**Funciones en JavaScript**

Una función en JavaScript es un bloque de código que realiza una tarea específica y puede ser reutilizado en diferentes partes de un programa a voluntad.

**Características de las Funciones**

* **Encapsulamiento:** Las funciones encapsulan un bloque de código, lo que facilita la reutilización y la organización del código.
* **Ámbito (Scope):** Las funciones crean su propio ámbito. Las variables definidas dentro de una función no están disponibles fuera de ella, a menos que sean retornadas o pasadas.
* **Valor de Retorno**: Las funciones pueden devolver un valor mediante la palabra clave **return**. Si no se especifica un valor de retorno, la función devuelve undefined por defecto.
* **Objetos de Primera Clase**: Las funciones en JavaScript son objetos de primera clase, lo que significa que pueden ser asignadas a variables, pasadas como argumentos y retornadas desde otras funciones.

**Tipos de Funciones en JavaScript:** En JavaScript, hay dos formas principales de crear funciones: expresión de funciones y declaración de funciones. Aunque ambas permiten definir funciones en tu código, difieren en cómo se crean y cómo se comportan en términos de elevación (hoisting) y disponibilidad en el ámbito.

1. **Declaraciones de funciones:** No hay proceso de asignación. Una declaración de función es la manera más sencilla de definir una función en JS. Se hace con la palabra clave function, seguida por el identificador de la función.

* **Hoisting:** Estas funciones se "elevan" al principio del contexto, por lo que puedes llamarlas antes de que estén definidas en el código.
* **Constructora:** Pueden actuar como una función constructora cuando se invoca con new.
* **This:** Disponible dentro de la función; su valor depende de cómo se invoque la función.
* **Prototype:** Las funciones declaradas tienen una propiedad prototype, que es utilizada cuando la función es usada como constructora.
* **Arguments:** Dentro de las funciones declaradas, existe un objeto arguments que contiene todos los argumentos pasados a la función.
* No se pueden utilizar como callback.

Texto

Descripción generada automáticamente

1. **Expresiones de funciones:** Hay un proceso de asignación. Una **expresión de función** es cuando una función se define dentro de una expresión, generalmente asignándola a una variable. Existen dos tipos principales de expresiones de funciones: **anónimas** y **nombradas**.

* **Hoisting:** Las expresiones de funciones no son elevadas en el ámbito en el que se encuentran. Deben declararse antes de usarse. Intentar llamar a una expresión de función antes de su definición resultará en un error.
* **Constructora:** Sí, pueden actuar como constructoras si se invocan con new.
* **This:** Disponible dentro de la función.
* **Prototype:** Las funciones nombradas o anónimas tienen una propiedad prototype.
* **Arguments:** Disponible dentro de la función, igual que en las funciones declaradas

1. **Funciones anónimas:** Es una función sin identificador que se asigna a una variable.

**Texto

Descripción generada automáticamente**

1. **Funciones nombradas:** Es una función que posee un identificador y se asigna a una variable. Podemos acceder a ella desde la misma función, es decir, recursividad.

**Texto

Descripción generada automáticamente**

1. **Funciones flecha:** Son expresiones de funciones porque hay un proceso de asignación pero se comportan de manera diferente. Son una forma más concisa de escribir funciones. Introducidas en ES6 (ECMAScript 2015). Simplifican la sintaxis de las funciones, especialmente cuando se trata de funciones anónimas o callbacks.

Son menos complejas y más ligeras. Por eso es que no tienen acceso a propiedades como this, prototype, arguments,etc.

* **Hoisting:** Las expresiones de funciones no son elevadas en el ámbito en el que se encuentran. Deben declararse antes de usarse. Intentar llamar a una expresión de función antes de su definición resultará en un error.
* **Constructora:** No, no pueden ser usadas como constructoras; lanzarían un error si se invocan con new.
* **This**: No tienen su propio this; heredan el this del contexto donde fueron definidas.
* **Prototype**: No tienen una propiedad prototype.
* **Arguments**: No tienen un objeto arguments; si necesitas acceder a los argumentos, debes usar el **spread operator (…)**.

**Una captura de pantalla de un celular con texto e imagen

Descripción generada automáticamente con confianza media**

1. **Callbacks:** Los **callbacks** no son un tipo especial de función, sino una función pasada como argumento a otra función. Pueden ser cualquiera de los tipos mencionados (declaración de función, expresión de función o arrow function). Se usan ampliamente en el desarrollo web, especialmente para manejar tareas asíncronas, como llamadas a una API, temporizadores, o eventos del usuario.

**Texto

Descripción generada automáticamente** **Una captura de pantalla de un celular con texto e imagen

Descripción generada automáticamente con confianza media**

Los callbacks son cruciales para manejar tareas que toman tiempo en completarse, como leer archivos, esperar respuestas de un servidor, o temporizadores. JavaScript es un lenguaje asíncrono, lo que significa que no espera a que una tarea termine antes de pasar a la siguiente. Esto es bueno para la eficiencia, pero significa que necesitamos una manera de manejar tareas que toman tiempo, y ahí es donde entran los callbacks.

Las **promesas** son una forma de manejar la asincronía de manera más controlada y estructurada que los callbacks. Una promesa es un objeto que representa la eventual **resolución** o **rechazo** de una operación asíncrona.

1. **Función Constructora:** Se utiliza para crear objetos. Se define con la palabra clave new y utiliza la palabra clave this para asignar propiedades al objeto creado.

**Texto

Descripción generada automáticamente**

**Diferencia entre funciones y métodos:** La diferencia entre una función y un método radica en su contexto de uso, pero se utilizan en contextos ligeramente diferentes según el paradigma de programación que estés utilizando. Una función es independiente y no pertenece a ningún objeto, en cambio, los métodos son funciones que están vinculadas a un objeto o una clase en el contexto de la programación orientada a objetos y se invoca a través de él. Los métodos pueden acceder a los datos internos (propiedades) del objeto en el que se invocan y además poseen Prototype que es un atributo dentro de la clase.

* **Contexto:** Una función puede existir de manera independiente y no está necesariamente vinculada a un objeto o clase en particular. Un método, por otro lado, está vinculado a un objeto o una clase y opera en el contexto de ese objeto.
* **Invocación:** Una función se invoca directamente usando su nombre, mientras que un método se invoca en el contexto de un objeto utilizando la notación de punto **(objeto.metodo()).**
* **Acceso a datos:** Los métodos pueden acceder a las propiedades y datos internos de un objeto en el que se invocan. Las funciones pueden tener acceso a los datos pasados como argumentos, pero no necesariamente tienen acceso a los datos internos de otros objetos.

**Arguments:** Es un objeto especial que se encuentra dentro de las funciones. Este objeto contiene todos los argumentos que se pasan a la función cuando se llama. Es como una lista de todos los valores que la función recibe. Se puede usar para crear funciones que manejen un número variable de parámetros y para iterar sobre los argumentos pasados a la función.

**Características principales de arguments**

* **Objeto tipo Array-like:** Aunque no es exactamente un array, se comporta de manera similar a uno. Puedes acceder a sus elementos usando índices (arguments[0], arguments[1], etc.). Tiene una propiedad llamada length que indica cuántos argumentos se pasaron a la función.
* **No está disponible en las Arrow Functions:** En las arrow functions, no tienes acceso al objeto arguments. En su lugar, puedes usar el operador rest (...args).

**Sobrecarga de Funciones en JavaScript:** JavaScript no soporta sobrecarga de funciones de forma nativa como algunos otros lenguajes, donde puedes definir múltiples versiones de una función con el mismo nombre pero con diferentes parámetros. Sin embargo, puedes emular la sobrecarga de funciones utilizando técnicas como la verificación de tipos de parámetros y el manejo de diferentes números de argumentos.

**Sobrecarga Simulada mediante Argumentos:** Puedes usar la cantidad y el tipo de argumentos dentro de una función para simular sobrecarga.

**Texto

Descripción generada automáticamente**