CICLO DAW DISTANCIA- CURSO 23/24

PRACTICA 2: INTRODUCCIÓN A JAVASCRIPT

OBJETIVO

- Iniciar al alumnado en el manejo de los tipos de datos JavaScript
- Iniciar al alumnado en las funciones
- Consolidar la sintaxis del lenguaje javascript, declaración de variables, sentencias condicionales, bucles, etc.

RECURSOS

TIPOS DE DATOS EN JAVASCRIPT

Javascript como lenguaje de programación, tiene interesantes características que lo hacen versatil y funcional, una de ellas es como gestiona sus tipos de datos. En Javascript los tipos no se definen, es decir hablamos que es un lenguaje de tipado débil, pero eso no significa que no tenga tipos.

En Javascript disponemos de muchos tipos de datos, sin embargo se suelen catalogar en dos grupos, **tipos de datos primitivos** (básicos) y **tipos de datos no primitivos** (complejos)

Tipos de datos primitivos

- **String** para gestión variables con valores alfanuméricos y cualquier otro carácter.
- **Number** para representar los números ya sean enteros o decimales.
- **BigInt**, usado para representar números enteros.
- **Boolean** para manipular los valores lógicos (true y false)
- **Object**, con el fin de poder gestionar y representar objetos, conocidos por sus propiedades y métodos
- **Symbol**, son un nuevo tipo de tipo de datos primitivo introducido en la versión ES6 del lenguaje. Se utilizan para representar valores únicos que se pueden utilizar como identificadores o claves en los objetos. También se utilizan para crear propiedades y métodos privados en las clases.
- **Null**, El valor nulo representa la ausencia intencional de cualquier valor de objeto. Existe como tipo de dato primitivo de JavaScript y se trata como falso para las operaciones booleanas.
- **undefined**, es un valor primitivo automáticamente asignado a las variables que solo han sido declarados o a los argumentos formales para los cuales no existe argumentos reales. Es decir si no asignamos ningún valor a una variable, está tendrá ese valor undefined.

Tipos de datos no primitivos

- **Object,** es usado para guardar una colección de datos definidos y entidades más complejas
- **Function**, es un tipo de dato especial usado para reconocer cuando trabajamos sobre un función, que es un fragmento de código que realiza una tarea específica y devuelve un valor.

En Javascript como comentamos al anteriormente, la gestión de tipos de datos es débil, no es necesario indicar el tipo de dato cuando declaramos una variable, por ello es importante saber como determinar el tipo de dato a una variable declarada.

Hay varias formas de saber que tipo de dato tiene una variable en Javascript

- **El operador typeof:** La función typeof nos devuelve el tipo de dato primitivo de la variable que le pasemos por parámetro.

```
const text = "Hola que tal estas!";
typeof text;      // Devuelve "String"
const number = 40;
typeof number;      // Devuelve "Number"

let notDefined;
typeof notDefined;      // Devuelve undefined
```

Un punto importante a recordar es que Javascript hace conversión automática de los tipos, a partir de un valor asignado, pero existen formas para asignar tipos de forma implicita, esto lo logramos usando clases como:

Number, con esta clase conseguimos transformar un valor numérico en un tipo de dato explicito Number, cuando el valor no es un número válido será retornado el valor especial NaN.

CONVERSIÓN Y COERCIÓN DE DATOS EN JAVASCRIPT

La conversión de tipos de JavaScript es una faceta crucial del lenguaje. En concreto, la conversión de tipos automática o implícita de JavaScript se conoce a menudo como **coerción** de tipos, un concepto igualmente importante que debes conocer.

La **coerción** es un concepto que se basa en la conversión de tipos. En JavaScript, cuando un operador o sentencia espera un tipo de datos concreto pero encuentra otro, convierte automáticamente, o coacciona, este tipo de datos en el esperado.

Esto puede parecer similar a la conversión de tipos, y en muchos aspectos lo es. Sin embargo, la diferencia crucial radica en el control y la intención. Mientras que la conversión de tipos puede ser explícita (realizada manualmente por el programador) o implícita (realizada automáticamente), la coacción siempre es implícita.

Es la forma que tiene el lenguaje JavaScript de simplificar las interacciones entre distintos tipos de datos y garantizar una experiencia de programación más fluida.

Ejemplo:

```
let númeroUno = 7;
let númeroDos = "42";
let resultado = númeroUno + númeroDos;
```

El resultado de la ejecución de estas intrucciones no es un error, si no que Javascript coacciona el número 7 para convertirlo en una cadena, lo que da como resultado una concatenación de cadenas que produce **"742".**

FUNCIÓN EN JAVASCRIPT

1. **Definición**:

   Una función en JavaScript es un bloque de código o un conjunto de instrucciones que realiza una tarea específica y que puede reutilizarse a voluntad. Por lo tanto, una función es uno de los bloques de construcción fundamentales de JavaScript.

2. Tipos de funciones

Existen diversas formas de crear una función en JavaScript:

- **Por declaración:** Es probablemente la más utilizada y la más fácil de recordar, sobre todo si ya conoces algún otro lenguaje de programación. Consiste en declarar la función con un nombre y sus parámetros de entrada entre paréntesis.

  Esta forma permite declarar una función que existirá a lo largo de todo el código. De hecho, podríamos ejecutar la función sumar() de haberla creado y funcionaría correctamente, ya que Javascript primero busca las declaraciones de funciones y luego procesa el resto del código.

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

  Ejecutar: sumar(5,6); Resultado: //11

- **Por expresión:** Este tipo ha tomado popularidad y consiste básicamente en guardar una función en una variable, para así ejecutar la variable como si fuera una función.

  Con este nuevo enfoque, estamos creando una función **en el interior de una variable**, lo que nos permitirá posteriormente ejecutar la variable (*como si fuera una función*).

  Observa que el nombre de la función (*en este ejemplo: saludar*) pasa a ser inútil, ya que si intentamos ejecutar `sumar()` nos dirá que no existe y si intentamos ejecutar plus() funciona correctamente.

  ```
  const plus = function sumar(a, b){ return a+b;};plus(5,6)
  ```

  ¿Qué ha pasado? Ahora el nombre de la función pasa a ser el nombre de la variable, mientras que el nombre de la función desaparece y se omite, dando paso a lo que se llaman las **funciones anónimas** (*o funciones lambda*).

  ```
  const plus = function (a, b){ return a+b;};plus(5,6)
  ```

3. Declaración de funciones

Una definición de función (también denominada declaración de función o expresión de función) consta de la palabra clave function, seguida de:

- El nombre de la función.
- Una lista de parámetros de la función, entre paréntesis y separados por comas.
- Las declaraciones de JavaScript que definen la función, encerradas entre llaves, { ... }.

```
function square(number) {
  return number * number;
}
```

La función square toma un parámetro, llamado number. La función consta de una declaración que dice devuelva el parámetro de la función (es decir, number) multiplicado por sí mismo. La instrucción return especifica el valor devuelto por la función

Los parámetros primitivos (como un number) se pasan a las funciones por valor; el valor se pasa a la función, pero si la función cambia el valor del parámetro, este cambio no se refleja globalmente ni

en la función que llama.

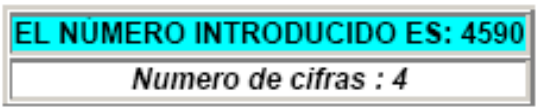
Si pasas un objeto (es decir, un valor no primitivo, como Array o un objeto definido por el usuario) como parámetro y la función cambia las propiedades del objeto, ese cambio es visible fuera de la función

SECUENCIA/DESARROLLO

1. ***Ejercicio 1:****Realiza un script al que se le introduce un número e indique si dicho número es primo. En ese caso mostrará una lista con todos los números primos menores que dicho número.*
   - *El usuario introduce el número utilizando el método prompt*
   - *Comprueba si el número es primo. Para ello debes crear una función* ***esPrimo()*** *que devuelve un valor true o false.*
   - *Por último comprobar y mostrar todos los números primos menor que el. Para ello utilizar el método document.write.*
2. ***Ejercicio 2*** *: Realiza un script que lea una secuencia de números introducidos por teclado y calcula y visualiza su suma y media. La secuencia terminará cuando el número introducido sea cero.*
3. ***Ejercicio 2:*** *Escribir un programa que lea un número entero positivo mayor que cero y calcule y visualice el número de cifras que tiene. Por ejemplo si escribimos el número 4590 el número de cifras ha de ser 4. Un ejemplo de la salida del ejercicio puede ser la siguiente:*

   Para introducir el número debemos utilizar el método prompt() de la clase Window.

   Debe calcularse el numero de cifras, teniendo en cuenta que dicho número no es un string, sino un entero, por lo que la solución no puede ser indicar la longitud de la cadena



Nota: La función floor(n) devuelve el entero obtenido de redondear 'n' al valor entero inmediatamente inferior. Para usarla se necesita anteponer el objeto Math (Math.floor(n)).