d)

JavaScript es conocido por su flexibilidad y dinamismo, pero esta flexibilidad a menudo lleva a comportamientos peculiares cuando se trata de conversiones automáticas entre tipos de datos (coerción). Algunas de estas peculiaridades pueden causar errores inesperados si no se entienden bien. A continuación, detallo algunos aspectos clave relacionados con la conversión de tipos en JavaScript:

1. Coerción implícita:

- JavaScript realiza conversiones automáticas entre tipos de datos cuando se combinan o comparan tipos diferentes. Esta coerción puede ser confusa ya que los resultados no siempre son intuitivos.

2. Conversión entre números y strings:

- **Suma de número y string**: Cuando sumamos un número con una cadena de texto, JavaScript convierte ambos operandos a una cadena y realiza la concatenación.

```
js
Copiar código
console.log(5 + "3"); // "53"
```

- **Resta entre número y string**: Curiosamente, cuando restamos, JavaScript trata de convertir el string a número.

```
js
Copiar código
console.log(5 - "3"); // 2
```

3. Comparaciones sueltas (==) versus comparaciones estrictas (===):

- JavaScript permite dos tipos de comparaciones: sueltas (==) y estrictas (===). Las comparaciones sueltas realizan coerción implícita antes de comparar los valores.

```
js
Copiar código
console.log(5 == "5");  // true  (coerción implícita de la cadena "5" al número 5)
console.log(5 === "5"); // false (sin coerción, comparando un número con un string)
```

- Las comparaciones estrictas no convierten tipos, por lo que son más seguras y predecibles. Generalmente, es recomendable usar === para evitar errores inesperados.

4. Conversión de tipos en valores booleanos:

- JavaScript tiene reglas específicas para convertir tipos a booleanos, conocidas como **"truthy" y "falsy"**.
- **Valores "falsy"**: false, 0, "" (cadena vacía), null, undefined, y NaN son tratados como false cuando se convierten a booleanos.

```
js
Copiar código
console.log(Boolean(0));        // false
console.log(Boolean(""));       // false
console.log(Boolean(undefined));// false
```

- Cualquier otro valor es considerado **"truthy"**, es decir, se convierte a true:

```
js
Copiar código
console.log(Boolean(42));       // true
console.log(Boolean("hello"));  // true
```

5. Conversión de objetos a tipos primitivos:

- Cuando un objeto necesita ser convertido a un tipo primitivo (por ejemplo, durante una concatenación o comparación), JavaScript intenta primero convertirlo a un string o número.
- Los objetos se convierten a strings llamando su método toString() y a números usando valueOf(), si están definidos.

```
js
Copiar código
console.log([1, 2] + 3); // "1,23" (el array [1, 2] se convierte a la cadena "1,2")
console.log({} + 2);     // "[object Object]2"
```

6. NaN:

- El valor especial NaN (Not-a-Number) resulta cuando una operación matemática no tiene sentido. Por ejemplo, dividir un número por una cadena no numérica o intentar convertir una cadena no numérica a número genera NaN.

```
js
Copiar código
console.log(5 / "hello");  // NaN
console.log(Number("abc")); // NaN
```

- **Peculiaridad**: NaN **no es igual a sí mismo**, lo que significa que NaN === NaN es false.

```
js
Copiar código
console.log(NaN === NaN); // false
```

7. Peculiaridades del operador + con cadenas y números:

- El operador + se usa tanto para concatenar cadenas como para sumar números, lo que genera conversiones no siempre obvias. En contextos donde se combina un número con una cadena, JavaScript opta por la concatenación:

```
js
Copiar código
console.log("5" + 1); // "51" (el número 1 se convierte en string y se concatena)
```

8. Conversión de null y undefined:

- null se convierte a 0 en operaciones aritméticas, pero se convierte a false en contextos booleanos:

```
js
Copiar código
console.log(null + 5); // 5
console.log(Boolean(null)); // false
```

- undefined es más problemático, ya que suele resultar en NaN cuando se usa en operaciones aritméticas:

```
js
Copiar código
console.log(undefined + 5); // NaN
console.log(Boolean(undefined)); // false
```

9. Conversión explícita:

- Para evitar la coerción implícita, es buena práctica usar conversiones explícitas con funciones como Number(), String(), Boolean(), o usando operadores específicos como parseInt(), parseFloat() para convertir cadenas a números.

```
js
Copiar código
let str = "123";
let num = Number(str);  // Conversión explícita
```

Conclusión:

JavaScript tiene muchas peculiaridades en cuanto a la coerción de tipos, lo que puede llevar a resultados inesperados si no se comprenden bien. Por ello, es recomendable usar **comparaciones estrictas** (===), realizar conversiones explícitas de tipos cuando sea posible, y estar consciente de los valores que se consideran "falsy" y "truthy".

EJERCICIOS

1. Coerción implícita

Ejercicio: ¿Cuál es el resultado de la siguiente operación? Explica por qué sucede.

```
js
Copiar código
let resultado = "10" * 2;
console.log(resultado);
```
Respuesta → El resultado sería 20 puesto que JavaScript aplica coerción de tipos de manera automática, como tiene el operador * que necesita n.º trata de cambiar el string a numérico y como es posible porque tiene valores numéricos lo hace.

2. Conversión entre números y strings

Ejercicio: ¿Qué imprimen las siguientes líneas de código y por qué?

```
js
Copiar código
let a = 5 + "2";
let b = 5 - "2";
console.log(a);
console.log(b);
```

Respuesta → En el primer caso el operador + cuando hay String se usa para concatenar, por lo que en este caso mostraría la concatenación de 5 + 2 y sería 52.

En el segundo caso, como el operador – se utiliza con n.º, realiza la coerción de tipos y como es posible resta mostrando 3.

3. Comparaciones sueltas (==) versus comparaciones estrictas (===)

Ejercicio: Evalúa las siguientes comparaciones. ¿Qué resultado esperas en cada caso?

```
js
Copiar código
console.log(0 == "0");   // Comparación suelta
console.log(0 === "0");  // Comparación estricta
```
Respuesta → En el primer caso te daría true, porque con 2 iguales realiza una coerción de tipos antes de comparar y como “0” lo puede convertir a n.º ya sería 0 = 0 por lo que da true.
En el segundo caso con 3 iguales no realiza coerción de tipos, por lo que compararía en este caso un n.º con un String dando false.

4. Conversión de tipos en valores booleanos

Ejercicio: ¿Qué imprimen las siguientes líneas de código y por qué?

```
js
Copiar código
console.log(Boolean("false"));   // Cadena no vacía
console.log(Boolean(0));         // Número cero
```

```
console.log(Boolean(""));        // Cadena vacía
console.log(Boolean([]));        // Array vacío
```

Respuesta →

1. Te devuelve True porque es una cadena de texto que aunque contenga la palabra false, no está vacía.
2. Este valor numérico te da false al pasar a booleano.
3. Te da false porque es una cadena de texto vacía
4. Te da true porque aunque el array esté vacío es un objeto y siempre se inicializa a true.

---

5. Conversión de objetos a tipos primitivos

Ejercicio: ¿Qué imprimen las siguientes operaciones y por qué?

js
Copiar código

```
let arr = [1, 2];
let obj = { key: "value" };
console.log(arr + 3);
console.log(obj + 3);
```

Respuesta →

1. El resultado será 1,23 porque el array pasa a cadena porque el 3 es un tipo primitivo.
2. Cuando intentas sumar un objeto con un n.º se convierte el objeto a tipo primitivo con el toString() y el valueOf() y lo concatena con el otro tipo primitivo. Por lo que el resultado es [Object Object]3

---

6. NaN

Ejercicio: ¿Qué sucede en la siguiente operación? ¿Cuál es el valor de resultado?

js
Copiar código

```
let resultado = "hello" / 2;
console.log(resultado);
```

Respuesta → El resultado esperado es NaN porque no se puede dividir un String entre un valor numérico.

7. Peculiaridades del operador + con cadenas y números

Ejercicio: ¿Cuál es el valor de resultado1 y resultado2? Explica por qué.

js
Copiar código

```
let resultado1 = "5" + 1;
let resultado2 = 5 + 1 + "3";
console.log(resultado1);
console.log(resultado2);
```

Respuesta →

1. En el primer caso se concatena 5 + 1 y daría 51. Puesto que como usa el operador + y tiene una cadena pues no realiza coerción.

2. En el segundo caso realiza primero una suma y luego el resultado lo concatena con el String 3 por la misma razón que el anterior. Dando como resultado 63.

8. Conversión de null y undefined

Ejercicio: ¿Qué imprimen las siguientes líneas y por qué?

```
js
Copiar código
console.log(null + 5);
console.log(undefined + 5);
```

Respuesta →

1.En este caso daría como resultado 5 porque null es vacío, por lo que se puede poner como numérico y le suma el valor 5.

2. En este caso undefined significa que no se le ha asignado valor a la variable por lo que cuando se quiere pasar a numérico te dice que es NaN y al sumar eso con 5 el resultado es NaN.

9. Conversión explícita

Ejercicio: Completa el código siguiente para convertir el valor de str en un número de manera explícita y realizar una suma correcta.

```
js
Copiar código
let str = "42";
let num = 8;
let resultado = /* tu código aquí */;
console.log(resultado); // Debería imprimir 50
```

Respuesta →

let resultado = parseInt(str) + num; Esto dará 50 porque pasa el string str a valor numérico. En caso de no poder arrojará un NaN.