# Herramientas de Desarrollo Scripts (JS – TS)


JavaScript es una herramienta de programación, que te permite realizar actividades complejas en una página web, a medida que ha avanzado el tiempo se ha hecho su aporte en muchos desarrollos web, que hace más atractivas las presentaciones dentro de las páginas, con este lenguaje de desarrollo se aleja de solo mostrar información estática, también permite la visualización y actualización de contenido en el momento, interactuar con mapas, animaciones gráficas 2D/3D, etc.".

JS Forma parte del mundo de las aplicaciones móviles y las páginas web. Los desarrolladores **full-stack** y **front-end**, encargados del ecosistema de los navegadores web, están directamente relacionados con él. Ha hilvanado la unión entre las aplicaciones para iOS y Android en un solo lenguaje de programación, ¿**sabes qué significan las aplicaciones híbridas y nativas**?

- ✓ Las primeras son aquellas que se ejecutan con el navegador integrado en la App, lo que se conoce como **WebView**,
- ✓ Las segundas compilan el código nativo, además de poder ejecutar un desarrollo de Apps por medio de un **WebView** también.

En definitiva, JavaScript nos brinda la posibilidad de hacer cosas muy diversas, desde crear contenido nuevo y dinámico, hasta controlar los archivos multimedia, crear imágenes animadas y mucho más. Unas pocas líneas de código de JavaScript pueden sorprender desde el desarrollador hasta el usuario final. En definitiva, lo que aporta es un comportamiento dinámico y la posibilidad de almacenar valores útiles dentro de algunas variables, entre otras cosas.

Entre las ventajas del uso del **JS** están:

- Sencillez al entender y al aplicar
- Rapidez
- Multiplataforma
- Full-Stack

## Estructuras de control de flujo en JavaScript

Este tipo de programas son necesarias las estructuras de control de flujo o bien también llamadas estructuras condicionales, las cuales son instrucciones del tipo "**SI SE CUMPLE ESTA CONDICIÓN, HAZLO; SI NO SE CUMPLE, HAZ ESTO OTRO**". También existen instrucciones del tipo "**HAZ ESTO MIENTRAS SE CUMPLA ESTA CONDICIÓN**".

Si se utilizan estructuras de control de flujo, los programas dejan de ser una sucesión lineal de instrucciones para convertirse en programas inteligentes que pueden tomar decisiones en función del valor de las variables.

### Estructura Selectiva Si (if)

La estructura más utilizada en JavaScript y en la mayoría de lenguajes de programación es la estructura if. Se emplea para tomar decisiones en función de una condición. Su definición formal es:

```
if(condición) {
  ...
}
```

Si la condición se cumple (es decir, si su valor es true) se ejecutan todas las instrucciones que se encuentran dentro de {...}. Si la condición no se cumple (es decir, si su valor es false) no se ejecuta ninguna instrucción contenida en {...} y el programa continúa ejecutando el resto de instrucciones del script.

**Ejemplo:**

```
var verMsg = true;
if(verMsg) {
  alert("Hola Mundo");
}
```

En el ejemplo anterior, el mensaje sí que se muestra al usuario ya que la variable **verMsg** tiene un valor de true y por tanto, el programa entra dentro del bloque de instrucciones del if.

El ejemplo se podría reescribir también como:

```
var verMsg = true;
if(verMsg == true) {
  alert("Hola Mundo");
}
```

En este caso, la condición es una comparación entre el valor de la variable **verMsg** y el valor true. Como los dos valores coinciden, la igualdad se cumple y por tanto la condición es cierta, su valor es true y se ejecutan las instrucciones contenidas en ese bloque del if.

La comparación del ejemplo anterior suele ser el origen de muchos errores de programación, al confundir los operadores lógicos **==** y **=**. Las comparaciones siempre se realizan con el operador lógico **==,** ya que el operador lógico **=** solamente asigna valores:

```
var verMsg = true;

// Se comparan los dos valores
if(verMsg == false) {
  ...
}
// Error - Se asigna el valor "falso" a la variable
if(verMsg = false) {
  ...
}
```

La condición que controla el if() puede combinar los diferentes operadores lógicos y relacionales mostrados anteriormente:

```
var mostrado = false;

if(!mostrado) {
  alert("Es la primera vez que se muestra el mensaje");
}
```

Los operadores AND y OR permiten encadenar varias condiciones simples para construir condiciones complejas:

```
var mostrar = false;
var usrVerMsg = true;

if(!mostrar && usrVerMsg) {
  alert("Es la primera vez que se muestra el mensaje");
}
```

La condición anterior está formada por una operación AND sobre dos variables. A su vez, a la primera variable se le aplica el operador de negación antes de realizar la operación AND. De esta forma, como el valor de mostrar es false, el valor !mostrar sería true. Como la variable usrVerMsg vale true, el resultado de **!mostrar && usrVerMsg** sería igual a true && true,

por lo que el resultado final de la condición del if() sería true y por tanto, se ejecutan las instrucciones que se encuentran dentro del bloque del if().

**Ejercicio #1**
Completar las condiciones de los if del siguiente script para que los mensajes de los alert() se muestren siempre de forma correcta:

```
var numero1 = 5;
var numero2 = 8;

if(___) {
  alert("numero1 no es mayor que numero2");
}
if(___) {
  alert("numero2 es positivo");
}
if(___) {
  alert("numero1 es negativo o distinto de cero");
}
if(___) {
  alert("Incrementar en 1 unidad el valor de numero1 no lo hace mayor o igual que numero2");
}
```

**Estructura Si…Entonces < if...else>**
En ocasiones, las decisiones que se deben realizar no son del tipo "si se cumple la condición, hazlo; si no se cumple, no hagas nada". Normalmente las condiciones suelen ser del tipo "si se cumple esta condición, hazlo; si no se cumple, haz esto otro". Para este segundo tipo de decisiones, existe una variante de la estructura if llamada if...else. Su definición formal es la siguiente:

```
if(condicion) {
  ...
}
else {
  ...
}
```

Si la condición se cumple (es decir, si su valor es true) se ejecutan todas las instrucciones que se encuentran dentro del if(). Si la condición no se cumple (es decir, si su valor es false) se ejecutan todas las instrucciones contenidas en else { }. Ejemplo:

```
var edad = 18;

if(edad >= 18) {
  alert("Eres mayor de edad");
}
else {
  alert("Todavía eres menor de edad");
}
```

Si el valor de la variable edad es mayor o igual que el valor numérico 18, la condición del if() se cumple y por tanto, se ejecutan sus instrucciones y se muestra el mensaje "Eres mayor de edad". Sin embargo, cuando el valor de la variable edad no es igual o mayor que 18, la condición del if() no se cumple, por lo que automáticamente se ejecutan todas las instrucciones del

bloque else { }. En este caso, se mostraría el mensaje 'Todavía eres menor de edad". El siguiente ejemplo compara variables de tipo cadena de texto:

```
var nombre = "";

if(nombre == "") {
  alert("Aún no nos has dicho tu nombre");
}
else {
  alert("Hemos guardado tu nombre");
}
```

La condición del if() anterior se construye mediante el operador ==, que es el que se emplea para comparar dos valores (no confundir con el operador = que se utiliza para asignar valores). En el ejemplo anterior, si la cadena de texto almacenada en la variable nombre es vacía (es decir, es igual a "") se muestra el mensaje definido en el if(). En otro caso, se muestra el mensaje definido en el bloque else { }. La estructura if...else se puede encadenar para realizar varias comprobaciones seguidas:

```
if(edad < 12) {
  alert("Todavía eres muy pequeño");
}
else if(edad < 19) {
  alert("Eres un adolescente");
}
else if(edad < 35) {
  alert("Aun sigues siendo joven");
}
else {
  alert("Piensa en cuidarte un poco más");
}
```

## Ejercicio #2

El cálculo de la letra de un documento de Identidad (c.c) es un proceso matemático sencillo que se basa en obtener el resto de la división entera del número de documento de Identidad y el número 23. A partir del resto de la división, se obtiene la letra seleccionándola dentro de un array **(¿de que tipo?**)de letras.

El array de letras es:

var letras = ['T', 'R', 'W', 'A', 'G', 'M', 'Y', 'F', 'P', 'D', 'X', 'B', 'N', 'J', 'Z', 'S', 'Q', 'V', 'H', 'L', 'C', 'K', 'E', 'T'];

Por tanto si el resto de la división es 0, la letra del documento de Identidad es la T y si el resto es 3 la letra es la A. Con estos datos, elaborar un pequeño script que:

Almacene en una variable el número de DNI indicado por el usuario y en otra variable la letra del DNI que se ha indicado. (Pista: si se quiere pedir directamente al usuario que indique su número y su letra, se puede utilizar la función **prompt()**➔otra manera de alerta)
En primer lugar (y en una sola instrucción) se debe comprobar si el número es menor que 0 o mayor que 99999999. Si ese es el caso, se muestra un mensaje al usuario indicando que el número proporcionado no es válido y el programa no muestra más mensajes.

Si el número es válido, se calcula la letra que le corresponde según el método explicado anteriormente.
Una vez calculada la letra, se debe comparar con la letra indicada por el usuario. Si no coinciden, se muestra un mensaje al usuario diciéndole que la letra que ha indicado no es correcta. En otro caso, se muestra un mensaje indicando que el número y la letra del documento de Identidad son correctos.

**Estructura Iterativas Para (for)**

Las estructuras if e if...else no son muy eficientes cuando se desea ejecutar de forma repetitiva una instrucción. Por ejemplo, si se quiere mostrar un mensaje cinco veces, se podría pensar en utilizar el siguiente if:

```
var veces = 0;

if(veces < 4) {
  alert("Mensaje");
  veces++;
}
```

Se verificar si la variable **veces** es menor que 4. Si se cumple, se entra dentro del **if()**, se muestra el mensaje y se incrementa el valor de la variable veces. Así se debería seguir ejecutando hasta mostrar el mensaje las cinco veces deseadas. Sin embargo, el funcionamiento real del script anterior es muy diferente al deseado, ya que solamente se muestra una vez el mensaje por pantalla. La razón es que la ejecución de la estructura **if()** no se repite y la comprobación de la condición sólo se realiza una vez, independientemente de que dentro del **if()** se modifique el valor de la variable utilizada en la condición.

La estructura for permite realizar este tipo de repeticiones (también llamadas bucles,loops, iteraciones o ciclos) de una forma muy sencilla. No obstante, su definición formal no es tan sencilla como la de **if():**

```
for(inicializacion; condicion; actualizacion) {
  ...
}
```

La idea del funcionamiento de un bucle for es la siguiente: "mientras la condición indicada se siga cumpliendo, repite la ejecución de las instrucciones definidas dentro del **for**. Además, después de cada repetición, actualiza el valor de las variables que se utilizan en la condición".

La "inicialización" es la zona en la que se establece los valores iniciales de las variables que controlan la repetición.
La "condición" es el único elemento que decide si continua o se detiene la repetición.
La "actualización" es el nuevo valor que se asigna después de cada repetición a las variables que controlan la repetición.

```
var mensaje = "Hola, estoy dentro de un bucle";

for(var i = 0; i < 5; i++) {
  alert(mensaje);
}
```

La parte de la inicialización del bucle consiste en: var i = 0;

Por tanto, en primer lugar se crea la variable i y se le asigna el valor de 0. Esta zona de inicialización solamente se tiene en consideración justo antes de comenzar a ejecutar el bucle. Las siguientes repeticiones no tienen en cuenta esta parte de inicialización, la zona de condición del bucle es: **i < 5**. Los bucles se siguen ejecutando mientras se cumplan las condiciones y se dejan de ejecutar justo después de comprobar que la condición no se cumple. En este caso, mientras la variable i valga menos de 5 el bucle se ejecuta indefinidamente.

Como la variable i se ha inicializado a un valor de 0 y la condición para salir del bucle es que i sea menor que 5, si no se modifica el valor de i de alguna forma, el bucle se repetiría indefinidamente. Por ese motivo, es imprescindible indicar la zona de actualización, en la que se modifica el valor de las variables que controlan el bucle: **i++**

En este caso, el valor de la variable i se incrementa en una unidad después de cada repetición. La zona de actualización se ejecuta después de la ejecución de las instrucciones que incluye el **for**. Así, durante la ejecución de la quinta repetición el valor de i será 4. Después de la quinta ejecución, se actualiza el valor de i, que ahora valdrá 5. Como la condición es que i sea menor que 5, la condición ya no se cumple y las instrucciones del **for** no se ejecutan una sexta vez.

Normalmente, la variable que controla los bucles **for** se llama **i**, ya que recuerda a la palabra índice y su nombre tan corto ahorra mucho tiempo y espacio.

El ejemplo anterior que mostraba los días de la semana contenidos en un array se puede rehacer de forma más sencilla utilizando la estructura for:

```
var dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"];

for(var i=0; i<7; i++) {
  alert(dias[i]);
}
```

## Ejercicio #3

El factorial de un número entero n es una operación matemática que consiste en multiplicar todos los factores n x (n-1) x (n-2) x ... x 1. Así, el factorial de 5 (escrito como 5!) es igual a: 5! = 5 x 4 x 3 x 2 x 1 = 120

Utilizando la estructura for, crear un script que calcule el factorial de un número entero.

**Estructura Para Extendido (for...in ➔ conocido por algunos como foreach)**

Una estructura de control derivada de for es la estructura for...in. Su definición exacta implica el uso de objetos, que es un elemento de programación avanzada que no se va a estudiar. Por tanto, solamente se va a presentar la estructura for...in adaptada a su uso en arrays. Su definición formal adaptada a los arrays es:

```
for(indice in array) {
  ...
}
```

Si se quieren recorrer todos los elementos que forman un array, la estructura for...in es la forma más eficiente de hacerlo, como se muestra en el siguiente ejemplo:

```
var dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"];

for(i in dias) {
  alert(dias[i]);
}
```

La variable que se indica como indice es la que se puede utilizar dentro del bucle for...in para acceder a los elementos del array. De esta forma, en la primera repetición del bucle la variable i vale 0 y en la última vale 6.

Esta estructura de control es la más adecuada para recorrer arrays (y objetos), ya que evita tener que indicar la inicialización y las condiciones del bucle for simple y funciona correctamente cualquiera que sea la longitud del array. De hecho, sigue funcionando igual aunque varíe el número de elementos del array.

## Otros puntos de interés:

### Funciones y propiedades básicas de JavaScript

JavaScript incorpora una serie de herramientas y utilidades (llamadas funciones y propiedades, como se verá más adelante) para el manejo de las variables. De esta forma, muchas de las operaciones básicas con las variables, se pueden realizar directamente con las utilidades que ofrece JavaScript.

### Funciones útiles para cadenas de texto

A continuación se muestran algunas de las funciones más útiles para el manejo de cadenas de texto:

**length**, calcula la longitud de una cadena de texto (el número de caracteres que la forman)

```
var mensaje = "Hola Mundo";
var numeroLetras = mensaje.length; // numeroLetras = 10 +, se emplea para
                                   //concatenar varias cadenas de texto

var mensaje1 = "Hola";
var mensaje2 = " Mundo";
var mensaje = mensaje1 + mensaje2; // mensaje = "Hola Mundo"
```

Además del operador +, también se puede utilizar la función **concat()**

```
var mensaje1 = "Hola";
var mensaje2 = mensaje1.concat(" Mundo"); // mensaje2 = "Hola Mundo"
```

Las cadenas de texto también se pueden unir con variables numéricas:

```
var variable1 = "Hola ";
var variable2 = 3;
var mensaje = variable1 + variable2; // mensaje = "Hola 3"
```

Cuando se unen varias cadenas de texto es habitual olvidar añadir un espacio de separación entre las palabras:

```
var mensaje1 = "Hola";
var mensaje2 = "Mundo";
var mensaje = mensaje1 + mensaje2; // mensaje = "HolaMundo"
```

Los espacios en blanco se pueden añadir al final o al principio de las cadenas y también se pueden indicar forma explícita:

```
var mensaje1 = "Hola";
var mensaje2 = "Mundo";
var mensaje = mensaje1 + " " + mensaje2; // mensaje = "Hola Mundo"
```

toUpperCase(), transforma todos los caracteres de la cadena a sus correspondientes caracteres en mayúsculas:

```
var mensaje1 = "Hola";
var mensaje2 = mensaje1.toUpperCase(); // mensaje2 = "HOLA"
```

toLowerCase(), transforma todos los caracteres de la cadena a sus correspondientes caracteres en minúsculas:

```
var mensaje1 = "HolA";
var mensaje2 = mensaje1.toLowerCase(); // mensaje2 = "hola"
```

charAt(posicion), obtiene el carácter que se encuentra en la posición indicada:

```
var mensaje = "Hola";
var letra = mensaje.charAt(0); // letra = H
letra = mensaje.charAt(2);     // letra = l
```

indexOf(caracter), calcula la posición en la que se encuentra el carácter indicado dentro de la cadena de texto. Si el carácter se incluye varias veces dentro de la cadena de texto, se devuelve su primera posición empezando a buscar desde la izquierda. Si la cadena no contiene el carácter, la función devuelve el valor -1:

```
var mensaje = "Hola";
var posicion = mensaje.indexOf('a'); // posicion = 3
posicion = mensaje.indexOf('b');     // posicion = -1
```

Su función análoga es lastIndexOf():

lastIndexOf(caracter), calcula la última posición en la que se encuentra el carácter indicado dentro de la cadena de texto. Si la cadena no contiene el carácter, la función devuelve el valor -1:

```
var mensaje = "Hola";
var posicion = mensaje.lastIndexOf('a'); // posicion = 3
posicion = mensaje.lastIndexOf('b');     // posicion = -1
```

La función lastIndexOf() comienza su búsqueda desde el final de la cadena hacia el principio, aunque la posición devuelta es la correcta empezando a contar desde el principio de la palabra.

substring(inicio, final), extrae una porción de una cadena de texto. El segundo parámetro es opcional. Si sólo se indica el parámetro inicio, la función devuelve la parte de la cadena original correspondiente desde esa posición hasta el final:

```
var mensaje = "Hola Mundo";
var porcion = mensaje.substring(2); // porcion = "la Mundo"
porcion = mensaje.substring(5);     // porcion = "Mundo"
porcion = mensaje.substring(7);     // porcion = "ndo"
```

Si se indica un inicio negativo, se devuelve la misma cadena original:

```
var mensaje = "Hola Mundo";
var porcion = mensaje.substring(-2); // porcion = "Hola Mundo"
```

Cuando se indica el inicio y el final, se devuelve la parte de la cadena original comprendida entre la posición inicial y la inmediatamente anterior a la posición final (es decir, la posición inicio está incluida y la posición final no):

```
var mensaje = "Hola Mundo";
var porcion = mensaje.substring(1, 8); // porcion = "ola Mun"
porcion = mensaje.substring(3, 4);     // porcion = "a"
```

Si se indica un final más pequeño que el inicio, JavaScript los considera de forma inversa, ya que automáticamente asigna el valor más pequeño al inicio y el más grande al final:

```
var mensaje = "Hola Mundo";
var porcion = mensaje.substring(5, 0); // porcion = "Hola "
porcion = mensaje.substring(0, 5);     // porcion = "Hola "
```

split(separador), convierte una cadena de texto en un array de cadenas de texto. La función parte la cadena de texto determinando sus trozos a partir del carácter separador indicado:

```
var mensaje = "Hola Mundo, soy una cadena de texto!";
var palabras = mensaje.split(" ");
// palabras = ["Hola", "Mundo,", "soy", "una", "cadena", "de", "texto!"];
```

Con esta función se pueden extraer fácilmente las letras que forman una palabra:

```
var palabra = "Hola";
var letras = palabra.split(""); // letras = ["H", "o", "l", "a"]
```

**Funciones útiles para arrays**

A continuación se muestran algunas de las funciones más útiles para el manejo de arrays:

**length**, calcula el número de elementos de un array

```
var vocales = ["a", "e", "i", "o", "u"];
var numeroVocales = vocales.length; // numeroVocales = 5
```

**concat()**, se emplea para concatenar los elementos de varios arrays

```
var array1 = [1, 2, 3];
array2 = array1.concat(4, 5, 6);   // array2 = [1, 2, 3, 4, 5, 6]
array3 = array1.concat([4, 5, 6]); // array3 = [1, 2, 3, 4, 5, 6]
```

**join(separador)**, es la función contraria a split(). Une todos los elementos de un array para formar una cadena de texto. Para unir los elementos se utiliza el carácter separador indicado

```
var array = ["hola", "mundo"];
var mensaje = array.join(""); // mensaje = "holamundo"
mensaje = array.join(" ");    // mensaje = "hola mundo"
```

**pop()**, elimina el último elemento del array y lo devuelve. El array original se modifica y su longitud disminuye en 1 elemento.

```
var array = [1, 2, 3];
var ultimo = array.pop(); // ahora array = [1, 2], ultimo = 3
```

**push()**, añade un elemento al final del array. El array original se modifica y aumenta su longitud en 1 elemento. (También es posible añadir más de un elemento a la vez)

```
var array = [1, 2, 3];
array.push(4); // ahora array = [1, 2, 3, 4]
```

**shift()**, elimina el primer elemento del array y lo devuelve. El array original se ve modificado y su longitud disminuida en 1 elemento.

```
var array = [1, 2, 3];
var primero = array.shift(); // ahora array = [2, 3], primero = 1
```

**unshift()**, añade un elemento al principio del array. El array original se modifica y aumenta su longitud en 1 elemento. (También es posible añadir más de un elemento a la vez)

```
var array = [1, 2, 3];
array.unshift(0); // ahora array = [0, 1, 2, 3]
```

**reverse()**, modifica un array colocando sus elementos en el orden inverso a su posición original:

```
var array = [1, 2, 3];
array.reverse();// ahora array = [3, 2, 1]
```

**Funciones útiles para números**

A continuación se muestran algunas de las funciones y propiedades más útiles para el manejo de números.

**NaN**, (del inglés, "Not a Number") JavaScript emplea el valor NaN para indicar un valor numérico no definido (por ejemplo, la división 0/0).

```
var numero1 = 0;
var numero2 = 0; alert(numero1/numero2); // se muestra el valor NaN
```

**isNaN()**, permite proteger a la aplicación de posibles valores numéricos no definidos

```
var numero1 = 0;
var numero2 = 0;
if(isNaN(numero1/numero2)) {
  alert("La división no está definida para los números indicados");
}
else {
  alert("La división es igual a => " + numero1/numero2);
}
```

**Infinity**, hace referencia a un valor numérico infinito y positivo (también existe el valor –Infinity para los infinitos negativos)

**var numero1 = 10;**
**var numero2 = 0;**
**alert(numero1/numero2); // se muestra el valor Infinity**

**toFixed(digitos)**, devuelve el número original con tantos decimales como los indicados por el parámetro digitos y realiza los redondeos necesarios. Se trata de una función muy útil por ejemplo para mostrar precios.

**var numero1 = 4564.34567;**
**numero1.toFixed(2); // 4564.35**
**numero1.toFixed(6); // 4564.345670**
**numero1.toFixed(); // 4564**

**Provemos node.js con js**
Crea y ejecuta este script dentro de tu terminal de vscode para determinar si Node.js fue instalado correctamente, crear un archivo llamado **HolaMundo.js** con la siguiente línea:

**console.log('!!!Hola Mundo Node.js!!!');**

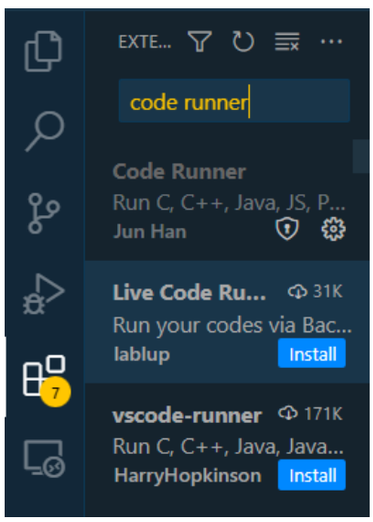
Corre el script usando el comando node.

**node HolaMundo.js**

**R/ !!!Hola Mundo Node.js!!!**

Si Node.js está instalado correctamente, verás que la consola responde con **!!!Hola Mundo Node.js!!!**

**Actividad en clase:**
Una vez instaladas las herramientas de **Node.js** y **VsCode**, realice los siguientes pasos :




✓Instala la extensión de code runner en tu vs code.
✓Dentro de Visual Studio Code crea un nuevo proyecto que se llamará "**Scripts**"
✓Dentro de él se creará los diferentes archivos de extensión js de los ejercicio y ejemplos propuestos en la clase
✓Con tu archivo js abierto, seleccionando el bloque de código deseado, click derecho y luego click en run code.
✓La terminal integrada en vs code mostrará el código en ejecución.
✓Una vez realizada la actividad, proceda a crear una carpeta con sus nombres y apellidos completos, dentro de ella coloque otra carpeta llamada C1 y aloje las evidencias dentro de ellas.

En la Sesion #6 haremos actividades correspondientes a TypeScript con **node.js** y **vscode**, dejaremos el siguiente dato:

## ¿Qué diferencias hay entre JavaScript y TypeScript?

| *JAVASCRIPT* | TYPESCRIPT |
|---|---|
| ✓*No admite parametros opcionales*<br>✓*Es un lenguaje interpretado, por eso resalta los errores en tiempo de ejecución.*<br>✓*JavaScript no admite módulos.*<br>✓*JavaScript no admite genéricos.*<br>✓*Los números y cadena son de tipo objetos.* | ✓Admite parámetros opcionales.<br>✓Compila el código y resalta los errores durante el tiempo de desarrollo.<br>✓TypeScript brinda soporte para módulos.<br>✓Permite **genericos**.<br>✓Los números y cadena son de tipo interface. |

### ¿Qué es un tipo de dato genérico?

Un tipo genérico es un elemento de programación único que se adapta para ejecutar la misma funcionalidad para distintos tipos de datos. Cuando se define una clase o un procedimiento genérico, no es necesario definir una versión independiente para cada tipo de datos para el que quiera ejecutar esa funcionalidad.

### ¿Qué son interfaces en TypeScript?

El uso de interface en la construcción de Objetos.

### ¿Apps híbridas Qué son?

Las apps híbridas son aquellas que se pueden usar en diferentes sistemas operativos, en Smartphone o en tableta. No son, por tanto, exclusivas de una marca comercial o de un modelo concreto de dispositivo: el Frameworks con el que se diseñan sirve para diversos gadgets.
Dentro de las ventajas de las apps híbridas es que con el mismo código podemos desplegar en diferentes dispositivos y sistemas operativos: iOS, Android, web… Esta característica las convierte en una opción más económica frente a una app nativa, además de que el "**time to market**[1]" es mucho más reducido.

[1] Tiempo que transcurre desde que se concibe un producto o servicio hasta que se lanza al mercado