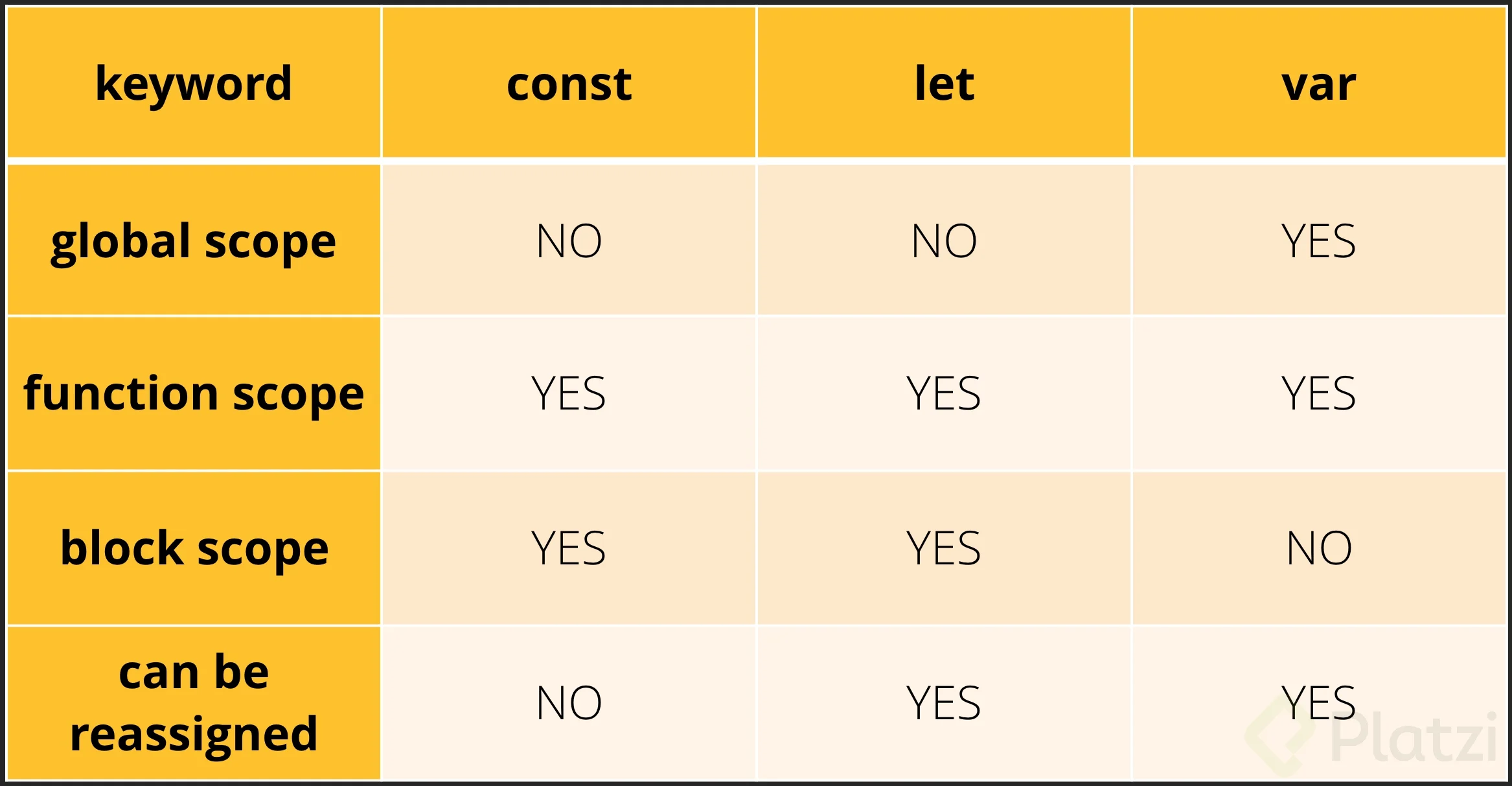
**ECMAScript 6+**

ECMAScript es la especificación del lenguaje JavaScript propuesto por ECMA Internacional, que es la institución encargada de los estándares, y JavaScript, es el lenguaje de programación que utiliza las especificaciones propuestas, que van siendo añadidas cada año a partir del 2015, cuando fue lanzado ES6.

**LET y CONST, Multilínea, Spread Operator y Desestructuración**



**Arrow Functions, Promesas y Parámetros en objetos**

* composicion - newPromise()
  + resolve() - comportamiento al cumplir la promesa
  + reject() - comportamiento al fallar la promesa
  + .then (⇒) - obtener el valor del resolve
  + .error(⇒) - obtener el valor del reject()

**Clases, Módulos y Generadores**

La razón por la que no se corren los módulos adecuadamente es porque la extensión Code Runner usa nodejs para correr el código. Nodejs usa la sintaxis de CommonJS Module. Entonces, la exportación se podría hacer así:

**function hello() {**

**return 'Hello!'**

**}**

**module.exports = hello**

Y la importación, así:

Si quieren que en la consola se vea el “hello!”, recuerden:

const hello = require('./module')

console.log('Hello Module -> ', hello());

Por lo que entiendo, para poder hacerlo con la sintaxis de ES6 Module (la que enseña Oscar), se necesitaría que la extensión usara Babel para transpilar el código.

**¿Qué se implementó en ES7?**

****

En ES7 que salió en junio de 2016 se añadieron nuevas funcionalidades a JS, dentro de estas está Includes y Potency.

Includes() permite identificar un valor dentro de un arreglo o una variable. Potencia, es simplemente una notación para realizar una potencia y se hace colocando un doble asterisco \*\* .

Anteriormente a ES7, usábamos otro método para saber si un elemento estaba en un array y es el siguiente:

**const** arr = [1, 2, 3, 4, 5];

arr.indexOf(1) !== -1; // true

arr.indexOf(10) !== -1; // false

Pero ahora estoy se volvió más legible, tan solo con un includes

**const** arr = [1, 2, 3, 4, 5];

arr.includes(1) // true

arr.includes(10) // false

**¿Qué se implementó en ES8?**

**Async Await**

La expresión await provoca que la ejecución de una función async sea pausada hasta que una Promise sea terminada o rechazada, y regresa a la ejecución de la función async después del término. Al regreso de la ejecución, el valor de la expresión await es la regresada por una promesa terminada.

Si la Promise es rechazada, el valor de la expresión await tendrá el valor de rechazo.

Si el valor de la expresión seguida del operador await no es una promesa, será convertido a una resolved Promise.

**¿Qué se implementó en ES9?**

* **\*\*spread \*\*:** permite manipular partes de un objeto es decir destructurar o estructura un objeto

// Utilizando propiedades de propagación se pueden añadir multiples objetos a otros objetos mediante ...nombre\_objeto

* **then/catch/finally**: ahora podemos tener un funcion que permita saber cuando terminó una promesa.
* **"/miRegexString/.exec() "**: ahora permite obtener los grupos en un arreglo.

**¿Qué se implementó en ES10?**

* **Array.prototype.flat(nivel\_de\_profundidad)**: un nuevo método que nos permite aplanar arreglos.
* **Array.prototype.flatMap()** lo mismo que flat con el beneficio de que primero manipular la data para luego poder aplanar.
* **String.prototype.trimStart() | String.prototype.trimEnd()** permite quitar los espacios al inicio o al final dependiendo de la funcion.
* **try/catch**: ahora puedes utilizarlo sin necesidad de especificaro como catch(error) sino directamente usarlo en el scope del catch.
* **Object.fromEntries()** lo inverso a Object.entries(), es decir podemos convertir un objeto en una matriz clave/valor con Object.entries(), y hace lo inverso es decir de una matriz clave/valor a un objeto con Object.fromEntries().
* **Symbol.prototype.description**: permite regresar el descripcion opcional del Symbol