1. Conocimiento el entorno de desarrollo: Node y JavaScript

1.1. Instalación de Visual Studio Code y Node.js

https://code.visualstudio.com/

1.2. Instalación de Node.js

https://nodejs.org/

1.3. Visual Studio Code

Configuración

1.4. Variables y tipos de datos

- ¿Qué es una variable?:

Definición formal: es un espacio reservado de la memoria en donde se pueden almacenar distintos datos bajo un nombre único que está asociado a dicho dato.

La memoria RAM es la encargada de almacenar toda la información necesaria para utilizar en la computadora mientras está encendida, por lo que al apagarla, esa información se pierde, ya que la memoria RAM retiene la información de manera temporal.

<table>
<tr><td colspan="13">Memoria RAM</td></tr>
<tr><td></td><td colspan="4" rowspan="3">Escritura de un documento</td><td></td><td></td><td colspan="4" rowspan="3">Páginas web abiertas</td><td></td></tr>
<tr><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td colspan="4" rowspan="2">Juego</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="3">Variable</td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>

Al declarar una variable, lo que se está haciendo, técnicamente, es ordenar a la computadora que agarre un espacio que tenga disponible en la memoria y lo guarde mediante un nombre, indicado por el programador, y lo reserve para que a futuro se pueda colocar cualquier dato que sea necesario persistir.
Como su nombre lo indica, son VARIABLES, lo que significa que en cualquier momento se puede eliminar y cargar otro dato o valor diferente, siempre teniendo en cuenta que una variable puede almacenar un solo dato.

Consideraciones importantes:
Cuando se crea una variable, no se decide en qué espacio de la memoria RAM se va a guardar el dato, sino que la computadora es la encargada de agarrar algún lugar o espacio disponible y reservarlo.

Sobre lo que si hay decisión es el nombre que se le va a poner a la variable, lo que se conoce como IDENTIFICADOR.

Por lo tanto una variable es un espacio de toda la memoria RAM que se reserva bajo un único nombre para persistir datos de manera temporal.

(Ver archivo)

Para crear variables se hace uso de ciertas “palabras reservadas”, que son palabras clave necesarias para que la computadora comprenda de qué orden o instrucción se hace referencia al momento de darle un accionar o que deba realizar.

- **let nombre;** :

La palabra reservada ***let*** indica a la computadora que se quiere guardar un espacio en memoria porque se va a crear una variable, ya que, a futuro, se va a crear un dato ahí. La instrucción termina y se cierra con un punto y coma (;), aunque en otros lenguajes es necesario terminar así las instrucciones, JavaScript no lo hace necesario.

Reglas para nombrar variables:

- Los nombres deben ser representativos o descriptivos del valor que la variable va a contener.
- Los nombres no pueden tener espacios. Se pueden identificar usando guiones o mayúsculas: numero_uno, numeroUno.
- Los identificadores tampoco pueden comenzar con caracteres especiales o números, pueden contenerlos pero no empezar con ellos.

Esta acción de decirle a la computadora que reserve un espacio en memoria se llama: **DECLARACIÓN DE VARIABLES**.

Luego, se puede llamar al identificador y cargar un dato:

- **nombre = valor;** :

Esto se llama **ASIGNACIÓN**, y consiste en cargar o asignar un dato a la variable y consecuentemente, cargarlo en la memoria.

- **let nombre = valor;** :

Se puede declarar y asignar un valor en la misma instrucción.

- Tipos de datos:

El tipo de dato es la clasificación del valor del dato que se está cargando en la variable.

- Enteros/Integer:

Números o datos numéricos enteros, que no tienen coma ni decimales: 1, 2, 3, 100….

- Flotante/Float:

Son números con coma o con decimales: 14.15, 16.78, 34,44….

- Booleano/Boolean:

Tipo de dato que solo acepta dos valores: verdadero o false, true o false.

- Caracter/String:

Se escriben entre comillas, son cadenas de texto: “texto”, ‘texto’, `texto`.

- null:

Es una palabra reservada, un tipo de dato o valor especial que se asigna a variables para indicar explícitamente que no contiene ningún valor, o para inicializarla o limpiarla antes de asignarle un valor real.

- undefined:

Expresa el valor de una variable que ha sido creada o declarada, pero a la que no se le ha asignado ningún valor.

Luego de crear las variables, se pueden controlar sus datos y si realmente el código está haciendo lo que se supone que tendría que hacer, haciendo uso de la terminal.

- **console.log(contenido);** :

Hace una impresión del contenido entre paréntesis, puede ser cualquier cosa, desde datos en su diferentes tipos hasta variables y métodos.

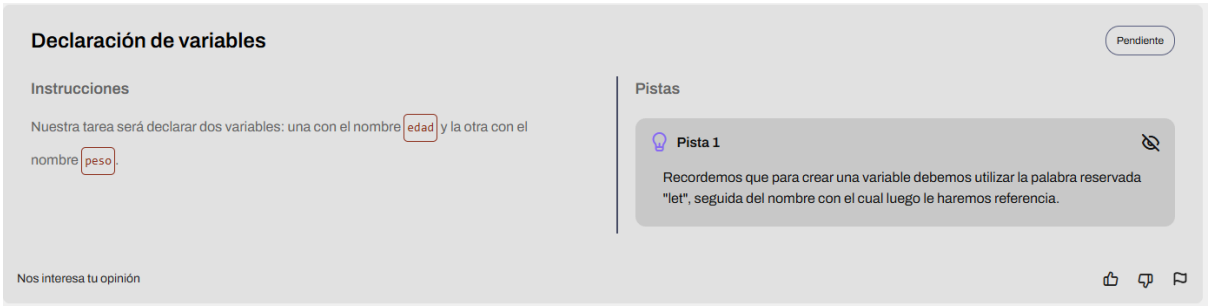
- **node archivo.js**:

Comando para ejecutar desde nodejs, archivos javascript en consola.

- **clear** o **cls**:

Comando para limpiar la terminal.

### 1.5. Declaración de variables




### 1.6. Asignando valores a las variables


### 1.7. Concatenación e interpolación

(Ver archivo)

- Comentarios:

- // **contenido** : Comentarios en línea.
- /* **contenido** */: Comentarios en párrafo.

Alternativas para unir dos variables:

- Concatenación

Es la unión de dos o más cadenas de texto en una sola cadena que se vuelve más larga. Para aplicarla se utiliza el operador "**+**" para unificar las cadenas. Se puede concatenar un espacio vacío entre variables: **variable1 + " " + variable2**.

- Interpolación:

La diferencia utiliza plantillas para unir cadenas textos utilizando comillas graves o comillas invertidas ``. Permite incrustar valores de variables directamente dentro de una cadena de texto utilizando una sintaxis específica, se pueden agregar dentro de la cadena de caracteres dentro del código **${variable}**: **`texto ${variable1} texto ${variable2}`**.

### 1.8. Operadores aritméticos

(Ver Archivo)

Sirven para realizar operaciones matemáticas. Solo pueden ser utilizados con tipos de datos numéricos o flotantes, es decir números enteros y números reales.
Al igual que sucede en matemáticas, al hacer operaciones entre números enteros, el resultado será un número entero, y cuando se hagan operaciones con algún valor que sea un número real, el resultado puede ser un número real.

Cuando se utilizan los operadores aritméticos no es obligatorio poner, de manera manual, un valor fijo, sino que se puede hacer con variables.

Los operadores son exactamente los mismos que se acostumbran a usar en matemáticas:

- **Resta: -**:

Se utiliza el guión medio: **10 - 5; // 5**.

- **Multiplicación: ***:

Se utiliza el asterisco: **10 * 10; // 100**.

- **División: /**:

Se utiliza la barra invertida: **10 / 2; // 5**.

- **Suma: +**:

Se utiliza el símbolo más: **10 + 10; // 20**.

- **Módulo: %**:

Se utiliza el símbolo de porcentaje. Devuelve el resto de una operación: **11 % 2; // 1**.

Al trabajar con diferentes tipos de datos numéricos (enteros y reales), los resultados pueden cambiar, ya sea porque haya un número entero y otro real, o porque el resultado sea un necesario un número real, como en el caso de las divisiones. Por ejemplo:

- **10 + 10 = 20**: entero + entero, resultado entero.
- **10 + 10.5 = 20.5**: entero + real, resultado real.
- **10.0 + 10.0 = 20:** real + real, resultado entero.

Sin embargo, si el tipo de dato primitivo es diferente, como en el caso de sumar (concatenar) una cadena de texto con un número, el resultado es una cadena de texto concatenada:

- **10 + "10" = "1010"**: numero + string, resultado string.

Por otro lado, al utilizar los demás operadores, el tipo de dato string, se convierte en numérico, y por lo tanto, devuelve un tipo de dato número con el resultado también numérico:

- **“10” - 10 = 20**: string, operación, número, resultado número.

### 1.9. Ejercitación con operadores

**Práctica de código**

**Instrucciones**

Vimos cómo declarar una variable y asignarle un valor, y probablemente surgió la siguiente pregunta: ¿para qué sirve almacenar datos en variables? Las variables nos permiten reutilizar el dato asignado en la misma con solo invocar su nombre.

```
let unNumero = 123;

console.log(unNumero); // imprimirá el valor asignado a unNumero, o sea 123
```

También, algo muy importante, así como podemos hacer operaciones matemáticas como sumar (+) o restar (-) números, podemos hacer lo mismo con las variables que las referencien.

Ejemplo:

```
// Esto es una asignación estática (valor fijo)
let unNumero = 124;

// Esto es una asignación dinámica (resultado de una operacion)
let siguienteNumero = unNumero + 1;

// Imprime en pantalla el valor asignado, o sea 124
console.log(siguienteNumero);
```

Ahora si, vayamos al ejercicio:

Declara y asigna valor a dos variables: primerNumero y segundoNumero.

Luego, declara las variables resultadoSuma, resultadoResta, resultadoMultiplicacion, resultadoDivision y en cada una almacena el resultado que corresponda.

Por último, imprimí cada uno de los resultados utilizando `console.log(nombreDeLaVariable)` para ver como se ven los mismos.

```
let primerNumero = 100;
let segundoNumero = 5;

// resultadoSuma será igual a 105
// resultadoResta será igual a 95 (En este caso a primerNumero debemos restarle segundo)
// resultadoMultiplicacion será igual a 500 (En este caso el orden de los factores no altera el producto 😊)
// resultadoDivision será igual a 20 (En este caso debemos dividir a primerNumero por el segundo)
```

### 1.10. Operadores relacionales

(Ver Archivo)

Los operadores relacionales son símbolos que permiten realizar comparaciones en el código, ya sean de valores fijos ingresados manualmente o de valores contenidos dentro de una variable. Se le puede preguntar a una computadora si, por ejemplo, si dos variables son el mismo valor, son distintas, si una es mayor que la otra, arrojando o respondiendo un valor valor booleano, mediante lo que se denomina como un true o un false.

- Mayor que: **>**
- Menor que: **<**
- Mayor o igual: **>=**
- Menor o igual: **<=**
- Igualdad: **==** (compara valor del dato pero no el tipo de dato)
- Igualdad estricta: **===** (compara tanto el valor como el tipo de dato)

- Desigualdad: **!=**
- Desigualdad estricta: **!==**

### 1.11. Operadores lógicos

(Ver Archivo)

Permitan llevar las consultas que se hagan mediante operadores relacionales, un paso más allá, ya que van a funcionar como palabras especiales que van a permitir combinar dos o más consultas o condiciones bajo distintos criterios.

- and: **&&** (ampersand): todas las condiciones deben ser verdaderas para devolver un verdadero. Si una o ambas condiciones son falsas, el resultado será falso.
- or: **||**: una o ambas condiciones deben ser verdaderas para devolver un verdadero. Solo si ambas condiciones son falsas, el resultado será falso.
- not: **!**: cambia el valor de verdad que tiene una consulta específica. Si es verdadera la transforma en falsa y viceversa.

## 2. Trabajando con funciones

### 2.1. Funciones

(Ver Archivo)

Las funciones son herramientas que permiten reutilizar código siempre que sea necesario para evitar repetir instrucciones que ya se hayan escrito anteriormente. Uno de los componentes del pensamiento computacional era el de *detectar patrones*, el cual consiste en encontrar aquellas cosas que se repiten. En el desarrollo de código se pueden encontrar algoritmos e instrucciones que promoverán la necesidad de escribirlos más de una vez.

Las funciones tienen dos etapas:

- Definición:

**function nombre() {...};**.

Se las define, crea, o declara con la palabra reservada **function**, un nombre descriptivo, paréntesis, y llaves, que representan un bloque específico, dentro de las cuales van todas las instrucciones que se van a ejecutar.

- Invocación:

**nombre();**

Cuando se llama a una función se está dando la instrucción de realizar todo lo que se encuentre dentro de la función.

En conclusión, las funciones son un bloque de código que se crea una sola vez, al que se le asigna el algoritmo que sea necesario. Toda función siempre cumple una tarea específica y, una vez que está declarada, se la podrá reutilizar cada vez que sea necesario, evitando repetir código y haciendo que los proyectos sean reutilizables.

### 2.2. Parámetros

(Ver Archivo)

Los parámetros de una función son valores que se le pueden brindar a la función para que esta pueda operar con ellos de la manera que se le indique, permitiendo que la función sea mucho más flexible y reutilizable.

**function nombre(parámetro 1, parámetro 2, …) {...};**

**nombre(valor 1, valor 2, …);**

El nombre de los parámetros puede ser cualquiera, aunque se recomienda que sea descriptivo.
Luego, al invocar a la función, se deben ingresar los valores de los parámetros, respetando su orden ingresado en su declaración.

**function nombre(parámetro = valor) {...};**
Los parámetros pueden definirse con valores por defecto, de manera tal de que si no es declarado en la función, tendrá ese valor por defecto.

### 2.3. Scope y Retornos

(Ver Archivo)
Una vez que se terminan de ejecutar todas las instrucciones dentro de una función, se suele llegar a un valor que es necesario persistir en una variable para su uso futuro. En esos casos se hace uso de los retornos.

El Scope de una variable, se refiere a su alcance, dado que no en todas las situaciones se puede acceder a ellas:

- Variables globales:

Se definen variables en el contexto o ámbito general del código, y pueden ser accedidas desde cualquier parte.

- Variables locales:

La creación de variables se encuentra dentro ciertos bloques de código, por ejemplo, dentro del ámbito de una función, y existen solo dentro de ellos. No pueden ser accedidas fuera de ese ámbito.

Para poder acceder a valores de variables locales en otros ámbitos o scopes, se utiliza la palabra reservada **return**, y el valor o la variable que se quiere devolver. Cuando el código se encuentre con el llamado a la función, realizará todas las instrucciones y luego devolverá o arrojará lo que se encuentre en el **return**, pudiendo así sacar del contexto local, variables hacia otros contextos.

```
function nombreFunc(parámetros, …) {
        …
        return …;
};

let nombreVariable = nombreFunc(parámetros);
```

## 3. Controlando el flujo de la aplicación

### 3.1. If / Else

(Ver Archivo)
Las estructuras condicionales son herramientas que permiten tomar decisiones en el código, haciendo que el programa sea más inteligente y capaz de adaptarse a diferentes situaciones.Las estructuras condicionales se representan con un rombo en los diagramas de flujo, dentro de los cuales se manifestaba una pregunta cuya respuesta solamente podía ser SI o No o verdadero o falso.

Todo esto indica que hay una condición que debe ser analizada y, a raíz de la respuesta, se van a tener que tomar diferentes decisiones. Utiliza la palabra reservada **if**, seguido de una condición, y luego el código o algoritmo a ejecutar si la condición es verdadera. Si la condición es falsa, el algoritmo o las instrucciones dentro del bloque, no se ejecutan:

```
if(condición) {
        algoritmo;
};
```

Para ejecutar instrucciones con condiciones que no se cumplan en el primer bloque **if**, se utiliza la palabra reservada, **else**:

```
if(condición) {
        algoritmo;
} else {
        algoritmo;
};
```

### 3.2. Condicionales múltiples y anidados

(Ver Archivo)

Estructura **else if**, es una estructura que se coloca en el medio entre un **if** y un **else** y es, al igual que un **else**, un acompañamiento de la estructura **if**, es decir, no se la puede comenzar con ella. Lo que hace es permitir agregar una consulta más a la estructura condicional. Pueden existir tantas estructuras condicionales múltiples como sean necesarias:

```
if(condición) {
        algoritmo;
} else if (condición) {
        algoritmo;
} else if (condición) {
        algoritmo;
} … else {
        algoritmo;
};
```

Las estructuras anidadas consisten en estructuras condicionales dentro de estructuras condicionales, ya sea dentro de un **if** o dentro de un **else**. Se pueden anidar tantas condiciones como sean necesarias, tanto en un **if**, como en un **else** y hasta en un **if else**:

```
if(condición) {
        if(condición) {
                algoritmo;
        }
        algoritmo;
} else {
        if(condición) {
                algoritmo;
        }
        algoritmo;
};
```

### 3.3. If Ternario

(Ver Archivo)

También llamado *operador ternario*, consiste en una forma concisa y rápida de escribir declaraciones **if else** en una sola línea, lo que puede ayudar a hacer el código mucho más reducido, legible y eficiente:

**condición ? valor si es verdadero : valor si es falso;**

Se desarrolla una condición, y luego se separa con un signo de pregunta (**?**) las instrucciones o valores si la condición es verdadera, y luego con dos puntos (**:**) las instrucciones si la condición es falsa.

Se puede usar para asignar valores a variables:

**let variable = condición ? valor si es verdadero : valor si es falso;**

También se pueden agregar más de una instrucción utilizando paréntesis y separando por comas los valores y/o instrucciones:

**condición ?**
**(valor si es verdadero, otro valor si es verdadero) :**
**(valor si es falso, otro valor si es falso);**

### 3.4. Switch

(Ver Archivo)

Es una estructura condicional que también permite controlar el flujo de una aplicación y decidir qué bloque o sectores del código se desea ejecutar, con la diferencia principal de que no se estará decidiendo qué instrucción ejecutar en base a una condición, sino que se ejecutará en función al contenido o de una variable o, mejor dicho, en base al valor de una expresión.

Es una estructura de control que permite evaluar una expresión y ejecutar un bloque de código basado en distintas opciones. Esto significa que va a existir una variable con un valor específico que puede cambiar de acuerdo a una situación, y el switch va a elegir un camino o accionar diferente acorde a cada valor de esa variable. Es una situación que se puede resolver con una estructura **if** tradicional, pero al existir muchos caminos a analizar, puede que queden muchos difícil de leer:

```
switch(expresión) {
        case valor:
                algoritmo;
                break;
        case valor:
                algoritmo;
                break;
        …
        default:
                algoritmo;
                break;
}
```

La palabra reservada **break** separa las instrucciones de los diferentes casos e indica al algoritmo que tiene que frenar la ejecución. Si no se expresa **break**, el código sigue ejecutando las siguientes instancias del **switch**.

Finalmente, existe el caso específico **default** que ejecuta instrucciones que no estén contempladas en los **cases** del **switch**.

4. Duelo de variables
    4.1. Let vs Var

(Ver Archivo)
**let** y **var** son palabras reservadas que sirven para crear variables diferenciándose en cuanto a su alcance y comportamiento.

En versiones anteriores de JavaScript, **var** era la única palabra reservada que existía para declarar variables. Su principal inconveniente es que tiene un alcance global, lo que significa que puede ser accedida desde cualquier parte del código generando, quizás, comportamientos inesperados debido la posible reutilización de mismos nombres de variables pudiendo producir reasignaciones y no declaraciones.

Recordar el concepto de **scope**, son los bloques de código entre llaves **{...}**, es decir, es el alcance que tiene la declaración de instrucciones dentro de un bloque. Las variables locales son aquellas que pertenecen a un bloque específico, y las globales se crean con la palabra reservada **var** que, a pesar de ser creada dentro de un bloque particular, se pueden acceder a ellas en cualquier sector.

**let** es una palabra reservada de versiones más recientes de JavaScript que soluciona los conflictos de alcance que tiene **var**. **let** tiene un alcance de bloque, es decir, su alcance se limita solo al bloque donde fue creado. De esta manera evita inconvenientes de declaraciones duplicadas y reasignaciones erróneas.

5. Integración de contenidos
    5.1. Desafio práctico
        5.1.1. Validador de contraseña:

Crea una variable llamada ‘contraseña’ donde guardes un valor específico. Luego, crea una función que reciba por parámetro un dato y utilizando operadores lógicos y condicionales, verifica si dicho parámetro coincide con la variable llamada ‘contraseña’. La función deberá mostrar por consola si la contraseña recibida es válida o no.

        5.1.2. Calculadora de Índice de Masa Corporal (IMC):

Crea una función que calcule el IMC utilizando los parámetros de peso y altura. Usa operadores aritméticos y condicionales para evaluar diferentes rangos de IMC (bajo peso, peso normal, sobrepeso, etc.).

        5.1.3. Conversor de Monedas:

Crea una función que convierte una cantidad de dinero de una moneda a otra. Usa variables para almacenar tasas de cambio y condicionales para manejar diferentes tipos de monedas.

#### 5.1.4. Evaluador de Rango de Edad:

Escribe una función que determine en qué categoría de edad se encuentra una persona. Utiliza las estructuras que consideres necesarias para clasificar a las personas en rangos como niño, adolescente, adulto joven, adulto, etc., según su edad.

#### 5.1.5. Generador de Mensajes Personalizados:

Crea una función que genere un mensaje personalizado para un usuario según su nombre y el momento del día (solo tendremos en consideración si es de mañana, o es de tarde). Utiliza concatenación e interpolación para construir el mensaje, y condicionales (if ternario) para incluir diferentes saludos.

### 5.2. Resolución de desafío práctico

(Ver Archivo)

## 6. Aplicación de habilidades adquiridas

### 6.1. Calculadora - Parte 1

(Ver Archivo)

### 6.2. Calculadora - Parte 2

(Ver Archivo)

## 7. Checkpoint de conocimientos

- ¿Cuál es la principal diferencia entre la concatenación de strings y la interpolación de strings en JavaScript?

La interpolación de strings permite la inclusión de expresiones JavaScript dentro de las cadenas, mientras que la concatenación no.

- ¿Cuál es la diferencia entre “if”, “else if” y “switch” en términos de su comportamiento?

“if” y “else if” evalúan condiciones booleanas, mientras que “switch” se utiliza para comparaciones múltiples de una expresión.

- ¿Cuál es la diferencia entre == y === en JavaScript?’

== compara solo los valores, mientras que === compara los valores y los tipos de datos.

- ¿Qué operador se utiliza para concatenar cadenas de texto en JavaScript?

El caracter es: +

- ¿Cuál es la sintaxis correcta para una función en JavaScript?

function myFunction() {}

- ¿Cuál es el resultado de 11 % 2 en JavaScript?

El resultado es: 1

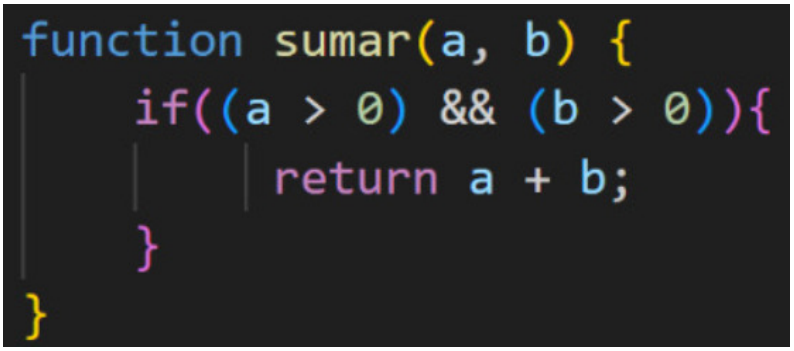
- ¿Cuál es la diferencia principal entre return, console.log() y undefined en una función en JavaScript?

return se usa para finalizar la ejecución de una función y devolver un valor, console.log() muestra mensajes en la consola y undefined es un valor predeterminado cuando una variable no está definida.

- ¿Cuál es el propósito principal del uso de la palabra clave return en una función de JavaScript?

Para devolver un valor específico de la función y finalizar su ejecución.

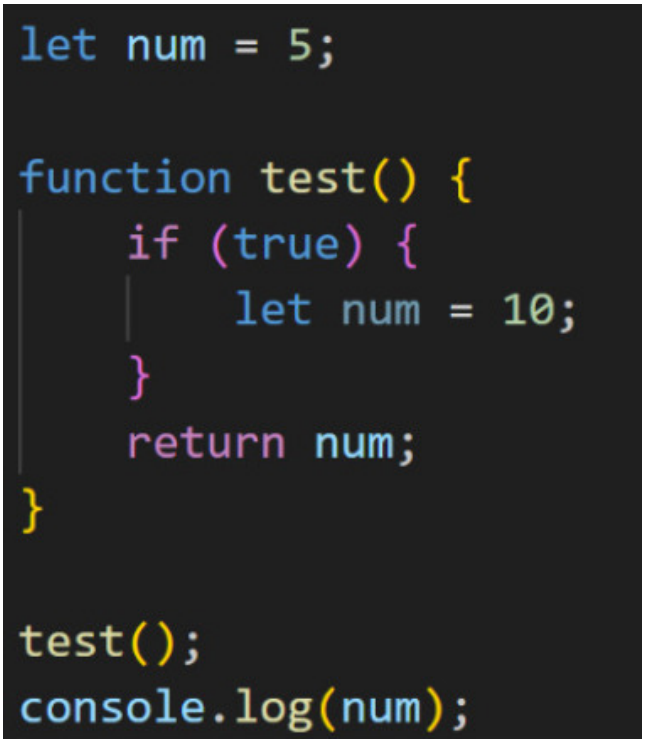
- ¿Qué hace la siguiente función en JavaScript?



```
function sumar(a, b) {
    if((a > 0) && (b > 0)){
        return a + b;
    }
}
```

Retorna la suma de a y b solo si ambos son números positivos.

- ¿Qué se visualizará por consola luego de ejecutar este código?



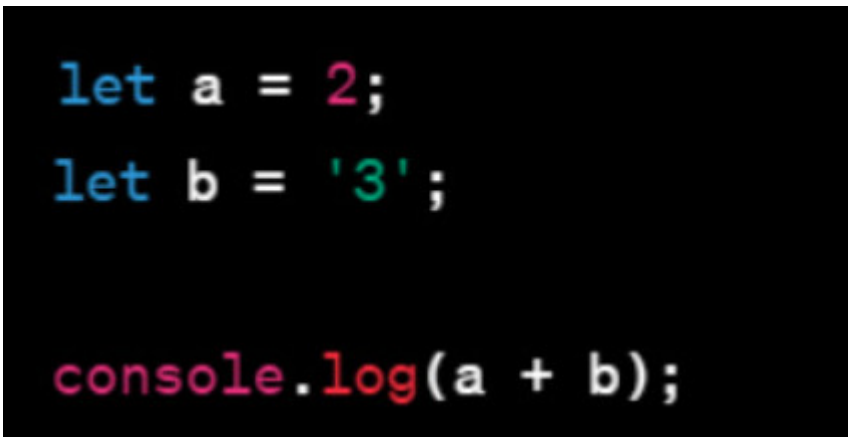
```
let num = 5;

function test() {
    if (true) {
        let num = 10;
    }
    return num;
}

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

Se visualizará: 5

- ¿Qué se imprimirá en la consola al ejecutar este código?

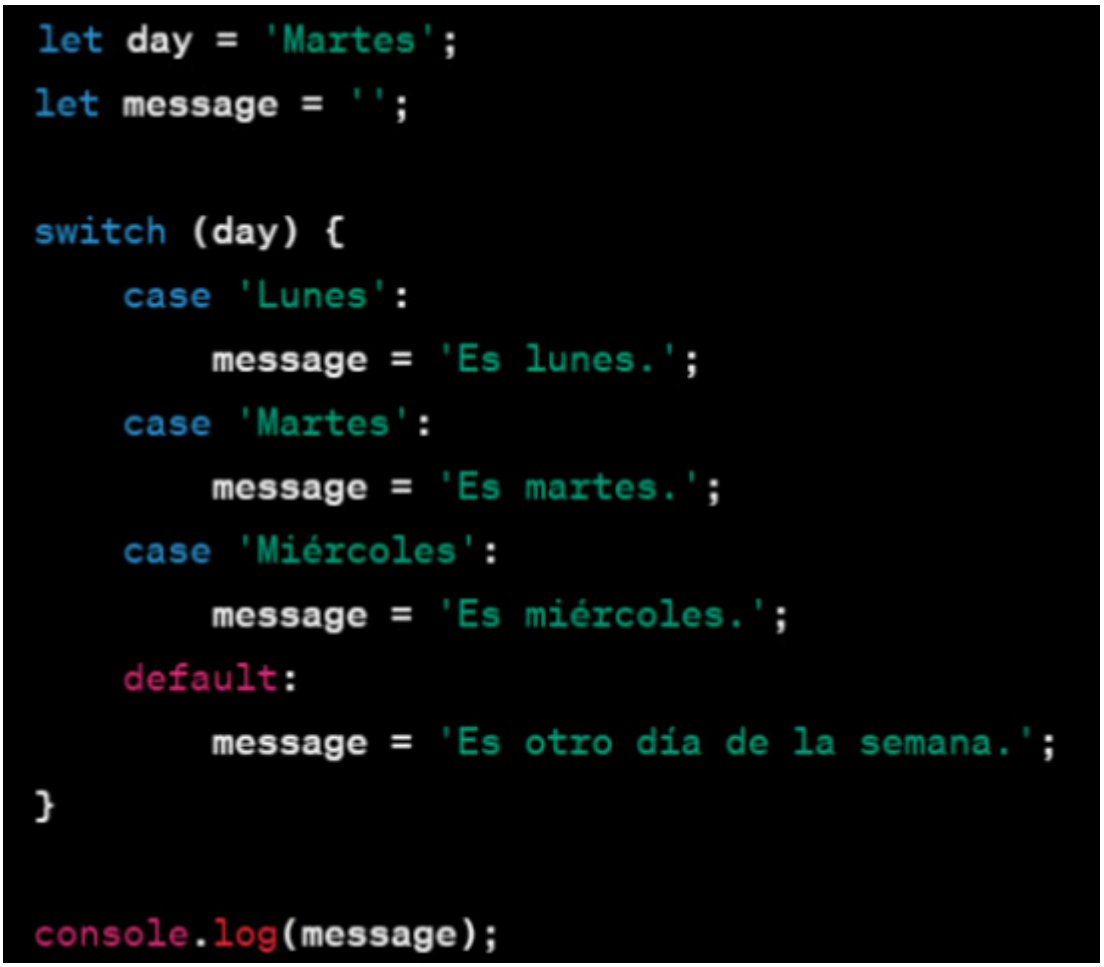


```
let a = 2;
let b = '3';

console.log(a + b);
```

Se imprimirá: "23"

- ¿Qué se imprimirá en la consola al ejecutar este código?



```
let day = 'Martes';
let message = '';

switch (day) {
    case 'Lunes':
        message = 'Es lunes.';
    case 'Martes':
        message = 'Es martes.';
    case 'Miércoles':
        message = 'Es miércoles.';
    default:
        message = 'Es otro día de la semana.';
}

console.log(message);
```

'Es otro día de la semana.'