**DIFERENCIAS ENTRE VAR, LET Y CONST**

““var”” es la manera más antigua de declarar variables. No es muy estricta en cuanto al alcance, ya que al declarar variables de esta forma, dichas variables podrán ser accedidas, e incluso modificadas, tanto dentro como fuera de los bloques internos en una función.

Con ““let”” por otra parte, el alcance se reduce al bloque (las llaves) en el cual la variable fue declarada. Fuera de este bloque la variable no existe. Una vez declarada la variable con let, no se puede volver a declarar con en ninguna otra parte de la función.

““const”” al igual que ““let”” se define en el contexto o alcance de un bloque, a diferencia de let y var, las variables definidas como constantes (const), ya no podrán ser modificadas ni declaradas nuevamente, en ninguna otra parte de la función o el contexto en el que ya existen.

La recomendación es reducir siempre al mínimo el alcance de nuestras variables, por lo que se debe usar let en lugar de var mientras sea posible.

--Aporte Estudiante

• Cuando declaramos variables con ‘var’ siempre conviene declararlas ‘arriba’ del código en el que sea claro cuáles van a ser las variables que se van a usar dentro de nuestra función o programa.

• Dentro de una función javascript detecta todas las variables declaradas con ‘var’ y las ‘declara’ por sí solo como si estuvieran ‘arriba’ en el código. Por lo que si se declara un var dentro de un bloque else y este no es accedido por el condicional, la variable ‘var’ declarada dentro de ese else existe de todas maneras.

• Si utilizamos ‘let’, el alcance de esa variable se ve reducido únicamente al bloque de código donde es utilizado.

• ‘const’ se comporta parecido a let sólo que no es posible reasignarlo.

• Es posible modificar una variable ‘const’ en el caso de un array[ ] con el método push() por ejemplo.

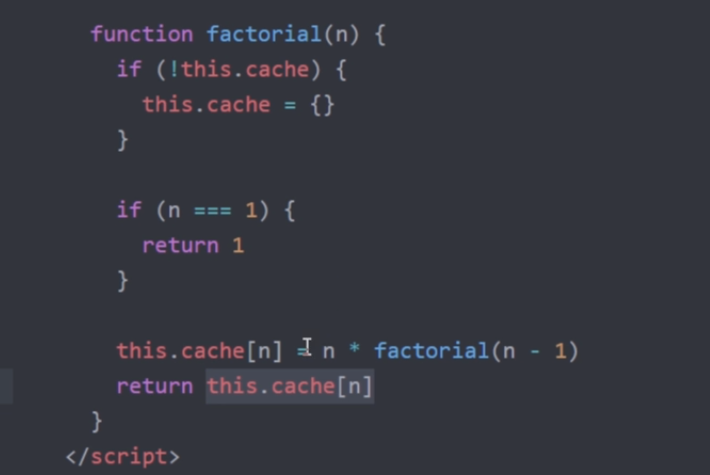
• Reducir siempre al mínimo el alcance de nuestras variables.

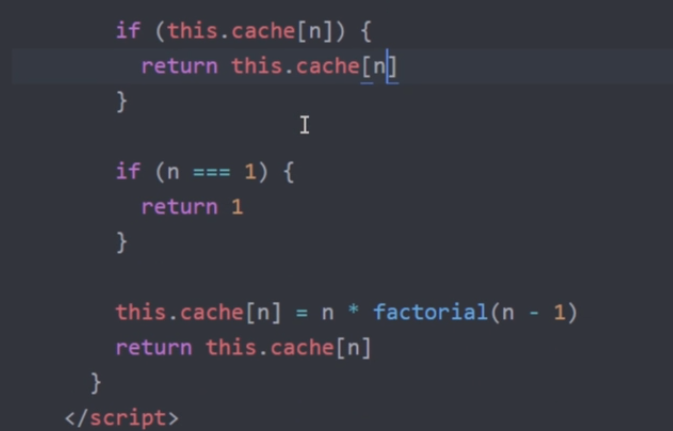
• Utilizar 'let’ si tenemos que reasignar una variable.

• Si nunca tenemos que reasignar una variable usamos ‘const’.

**MEMORIZACIÓN: AHORRANDO CÓMPUTO**

La memorización es una técnica de programación que nos permite ahorrar cómputo o procesamiento en JavaScript, al ir almacenando el resultado invariable de una función para que no sea necesario volver a ejecutar todas las instrucciones de nuevo, cuando se vuelva a llamar con los mismos parámetros. Es similar a usar memoria cache.



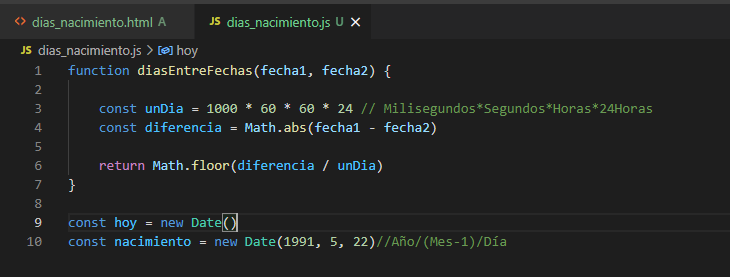


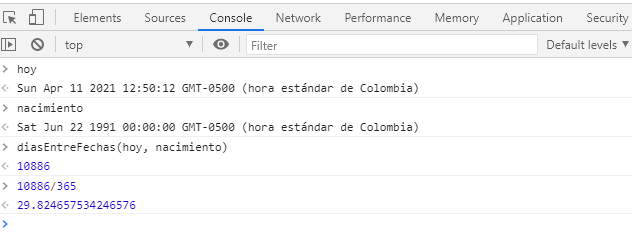
La Memoización es una clave en la Programación Dinámica, les dejo una serie de retos de Dynamic Programming, en uno de esos problemas viene un problema similar al visto en ésta clase

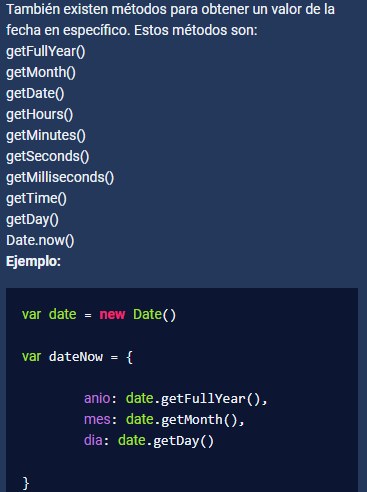
<https://app.codesignal.com/interview-practice/topics/dynamic-programming-basic>

¿Hace cuántos días naciste?

Con variables de tipo Date, se pueden realizar operaciones de suma y resta similares a las que se realizan con números. El resultado que se obtiene está en milisegundos, por lo que luego hay que hacer algunas operaciones adicionales para llevarlos a días, meses o años según queramos. También aplica para Horas, Minutos, Segundos y Milisegundos.

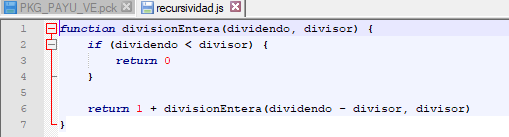






**Recursividad**

La recursividad es un concepto muy importante en cualquier lenguaje de programación. Una función recursiva es básicamente aquella que se llama (o se ejecuta) a sí misma de forma controlada, hasta que sucede una condición base.

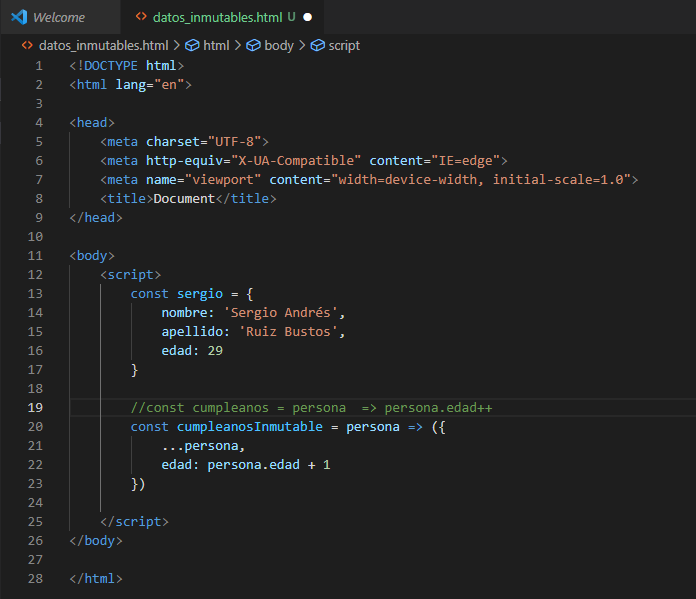


**ENTIENDE LOS CLOSURES DE JAVASCRIPT**

Un closure, básicamente, es una función que recuerda el estado de las variables al momento de ser invocada, y conserva este estado a través de reiteradas ejecuciones. Un aspecto fundamental de los closures es que son funciones que retornan otras funciones.

**ESTRUCTURAS DE DATOS INMUTABLES**

Las estructuras de datos inmutables forman parte de los principios de la Programación Funcional y nos permiten evitar tener efectos colaterales en los datos. En otras palabras, que haya modificaciones en las variables sin nuestro consentimiento, produciendo comportamientos inesperados en el programa.

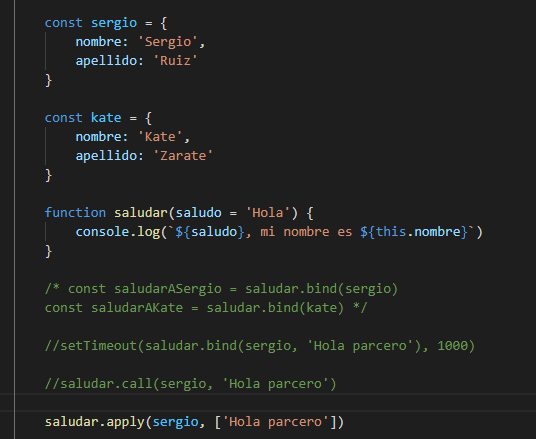


**CAMBIANDO DE CONTEXTO AL LLAMAR A UNA FUNCIÓN**

El contexto (o alcance) de una función es por lo general, window. Así que, en ciertos casos, cuando intentamos referirnos a this en alguna parte del código, es posible que tengamos un comportamiento inesperado, porque el contexto quizás no sea el que esperamos.

Existen al menos tres maneras de cambiar el contexto de una función.

* Usando el método .bind, enviamos la referencia a la función sin ejecutarla, pasando el contexto como parámetro.
* Usando el método .call, ejecutamos inmediatamente la función con el contexto indicado
* Usando el método .apply, es similar a .call pero los parámetros adicionales se pasan como un arreglo de valores



**¿CUÁNDO HACE FALTA PONER EL PUNTO Y COMA AL FINAL DE LA LÍNEA?**

El punto y coma es opcional en JavaScript, excepto en algunos casos:

* Cuando usamos varias instrucciones en una misma línea
* Al comenzar la próxima línea con un array
* Al comenzar la próxima línea con un template string