01 - ¿Qué hace que Javascript sea diferente de cualquier otro lenguaje de programación?

- Javascript es el único lenguaje que se puede usar sin necesidad de ejecutarlo desde un servidor o tener que instalar ningún software extra en el equipo, es decir, se puede ejecutar directamente desde el navegador (Chrome, Firefox, Opera, etc). Esto quiere decir que puedes crear un archivo con formato "js" en cualquier equipo, ejecutarlo en el navegador y funcionara correctamente sin necesidad de hacer ningún otro proceso.

- Muchas de las aplicaciones y webs más importantes están integradas usando Javascript. Sitios como Gmail o Facebook usan Javascript para crear sistemas propios, además gracias a ello existen muchas librerías y documentación detallada disponible.

- Es posible usar Javascript para crear aplicaciones para dispositivos móviles, tanto para iOS como Android, sin necesidad de usar los lenguajes específicos para cada sistema.

- JavaScript es un lenguaje dinámico y flexible, es decir, permite agregar y modificar funciones y propiedades a la vez que se está ejecutando.

- JavaScript tiene una amplia variedad de bibliotecas (jQuery, AngularJS, React, Node.js, etc) que ayudan con el desarrollo para ahorrar tiempo y líneas de código.

- JavaScript soporta programación orientada a eventos, es decir, permite responder a acciones del usuario de forma dinámica y en tiempo real.

- Javascript permite la automatización de flujos de trabajo para que de esta forma pueda hacer tareas repetitivas más rápido

Fuentes:

- Hackaboss
- The Power Business School
- Campus MVP

02 - ¿Cuáles son algunos tipos de datos JS?

- booleano
  - Solo puede contener dos valores, true o false
  - Ejemplo: let myBooleano = true;

- number
  - Contiene números enteros y decimales con una limitación de 64 bits, esto quiere decir que tiene una precisión limitada en cuanto a cálculos de coma flotante y números muy largos (Tanto en positivo como en negativo).
  - Ejemplo: let myNumber = 150;

- bigint
  - Contiene números sin limitación en cuanto a su tamaño pero a diferencia del tipo “Number” solo admite números enteros, no decimales o de coma flotante.
  - Es importante remarcar que no se pueden hacer operaciones entre variables tipo “bigint” y “number”, la variables “bigint” solo se pueden operar con otras variables “bigint”
  - Para que una variable sea “Bigint” y no “Number” se debe terminar con una “n” final antes del punto y coma “;”
  - Ejemplo let myBigint = 1234567890123456789012345678901234567890n;

- string
  - Contiene una cadena de caracteres que pueden ser letras, números y símbolos.
  - Ejemplo: let myString = ‘Esto es una cadena de texto, con números (123) y simbolos (@=)’;

- array
  - Contiene una lista ordenada de elementos que pueden ser de diferentes tipos de datos. Su longitud es variable y se pueden eliminar y añadir elementos en cualquier momento.
  - Ejemplo: let myArray = [‘Dato1’, ‘Dato2’];

- object
  - Contiene una lista con índices para identificar a cada uno de los elementos. Al igual que los “Arrays” su longitud es variable y se pueden eliminar y añadir elementos en cualquier momento, además también pueden ser elementos de diferentes tipos.
  - Ejemplo: let myObject = {Nombre1: “Valor de Nombre1”, Nombre2: “Valor de Nombre2”};

- null
  - Este tipo de variable se usa para representar la ausencia de un valor, es decir, que una variable tipo “null” se considera una variable vacía y que no tiene ningún valor.
  - Ejemplo let myNull = null;

- undefined
  - Es una variable vacia y que no tiene ningún valor, pero a diferencia del tipo “null”, las variables “undefined" son creadas sin valor y quedan a la espera de asignárselo.
  - Ejemplo 1:
    - let myUndefined;
  - Ejemplo 2:
    - let myUndefined = undefined;

- symbol
  - Es una variable utilizada para datos únicos e inmutables, no hay dos variables de este tipo iguales, ni siquiera dando el mismo valor, esto hace que puedan ser usadas como identificadores o claves de objetos.
  - La inclusión de un identificador es opcional y no afecta a su uso.
  - Ejemplo 1:
    - const mySymbol = symbol(‘dato’);
  - Ejemplo 2:
    - Const mySymbol = symbol();

- function
  - Este tipo de variable contiene una función que puede ser ejecutada en cualquier momento.
  - Ejemplo: const sumar = function() {}

Fuentes:

- Developer Mozilla
- Manual Web

03 - ¿Qué son las tres funciones de cadena JS?

No se debe confundir funciones con atributos, Las funciones llevan paréntesis al final que es donde se encapsula la información con la que va a trabajar, y los atributos no llevan ningún tipo de paréntesis final. Se verían de la siguiente forma:

- Atributo:
  - let myText = 'Texto de prueba';
  - myText.length;
  - Devuelve: 15

- Función:
  - let myText = 'Texto de prueba';
  - myText. charAt(10);
  - Devuelve: "r"

Alguna de las funciones relacionadas con cadenas son las siguientes:

- charAt()
  - Sirve para devolver el carácter de la posición especificada dentro del paréntesis, es decir, entre paréntesis debemos informar de la posición en la que estemos interesados y nos devolverá el carácter que este en dicha posición.
  - Ejemplo
    - let myText = 'Texto de prueba';
    - console.log(myText.charAt(10));
      - Devuelve: "r"

- indexOf()
  - Sirve para devolver la primera ocurrencia de un elemento en una cadena, es decir, hará una búsqueda en una cadena de lo que se le informe entre los paréntesis y te dirá la posición en la que se encuentra esa primera ocurrencia.
  - En el caso de que existan en la cadena varias ocurrencias de lo que se marca entre paréntesis, únicamente informara de la posición de la primera y no buscara más.
  - Ejemplo
    - let myTexto = 'Texto de prueba';
    - console.log(myTexto.indexOf('de'));
    - Devuelve: 6

- lastIndexOf()
  - Sirve para devolver la última ocurrencia de un elemento en una cadena, es decir, hará una búsqueda en una cadena de lo que se le informe entre los paréntesis y te dirá la posición en la que se encuentra esa última ocurrencia.
  - En el caso de que existan en la cadena varias ocurrencias de lo que se marca entre paréntesis, únicamente informara de la posición de la última.
  - Ejemplo
    - let myTexto = 'Texto de prueba para funciones de cadena';
    - console.log(myTexto.lastIndexOf('de'));
    - Devuelve: 36

- concat()
  - Sirve para añadir cadenas de texto al final de una variable de tipo "string". Es importante tener en cuenta que el uso de esta función no cambia la variable original por sí misma, sino que debemos o bien sobrescribir la función original con la nueva cadena o guardarla en una nueva variable.
  - Es posible añadir una cadena nueva entre comillas o también añadir una cadena anteriormente guardada en una variable, admite ambos orígenes de cadenas.
  - Ejemplo 1 – Usando una cadena entre comillas:
    - let myText = 'Texto original';
    - myText = myText.concat(' y texto nuevo');
    - console.log(myText);

  - Ejemplo 2 – Usando una variable ya creada:
    - let myText = 'Texto original';
    - let myOtherText = ' y texto nuevo';
    - let myNewText = myText.concat(myOtherText);
    - console.log(myNewText);

  - En ambos ejemplos devuelve "Texto original y texto nuevo"

- includes()
    - Sirve para comprobar si una cadena esta dentro de otra, devuelve un valor booleano por lo que obtendremos “true” si encuentra la cadena y “false” si no la encuentra.
    - Ejemplo:
        - let myText = ‘Texto original’;
        - console.log(myText.includes(‘pru’));
        - Devuelve: true

- endsWith()
    - Sirve para comprobar si una cadena termina con la cadena que se le informe, devuelve un valor booleano por lo que obtendremos “true” si termina con la cadena solicitada y “false” si no termina.
    - Se puede usar tanto una cadena entre comillas como una cadena guardada en una variable.
    - Ejemplo 1 – Usando una cadena entre comillas:
        - let myText = ‘Texto de prueba’;
        - console.log(myText.endsWith(‘prueba’));

    - Ejemplo 2 – Usando una variable ya creada:
        - let myText = ‘Texto de prueba’;
        - let myOtherText = ‘prueba’;
        - console.log(myText.endsWith(myOtherText));

    - En ambos ejemplos devolverá true.

- startWith()
    - Sirve para comprobar si una cadena comienza con la cadena que se le informe, devuelve un valor booleano por lo que obtendremos “true” si comienza con la cadena solicitada y “false” si no comienza.
    - Se puede usar tanto una cadena entre comillas como una cadena guardada en una variable.
    - Ejemplo 1 – Usando una cadena entre comillas:
        - let myText = ‘Texto de prueba’;
        - console.log(myText.startWith(‘Texto’));

    - Ejemplo 2 – Usando una variable ya creada:
        - let myText = ‘Texto de prueba’;
        - let myOtherText = ‘Texto’;
        - console.log(myText. startWith(myOtherText));

    - En ambos ejemplos devolverá true.

- repeat()
    - Sirve para hacer repeticiones de una cadena tantas veces como se le índice.
    - Ejemplo:
        - let myText = ‘Texto de prueba’;
        - console.log(myText.repeat(3));
        - Devuelve: “Texto de prueba Texto de prueba Texto de prueba”

- replace()
    - Sirve para reemplazar una cadena de texto por otra.
    - Se deben pasar dos argumentos dentro del paréntesis. El primer argumento es la cadena que debe buscar y es la que va a quitar, el segundo argumento es la cadena sustituta, la que va a añadirse a la cadena original.
    - Ejemplo:
        - let myText = ‘Texto de prueba’;
        - console.log(myText.replace(‘prueba’, ‘reemplazo’));
        - Devuelve: “Texto de reemplazo”.

- slice()
  - Sirve para extraer una parte de la cadena original, sin necesidad de que sea únicamente el inicio o final de la cadena.
  - Se deben pasar dos argumentos dentro del paréntesis. El primer argumento es la posición inicial desde donde se comenzara a extraer y el segundo argumento es la posición final de extracción.
  - Si únicamente se pone un solo argumento, ese argumento será la posición inicial de extracción y extraerá hasta el final de la cadena
  - Si no se usan argumentos, extraerá la cadena completa.
  - Ejemplo – Usando dos argumentos:
    - let myText = 'Texto de prueba';
    - console.log(myText.slice(2, 12))
    - Devuelve: "xto de pru"

  - Ejemplo – Usando un argumento:
    - let myText = 'Texto de prueba';
    - console.log(myText.slice(2))
    - Devuelve: "xto de prueba"

  - Ejemplo – Sin argumentos:
    - let myText = 'Texto de prueba';
    - console.log(myText.slice())
    - Devuelve: "Texto de prueba"

- trim()
  - Sirve para eliminar espacio en blanco tanto al principio como al final de una cadena, esto es útil para hacer limpieza de datos.
  - Ejemplo:
    - Let myText = ' Texto de prueba ';
    - console.log(myText.trim());
    - Devuelve: "Texto de prueba"

Fuentes:

- Developer Mozilla
- Aprender a Programar
- Blog Hubspot

04 - ¿Qué es un condicional?

Las condicionales son estructuras de comprobación que nos permiten realizar diferentes acciones dependiendo de si una condición se cumple o no. Las condicionales nos permiten tomar decisiones y ejecutar bloques de código diferentes en base a esas decisiones.

Existen tres niveles a la hora de crear condicionales, if, else y else if.

- if:
  - Es la condicional básica que nos permite ejecutar un bloque de código si una condición es verdadera.
  - Ejemplo:
    - let edad = 18;
    - if (edad >= 18) {
      - console.log("Eres mayor de edad");
    - }
  - En este ejemplo, si la variable edad es mayor o igual a 18, nos devolverá el mensaje "Eres mayor de edad".

- else:
  - Se puede combinar con la condicional if para ejecutar un bloque de código en el caso de que la condición sea falsa.
  - Al ser el opuesto al "if" inicial, no se deben poner condiciones ya que si la condición inicial del "if" es falsa, automáticamente se ejecutara el código dentro de "else"
  - Ejemplo:
    - let edad = 15;
    - if (edad >= 18) {
      - console.log("Eres mayor de edad");
    - } else {
      - console.log("Eres menor de edad");
    - }
  - En este ejemplo, si la variable edad es menor de 18, nos devolverá el mensaje "Eres menor de edad".

- else if:
  - Nos permite agregar condicionales adicionales en el caso de que la primera condición no se cumpla.
  - Se pueden añadir tantos "else if" como sean necesarios y a diferencia de "else", con "else if" si se deben añadir condiciones que se deban cumplir.
  - Ejemplo:
    - let edad = 25;
    - if (edad < 18) {
      - console.log("Eres menor de edad");
    - } else if (edad >= 18 && edad < 65) {
      - console.log("Eres adulto");
    - } else {
      - console.log("Eres sénior");
    - }
  - En este ejemplo, se comprueban tres condiciones distintas, una dentro del "if" inicial y dos dentro del "else if", dependiendo de la edad se imprimirá en consola "Eres menor de edad", "Eres adulto" o "Eres sénior".

Las condicionales dan la posibilidad de poder anidarse una dentro de otra para poder tener más opciones dependiendo de lo que necesitemos que haga nuestro código y que información vaya a tener. Se pueden anidar tantas condicionales como se necesiten.

- Usando los ejemplos anteriores
  - let edad = 25;
  - let licencia = true
  - if (edad < 18) {
    - console.log("Eres menor de edad");
  - } else if (edad >= 18 && edad < 65) {
    - if (licencia === true) {
      - console.log("Eres adulto y puedes conducir");
    - } else {
      - console.log(“Eres adulto pero no puedes conducir”);
  - } else {
    - console.log("Eres sénior");
  - }

- En este ejemplo hemos anidado una nueva condicional dentro del “else if” en la que nos devuelve que eres adulto y dependiendo de si se tiene o no licencia de conducir, puedes o no conducir.

Se puede anidar una condicional dentro de cualquiera de las partes de una condicional superior, “if”, “else if” o “else”

Fuentes:

- Developer Mozilla
- Make it Real Camp
- Programador Web Valencia

05 - ¿Qué es un operador ternario?

Un operador ternario es una forma abreviada de escribir una condicional en menos espacio que con la estructura habitual.

Consiste en un operador de tres partes: la condición, la expresión que se ejecuta si la condición es verdadera y la expresión que se ejecuta si la condición es falsa.

Por ejemplo, queremos hacer una condicional para detectar si un número es mayor o menor de 100:

- Estructura habitual:
  - let myNumber = 50;
  - if (myNumber >= 100) {
    - console.log('Es mayor o igual que 100');
  - } else {
    - Console.log('Es más pequeño que 100');

- Estructura con operador ternario:
  - let myNumber = 50;
  - myNumber >= 100 **?** console.log('Es mayor o igual que 100') **:** console.log('Es más pequeño que 100');

- En ambos casos devuelve "Es más pequeño que 100"

Vamos a analizar la línea:

- La primera parte es la condición que debe cumplir, en nuestro ejemplo "myNumber >= 100".
- Para identificar hasta donde llegan las condiciones se usa el carácter "?". Todo lo que este antes de este carácter son las condiciones y todo lo que este después son las expresiones que van a ejecutarse.
- Después del símbolo "?" tenemos la expresión que se va a ejecutar si las condiciones son validas, en nuestro ejemplo tenemos "console.log('Es mayor o igual que 100')"
- Para identificar que expresión va a ejecutarse si la condicione es verdadera o es falsa, se usa el carácter ":". Todo lo que este después de "?" y antes de ":" es la expresión que se va a ejecutar si las condiciones son verdaderas.
- Y por ultimo tenemos la expresión después de ":" que es lo que va a ejecutarse cuando la condición sea falsa.

Es posible tener más de una condición usando los operadores ternarios, por ejemplo, aprovechando el ejemplo anterior, queremos una condicional que dependiendo del número que le proporcionemos nos diga si es una temperatura en la que el agua es liquida o está en otro estado.

Se haría de la siguiente forma:

- Estructura habitual:
  - let myNumber = 50;
  - if (myNumber > 0 && myNumber < 100) {
    - console.log("Es liquida");
  - } else {
    - console.log("No es liquida");

- Estructura con operador ternario:
  - let myNumber = 10;
  - myNumber > 0 && myNumber < 100 ? console.log("Es liquida") : console.log("No es liquida');

- En ambos casos devuelve "Es liquida".

Tal como se aprecia, lo que se ha puesto antes del carácter "?" es "myNumber > 0 && myNumber < 100" lo que significa que para que las condiciones sean verdaderas, el numero de la variable "myNumber" debe ser mayor que cero y además menor que cien, es decir, se deben cumplir ambas condiciones para que el resultado sea verdadero y en nuestro ejemplo nos retorne "Es liquida"

Fuentes:

- Developer Mozilla
- Lenguaje js
- Desarrollo Web

06 - ¿Cuál es la diferencia entre una declaración de función y una expresión de función?

Una declaración de función es cuando se define una función utilizando la palabra clave “function” seguida del nombre de la función y su código. Por ejemplo:

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

Y una expresión de función es cuando se define una función como una variable. Por ejemplo:

- const sumar = function(a, b) {
    - return a + b;
- }

La principal diferencia entre ambas es que las declaraciones de función se pueden llamar antes de su declaración en el código, mientras que las expresiones de función deben ser declaradas antes de ser llamadas.

Las declaraciones de función están disponibles desde cualquier parte del código sin necesidad de tener que hacer la llamada por debajo de la propia función gracias al Hoisting.

Además las expresiones de función pueden asignarse como valores a variables, pasar como argumentos a otras funciones, etc, con las declaraciones de función no es posible hacerlo.

Fuentes:

- Javascript Info
- Escuela Frontend
- Codemente

07 - ¿Cuál es la palabra clave 'this'?

La palabra clave "this" se refiere al objeto que está siendo ejecutado en ese mismo momento. El valor de "this" puede distinto y dependerá de la ejecución del código.

Hay cuatro maneras principales en las que se puede utilizar la palabra clave "this".

1 - Dentro de un método de un objeto. Cuando se utiliza "this" dentro de un método de un objeto, hace referencia al objeto que contiene ese método. Por ejemplo:

- const persona = {
    - nombre: 'Juan',
    - saludar: function() {
        - console.log(`Hola, mi nombre es ${this.nombre}.`);
    - }
- };
- persona.saludar();
- Devuelve: “Hola, mi nombre es Juan”

2 - En una función normal. Si se utiliza "this" dentro de una función normal, su valor será el objeto global. Por ejemplo:

- function mostrarNombre() {
    - console.log(`Mi nombre es ${this.nombre}.`);
- }
- const nombre = 'María';
- mostrarNombre();
- Devuelve: “Mi nombre es María”

3 - En un constructor de objetos (usando la palabra clave "new"): Cuando se crea un objeto usando la palabra clave "new", el valor de "this" dentro del constructor hace referencia al objeto recién creado. Por ejemplo:

- function Persona(nombre) {
    - this.nombre = nombre;
    - this.saludar = function() {
        - console.log(`Hola, mi nombre es ${this.nombre}.`);
    - };
- }
- const persona1 = new Persona('Carlos');
- persona1.saludar();
- Devuelve: Hola, mi nombre es Carlos.

4 - En un evento: Cuando se utiliza "this" dentro de un controlador de eventos, hace referencia al elemento que desencadenó el evento. Por ejemplo:

- <button onclick="console.log(this.id)" id="boton">Haz clic</button>

En este caso, "this" hará referencia al botón con el id "boton".

En resumen, la palabra clave "this" en JavaScript se utiliza para hacer referencia al objeto que está siendo ejecutado en un determinado momento, dependiendo del contexto en el que se encuentre.

Fuentes:

- Developer Mozilla
- Freecode Camp