**Qué es el scope en JavaScript?**

El **scope** puede definirse como **el alcance que una variable tendrá en tu código**. En otras palabras, el scope **decide a qué variables tienes acceso** en cada parte del código. Existen dos tipos de scope, el **scope global** y el **scope local**. A continuación te mostraré un par de ejemplos explicando en qué consiste cada uno de ellos.

**Qué es el Scope Local**

**Cuando puedes acceder a una variable únicamente en cierta parte del código**, se dice que esa variable está declarada en el **scope local**. Estas también son conocidas como **variables locales**. Un ejemplo de esto es **cuando una variable está declarada dentro de un bloque o una función**. Si tratas de acceder a esta variable fuera de esta función o bloque, tendrás un error que dirá que la variable no está definida.

**function** **platzi**() {

**const** soyEstudiante = **true**;

console.log(soyEstudiante);

}

platzi(); // true

console.log(soyEstudiante); // soyEstudiante is not defined

**Qué es el Scope Global**

Se dice que una variable está en el **sope global** cuando **está declarada fuera de una función o de un bloque**. También son conocidas como **variables globales**. Puedes acceder a este tipo de variables desde cualquier parte de tu código, ya sea dentro o fuera de una función.

**const** soyEstudiante = **true**;

**function** **platzi**() {

console.log(soyEstudiante);

}

platzi(); //true

console.log(soyEstudiante); //true

A pesar de que JavaScript nos permite declarar una variable como global, no es una buena práctica. Una de las razones es porque tenemos la posibilidad de declarar dos variables globales en distintas partes del código con el mismo nombre sin notarlo.

**Diferencias entre *let*, *const* y *var***

* **Variables declaradas con *let* o *const*.** Cuando declaramos una variable ***let* o *const* dentro de un bloque, el alcance o scope que tendrá será sólo dentro de ese bloque**. Además, si existe una colisión de nombres entre dos variables *let* o *const* recibiremos un mensaje de error. Esto hace que seamos menos susceptibles a cometer errores al utilizarlas, es por esto que desde que salió ES6 es más recomendable utilizar *let* o *const* a la hora de declarar una variable en lugar de usar *var*.

**let** platzi = 'Esto está fuera del bloque';

**if** (**true**) {

**let** platzi = 'Esto está dentro del bloque';

console.log(platzi); //Esto está dentro del bloque

}

console.log(platzi) //Esto está fuera del bloque

Como podemos ver en el código anterior 👆🏻, aún y cuando declaré dos variables con el mismo nombre no se pisaron, ya que una de ellas está declarada dentro de un bloque y sólo es posible acceder a su valor dentro de éste. Por otro lado, si ambas variables estuvieran declaradas dentro del mismo scope, un mensaje de error sería mostrado en nuestra consola.

**let** platzi = 'Esto está afuera del bloque';

**if** (**true**) {

...

}

**let** platzi = 'Misma variable declarada dos veces'; // Error, platzi has already been declared

* **Variables declaradas con *var***. Si en los ejemplos anteriores, cambiamos las variables declaradas con *let* por variables declaradas con *var*, su valor sí se va a rescribir. Obviamente no es nada recomendable utilizar este tipo de variables porque puede ser más susceptible a bugs.

**var** platzi = 'Esto está afuera del bloque';

**if** (**true**) {

**var** platzi = 'Esto está dentro del bloque';

console.log(platzi); //Esto está dentro del bloque

}

console.log(platzi); //Esto está dentro del bloque

**Otros ejemplos**

Puede ser común también que tengas una función dentro de otra. En este caso, la función que está por fuera no tendrá acceso a las variables que se encuentran en la función de adentro, sin embargo, la función de adentro sí tendra acceso a las variables que se encuentran declaradas en la función de afuera.

**function** **afuera**() {

**let** variable\_afuera = 'Esta variable está declarada afuera';

**function** **adentro**() {

**let** variable\_adentro = 'Esta variable está declarada adentro';

console.log(variable\_afuera); // Esta variable está declarada afuera

}

console.log(variable\_adentro); // variable\_adentro is not defined

}

En conclusión, el término **scope** no es tan extremadamente difícil de entender, es simple una vez que sabes la diferencia entre los distintos tipos que existen. Recuerda: **cuando declaras una variable en una función sólo podrás acceder a su valor dentro de esa función**. Si después de haber leído este tutorial sigues confundido con este concepto, no dudes en dejar tus preguntas en los comentarios.