**¿Qué es JavaScript?**

**¿Cómo nace Javascript?**  
Nace con la necesidad de generar dinamismo en las páginas web y que a su vez los usuarios y las empresas pudieran interactuar unos con otros.  
**¿Qué es Javascript?**  
Es un lenguaje interpretado, orientado a objetos, débilmente tipado y dinámico.  
**Débilmente tipado**  
Se pueden hacer operaciones entre tipos distintos de datos (enteros con strings, booleanos con enteros, etc). Ejemplo:

4 + "7"; // 47

4 \* "7"; // 28

2 + **true**; // 3

**false** - 3; // -3

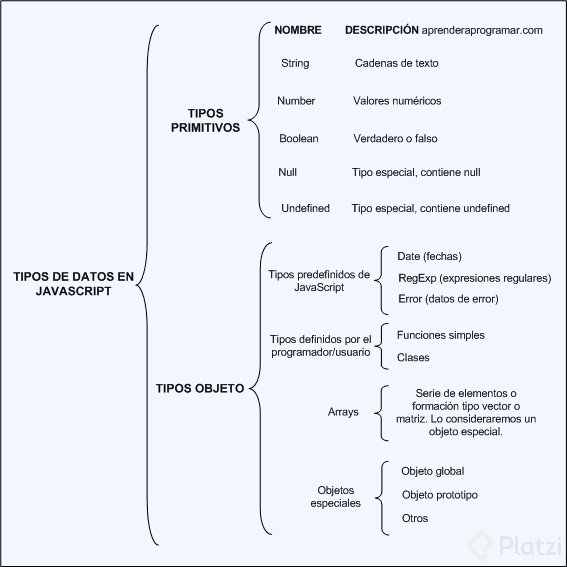
**Dinámico**  
Corre directamente en la etapa de Runetime sin una etapa de compilación previa. Esto permite probar nuestro código inmediatamente; pero también es lo que hace que los errores se muestren hasta que se ejecuta el programa.  
**¿Realmente es Javascript un lenguaje interpretado?**  
Si, y la razón es que le navegador lee linea por linea nuestro código el cuál le indica lo que tiene que hacer, sin la necesidad de compilar. Todo esto es controlado por el motor de Javascript V8 del navegador  
**Javascript es Basckwards Compatible**  
Todas las funciones nuevas que salen de Javascript no dañarán el trabajo ya hecho, pero no se podrá utilizar en nuestro entorno de trabajo inmediatamente. Para solucionar esto está **Babel** que permite utilizar las nuevas características del lenguaje pero lo transforma a una versión que el navegador pueda entender.

# ¿Por qué JavaScript?

1.- JavaScript tiene una **comunidad enorme** de desarrolladores que te pueden ir ayudando a generar diferentes cosas.

1. Si solo estuvieras interesado en trabajar **aplicaciones web** tienes muchos frameworks y librerías construidas en JavaScript que te van a ayudar a hacer proyectos de forma mucho mas rápida, eficiente y robusta (Angular, View, React,entre otros)
2. Si no quieres trabajar solo en aplicaciones Web puedes utilizar JavaScript con un framework que se llama React Native para poder **construir aplicaciones nativas** como Android y IOS.
3. Puedes construir **aplicaciones de escritorio** con JavaScript, usando un framework llamado Electron, pueden correr en Mac o Windows.
4. También puedes trabajar en la parte del **Back-end** o \*\*IOT \*\*(Internet Of Things) es un concepto que se refiere a una interconexion digital de objetos cotidianos con Internet. Esto con un Framework llamado NodeJS, el cual es un entorno de ejecución de JavaScript que corre directamente en el Back-end.

# Elementos de un Lenguaje de Programación: Variables, Funciones y Sintaxis



# Variables

Dentro de JavaScript tenemos tres formas de declarar una variable las cuales son: **var**, **const** y **let.**

1. **Var:** Era la forma en que se declaraban las variables hasta ECMAScript 5. Casi ya no se usa porque es de forma global y tiene las siguientes características:

**o Se puede reinicializar:** osea todas las variables se inicializan, por ejemplo:  
Var pokemonType = ‘electric’ entonces reinicializar es:  
Var pokemonType = ‘grass’ osea la misma variable con diferentes datos el último dato predomina.  
**o Se puede reasignar:** osea la variable ya inicializada le reasignamos otro valor por ejemplo: inicializamos la variable: Var pokemonType = ‘electric’ ahora la reasignamos pokemonType = ‘grass’ ya no va var  
**o Su alcance es función global:** osea inicializamos la variable, pero la podemos llamar desde cualquier bloque (una llave abierta y una cerrada {}) pero hay que tener mucho cuidado con ello ya que puede haber peligro, no es recomendable usar VAR.

**const y let es la forma en que se declaran las variables a partir de ECMAScript 6,**

1. **const:** sirve para declarar variables que nunca van a ser modificadas:  
   **o No se puede reinicilizar:** es una const única no puede haber otra inicializada con el mismo nombre. const pokemonType = ‘electric’ no puede haber:  
   const pokemonType = ‘grass’  
   o **No se pude re asignar:** una vez que la hayamos inicializado no la podemos reasignar solo con su nombre: const pokemonType = ‘electric’ no puede ejecutarse:  
   pokemonType = ‘grass’  
   o **No es inmutable:** osea no puede cambiar con objetos:
2. **Let:** Son variables que pueden ser modificadas, se pueden cambiar:  
   **o No se puede reinicilizar:** es una const única no puede haber otra inicializada con el mismo nombre. let pokemonType = ‘electric’ no puede haber:  
   let pokemonType = ‘grass’  
   o **Se puede reasignar:** Osea la variable ya inicializada le reasignamos otro valor por ejemplo: inicializamos la variable: let pokemonType = ‘electric’ ahora la reasignamos pokemonType = ‘grass’  
   o **Su contexto de es bloque:** Solo funciona dentro de un bloque {}, fuera de ello no.

# ¿Cuándo utilizar una función declarativa y una expresiva?

Cuando hablamos de funciones en JavaScript, tenemos dos tipos de funciones: Funciones Declarativas (function declaration / function statement) y Expresiones de función (function expression / funciones anónimas).

### Funciones Declarativas:

En las funciones declarativas, utilizamos la palabra reservada function al inicio para poder declarar la función:

**function** **saludar**(nombre) {

console.log(`Hola ${nombre}`);

}

saludar('Diego');

#### Expresión de función:

En la expresión de función, la declaración se inicia con la palabra reservada var, donde se generará una variable que guardará un función anónima.

**var** nombre = **function**(nombre){

console.log(`Hola ${nombre}`)

}

nombre(‘Diego’);

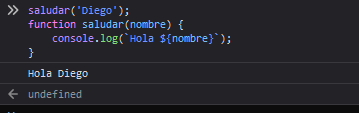
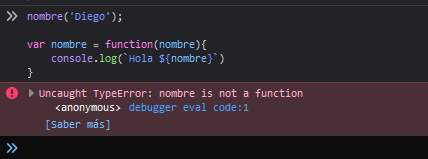
En la expresión de función, la función podría o no llevar nombre, aunque es más común que se hagan anónimas.

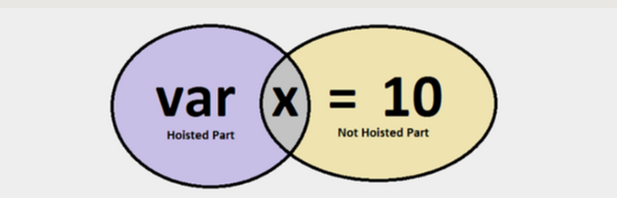
### Diferencias:

A las funciones declarativas se les aplica hoisting, y a la expresión de función, no. Ya que el hoisting solo se aplica en las palabras reservadas var y function.

Lo que quiere decir que con las funciones declarativas, podemos mandar llamar la función antes de que ésta sea declarada, y con la expresión de función, no, tendríamos que declararla primero, y después mandarla llamar.

Cuando se refiere a que el **hoisting** afecta a una y a la otra no:

1. si intentamos usar primero una **función declarativa** antes de ser declarada:  
   
2. si intentamos usar primero una **expresión de función** antes de ser declarada:  
   

Y esto básicamente se debe a que:  
  
**hoisting** detecta primero **declaraciones** de **variables** y **funciones** para poder ser utilizadas, pero no su **definición (valor)**.

Y la **expresión de función** al ser una función **declarada** dentro de una variable, detecta solamente **var nombre**; (sin valor) por lo tanto la función **NO** es detectada en un principio.

# Scope

El scope es el alcance que tiene las variables dentro de un archivo .js

Scope Global: son las variables que están definidas de manera globlal.

Scope Local: son las variables que están definidas dentro de una función.

# Hoisting

El **Hoisting** es un proceso del compilador de JavaScript, que consiste en que la declaracion de las variables y las funciones son llevadas al inicio del codigo, sin importar su posicion, para su procesamiento, sin embargo, la inicializacion de las variables no es llevada al inicio del codigo para su compilacion, sino solo su declaracion, lo cual suele dar espacio a errores cuando se declara una variable sin inicializarla y se procesa en el codigo antes de haber llegado a su inicializacion, lo cual nos suele dar una variable con valor **undefined**, ya que la variable sí fue almacenada en memoria, pero no se le asigno un valor hasta despues de su ejecución.

# Coerción

Coerción es la forma en la que podemos cambiar un tipo de valor a otro, existen dos tipos de coerción:  
Coerción implícita = es cuando el lenguaje nos ayuda a cambiar el tipo de valor.  
Coerción explicita = es cuando obligamos a que cambie el tipo de valor.

# Valores: Truthy y Falsy

