HINTS

USO DE VARIABLES DENTRO DE LAS FUNCIONES

En JavaScript se pueden declarar dos tipos de variables: globales y locales. Las variables declaradas dentro de una función se vuelven *locales*, y solo se pueden acceder dentro de la función. Las variables declaradas fuera de una función se convierten en variables globales, lo que significa que todas las funciones de una página web pueden acceder a ella.

```
var nombreVariable = "esta es una variable global"
//Las variables globales se pueden utilizar en cualquier parte de nuestra
página web.

function nombreFuncion(){
    var nombreVariable = 'esta es una variable local'; //variable local
    //La variable local SOLO se puede usar dentro de la función
    //La variable global se puede aqui tambien.
}
```

USO DE LOS PARENTESIS ()

Al momento de invocar a una función, debemos saber la manera adecuada de hacerlo.

```
function hola(){
    alert("hola");
}

hola; // Aquí solo llamamos al objeto.

hola(); // Aquí llamamos al método y lograremos ver el alert.
```

Usando el ejemplo anterior, `hola` se refiere al objeto de función, y `hola()` se refiere al método que nos devuelve el comportamiento esperado de la función. Si queremos acceder a una función sin `()`, ésta devolverá el objeto en lugar del resultado.

VARIABLES Y SCOPE

En este punto de nuestro aprendizaje, ya podemos comenzar a revisar sobre 2 nuevos tipos de variables, y el concepto de alcance de variable.

Primero, hablemos del alcance. Éste se refiere al espacio de código entre dos llaves. Por ejemplo, cuando declaramos una función con su nombre, paréntesis y llaves, llamamos al interior de la función: "alcance de la función". El espacio fuera del ámbito se denomina "ámbito global", que significa el espacio de todo el documento.

Ahora, vamos a revisar las nuevas variables: "let" y "const", que forman parte de la última revisión de JavaScript. "let" nos permite declarar variables que solo son accesibles en su alcance, no globalmente. Y "const", por otro lado, nos da acceso a su contenido independientemente de su alcance, al igual que las declaraciones "var". A diferencia de éstas últimas, las variables "const" únicamente pueden recibir un valor cuando se declaran.

Por ejemplo, si declaramos una serie de variables dentro de un `if`, y luego mostramos sus valores mediante la consola, podremos ver la diferencia entre cada declaración de variable. En este caso, las variables `let` y `const` tienen alcance de bloque, lo que significa que no estarán disponibles para usarse fuera de las llaves del `if`.

```
1  if (true) {
2      var temaCuaderno = "JavaScript"
3      let cantiHojas = 192
4      const color = "azul"
5
6  }
7
8  //var tiene alcance de funcion, no hay funcion en este caso.
9  console.log(temaCuaderno)
10
11 // let y const tienen alcance de bloque.
12 console.log(cantiHojas)
13 console.log(color)
```

```
JavaScript                                          javascript.js:13
▶ Uncaught ReferenceError: cantiHojas is not defined  javascript.js:16
    at javascript.js:16:13
>
```

En cambio, con las variables var, vemos que tienen alcance de función, restringiendo su acceso solo a las llaves de una. Por ejemplo:

```
function func(){
    var paginasUsadas = 35
}
// Var tiene alcance de funcion, no se puede acceder fuera de ella.
console.log(paginasUsadas)
```

Esto nos resulta en el siguiente error en nuestra consola:

```
▶Uncaught ReferenceError: paginasUsadas is not defined    javascript.js:13
    at javascript.js:13:13
>
```

La causa de éste es su alcance. Si logramos entender bien el uso de alcance, o "scope" en inglés, podremos emplear estas nuevas declaraciones de variables a nuestro favor. A lo largo de este curso, aprenderemos más sobre la utilización de éstas. También te recomendamos investigar por tu cuenta sobre estos interesantes nuevos implementos.

FUNCIONES ANIDADAS

Otro concepto que podemos analizar en este momento es el de funciones anidadas. Se refiere a cuando empleamos o declaramos una función dentro de otra. En un ejemplo muy sencillo, podemos poner en práctica este concepto creando una función que nos calcula el total de un pedido de comida, y que en su interior utiliza una anidada para darnos el valor de la compra junto a su propina, que será el 10% del total.

```
function calcularTotal(valorPedido){

    // Funcion anidada, nos devuelve el valor con la propina
    function propina(valor) {
        return valor + (valor * 0.10)
    }

    // aqui usamos la funcion anidada.
    total = propina(valorPedido);
```

```
10      console.log("El total con propina es: $"+total);
11  }
12
13  calcularTotal(100);
```

Cómo indica la línea 13, si llamamos a nuestra función calcular total, y le pasamos 100 como su parámetro, deberíamos esperar ver el valor 110 como el total con la propina en nuestra consola.

```
El total con propina es: $110                    javascript.js:17
>
```

Cómo se puede observar, efectivamente logramos usar funciones anidadas para dividir procesos y usarlos de forma ordenada.

EL PROBLEMA DE LAS VARIABLES GLOBALES

Esto se debe a que otras secuencias de comandos sobrescriben fácilmente las variables globales. Éstas no son malas, ni un problema de seguridad, pero no deberían sobrescribir los valores de otra variable.

El uso de más variables globales en nuestro código, puede generar un problema de mantenimiento. Si agregamos una variable con el mismo nombre, en ese caso, debemos estar atentos para algunos errores graves.

Para no utilizarlas, podemos emplear las variables locales, y envolver el código en cierres. De esta forma, se podrá evitar causar serios problemas al código de otras personas, y a tus propios proyectos.

ALCANCE DE VARIABLES EN UN CICLO

Los ámbitos de variables pueden ser un tema complicado de dominar cuando se programa, ya que existen diferentes capas de comportamiento según la ubicación de una variable. Por ejemplo, ¿cuál es el alcance de una variable que está en un bucle?

Para responder a esto, tomaremos en cuenta lo que indica Mozilla, al considerar el alcance de una variable en un bucle for:

"Las variables declaradas con var no son locales para el ciclo, es decir, están en el mismo ámbito en el que se encuentra el ciclo for. Las variables declaradas con let son locales para la declaración".

Por lo tanto, dependiendo de cómo declaremos nuestras variables, estaremos definiendo sus ámbitos, alcances o scopes.