Clase 9: Estructura Básica de un programa y Variables

Índice

Estructura Básica de un programa

Estructura Básica de un programa

Expresiones y declaraciones... Son lo mismo?

Expresión:

- Es un fragmento de código que produce un valor
- Las expresiones pueden ser tan simples como un número o una cadena de texto, o pueden ser más complejas y anidadas.
- Una expresión entre paréntesis también es una expresión, como lo es un operador binario aplicado a dos expresiones o un operador unario aplicado a una sola.
- Las expresiones son similares a las sub-oraciones en el lenguaje humano, permitiendo construir cálculos complejos de manera anidada.

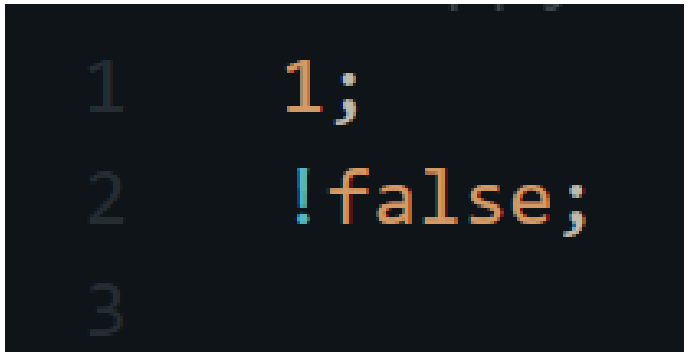
```
// Tipos de Expresiones:

// Literales
"Hola" // String
' ' // String vacio
false // Dato booleano
42; // Numero

// Variable
let a = 20;
let b = 10;

// Operacion Aritmetica
let suma = a + b // Esto si lo interpreta JS
let resta = a - b
let multiplicacion = a * b
let division = a / b // / = división % = modulo

// Operaciones de Cadena de texto
let hello = 'Hola'
let word = 'Mundo'
let concatenacion = hello + ' ' + word

// Expresiones Booleanas
let isEqual = (a === b) // Estrictamente igual
let Mayor = (a > b) // Si a es mayor que b

```

Estructura Básica de un programa

Expresiones y declaraciones... Son lo mismo?

Declaración:

- Una declaración en JavaScript corresponde a una "oración completa", es decir, es una instrucción independiente que realiza una acción o define algo en el programa.
- Un programa es esencialmente una lista de declaraciones.
- Las declaraciones pueden tener efectos secundarios, como modificar el estado del programa o interactuar con el entorno (por ejemplo, mostrando algo en la pantalla).
- El tipo más simple de declaración es una expresión con un punto y coma después ella (Aunque este código es un programa válido, no es útil ya que no produce efectos visibles o cambios en el estado.



Estructura Básica de un programa

Una Declaración es lo mismo que una Sentencia?

En el contexto de JavaScript, el término "sentencia" y "declaración" se usan de manera intercambiable, ya que ambos se refieren a instrucciones completas que realizan acciones en el programa.

Sin embargo, una sentencia puede ser más general y abarcar varios tipos de instrucciones, incluyendo declaraciones y otras estructuras de control de flujo.

Por lo tanto recuerden que:

- **Expresión** → Fragmento de código que produce un valor y puede ser parte de una declaración o sentencia.
- **Declaración** → Una instrucción completa (oración completa) que puede afectar el estado del programa o el entorno.
- **Sentencia** → Instrucción completa que realiza una acción (es similar a declaración).

Vinculaciones

Estructura Básica de un programa

Vinculaciones = Variable

- Las vinculaciones, también conocidas como variables en programación, son fundamentales para mantener un estado interno en un programa y recordar información.
- Las vinculaciones en JavaScript son nombres que se pueden usar para almacenar y acceder a valores.
- Estas vinculaciones pueden ser reasignadas y tienen diferentes comportamientos según la palabra clave utilizada (let, var, const).
- Ayudan a mantener un estado interno en el programa permitiendo que los valores sean recordados y manipulados a lo largo de la ejecución del código.

Declaración de Vinculación:

Una declaración de vinculación introduce un nombre (identificador) que puede ser usado para referirse a un valor en el programa.

Ejemplo: Aquí, let es una palabra clave que indica que se está definiendo una nueva vinculación llamada atrapado, que captura el resultado de la expresión 5 * 5.

```
let atrapado = 5 * 5;
```

Estructura Básica de un programa

Vinculaciones = Variable

Uso de Vinculaciones:

Después de definir una vinculación, su nombre puede ser usado como una expresión para acceder al valor que contiene.

Ejemplo:

```
let diez = 10;
console.log(diez * diez); // Imprime: 100
```

Reasignación de Vinculaciones:

Una vez definida, una vinculación puede ser reasignada usando el operador = para apuntar a un nuevo valor.

Ejemplo:

```
let humor = "ligero";
console.log(humor); // Imprime: ligero
humor = "oscuro";
console.log(humor); // Imprime: oscuro
```

JavaScript: Tipos de Vinculaciones (Variables y Constantes)

Los **Bloques** de Código normalmente son definidos por las llaves { }

Pueden cambiar a lo largo de la ejecución del programa.

Variables

Ambos son contenedores para almacenar datos

Constantes

No cambian una vez establecidos.

LET

- **Alcance**: Bloque (solo accesible dentro del bloque en el que se define)
- **Reasignable**: Sí (puede cambiar su valor después de la asignación inicial)
- **Redeclarable**: No (no se puede declarar nuevamente en el mismo alcance)
- **Uso común**: Cuando se espera que el valor cambie.

```
let nombre = "Juan";
nombre = "Pedro";  // Esto es válido
```

VAR

- **Alcance**: Global - Función (accesible dentro de la función donde se define)
- **Reasignable**: Sí (puede cambiar su valor después de la asignación inicial)
- **Redeclarable**: Sí (se puede declarar nuevamente en el mismo alcance)
- **Uso común**: Antes de ES6, era la única manera de declarar variables; todavía se usa en algunos casos por compatibilidad.

```
var edad = 25;
edad = 30;  // Esto es válido
var edad = 35;  // Esto también es válido
```

CONST

- **Alcance**: Bloque (solo accesible dentro del bloque en el que se define)
- **Reasignable**: No (el valor no puede cambiar después de la asignación inicial)
- **Redeclarable**: No (no se puede declarar nuevamente en el mismo alcance)
- **Debe ser inicializada**: Debe asignarse un valor en el momento de la declaración.
- **Uso común**: Para valores que no deberían cambiar, lo que ayuda a prevenir errores accidentales.

```
const longitud = 10;
const anchura = 5;

// Intentar reasignar valores a longitud y anchura generará un error
// longitud = 12;  // Error: Assignment to constant variable
// anchura = 7;    // Error: Assignment to constant variable
```

Nombres Vinculantes en JavaScript

Estructura Básica de un programa

Nombres Vinculantes en JavaScript

- Las variables son espacios de memoria en la computadora donde podemos almacenar distintos tipos de datos.
- Los nombres que se pueden usar para vinculaciones (variables) en JavaScript deben seguir ciertas reglas y convenciones:

Características de los Nombres Vinculantes:

Los nombres de las vinculaciones pueden ser cualquier palabra, pero deben seguir ciertas reglas:

1. Pueden incluir letras (mayúsculas y minúsculas), dígitos y los caracteres $ y _.
2. No pueden comenzar con un dígito.
3. No pueden incluir otros signos de puntuación o caracteres especiales (excepto $ y _).
4. Palabras Clave y Palabras Reservadas: Algunas palabras tienen un significado especial en JavaScript y no pueden ser usadas como nombres de vinculaciones. Estas son palabras clave.
   - Ejemplos de palabras clave incluyen let, var, if, function, entre otras.
   - También hay palabras reservadas para uso futuro en JavaScript que no se pueden utilizar como nombres de vinculaciones.

Estructura Básica de un programa

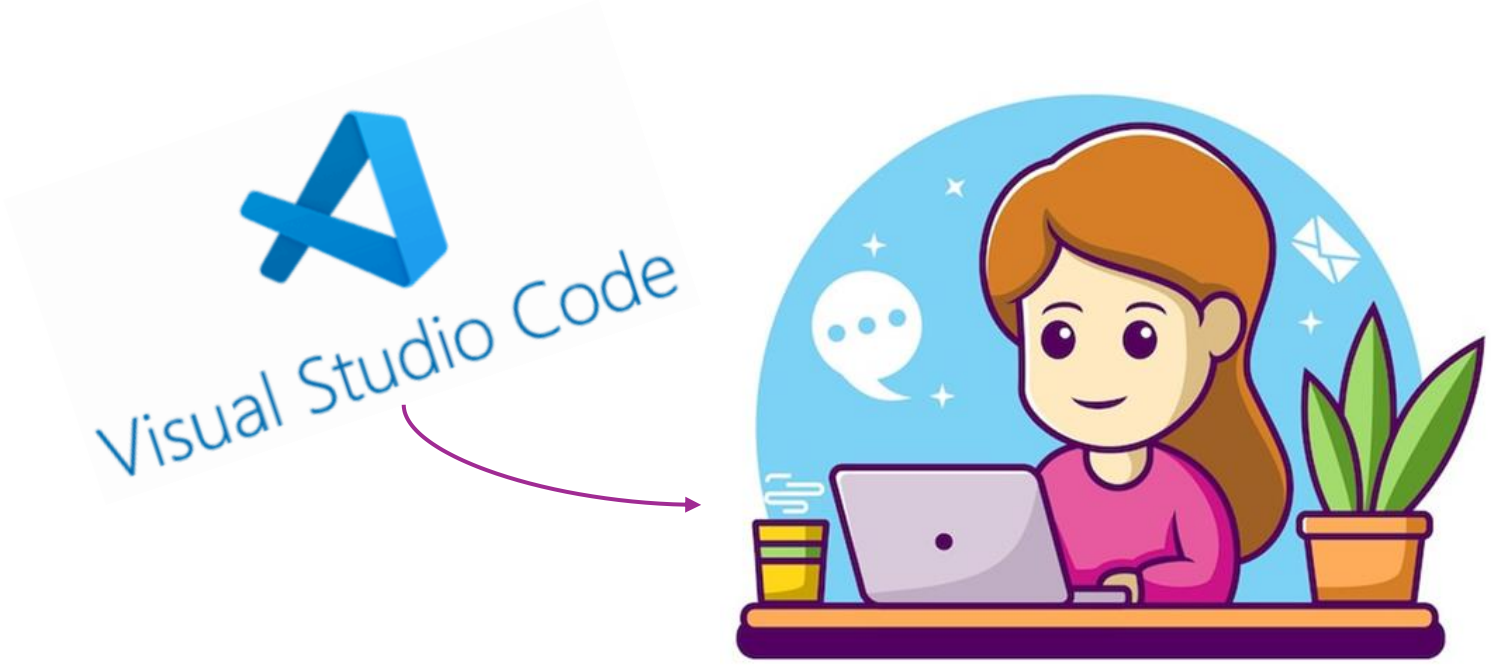
Nombres Vinculantes en JavaScript

Lista de Palabras Clave Comunes:
La lista completa de palabras clave y palabras reservadas es extensa e incluye términos como:

- **Break**
- **Case**
- **Class**
- **Const**
- **Continue**
- **Debugger**
- **Default**
- **Delete**
- **Else**
- **Enum**
- **Export**
- **Extends**
- **False**
- **Finally**
- **For**
- **Function**
- **If**
- **Implements**
- **Import**
- entre otros...

Recomendaciones:
No es necesario memorizar todas las palabras clave y reservadas.
Si al intentar definir una vinculación te encuentras con un error de sintaxis inesperado, verifica si estás intentando usar una palabra reservada o clave.

Veamos mas Ejemplos de nombres de vinculaciones en VSC!



Coerción de tipos en JavaScript

Coerción de tipos en JavaScript

Concepto

- Es un concepto fundamental que se refiere a la conversión automática de un tipo de dato a otro durante la evaluación de expresiones.
- Esto puede ocurrir de manera implícita (automáticamente por JavaScript) o explícita (mediante operadores o funciones).
- La coerción de tipos en JavaScript es poderosa pero puede conducir a resultados inesperados si no se comprende completamente.
- Saber cómo y cuándo ocurre la coerción, así como controlarla explícitamente cuando sea necesario, es fundamental para escribir código JavaScript robusto y libre de errores relacionados con tipos de datos.

Coerción de tipos en JavaScript

- Coerción Implícita y Explícita -

Coerción Implícita

Ocurre cuando JavaScript automáticamente convierte un tipo de dato a otro sin la intervención directa del desarrollador.
Esto suele ocurrir en contextos donde los tipos de datos esperados son diferentes. Algunos ejemplos comunes son:

- Concatenación de Cadenas y Conversión a String:

```
let num = 10;
let str = "El número es: " + num; // num se convierte implícitamente a string
console.log(str); // Resultado: "El número es: 10"
```

- Operaciones Aritméticas y Conversión a Number:

```
let result = "10" - 5; // "10" se convierte implícitamente a number
console.log(result); // Resultado: 5
```

Coerción de tipos en JavaScript

- Coerción Implícita y Explícita -

Coerción Explícita

Ocurre cuando el desarrollador fuerza la conversión de un tipo de dato a otro utilizando funciones o operadores específicos.
Esto permite controlar cómo se manejan los tipos de datos en ciertas situaciones.
Algunos ejemplos son:

- Convertir a String:

```
let num = 123;
let str = String(num); // Convertir num a string explícitamente
console.log(typeof str); // Resultado: "string"
```

- Convertir a Number:

```
let str = "456";
let num = Number(str); // Convertir str a number explícitamente
console.log(typeof num); // Resultado: "number"
```

Coerción de tipos en JavaScript

Comportamiento de Coerción en Operaciones

1. Operaciones de Comparación:

Las operaciones de comparación en JavaScript pueden conducir a resultados inesperados debido a la coerción automática de tipos.

Por ejemplo:

- Comparación con == (igualdad débil):

```
console.log(1 == '1'); // true, los tipos son convertidos automáticamente
```

- Comparación con === (igualdad estricta):

```
console.log(1 === '1'); // false, los tipos no son convertidos automáticamente
```

Coerción de tipos en JavaScript

Comportamiento de Coerción en Operaciones

2. Operaciones Aritméticas y Lógicas:

Las operaciones aritméticas y lógicas también pueden provocar coerción de tipos, especialmente cuando se combinan tipos diferentes:

- Suma de Número y String:

```
let result = 10 + "20"; // "10" se convierte a number y luego se concatena con "20"
console.log(result); // Resultado: "1020"
```

- Operaciones Lógicas con Tipos Diferentes:

```
console.log(true + 1); // 2, true se convierte a 1
console.log(false + ""); // "false", false se convierte a string vacío
```

Atajos con operadores lógicos

Atajos con operadores lógicos

- Los atajos con operadores lógicos (también conocidos como "short-circuit evaluation" en inglés) son una característica importante en muchos lenguajes de programación, incluido JavaScript.

- Esta característica permite que las expresiones lógicas se evalúen de manera más eficiente y permite escribir código más conciso y expresivo.

Concepto Básico:

- En JavaScript, los operadores lógicos **&& (AND)** y **|| (OR)** no solo evalúan las expresiones para determinar un valor booleano, sino que también pueden devolver uno de los operandos.

- Esto se debe a que estos operadores utilizan la evaluación de cortocircuito.

Atajos con operadores lógicos

Evaluación de Cortocircuito

Operador AND (&&):

- Si el primer operando es falso (false), el operador && no evalúa el segundo operando porque el resultado de toda la expresión será falso sin importar el valor del segundo operando. El valor de la expresión será el primer operando.

- Si el primer operando es verdadero (true), el operador && evalúa el segundo operando y el valor de la expresión será el segundo operando.

```
let a = false;
let b = true;
let result = a && b; // result será false porque a es false

a = true;
b = "Hello";
result = a && b; // result será "Hello" porque a es true y b se evalúa como true en contexto booleano

console.log(result); // Output: "Hello"
```

Atajos con operadores lógicos

Evaluación de Cortocircuito

Operador OR (||):

- Si el primer operando es verdadero (true), el operador || no evalúa el segundo operando porque el resultado de toda la expresión será verdadero sin importar el valor del segundo operando. El valor de la expresión será el primer operando.

- Si el primer operando es falso (false), el operador || evalúa el segundo operando y el valor de la expresión será el segundo operando.

```
// Si el primer operando es verdadero, el operador || no evalúa el segundo operando.
let a = true;
let b = "Hello";
let result = a || b; // result será true porque a es true

console.log(result); // Output: true

// Si el primer operando es falso, el operador || evalúa el segundo operando.
a = false;
b = "Hello";
result = a || b; // result será "Hello" porque a es false y b se evalúa como true en contexto booleano

console.log(result); // Output: "Hello"
```

Momento de poner a prueba lo aprendido!