

# Clase 8:

# Aprendiendo el

# lenguaje

# Índice

## Lenguaje JavaScript:

- Variables
- Constantes
- Tipos de datos
- Tipos de operadores

## Operadores Lógicos

- And
- Or
- Not

## Expresiones y sentencias

- Concepto
- Ejemplo en VSC

## Estándares de nomenclatura:

- camel case
- snake case
- pascal case
- kebab case

## Entorno:

- Concepto

## Control de flujo:

- Condicional:
- Estructura del if

## Como resuelvo los ejercicios de programación?

- Guía paso a paso para el éxito!



# Lenguaje JavaScript

# Variables

# JavaScript

## Variables

En JavaScript, una variable es un **contenedor que permite almacenar y hacer referencia a un valor específico que puede ser utilizado y manipulado dentro de un programa.**

Las variables actúan como nombres simbólicos asociados a espacios de memoria, lo que facilita el uso de datos de manera dinámica. Estos valores pueden ser de diferentes tipos, como números, cadenas de texto, booleanos, objetos, y más.



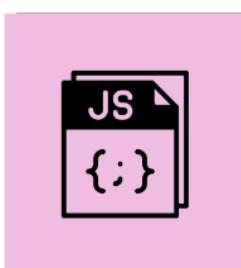
# JavaScript

## Tipos de Variables

En JavaScript existen estos tipos de variables:

- **let**
- **const**

Para declarar una variable escribimos el tipo y el nombre que le queremos dar a la variable:



```
let contador;  
const url;
```

Veamos cada parte con mas detalle...

# JavaScript

## Variables

### Declaracion de una variable

```
let nombreSignificativo;
```

#### Let:

La palabra reservada let le indica a JavaScript que vamos a declarar una variable de tipo let

#### Nombre:

Solo puede estar formado por letras, numeros y los simbolos \$(pesos) y (\_)guion bajo  
No pueden empezar con un numero  
No deben contener ñ ni caracteres con acentos



Es una buena practica que los nombres de las variables usen el formato camelCase, como variableEjemplo en vez de variableejemplo o variable\_ejemplo

# JavaScript

## Variables

### Declaracion de una variable

```
let miVariable;
```

No es lo mismo que:

```
let MiVariable;
```



JavaScript es un lenguaje que hace diferencia entre **MAYUSCULAS** y **minusculas**. Por eso, es bueno seguir un estandar a la hora de escribir nombres.

# JavaScript

## Variables

### Asignacion de un valor

```
let miApodo = 'LaMejor';
```

Nombre

El nombre que nos va a servir para identificar nuestra variable cuando necesitemos usarla.

Asignación

Le indica a JavaScript que queremos guardar el valor de la derecha en la variable de la izquierda.

Valor

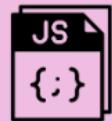
Lo que vamos a guardar en nuestra variable. En este caso, un texto.

# JavaScript

## Variables

### Asignacion de un valor

La primera vez que declaramos una variable es necesaria la palabra reservada `let`



```
let miApodo = 'LaMejor';
```

Una vez que la variable ya fue declarada, le asignamos valores **sin** `let`



```
miApodo = 'Super Programadora';
```

# JavaScript Variables

## Declaracion con const

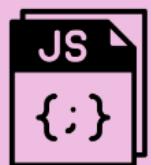
Las variables const se declaran con la palabra reservada **const**.



```
const email = "mi.email@hotmail.com";
```

Las variables declaradas con const funcionan igual que las variables let, estarán disponibles solo en el bloque de código en el que se hayan declarado.

Al contrario de let, una vez que les asignemos un valor, no podremos cambiarlo.



```
email = "mi.otro.email@hotmail.com";
// Error de asignación, no se puede cambiar
// el valor de un const
```

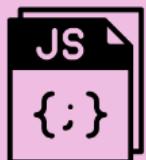
# JavaScript

## Variables

### Declaración con let o const

Como dijimos antes, tanto let como const son accesibles dentro del bloque donde son declaradas.

Por esta razón solo podemos declararlas una vez. Si volvemos a declararlas, JavaScript nos devolverá un error.



```
let contador = 0;  
let contador = 1;  
// Error de re-declaración de la variable  
  
const email = "mi.email@hotmail.com";  
const email = "mi.nuevo.email@hotmail.com";  
// Error de re-declaración de la variable
```

# JavaScript

## Variables

### Declaración con var

Existe otra forma de declarar variables que ya no se utiliza y no lo veremos en la carrear, pero que puede ser que la encuentres en códigos que tengan algunos años.

Estas variables se declaran de una manera similar, con la diferencia que utilizamos la palabra reservada `var`.



```
var contador = 1;
```

La principal diferencia entre `var` y `let` es que `var` será accesible de manera global por todo nuestro código y no estará limitado el acceso a solo el bloque de código donde fue declarado como es el caso de `let`. Por convención y buenas prácticas es recomendado el uso de `let`. Los bloques de código son normalmente determinados por las llaves {}.

# JavaScript Variables

## Declaración con var

Veamos un ejemplo:

**var**

```
if (true) {  
    var nombre = "Juan";  
}  
  
console.log(nombre);  
// Ok, muestra "Juan"
```

**let**

```
if (true) {  
    let nombre = "Juan";  
}  
  
console.log(nombre);  
// Ok, muestra "Juan"
```

Cuando usamos var, JavaScript ignora los bloques de código y convierte nuestra variable en global.

Eso quiere decir que si hay otra variable nombre en nuestro código, seguramente estemos pisando su valor.

Cuando usamos let, JavaScript respeta los bloques de código. Eso quiere decir que nombre no podrá ser accedida fuera del if. También quiere decir que podemos tener variables con el mismo nombre en diferentes bloques de nuestro código.

# Tipos de Datos

# JavaScript

## Variables



En JavaScript, los tipos de datos representan las **diferentes clases de valores que pueden almacenarse y manipularse en las variables**.

Cada valor en JavaScript pertenece a uno de estos tipos, lo que determina cómo puede ser utilizado en las operaciones y qué tipo de comportamientos se espera de él.

# JavaScript

## Tipos de Datos

### Numéricos (number)



```
let edad = 32; // número entero  
let precio = 150.65; // decimales  
let malaDivision = "35" / 2; // NaN - Not a Number, no es un  
número aunque es un valor de "tipo" number
```

IMPORTANT

Como JavaScript está escrito en inglés,  
usaremos un punto para separar los  
decimales.

### Cadenas de caracteres (string)



```
let nombre = 'Mamá Luchetti'; // comillas simples  
let ocupacion = "Master of the sopas"; // comillas dobles  
tienen el mismo resultado
```

### Lógicos o booleanos (boolean)



```
let laCharlaEstaReCopada = true;  
let hayAsadoAlFinal = false;
```

# JavaScript

## Tipos de Datos

### Undefined (valor sin definir)

Indica la ausencia de valor.

Las variables tienen un valor indefinido hasta que les asignamos uno.



```
let saludo; // undefined, no tiene valor  
saludo = "¡Hola!"; // Ahora si tiene un valor
```

### Null (valor nulo)

Lo asignamos nosotros para indicar un valor vacío o desconocido.



```
let temperatura = null; // No llegó un dato, algo falló
```

# JavaScript

## Tipos de Datos

### NaN (Not-A-Number)

La propiedad global NaN es un valor de tipo numérica que representa Not-A-Number. Indica que el valor pasado no es un número.



```
let nombre = "Esteban"  
console.log("Esteban"-1)
```

```
PS C:\Users\esteb\Desktop> node nan.js  
NaN
```

# JavaScript

## Tipos de Datos

Los **comentarios** son partes de nuestro código que no se ejecutan.

Siempre comienzan con dos barras inclinadas //

Los usamos para explicar lo que estamos haciendo y **dejar información útil** para nuestro equipo o para nuestro yo del futuro.



```
// Math.round() retorna el valor  
redondeado al entero más cercano.  
let redondeado = Math.round(20.49);
```

# Operadores

# JavaScript

## Operadores



En JavaScript, los **operadores** son símbolos o palabras clave que se utilizan para realizar operaciones sobre valores o variables, permitiendo manipular datos y crear expresiones lógicas o matemáticas.

Los operadores se dividen en varias categorías, cada una con un propósito específico.

# JavaScript

## Operadores

### De asignación

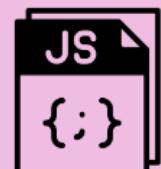
Asignan el valor de la derecha en la variable de la izquierda.



```
let edad = 35; // Asigna el número 35 a edad
```

### Aritméticos

Nos permiten hacer operaciones matemáticas, devuelven el resultado de la operación.



```
10 + 15 // Suma → 25  
10 - 15 // Resta → -5  
10 * 15 // Multiplicación → 150  
15 / 10 // División → 1.5
```

# JavaScript Operadores

## Aritméticos (continuación)

Nos permiten hacer operaciones matemáticas, devuelven el resultado de la operación.



```
15++ // Incremento, es igual a 15 + 1 → 16  
15-- // Decremento, es igual a 15 - 1 → 14
```



```
15 % 5 // Módulo, el resto de dividir 15 entre 5 → 0  
15 % 2 // Módulo, el resto de dividir 15 entre 2 → 1
```

$$15 \quad | \quad 5 \\ 0 \quad 3$$

$$15 \quad | \quad 2 \\ 1 \quad 7$$

El operador de módulo % nos devuelve el resto de una división. Los operadores aritméticos siempre devolverán el resultado numérico de la operación que se esté realizando.

# JavaScript

## Operadores

### De concatenación

Sirven para unir cadenas de texto. Devuelven otra cadena de texto.



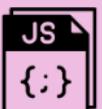
```
let nombre = 'Teodoro';
let apellido = 'García';
let nombreCompleto = 'Me llamo ' + nombre + ' ' + apellido;
```

**IMPORTANT**

#### Template literals

Existe otra forma de armar strings a partir de variables, y es con los template literals utilizando backtick en lugar de comillas y las variables entre llaves a continuación del símbolo \$. En este ejemplo lo escribiríamos así: `Me llamo \${nombre} \${apellido}`

Si mezclamos otros tipos de datos, estos se convierten a cadenas de texto.



```
let fila = 'M';
let asiento = 7;
let ubicacion = fila + asiento; // 'M7' como string
```

# JavaScript Operadores

## De comparación simple

Comparan dos valores, devuelven verdadero o falso.



```
10 == 15 // Igualdad → false  
10 != 15 // Desigualdad → true
```

## De comparación estricta

Comparan el valor y el tipo de dato también.



```
10 === "10" // Igualdad estricta → false  
10 !== 15 // Desigualdad estricta → true
```

En el primer caso el valor es 10 en ambos ejemplos, pero los tipos de datos son `number` y `string`. Como estamos comparando que ambos (valor y tipo de dato) sean iguales, el resultado es `false`.

# JavaScript

## Operadores

### De comparación (continuación)

Comparan dos valores, devuelven verdadero o falso. Los operadores de comparación siempre devolverán un booleano, es decir, true o false, como resultado.



```
15 > 15 // Mayor que → false  
15 >= 15 // Mayor o igual que → true  
10 < 15 // Menor que → true  
10 <= 15 // Menor o igual que → true
```

**IMPORTANT**

Siempre debemos escribir el símbolo mayor (>) o menor (<) antes que el igual (>= o <=). Si lo hacemos al revés (=> o =<), JavaScript lee primero el operador de asignación = y luego no sabe qué hacer con el mayor (>) o el menor (<).

# Operadores Lógicos

# JavaScript

## Operadores Lógicos



Los **operadores lógicos** y de **comparación** nos ayudan a **controlar** ciertos aspectos de nuestras aplicaciones, así como también a hacerlas más eficientes.

# Operadores Lógicos

## Lógicos

Permiten combinar valores booleanos, el resultado también devuelve un booleano.

Existen tres operadores y (and), o (or), negación (not).

- 1 AND (`&&`) → todos los valores deben evaluar como `true` para que el resultado sea `true`.

A screenshot of a JS code editor. On the left, there's a pink icon representing JS code. To its right is a black code area containing the following code:  
`(10 > 15) && (10 != 20) // false`



A screenshot of a JS code editor. On the left, there's a pink icon representing JS code. To its right is a black code area containing the following code:  
`(12 % 4 == 0) && (12 != 24) // true`



# Operadores Lógicos (continuación)

- 2 OR ( || ) → al menos un valor debe evaluar como true para que el resultado sea true.

 `(10 > 15) || (10 != 20) // true`



 `(12 % 5 == 0) && (12 != 12) // false`



- 3 NOT ( ! ) → niega la condición. Si era true, será false y viceversa.

 `!false // true, Porque se niega su valor como 'False'`  
`!(20 > 15) // false, Porque negamos una afirmacion verdadera`

# Clasificación completa de Operadores

# Tipos de Operadores

## Aritméticos:

- Utilizados para operaciones matemáticas

OPERADOR	DEFINICIÓN	EJEMPLO	
-	Restar números	Datos: A=6-3	Resultado:A=3
+	Sumar números	Datos: A=6+3	Resultado:A=9
*	Multiplicar números	Datos: A=6*3	Resultado:A=18
/	Dividir números	Datos: A=6/3	Resultado:A=2
^	Elevar a una potencia un numero	Datos: A=6^3	Resultado:A=216
--	Quitar una unidad a un numero	Datos: A=6; A=--	Resultado:A=5
++	Aumentar una unidad a un numero	Datos: A=6; A=++	Resultado:A=7
%	Obtener el modulo de una división, es decir, el residuo de dicha división	Datos: A=6 , B=4; A=A&B	Resultado:A=2

# Tipos de Operadores

## Relacionales:

- Utilizados para comparar dos o mas valores y determinar si el resultado es verdadero o falso

OPERADOR	DEFINICIÓN	EJEMPLO		
<	Menor que	Datos: A=6, B=3	Comparación: If(A<B)	Resu=F
>	Mayor que	Datos: A=6, B=3	Comparación: If(A>B)	Resu=V
<=	Menor o igual que	Datos: A=6, B=3	Comparación: If(A<=B)	Resu=F
>=	Mayor o igual que	Datos: A=6, B=6	Comparación: If(A>=B)	Resu=V
<> ó !=	Diferente a	Datos: A=6, B=3	Comparación: If(A!=B)	Resu=V
=	Igual que	Datos: A=6, B=3	Comparación: If(A=B)	Resu=F

# Tipos de Operadores

## Lógicos:

- Arrojan un resultado Verdadero o Falso al comparar uno o mas valores numéricos que pueden estar a su vez vinculados con operadores racionales

OPERADOR	DEFINICIÓN	EJEMPLO
&& AND	Al comparar valores o expresiones, si ambos son verdaderos obtiene un “true”	Datos: A=6; B=6 Comparación: If(A>3 && B>4) Si se cumple “Cierto”, si no “Falso”
OR	al comparar valores o expresiones, si una es verdadera obtiene un “true”, si ambas son falsas se obtiene un “false”	Datos: A=6; B=6 Comparación: If(A>3    B>22) Si se cumple “Cierto”, si no “Falso”
!= NOT	Al comparar valores o expresiones, niega la respuesta o resultado obtenido.	Datos: A=6; B=6 Comparación: If(A>3 != B>22) Si se cumple “Cierto”, si no “Falso”

# Tipos de Operadores

## Asignación:

- Se utilizan para asignar valores a las variables.

OPERADOR	DEFINICIÓN	EJEMPLO
<code>+=</code>	la suma del valor de la izquierda más el de la derecha	Datos: A=6; B=2, Expresión: A+=B Resultado: A=6+2
<code>-=</code>	la resta del valor de la izquierda menos el de la derecha	Datos: A=6; B=2, Expresión: A-=B Resultado: A=6-2
<code>*=</code>	la multiplicación del valor de la izquierda por el de la derecha	Datos: A=6; B=2, Expresión: A*=B Resultado: A=6*2

# Tipos de Operadores

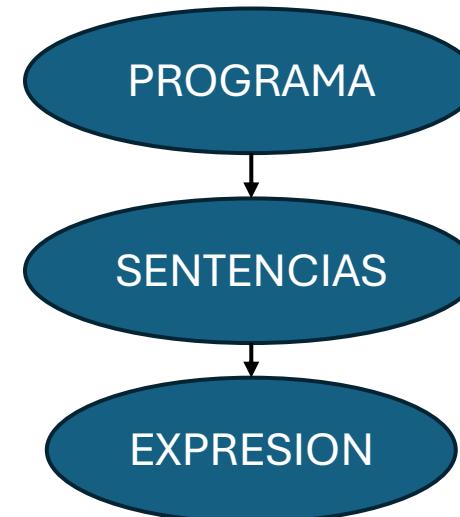


# Expresiones y Sentencias

# Expresiones y Sentencias

## Concepto

- Un fragmento de código que produce un valor, se denomina **expresión**.
- Cada valor que está escrito literalmente, como ser: números, cadenas de texto, etc, es una expresión.
- Podemos entender a las expresiones como una unidad mínima de código.
- Siguiendo esta analogía, las **sentencias** representan las oraciones enteras.
- Un **programa** está compuesto por una lista de sentencias.



```
> !true  
false  
> !true;  
false  
> 9;  
9  
> console.log(10);  
10
```

# Veamos mas Ejemplos en VSC!



# Estándares de nomenclatura

# Estándares de Nomenclatura

- Una actividad muy recurrente y necesaria en la programación es la **interpretación de código**, sea este propio o ajeno.
- Por este motivo es muy importante que el mismo sea claro y ordenado.
- Una de las principales prácticas para lograr este objetivo es la utilización de **estándares de nomenclatura** a la hora de nombrar variables, funciones, objetos, etc.
- Vamos a ver algunos de los más usados:

<b>camelCase</b>	→ precioProductoUno;
<b>SNAKE_CASE</b>	→ precio_producto_uno;
<b>PascalCase</b>	→ PrecioProductoUno;
<b>snake_case</b>	→ precio_producto_uno;
<b>kebab-case</b>	→ precio-producto-uno

Variables, propiedades, métodos.  
Constantes.  
Clases.  
Archivos.  
URLs, archivos.

# Entorno

# ¿Qué es “El Entorno” en JavaScript?

- El concepto de "entorno" en programación se refiere a la colección de variables, funciones y sus valores que están disponibles en un momento dado durante la ejecución de un programa.
- En JavaScript, cuando hablamos del entorno, estamos hablando del conjunto de enlaces (bindings) que existen y que se pueden utilizar.
- Cuando un programa arranca, el entorno no está vacío, sino que contiene todos los enlaces que forman parte del estándar del lenguaje, y también los que permiten interactuar con el sistema en donde se esté ejecutando.
- Por ejemplo, métodos que permiten “escuchar” el click de un mouse.

```
1 // Ejemplo de un evento de click
2 document.addEventListener('click', function() {
3     console.log("El mouse fue clickeado");
4 });
5
6 // Ejemplo de un evento de tecla presionada
7 document.addEventListener('keydown', function(event) {
8     console.log("Una tecla fue presionada: " + event.key);
9 });
```

Un **binding** es un enlace o conexión entre un nombre y un valor o una referencia en la memoria.

Por ejemplo, cuando declaramos una variable o una función, estamos creando un binding.

# Condicionales

# ¿Qué es un Condicional / Control de Flujo?

## Situaciones Reales:

- ¿Qué pasa si, estando en casa, se corta la luz y no podemos seguir viendo ese capítulo de la serie en Netflix? ¿Tomamos un libro y empezamos a leerlo o nos damos una pausa y salimos al balcón a respirar un poco de aire?
- Ese tipo de decisiones las podemos tomar también al momento de escribir código al programar.
- Para ello, se usan los mecanismos de **control de flujo**, también llamados **condicionales**.
- Estos nos permiten evaluar condiciones y realizar diferentes acciones según el resultado de dicha evaluación.



# Control de Flujo de Ejecución

## ¿Como Funciona?

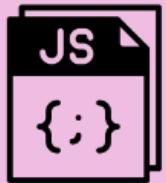
- El intérprete de JavaScript lee el código desde arriba hacia abajo, y desde la izquierda hacia la derecha, exactamente igual que en las lenguas occidentales.
- Sin embargo, no todos los programas se ejecutan de manera lineal, hay diversas formas de crear ramificaciones del flujo lineal de ejecución de un programa.
- Una de las herramientas que permite realizar esta ejecución condicional cae en la categoría de los condicionales, cuya naturaleza es la de ejecutar ciertas sentencias de código basando su decisión en el cumplimiento o no de una condición.
- Esta condición es una expresión lógica, cuyo resultado es un valor booleano, determinando así el camino a tomar.
- La herramienta que ofrece JavaScript para este propósito es la sentencia IF



# Componentes de un IF

## 1 Condicional simple

Versión más básica del if. Establece una condición y un bloque de código a ejecutar en caso de que sea verdadera.



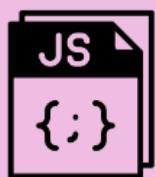
```
if (condición) {  
    // código a ejecutar si la condición es verdadera  
}
```

# Componentes de un IF

## 2 Condicional con bloque else

Igual al ejemplo anterior, pero agrega un bloque de código a ejecutar en caso de que la condición sea falsa.

Es importante tener en cuenta que el bloque else es opcional.



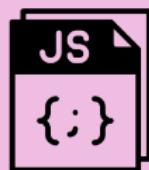
```
if (condición) {  
    // código a ejecutar si la condición es verdadera  
} else {  
    // código a ejecutar si la condición es falsa  
}
```

# Componentes de un IF

3

## Condicional con bloque else if

Igual que el ejemplo anterior, pero agrega un **if** adicional. Es decir, otra condición que puede evaluarse en caso de que la primera sea falsa. Podemos agregar todos los bloques **else if** que queramos, solo uno podrá ser verdadero. De lo contrario, entrará en acción el bloque **else**, si existe.



```
if (condición) {  
    // código a ejecutar si la condición es verdadera  
} else {  
    // código a ejecutar si la condición es falsa  
} else {  
    // código a ejecutar si todas las condiciones son falsas  
}
```

# Veamos mas Ejemplos en VSC!



# Funcionamiento de un IF

## {código}

```
let edad = 19;  
let acceso = '';
```

Declaramos la variable **edad** y le asignamos el número 19.

```
if (edad < 16) {  
    acceso = 'prohibido';  
} else if (edad >= 16 && edad <= 18) {  
    acceso = 'permitido sólo acompañado de un mayor';  
} else {  
    acceso = 'permitido';  
}
```

# Funcionamiento de un IF

## {código}

```
let edad = 19;  
let acceso = '';  
  
if (edad < 16) {  
    acceso = 'prohibido';  
} else if (edad >= 16 && edad <= 18) {  
    acceso = 'permitido sólo acompañado de un mayor';  
} else {  
    acceso = 'permitido';  
}
```

Declaramos la variable acceso y le asignamos un string vacío, con la intención de asignarle un nuevo valor según el resultado que arrojen los condicionales declarados a continuación.

# Funcionamiento de un IF

## {código}

```
let edad = 19;  
let acceso = '';  
  
if (edad < 16) {  
    acceso = 'prohibido';  
} else if (edad >= 16 && edad <= 18) {  
    acceso = 'permitido sólo acompañado  
de un mayor';  
} else {  
    acceso = 'permitido';  
}
```

Iniciamos el condicional.

Nuestra primera condición evalúa si edad es menor a 16.

En caso de ser verdadera, le asignamos el string 'prohibido' a la variable acceso.

En este caso, la condición es falsa, por lo tanto, JavaScript pasa a evaluar la siguiente condición.

# Funcionamiento de un IF

## {código}

```
let edad = 19;  
let acceso = '';  
  
if (edad < 16) {  
    acceso = 'prohibido';  
} else if (edad >= 16 && edad <= 18) {  
    acceso = 'permitido sólo acompañado  
de un mayor';  
} else {  
    acceso = 'permitido';  
}
```

Declaramos un bloque else if para contemplar una segunda condición:

Esta condición va a ser compuesta y va a requerir:

- que edad sea mayor o igual a 16
- que edad sea menor o igual a 18

La condición nuevamente es falsa, por lo tanto, JavaScript continúa leyendo el condicional.

# Funcionamiento de un IF

## {código}

```
let edad = 19;  
let acceso = '';  
  
if (edad < 16) {  
    acceso = 'prohibido';  
} else if (edad >= 16 && edad <= 18) {  
    acceso = 'permitido sólo acompañado  
de un mayor';  
} else {  
    acceso = 'permitido';  
}
```

Como ninguna de las condiciones anteriores era verdadera, se ejecuta el código dentro del else.

Por lo tanto, ahora la variable acceso es igual al string 'permitido'.

# IF – ELSE IF – ELSE

IMPORTANT

Tene en cuenta:

- Que un if no siempre necesita de un else o else if
- Que puede tener muchos else if si asi lo quisieras
- Que siempre que implementes un else, debera existir solo uno de ellos



# Los Ejercicios de Programación... Como los resuelvo?



# ¿Cómo resuelvo los ejercicios de Programación?

Es completamente normal sentirnos un poco abrumadas en nuestro primer contacto con los ejercicios de programación, es por eso que les tengo algunos consejitos que les servirán de ayuda:

1. Lee el enunciado varias veces para comprenderlo completamente.
2. Escribe a manera de comentario el ejercicio en VS
3. Identifica los datos de entrada y salida.
4. Busca restricciones y condiciones.
5. Divide el problema en partes más pequeñas.
6. Escribe un pseudocódigo o boceto de la solución.



# Sigamos los pasos para resolver este ejercicio:

## Ejercicio: Calculadora de Descuento

Escribe un programa que calcule el precio final de un producto después de aplicar un descuento. Pide al usuario que ingrese el precio original del producto y el porcentaje de descuento, y muestra el precio final.

PASO 1: Lee el enunciado varias veces para comprenderlo completamente.

- Asegúrate de entender qué se pide: calcular el precio final después de aplicar un descuento.
- El usuario debe proporcionar dos datos: el precio original y el porcentaje de descuento.
- El programa debe devolver el precio final después del descuento.

# Sigamos los pasos para resolver este ejercicio:

## Ejercicio: Calculadora de Descuento

Escribe un programa que calcule el precio final de un producto después de aplicar un descuento. Pide al usuario que ingrese el precio original del producto y el porcentaje de descuento, y muestra el precio final.

## PASO 2: Escribe a manera de comentario el ejercicio en VSC

```
// Ejercicio: Calculadora de Descuento
// Escribe un programa que calcule el precio final de un producto después de aplicar un descuento.
// Pide al usuario que ingrese el precio original del producto y el porcentaje de descuento,
// y muestra el precio final.
```

# Sigamos los pasos para resolver este ejercicio:

## Ejercicio: Calculadora de Descuento

Escribe un programa que calcule el precio final de un producto después de aplicar un descuento. Pide al usuario que ingrese el precio original del producto y el porcentaje de descuento, y muestra el precio final.

## PASO 3: Identifica los datos de entrada y salida

### **Entrada:**

- Precio original del producto (número decimal).
- Porcentaje de descuento (número decimal).

### **Salida:**

- Precio final del producto después de aplicar el descuento (número decimal).

# Sigamos los pasos para resolver este ejercicio:

## Ejercicio: Calculadora de Descuento

Escribe un programa que calcule el precio final de un producto después de aplicar un descuento. Pide al usuario que ingrese el precio original del producto y el porcentaje de descuento, y muestra el precio final.

## PASO 4: Busca restricciones y condiciones

- El precio original debe ser un **número positivo**.
- El porcentaje de descuento debe estar entre **0 y 100**.

# Sigamos los pasos para resolver este ejercicio:

## Ejercicio: Calculadora de Descuento

Escribe un programa que calcule el precio final de un producto después de aplicar un descuento. Pide al usuario que ingrese el precio original del producto y el porcentaje de descuento, y muestra el precio final.

## PASO 5: Divide el problema en partes más pequeñas

1. Pedir al usuario que ingrese el precio original del producto.
2. Pedir al usuario que ingrese el porcentaje de descuento.
3. Calcular el monto del descuento.
4. Calcular el precio final restando el descuento del precio original.
5. Mostrar el precio final al usuario.

# Sigamos los pasos para resolver este ejercicio:

## Ejercicio: Calculadora de Descuento

Escribe un programa que calcule el precio final de un producto después de aplicar un descuento. Pide al usuario que ingrese el precio original del producto y el porcentaje de descuento, y muestra el precio final.

## PASO 6: Escribe un pseudocódigo o boceto de la solución

1. Pedir al usuario el **precio original del producto**.
2. Pedir al usuario el **porcentaje de descuento**.
3. Calcular el descuento usando la fórmula:

$$\text{descuento} = (\text{precioOriginal} * \text{porcentajeDescuento}) / 100$$

4. Calcular el precio final usando la fórmula:  
$$\text{precioFinal} = \text{precioOriginal} - \text{descuento}$$
5. Mostrar el precio final al usuario.

# Ejercicio Resuelto:

## Prompt-sync

Es una **biblioteca de Node.js** que permite realizar entradas síncronas desde la consola. Esto es especialmente útil cuando necesitas que tu programa interactúe con el usuario pidiéndole datos de entrada durante la ejecución.

## ParseFloat

Es una función incorporada en **JavaScript** que se utiliza para convertir una cadena de texto (string) en un número de punto flotante (decimal).

Esto es útil cuando trabajas con datos ingresados por el usuario, que normalmente son cadenas de texto, pero necesitas realizar operaciones numéricas con esos datos.

```
1 // Ejercicio: Calculadora de Descuento
2 // Escribe un programa que calcule el precio final de un producto después de aplicar un descuento.
3 // Pide al usuario que ingrese el precio original del producto y el porcentaje de descuento,
4 // y muestra el precio final.
5
6 const prompt = require('prompt-sync')();
7
8 // 1. Pedir al usuario el precio original del producto.
9 let precioOriginal = parseFloat(prompt('Ingresa el precio original del producto: '));
10
11 // 2. Pedir al usuario el porcentaje de descuento.
12 let porcentajeDescuento = parseFloat(prompt('Ingresa el porcentaje de descuento: '));
13
14 // 3. Calcular el descuento
15 let descuento = (precioOriginal * porcentajeDescuento) / 100;
16
17 // 4. Calcular el precio final
18 let precioFinal = precioOriginal - descuento;
19
20 // 5. Mostrar el precio final al usuario
21 console.log('El precio final después del descuento es:', precioFinal);
```

- \$ node EjercicioPower.js  
Ingresa el precio original del producto: 400  
Ingresa el porcentaje de descuento: 10  
El precio final después del descuento es: 360

# Ejercicio Resuelto:

## Por que ponemos ParseFloat después de la variable?

Porque las entradas del usuario obtenidas con prompt son siempre cadenas de texto, pero queremos realizar operaciones numéricas con esos valores.

## Por que ponemos Prompt después de la variable?

Porque estamos asignando el valor que el usuario ingresa a esa variable. Esto es esencial para capturar la entrada del usuario y utilizarla posteriormente en el programa.

```
1 // Ejercicio: Calculadora de Descuento
2 // Escribe un programa que calcule el precio final de un producto después de aplicar un descuento.
3 // Pide al usuario que ingrese el precio original del producto y el porcentaje de descuento,
4 // y muestra el precio final.
5
6 const prompt = require('prompt-sync')();
7
8 // 1. Pedir al usuario el precio original del producto.
9 let precioOriginal = parseFloat(prompt('Ingresa el precio original del producto: '));
10
11 // 2. Pedir al usuario el porcentaje de descuento.
12 let porcentajeDescuento = parseFloat(prompt('Ingresa el porcentaje de descuento: '));
13
14 // 3. Calcular el descuento
15 let descuento = (precioOriginal * porcentajeDescuento) / 100;
16
17 // 4. Calcular el precio final
18 let precioFinal = precioOriginal - descuento;
19
20 // 5. Mostrar el precio final al usuario
21 console.log('El precio final después del descuento es:', precioFinal);
```

- \$ node EjercicioPower.js  
Ingresa el precio original del producto: 400  
Ingresa el porcentaje de descuento: 10  
El precio final después del descuento es: 360



Momento de  
poner a prueba  
lo aprendido!