "pizza";

// Líneas más abajo
colorFavorito = "verde";

console.log(“Color favorito: “ + colorFavorito);
console.log(“Comida favorita: “ + comidaFavorita);
```

Ejecútalo ahora. ¡Da un error! ¿Cómo lo corregimos?
Bien para usar variables que tomarán diferentes valores a lo largo del programa se usa let. Comprobemos.

```
// Declaración con const
let colorFavorito = "azul";
const comidaFavorita = "pizza";

// Líneas más abajo
colorFavorito = "verde";

console.log(“Color favorito: “ + colorFavorito);
console.log(“Comida favorita: “ + comidaFavorita);
```

1.3.2 Asignaciones a variables

Ejemplo guiado

Aveces encontramos asignaciones del tipo X +=2; ó X =+2;

Visita el anexo AII.3. Asignaciones para saber más.

Pero, ¡pruebalo tu mismo! Crea un fichero con el código que a continuación se muestra y ejecútalo observando la salida por consola.

Vamos a crear un script que se llame asignaciones.js y con el código:

```
// Caso 1: Asignación simple
let x = 3;
console.log(x);

// Caso 2: Operador de asignación con suma (x = x + 2)
x += 2;
console.log(x);

// Caso 3: Asignación con operador unario positivo (equivale a x = +2)
x =+ 2;
console.log(x);
```

¿Qué salidas obtienes en cada caso?

Explica la diferencia entre el caso 2 y 3.

Esto lo hemos hecho con el operador suma, pero es válido para el resto. Para ver las operaciones que permite JavaScript ver Anexo AII.4. Operaciones .

Muy a menudo nos veremos en la necesidad de incrementar en una unidad una variable. Para ello usaremos el operador unario "++" o "—" si lo queremos decrementar. Pero este operador tiene implicaciones diferentes si se usa delante o detrás de la variable: ++i ó i++. Para saber su diferencia ve al anexo AII.5.f. Operadores Unarios. (Te recomiendo que ejecutes el ejemplo propuesto y comentes con tus compañeros).

Un operador útil en JavaScript es TypeOf(), que nos permite conocer el tipo de una variable.

¡Vamos a probarlo con un ejemplo! Crea un script llamado probandoTypeof.js con el código siguiente y ejecútalo.

```
var ageDavid, ageAna;
ageDavid = 34;
ageAna = 28;
var mayorDavid = ageDavid > ageAna;
console.log("El tipo de la edad de David es: " + typeof ageDavid);
console.log("El tipo de la edad de David es: " + typeof ageAna);
console.log("El tipo de la comparación de si es mayor es: " + typeof mayorDavid);
console.log("El tipo de la cadena 'David es mayor que Ana' es: " + typeof 'David es mayor que Ana');
```

Explica los distintos tipos que obtienes por consola.

1.4 Ejemplo propuesto

Crea un fichero javascript que clasifique a una persona como niño, adolescente, adulto o anciano.
Se pedirá el nombre, apellidos y año de nacimiento en formato dd/mm/aaaa.
Si detecta datos vacíos se le avisa al usuario y se le da 3 intentos para que lo introduczca.
Al introducir todos los datos se mostrará el mensaje, por ejemplo:
"Antonio Dorta. Nacistes en el 20 de dicmebre de 1975"
Comenta el código.

Sesión 3

Hasta ahora hemos visto variables (declaración, tipos), asignaciones. Pero ¿cómo tomo decisiones en JavaScript? Lo vamos a ver con la siguiente actividad pero antes echemos un ojo a los anexos AII.7. Decisiones .

1.4. Ejemplo guiado

1.4.1. Mi bebida favorita

Ejemplo guiado

Haz un programa, miBebidaFavorita.js, que pida al usuario su bebida favorita con prompt() y responde con un mensaje: "¡Mmm! A mi también me gusta la <nombre de la bebida>.". Comenta el código.

La primera duda que nos planteamos es ¿cómo paso parámetros a un fichero? Y ¿cómo accedo a esos pará metros?

A la primera pregunta visita Anexo V. Como pasar parámetros a un archivo javascript cuando lo ejecuto por prompt donde te explico cómo, pero para no dejarte en ascuas te diré que es así, desde el promt de la consola:

> node miScript.js parametro1 parametro2 parametro3

En cuanto a la segunda pregunta, puedes consultar Anexo VI. Como acceder a los parámetros pasados a un archivo javascript por promt.

Recordando, los argumentos vienen en el array process.argv[] a partir de la posición 2.
En nuestro ejemplo el código iría quedado:

```
const bebida = process.argv[2] ;
console.log("¡Mmmm! A mi también me gusta la " + bebida + ".");
```

1.4.2. Mini calculadora

Para las siguientes actividades necesitamos saber cómo se declaran Bucles en JavaScript, para ello mira el anexo AII.8. Bucles.

Ejemplo guiado

Crea una calculadora de operaciones básicas. Le pasaremos por consola la operación y los operandos. Mostrará por consola el resultado.
Comenta el código.

Lo primero será recuperar los parámetros pasados al fichero. Los operadores pasarlos a formato numérico con el método de Javascript Number().

Luego un switch para discriminar la operación y evitar la división por 0.

```
// Recuperación de parámetros
const op= process.argv[2] ;
const a = Number(process.argv[3]) ;
const b = Number(process.argv[4]) ;

// Comprobando si a y b son números
if (!Number.isFinite(a) || !Number.isFinite(b)) {
   console.log("Tipo de operandos no válidos");
   return;
}

// Según el operador realizamos el cálculo
switch (op) {
    case "+":
      console.log(a + " + " + b + " = " + (a + b));
      break;
    case "-":
      console.log(a + " - " + b + " = " + (a - b));
      break;
    case "*":
      console.log(a + " * " + b + " = " + (a * b));
      break;
    case "/":
      if (b === 0){
        console.log("División por cero");
      } else {
        console.log(a + " / " + b + " = " + (a / b));
      }
      break;
    default:
      console.log("Operación no soportada");
      break;
}
```

1.5. Ejemplo propuesto

Juego de adivinanza con número aleatorio.

El número a adivinar será uno aleatorio comprendido entre 1 y 100.
El jugador tendrá 5 intentos para adivinarlo.
Tan solo se indicará si el número a adivinar está por encima o por debajo del proporcionado por el usuario.
Mostrar al final cuántos intentos le sobraron si ganó.
Permitir reiniciar el juego si el usuario lo desea.

ANEXO I. Creando el entorno

AI.1.- Interfaz de desarrollo

Usaré Visual Studio Code: https://code.visualstudio.com/

AI.2.- Extensiones recomendadas

Material Icon Them de Philipp Kief

AI.3.- Navegador usado en el desarrollo

Google Chrome. Con F12 vemos las opciones de Desarrollador

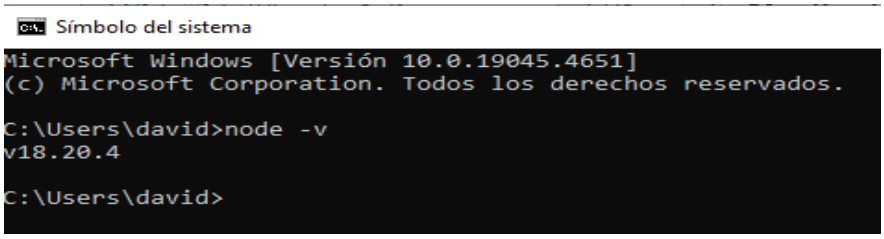
AI.4.- Otras aplicaciones usadas

AI.4.1.- Node JS

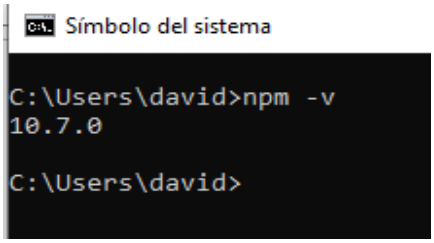
Para poder ejecutar Javascript sin la necesidad de un navegador usaremos NodeJS. Su página oficial es https://nodejs.org/en. Para instalar dirígete a Donwload y sigue los pasos.

Verifica su instalación (Abre una consola):

- Si está instalado node, comando_ node -v



- Si está instalado el manejador de paquetes npm, comando: npm -v



Si usas el IDE Visual Studio Code con la extensión Go Live puedes ahorrarte el instalr NodeJS.

ANEXO II. Variables, tipos de datos, asignaciones, operaciones, comentarios, sentencias, decisiones y bucles

AII.1. Variables

Las variables se pueden declarar de las siguientes maneras:

Modo	Descripción	Uso	Ejemplo
Automáticamente	Se le asigna un valor		x=5;
var	Usado en código de 1995 hasta 2015 y navegadores antiguos.	Solo si debe soportar navegadores antiguos.	var x = 5;
let	Se agregaron a JavaScript en 2015	Variables que pueden tomar diferentes valores en el tiempo. Si no puede usar const.	let x =5;
const		Si no cambia el valor o tipo	const x=5;

Uso concreto de **let** (se introdujo en ES6 (2015)).

Uso	Ejemplo

Tienen alcance de bloque	{
	  let x = 2;
	}
	// x can NOT be used here

Deben declararse antes de su uso	carName = "Saab"; let carName = "Volvo";
No se pueden redeclarar en el mismo ámbito	let x = "John Doe"; let x = 0;

AII.2. Tipos de datos

Los tipos soportados por Javascript son:

Tipo	Ejemplo	Almacenamiento
String	let color = "Yellow";	

Number	let length = 16;
	let weight = 7.5;
	let z = 123e-5;
		64 bits punto flotante.

Bigint	const bigNumber = BigInt("123456789012345678901234567890	Números integer más allá del límite 2^53 - 1 .

	0");	
Boolean	let x = true;	

Undefined	let x;
	if (x === undefined) {
	  text = "x is undefined";
	} else {
	  text = "x is defined";
	}
		Indica que una variable no tiene asignado un valor.

Null	Var persona = null; persona = {nombre=”José”, apellido=”Miró”}	Se aplica a objetos para indicar ue es nada o que no existe.

Object	const person = {firstName:"John",
	lastName:"Doe"};
	const cars = ["Saab", "Volvo", "BMW"];
	const date = new Date("2022-03-25");	

AII.2.a Conversiones de datos

Conversión a String

Ejemplo	Conversión a String	Resultado
let numero = 123;	let strNumero = String(numero);	“123”
let booleano = true;	et strBooleano = String(booleano);	“true”
let objeto = { nombre: "Juan" };	let strObjeto = String(objeto)	"[object Object]"

Podemos usar también:

```
let strNumero2 = numero.toString();  // "123"
let strBooleano2 = booleano.toString(); // "true"
```

Pero .toString() no funciona con null y undefined.

Conversión a Number, parseInt, parseFloat

Ejemplo	Conversión a Number	Resultado
let strNumero = “123”;	let num1 = Number(strNumero);	123
let strNoNumero = “123abc”	let num2 = Number(strNoNumero);	NaN*
let boolean= true;	let num3 = Number(boolean)	1**
	let entero = parseInt("123.45");	123
	let decimal = parseFloat("123.45");	123.45

* NaN significa Not a Number
** true → 1, false → 0

Conversión a Boolean

Ejemplo	Conversión a Boolean	Resultado
let strVacia = "";	let bool1 = Boolean(strVacia);	false
let strTexto = "Hola";	let bool1 = Boolean(strVacia);	true
let numeroCero = 0;	let bool1 = Boolean(strVacia);	false
let numeroPositivo = 123;	let bool1 = Boolean(strVacia);	true

Conversión a BigInt

Ejemplo	Conversión a BigInt	Resultado
let strNumero = "12345678901234567890";	let bigInt1 = BigInt(strNumero);	12345678901234567890n

AII.3. Asignaciones

Son las igualdades y algunas tienen una forma peculiar de trabajar.

Normalmente cuando a una misma variable queremos aumentar su valor haríamos lo siguiente:

X = X +2;

En Javascript, al igual que otros lenguajes, podemos escribir esto mismo de la forma:

X += 2;

Esta operación se lee como “incremento y luego asigno a X”.

Y no es lo mismo que:

X =+ 2;

Que en este caso estamos asignando un número positivo.

AII.4. Operaciones

AII.4.a. Aritméticas

Operador	Operación	Operador
Igual	x = y	=
suma	x = x + y	+=
resta	x = x - y	-=
multiplicación	x = x * y	*=
división	x = x / y	/=
módulo	x = x % y	%=
esponencial	x = x ** y	**=

AII.4.b. Desplazamiento de bits

Operador	Operador	Example	Equivale a
Desplaza a la izquierda con signo	<<=	x <<= y	x = x << y
Desplaza a la derecha con signo	>>=	x >>= y	x = x >> y
Desplazamiento sin signo	>>>=	x >>>= y	x = x >>> y

Un uso que se le suele dar es la obtención del peso de los colores primarios de un color dado. ¡Pruébalo tu mismo!

```
// Extraer componentes de color ARGB de un valor de 32 bits
let color = 0xFFA07A00; // Color naranja semitransparente

let alpha = color >>> 24;       // 0xFF (255)
let red = (color >>> 16) & 0xFF; // 0xA0 (160)
let green = (color >>> 8) & 0xFF; // 0x7A (122)
let blue = color & 0xFF;         // 0x00 (0)
```

AII.4.c. Operaciones bit a bit

Son las realizadas entre números binarios pero bit a bit.

Operator	Example	Same As
&=	x &= y	x = x & y
^=	x ^= y	x = x ^ y
|=	x |= y	x = x | y

AII.5.d. Operadores lógicos

Son las realizadas entre números binarios.

Operator	Example	Same As
&&=	x &&= y	x = x && (x = y)
||=	x ||= y	x = x || (x = y)
??=	x ??= y	x = x ?? (x = y)

AII.5.e. Type of

En javascript Typeof es un tipo, no una función como en otros lenguajes de programación.

Nos permite conocer el tipo de la variable.

AII.5.f. Operadores Unarios

Son los operadores de incremento y decremento en una unidad.

Operador	Ejemplo

++	let x = 5;
	x++;
	let z = x;

--	let x = 5;
	x--;
	let z = x;

¿Crees que será los mismo?

Let x = 5;

let y = ++x;

qué

let y = x++;

Vamos a ver este efecto con el siguiente código:

```
console.log("Tened en cuenta el orden en que se aplica el operador");
var edadCarmen = 18;
console.log("Aplico el opeorador DESPUES. La edad de Carmen: " + edadCarmen++);
console.log("Vuelvo a imprimir la edad de Carmen: " + edadCarmen);
console.log("--------------------");
edadCarmen = 18;
console.log("Aplico el opeorador ANTES. La edad de Carmen: " + ++edadCarmen);
console.log("Vuelvo a imprimir la edad de Carmen: " + edadCarmen);
```

Estudia el código y observa la diferencia entre edadCarmen++ y ++edadCarmen.

AII.5. Comentarios al código

Los comentarios de una sola línea comienzan con `//`.

Cualquier texto entre `//`y el final de la línea será ignorado por JavaScript (no se ejecutará).

Ejemplo: // Esto es un ejemplo

Los comentarios de varias líneas comienzan con `/*`y terminan con `*/`.

Cualquier texto entre `/*`y `*/`será ignorado por JavaScript.

Ejemplo:

```
/*
Esto es un ejemplo de bloque comentado
*/
```

AII.6. Sentencias

Las sentencias son instrucciones individuales que realizan acciones. Son los bloques básicos de construcción de un programa JavaScript.

Las declaraciones de JavaScript se componen de: Valores, operadores, expresiones, palabras clave y comentarios.

Las sentencias se ejecutan, una por una, en el mismo orden en que están escritas.

Cada línea termina en punta y coma (;), no es obligatorio pero sí recomendable.

Si usamos “;”podríamos escribir: a = 5; b = 6; c = a + b; en una sola línea.

Los espacios son ignorados por javascript, así que: **let person = "Hege"**; y **let person="Hege";** es lo mismo.

Una buena práctica es colocar espacios alrededor de los operadores (= + - * /):

```
let x = y + z;
```

Las declaraciones de JavaScript se pueden agrupar en bloques de código, dentro de llaves {...}. Usado en la declaración de funciones, por ejemplo.

```
function miFuncion() {
    // Código de la función
};
```

AII.7. Decisiones

Las decisones permiten controlar el flujo de ejecución del programa basado en condiciones.

AII.7.a. if….else

Ejecuta un bloque se código si una condición es verdadera.

Veamos algunos ejemplos para que los pruebes.

```
var nombre = 'Pablo';
var estadoCivil = 'soltero';
var estaCasado = true;

if (estaCasado){
    //si es verdadera la condición
    console.log(nombre + ' esta casado');
}else {
    //si es falsa la condición
    console.log(nombre + ' esta soltero');
}
```

Si se requieren más condiciones se usará **else if.**

```
var nombre = 'Pablo';
var edad = 8;

// edad < 12 es un niño.
// edad > 11, es < 18, es un adolescente.
// edad > 17, es < 60, es un adulto.
// edad > 60, es un anciano.

if (edad < 12){
    console.log(nombre + ' es un niño.');
}else if ((edad > 11) && (edad < 18)){
    console.log(nombre + ' es un adolescente.');
}else if ((edad > 17) && (edad < 60)){
    console.log(nombre + ' es un adulto.');
}else{
    console.log(nombre + ' es un anciano.');
}
```

Evaluación de variables lógicas

Nos referimos a cómo se evalúa una variable. Veamos un ejemplo:

```
var edad;
 //edad = 10;

 if(edad){
   console.log('Variable esta definida');
 }else{
   console.log('Variable no definida');
 }

 //operadores de igualda
if(edad === '10'){
  console.log('Variable con coersión');
}else{
  console.log('Variable sin coersión');
}
```

Aquí he definido la variable pero no le he dado un valor y resuta:

```
Variable no definida                    app.js:118
Variable sin coersión                   app.js:125
```

Ahora si yo doy el valor 10 a edad se devolverá:

```
Variable esta definida                  app.js:116
Variable sin coersión                   app.js:125
```

Resulta que el segundo if estoy comprando un número con una cadena, por ello me dice que no hay coersión. Escribamos if(edad === 10) a ver qué ocurre. Resulta:

```
Variable esta definida                  app.js:116
Variable con coersión                   app.js:123
```

Ahora sí que tiene un valor y la comparación del segundo if corresponde los tipos.

AII.7.b. Operador ternario

Otra forma de escribir el if, de forma más concisa es a través del operador ternario, que tiene una estructura como:

(condición a evaluar) ? (acción a realizar si es Verdadero) : (acción a realizar si es Falso) ;

Imagina que yo tuviese:

```
if (edad >= 18){
   console.log(nombre + ' es mayor de edad.');
}else {
   console.log(nombre + ' es un adolescente.');
```

Usando el operador ternario quedaría:

```
edad >= 18 ? console.log(nombre + ' es mayor de edad') :
                console.log(nombre + ' es un adolescente');
```

AII.7.c. Switch

Cuando sabemos que valores puede tomar una variable y por cada uno de ellos tenemos que realizar acciones usaremos la estructura switch.

Veamos un ejemplo. Capturemos el día de la semana e imprimirlo.

El método **getDay()** devuelve el día de la semana como un número entre 0 y 6.

(Domingo=0, Lunes=1, Martes=2..)

```
switch (new Date().getDay()) {
  case 0:
    day = "Sunday";
    break;
  case 1:
    day = "Monday";
    break;
  case 2:
     day = "Tuesday";
    break;
  case 3:
    day = "Wednesday";
    break;
  case 4:
    day = "Thursday";
    break;
  case 5:
    day = "Friday";
    break;
  case 6:
    day = "Saturday";
}
```

AII.8. Bucles

AII.8.a. for

Se ejecuta un código tantas veces como se cumpla una condición.

Sigue la sintaxis:

```
for (expression 1; expression 2; expression 3) {
  // code block to be executed
}
```

La expresión 1 se ejecuta (una vez) antes de la ejecución del bloque de código.
La expresión 2 define la condición para ejecutar el bloque de código.
La expresión 3 se ejecuta (cada vez) después de que se haya ejecutado el bloque de código.

Por ejemplo:

```
for (var i = 10;i >= 1;i--){
  console.log(i);
}
```

¿Qué se imprime por consola?

Otros usos de la expresión 1

Pero, la expresión 1 es opcional.

Puede omitir la expresión 1 cuando sus valores se establecen antes de que comience el bucle. En el siguiente ejemplo la variable “i” se inicializa a 2 antes del for, al igual que len.

```
let i = 2;
let len = cars.length;
let text = "";
for (; i < len; i++) {
  text += cars[i] + "<br>";
}
```

Podemos iniciar muchos valores en la expresión 1 (separados por coma):

```
for (i = 0, len = cars.length, text = ""; i < len; i++) {
  text += cars[i] + "<br>";
}
```

Otros usos de la expresión 3

La expresión 3 también se puede omitir (como cuando incrementas tus valores dentro del bucle). Véase el siguiente ejemplo donde “i” es incrementado dentro del bucle.

```
let i = 0;
let len = cars.length;
let text = "";
for (; i < len; ) {
  text += cars[i] + "<br>";
  i++;
}
```

Alcance de var y let

Veamos el ejemplo de var:

```
var i = 5;

for (var i = 0; i < 10; i++) {
  console.log(“Dentro del for i vale “ + i);
}
 console.log(“Fuera del for i vale “ + i);
```

Veamos el ejemplo de let:

```
let i = 5;

for (let i = 0; i < 10; i++) {
  console.log(“Dentro del for i vale “ + i);
}
 console.log(“Fuera del for i vale “ + i);
```

¿Qué observas? ¿Qué conclusiones obtienes?

AII.8.b. for/in

Declaración que recorre las propiedades de un objeto.

La sintaxis es:

```
for (key in object) {
  // code block to be executed
}
```

Veamos un ejemplo donde recorremos el objeto “persona”.

```
const person = {fname:"David", lname:"Aria", age:25};

let text = "";
for (let x in person) {
  text += person[x];
}
```

AII.8.c. for/of

Recorre los valores de un objeto iterable, como matrices, cadenas, mapas, listas de nodos y más.

Veamos el siguiente ejemplo:

```
const cars = ["BMW", "Volvo", "Mini"];

let text = "";
for (let x of cars) {
  text += x;
}

console.log(text);
```

¿Qué imprime el ejemplo?

AII.8.d. while

La sentencia while ejecuta un código mientras se cunpla una condición. Se tiene que inicializar la variable a chequear antes de iniciar el while.

```
var i = 10;
while(i >= 1){
  console.log(i);
  i--;
}
```

AII.8.e. do...while

La sentencia do...while es similar al while, solo que se ejecuata una vez el código que hay dentro.

```
var i = 3;
do{
  console.log(i);
  i++;
}while(i <= 10)
```

ANEXO III. Arrays, objetos y contexto this

AIII.1. Arrays

Un array es una colección ordenada de elementos. Puede contener cualquier tipo de dato, incluidos otros arrays u objetos.

La Declaración y acceso es la siguiente:

```
let frutas = ["manzana", "plátano", "pera"];
console.log(frutas[0]); // "manzana";
```

AIII.1.a. Métodos comunes

- `push()`: Añade un elemento al final.
- `pop()`: Elimina el último elemento.
- `shift()`, `unshift()`: Eliminan/añaden elementos al principio.
- `length`: Devuelve la longitud del array.
- `forEach()`: Recorre el array y ejecuta una función por cada elemento.

Ejemplo:

```
let numeros = [1, 2, 3];
numeros.push(4); // [1, 2, 3, 4]
numeros.forEach(num => console.log(num));
```

Crea un archivo con el código anterior y ejecútalo. ¿Qué obtenemos por consola?

AIII.1.b. Modificación de los elementos

Por ejemplo si en vegetales = new Array(‘Tomate’, ‘Lechuga’, ‘Zanahoria’); quiero cambiar Lechuga por Apio haría:

vegetales[1] = “Apio”;

AIII.1.c. Encontrar un elemento, indexOf() y findIndex()

La forma de localizar el índice de un elemento es la siguiente:
Si tengo el arreglos

```
const ceviche = [‘percado’, ‘marisco’, ‘cebolla’];
```

El índice de marisco sería:

```
ceviche.indexOf(‘marisco’);
```

Devolverá: 1.

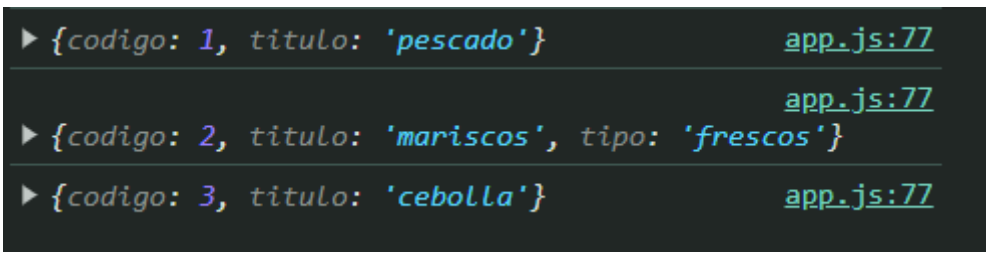
En el caso de tener un arreglo de obletos, la cosa cambia. Por ejemplo si tenemos:

```
const ceviche = [
   {
      codigo: 1,
      titulo: 'pescado'
   },
   {
      codigo: 2,
      titulo: 'mariscos',
      tipo: 'frescos'
   },
   {
      codigo: 3,
      titulo: 'cebolla'
   }
];
```

La forma de buscar un objeto es con findIndex(). Se usaría como:

```
const index = ceviche.findIndex(function(ing, index){
        console.log(ing);
});
```

Devuelve:



En el caso de buscar el ingrediente cebolla haría algo como:

```
const index = ceviche.findIndex(function(ing, index){
```

```
        return ing.titulo === 'cebolla';
    });
    console.log(index);
```

Que devolvería el índice 2.
Si fuese uno que no existiera devolvería -1.

AIII.1.d. Filtrar arreglo según una condición. Filter()

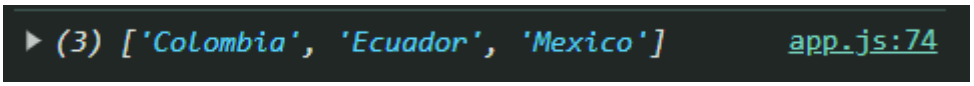
Imaginemos que tengo el arreglo de países;

```
const paises = ['Colombia', 'Ecuador', 'Perú', 'España', 'Mexico'];
```

Y quiero filtrar los que contenga la letra “o”.

```
console.log(paises.filter(pais => pais.includes('o')));
```

Devuelve:



Si quisiera los que comiencen por “Es”:

```
console.log(paises.filter(pais => pais.includes('Es')));
```

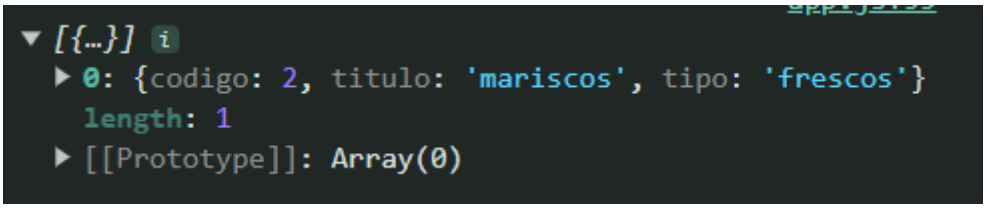
Devolvería:

Si fuese un arreglo con objetos tendríamos que hacer algo como:

Puedo crear una función para filtrar. Y en este caso filtro por el ingrediente que contenga las letras “mar”:

```
const filtrarIngredientes = ceviche.filter(function(ing, index){
   const ingredientes = ing.titulo.includes('mar');
   return ingredientes;
});
console.log(filtrarIngredientes);
```

Esto devuelve:



AIII.1.e. Ordenación. order()

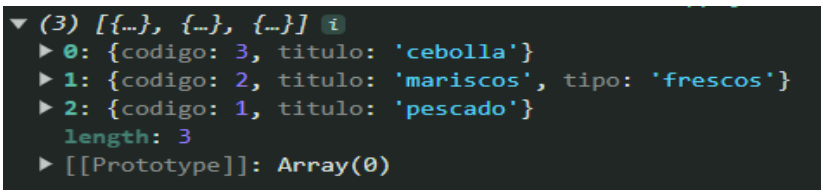
Para ordenar un arreglo de objeto y siguiendo con el ejemplo de cebiche:

```
const ceviche = [
   {
      codigo: 1,
      titulo: 'pescado'
   },
   {
      codigo: 2,
      titulo: 'mariscos',
      tipo: 'frescos'
   },
   {
      codigo: 3,
      titulo: 'cebolla'
   }
];
```

Para ordenarlo sigo lo que indica la teoría y es aplicar una función:

```
ceviche.sort(function(a,b){
   if(a.titulo > b.titulo){
      return 1;
   }
   if(a.titulo < b.titulo){
      return -1
   }
   return 0;
});
console.log(ceviche);
```

Que devuelve:



AIII.2. Objetos

Los objetos permiten agrupar datos mediante pares clave-valor.

AIII.2.a. Declaración de objetos

La forma de declaración es la siguiente:

```
objeto = {elemento1: ‘valor1’,
          elemento2: ‘valor2’}
```

Por ejemplo:

```
let persona = {
  nombre: "Ana",
  edad: 25,
  saludar: function() {
    console.log("Hola, soy " + this.nombre);
  }
};
```

AIII.2.b. Creación de objeto

Puedo crear el objeto con la ayuda de la **sintaxis Object** de la siguiente manera:
Si hiciéramos el ejemplo de persona:

```
let persona = new Object();
        persona.nombre = ‘David’;
        persona.edad= 20;
        persona.ciudad = ‘Puerto de la Cruz’;
        persona.gistos = [‘Tenis’, ‘Cine’, ‘Camping’];
        persona.escasado = true;
```

AIII.2.c. Acceso y modificación

Se realiza a través de *.propiedad*

```
console.log(persona.nombre); // "Ana"
persona.edad = 26;
persona.saludar();
```

AIII.3. El contexto `this`

this es una palabra clave que hace referencia al contexto en el que se ejecuta una función. Su valor depende de cómo y dónde se llama a la función.

Ejemplo:

```
const perro = {
  nombre: "Rex",
  ladrar: function() {
    console.log(`¡Guau! Soy ${this.nombre}`);
  }
};

perro.ladrar(); // "¡Guau! Soy Rex" (this = objeto `perro`)
```

En este caso this hace referencia al objeto perro.

this en eventos del DOM apunta al elemento que lo disparó, por ejemplo:

```
<button id="miBoton">Click aquí</button>

<script>
  document.getElementById("miBoton").addEventListener("click", function() {
    console.log(this); // El botón (`<button id="miBoton">`)
  });
</script>
```

En este caso this es el objeto botón.

ANEXO IV. Funciones (función flecha, callbacks, scope, clousures)

AIV.1. Funciones

Una función es un bloque que contiene un código a ejecutar que puede recibir parámetros. Las funciones pueden o no devolver parámetros.

La definición se hace con la palabra reservada **function**:

```
function miFuncion([parámetros de entrada){
        (código a ejecutar)
        [return parámetros de salida;]
}
```

La invocación de la función es llamándola por su nombre:

```
miFuncion([parámetros de entrada);
```

En caso de devolver algo se lo asignamos a una variable:

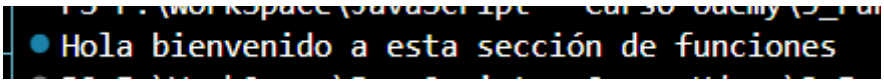
```
var valores = miFuncion([parámetros de entrada);
```

Veamos un ejemplo de una función que no recibe parámetros ni devuelve nada:

```
function bienvenida(){
    console.log("Hola bienvenido a esta sección de funciones");
}

bienvenida();
```

Si desde una terminal, dentro del proyecto, ejecuto el fichero con la sentencia "node app.js" tendré como resultado:



El mismo ejemplo, pero haciendo que devuelva el mensaje sería:

```
function bienvenida(){
    console.log(“Hola bienvenido a esta sección de funciones”);
}

var mensaje = bienvenida();
console.log(mensaje);
```

Obteniendo el mismo resultado.

AIV.1.a. Argumentos de entrada de una funciones

Son los parámetros que se le pasa a una función. Veamos un ejemplo:

```
function cuadradoNumero(numero){
   return numero*numero;
}

var num = 3;
var result = cuadradoNumero(num);
console.log(result);
```

Hemos creado la función y una variable, num, con el valor 3.
Llamamos a la función pasando la variable num: cuadradoNumero(num)

La respuesta de la función se guarda en la variable result, 9.

También podría haber hecho directamente:

```
console.log(cuadradoNumero(num));
```

AIV.1.b. Uso de funciones como expresiones

Es una forma de exprear la sintaxis de una función de manera distinta. Su sintaxis, haciendo uso de un ejemplo, es:

```
var prueba = function(){
        console.log(“Esto es una prueba.”);
}
```

Como puedes ver hay tres grandes diferencias:

- La función no tiene nombre
- La función se asigna a una variable
- La llamada se hace a la variable, pero con los ().

Estamos habituados a mostrar mensajes compuestos concatenando cadenas de la siguiente forma:

```
console.log("Tienes " + currentAge(yearBirthday) +  " años.");
```

Hay otra forma de mostrar la cadena de salida, haciendo uso de las bondades de javascript, el Template Screen sería:

```
console.log(`Tienes ${currentAge(yearBirthday)} años.`);
```

Nótese que las comillas son las acostada a la derecha y que la llamada a la función va entre llaves, precedido del símbolo $, es la manera de insertar variables o resultado de funciones de javascript en esta notación.

AIV.1.c. Argumentos undefine

Es cuando yo la creo una variable pero no le asigno valor y esta es pasada a una función.

```
var nombre;
```

Veamos el ejemplo:

```
var nombre;
var prueba = function(n){
   return 'Hola ' + n;
}
console.log(prueba(nombre));
```

Devuelve:

```
Hola undefined                                                    app.js:76
```

Javascript no devuelve error. Solo indica que no tiene un valor, undefine.

En este caso hay un puntero con una dirección de memoria pero que no contiene valor.

AIV.1.d. Argumentos nulos

En este caso la varible pasada es de valor nulo. Veamos que sucede:

```
var a;
a = null;

var valorNulo = function(n){
   return n
}
console.log(valoNulo(a));
```

Esto devuelve:

```
null                                                              app.js:87
```

En este caso hay un puntero con una dirección de memoria pero que contiene valor null. Es decir null se considera un valor y es legítimo usarlo en comparaciones.

AIV.1.e. Argumentos por defecto

Imaginemos que tenemos:

```
var sumar = function(a, b, c){
   return a +b + c;
}

console.log(sumar(10, 4));
```

Esto da como resultado:

```
NaN                                                               app.js:96
```

¿Por qué sucede esto? Porque c tiene un valor null y su tipo no es igual a y b.

Lo que puedo hacer es dar un valor por defecto de la siguiente manera:

```
var sumar = function(a, b, c=3){
   return a +b + c;
}

console.log(sumar(10, 4));
```

En la función yo doy un valor a la variable c. Si javascript que c no se le está pasando como parámetro le dará este valor de 3 y ejecutará el código. En este caso da como resultado:

```
17                                                    app.js:96
```

Pero si yo hiciese:

```
var sumar = function(a=5, b=3, c=3){
   return a +b + c;
}

console.log(sumar(10, null, 7));
```

Da como resultado:

```
17                                                    app.js:96
```

¿Por qué 17 y no 11? Porque null es considerado como valor 0.

AIV.1.f. Función anónima

Las funciones anónimas son funciones que no tienen un nombre definido. Se usan comúnmente en callbacks, IIFE (Immediately Invoked Function Expressions), closures y como argumentos de otras funciones.

Un ejemplo sencillo sería:

```
// Función anónima asignada a una variable
const saludar = function() {
  console.log("Hola mundo");
};
saludar(); // "Hola mundo"
```

Más adelante veremos eñ uso en callbacks e IIFE.

AIV.2. Función flecha

La función flecha es artificio que nos permite crear funciones un poco más dinámicas y que nos permiten ahorrar cierto código. Sus características son:

- No tienen su propio this (toman el this del contexto superior)
- No tienen arguments propio
- Muy útiles como callbacks

Veamos un ejemplo. Dado el año de nacimiento calcula la edad.

A lo que estamos habituado es:

```
Const years = [200, 2005, 2008, 2012];

var edad = years.map(function(el){
        return 2019 – el;
});

console.log(edad);
```

Lo anterior devuelve: 19, 14, 11, 7.

Haciendo uso de la función flecha:

```
const years = [200, 2005, 2008, 2012];

let edad = years.map(el => 2019 – el);

console.log(edad);
```

Devolviendo lo mismo.

En el caso de tener vario parámetros lo podremos entre paréntesis separados por comas:

```
const sumar = (a, b) => a + b;
```

En el caso de tener que ejecutar varias líneas las pondremos entre llaves:

```
const saludo = (nombre) => {
  let mensaje = "Hola " + nombre;
  return mensaje;
};
```

AIV.3. Callback

Un callback es una función que se pasa como argumento a otra función, y se ejecuta después.

Veámoslo mejor con un ejemplo:

```
function hacerAlgoDespues(nombre, callback) {
  console.log("Preparando saludo...");
  callback(nombre);
}

hacerAlgoDespues("Carlos", function(nombre) {
  console.log("Hola, " + nombre + " :)");
});
```

Vemos que declaramos una función llamada hacerAlgoDespues() al que se le pasa como segundo parámetro una función que en el cuerpo de la función es llamada pasándole “nombre”.

Más abajo invocamos a la función pasándole un nombre y una función anónima que muestra por consola “Hola Carlos”.

Se puede usar la función flecha cuando invocamos a hacerAlgoDespues(), sería:

```
hacerAlgoDespues("Carlos", nombre => console.log("Hola, " + nombre));
```

AIV.4. Scope

Hace referencia al ámbito de la variable. Esta puede ser:

- **Global**: fuera de cualquier función o bloque.
- **De función**: dentro de una función.
- **De bloque**: dentro de `{}` con `let` o `const`.

Un ejemplo donde podemos ver los tres tipos sería el siguiente:

```
let x = 10; // global

function demo() {
  let y = 20; // local
  if (true) {
    let z = 30; // de bloque
    console.log(x, y, z); // Esto funcionará
  }
    console.log(z); //  Esto dará Error
}
```

AIV.5. Clousures

Un closure es cuando una función interna recuerda el contexto (variables) en el que fue creada.

Un ejemplo típico es el siguiente:

```
function crearContador() {
  let contador = 0;

  return function() {
    contador++;
    console.log(contador);
  };
}

const contar = crearContador();
contar(); // 1
contar(); // 2
```

Aunque crearContador() terminó su ejecución, la función interna sigue teniendo acceso a contador, siguie mantenidendo el último valor.

ANEXO V. Como pasar parámetros a un archivo javascript cuando lo ejecuto por prompt

Cuando ejecutas un script con Node.js, los parámetros se pasan después del nombre del archivo, por ejemplo:

node miScript.js parametro1 parametro2 parametro3

ANEXO VI. Como acceder a los parámetros pasados a un archivo javascript por promt

Se accede a los parámetros a través del array **process.argv.** Por ejemplo:

Ejecuto el fichero calculadora para sumar dos números:

node calculadora.js sumar 5 3

El array process.argv[] tiene los siguientes valores:

process.argv[0]: ruta del ejecutable Node.
process.argv[1]: ruta del script que estás ejecutando.
process.argv[2]: parámetro sumar
process.argv[3]: parámetro 5
process.argv[4]: parámetro 3

El código del fichero calculadora.js sería:

```
const operacion = process.argv[2];
const num1 = parseFloat(process.argv[3]);
const num2 = parseFloat(process.argv[4]);

switch(operacion) {
  case 'sumar':
    console.log(`Resultado: ${num1 + num2}`);
    break;
  case 'restar':
    console.log(`Resultado: ${num1 - num2}`);
    break;
   / / otras operaciones...
  default:
    console.log('Operación no válida');
}
```