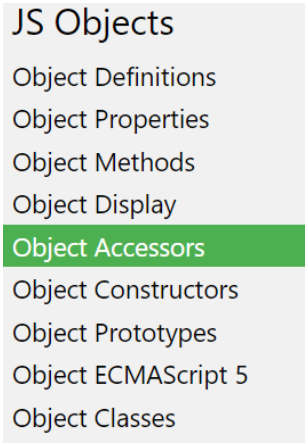
JS-Tarea12-Definición de Getters y Setters

DAM2 - DWEB

Realiza una **lectura comprensiva** del apartado del tutorial correspondiente a: Object Accessors (getters y setters)



Definición de captadores (getters) y establecedores (setters)

- ☐ Un getter (capturador) es un método que obtiene el valor de una propiedad específica.
- ☐ Un setter (establecedor) es un método que establece el valor de una propiedad específica.

En programación, los getters y setters son construcciones habituales de los objetos que permiten acceder a valores o propiedades sin revelar la forma de implementación de las clases. Esto es, permiten encapsular los objetos y evitar mantenimiento de las aplicaciones cuando la forma de implementar esos objetos cambia. En la práctica son simplemente métodos que te permiten acceder a datos de los objetos, para obtener o asignar nuevos valores. El getter (o captador) se encarga de obtener / recibir un valor y el setter (o establecedor) de asignar / establecer un valor.

Para definir getters y setters de objeto, todo lo que necesitas hacer es prefijar un método "captador" con get y un método "establecedor" con set. Lógicamente, el método get no debe esperar un parámetro, mientras que el método set espera exactamente un parámetro (el valor a establecer). Ahora bien, los getters y los setters de JavaScript aquí tratados, utilidad que forma parte de ES5, son un poco distintos de los tradicionales: básicamente son útiles para realizar el acceso a propiedades de los objetos que son resultado de computar otras propiedades existentes.

ECMAScript 5 admite las palabras clave get y set para definir propiedades calculadas

Los getters y setters en JavaScript se usan para definir las llamadas ***propiedades calculadas***, ***propiedades computadas*** o también llamadas ***accesadores***. Una propiedad calculada es aquella que ***usa una function*** para tomar (get) o establecer (set) un valor de object.

En principio, los get y set pueden ser definidos:

A. cuando se crea el objeto con notación literal o
B. agregado posteriormente a cualquier objeto en cualquier momento usando un método de adición para el captador o el establecedor *(defineProperty() y defineProperties)*.

A - Ejemplo en la creación del objeto con notación literal: el objeto persona podría tener un par de propiedades como el "nombre" y los "apellidos". Además podríamos desear conocer el nombre completo, pero realmente este no sería una nueva propiedad del objeto, sino el resultado de un cómputo de concatenación entre el nombre y los apellidos que tenemos en las dos propiedades anteriores. Es decir, "nombreCompleto" podría ser una ***propiedad computada***.

En los lenguajes tradicionales y como hemos visto en la tarea anterior también en Javascript, podemos implementar este valor computado con un método:

```
let persona = {

  nombre: 'Miguel Angel',

  apellidos: 'Alvarez Sánchez',

  getNombreCompleto: function() {

    return this.nombre + ' ' + this.apellidos;

  }

}
```

Y acceder al nombre completo con el siguiente código:

```
document.getElementById("demo").innerHTML = person.getNombreCompleto();
```

(es lo que en la anterior tarea hemos hecho con el método hablar() en Ejer5*)*

Sin embargo, con **las construcciones get y set de Javascript** la forma de definir este código cambia un poco: mediante la palabra reservada "get" puedo definir una función que se encarga de realizar el cómputo. Aquello que devuelve la función será el valor de *la propiedad computada*. El código queda así:

```
let persona = {

  nombre: 'Miguel Angel',

  apellidos: 'Alvarez Sánchez',

  get nombreCompleto() {

    return this.nombre + ' ' + this.apellidos;
```

```
  }

}
```

El uso de "get" en este nuevo código, nos sirve para definir una especie de método. Sin embargo, **no es un método** tradicional, sino una ***propiedad computada. Es una propiedad del objeto*** *que para resolver su valor tiene que ejecutar una función.* Esto lo debes ver claro con la siguiente línea de código en la que se usa la ***propiedad computada*** nombreCompleto, a diferencia del ejemplo anterior en el que para referirnos al método escribíamos *person.getNombreCompleto()*;

```
document.getElementById("demo").innerHTML = persona.nombreCompleto;
```

Fíjate que para acceder a ese "getter" usamos "persona.nombreCompleto", que es como si estuviéramos accediendo a una propiedad convencional, aunque internamente en nuestro objeto no existe tal propiedad, sino que se realiza un cómputo para poder evaluarla.

Ejer1. Piensa e implementa en JavaScript la definición de un objeto con la sintaxis literal de objeto JSON que mediante get obtenga el valor de una propiedad calculada *(el ejemplo ha de ser diferente al del tutorial y al de tu compañero).*

Ejer2. Piensa e implementa en JavaScript la definición de otro objeto al que mediante set establezcas el valor de una propiedad.

B - Object.defineProperty() y Object.defineProperties

Los captadores y establecedores también se pueden agregar a un objeto en cualquier momento después de su creación usando estos métodos. Básicamente:

- proporcionan otra forma de agregar getters y setters,
- pueden usarse en objetos una vez definidos
- definen una nueva propiedad sobre un objeto o modifican una ya existente, devolviendo el objeto modificado.

Sintaxis: *Object.defineProperty(obj, prop, descriptor)* donde

- *obj* es el objeto sobre el cual se define la propiedad.
- *prop* es el nombre de la propiedad que se añade o modifica.
- *descriptor* es el descriptor de la propiedad que está siendo añadida o modificada.

Cuando añadimos una propiedad de la “forma usual” a través de la asignación *(como en Tarea11-Ejer3 con persona.nacionalidad=“Americana”),* no controlamos el ***descriptor*** de la misma y estamos asumiendo su comportamiento por defecto. Es decir, los llamados ***indicadores*** de la propiedad (unos atributos especiales de la misma) toman los siguientes valores: *"writable": true, “enumerable": true, "configurable": true.* Esto permite que el *value* de la propiedad creada pueda modificarse, la propiedad pueda ser enumerada con un bucle *for...in,* y puede ser eliminada con el método delete. Ahora bien, en ocasiones para facilitar el logro de la herencia puede ser necesario tener control sobre dicho

comportamiento y cambiar la configuración predeterminada de los indicadores de la propiedad de un objeto. Es entonces cuando se recurre a definir las propiedades del objeto con *defineProperty() o defineProperties()*. Por ahora y en esta parte del curso, basta con que aprendas a definir y modificar propiedades haciendo uso de estos dos métodos estáticos y comprendas que su uso permite definir la propiedad como no enumerable, no modificable o incluso evitar que pueda ser eliminada del objeto.

Ejer3. Piensa e implementa en JavaScript la definición de un objeto y añade getters y setter al mismo haciendo uso del método Object.defineProperty(). Antes de hacerlo revisa e intenta entender el ejemplo propuesto acerca de este método en el tutorial. *El ejemplo en cuestión crea un objeto contador como el que sigue:*

```
// Define object
var obj = {counter : 0};
```

A posteriori haciendo uso del método ***defineProperty****, añade al objeto obj una nueva propiedad* `reset` *que retorna obj.counter = 0*
(ojo!, se está definiendo una nueva propiedad que nos va a servir para modificar el valor de otra)

```
Object.defineProperty(obj, "reset", {
  get : function () {this.counter = 0;}
});
```

Del mismo modo añade al objeto la propiedad `increment` *mediante otro getter que retorna obj.counter++*

```
Object.defineProperty(obj, "increment", {
  get : function () {this.counter++;}
});
```

Ídem con la propiedad `decrement` *(con un tercer get que retorna obj.counter--)*

```
Object.defineProperty(obj, "decrement", {
  get : function () {this.counter--;}
});
```

Las dos siguientes propiedades las añade mediante ***setters*** */ establecedores. Por un lado define una nueva propiedad add mediante un método set que incrementa el valor de obj.counter con el valor del parámetro:*

```
Object.defineProperty(obj, "add", {
  set : function (value) {this.counter += value;}
});
```

Y por otro añade la propiedad subtract que mediante set establece un nuevo valor a la propiedad counter (el resultado de restar a su valor el parámetro):

```
Object.defineProperty(obj, "subtract", {
```

```
  set : function (value) {this.counter -= value;}
});
```

Demo del uso de las definiciones anteriores:

```
obj.reset;              // Ejecuta el get que produce obj.counter = 0
obj.add = 5;             // Ejecuta el set que asigna 0+5 a la propiedad  'counter' (5)
obj.subtract = 1;       // Ejecuta el set que asigna 5-1 a la propiedad  'counter' (4)
obj.increment;          // Ejecuta el get, que produce un +1 (5)
obj.decrement;          // Ejecuta el get que produce un -1 (4)
```

*Los getters y setters también se pueden agregar en una única sentencia usando el método Object.**defineProperties**. En tal caso el ejemplo anterior lo veríamos así:*

```
<script>
// Define an object
var obj = {counter : 0};

// Define Setters and Getters
Object.defineProperties(obj, {
        "reset": { get : function () {this.counter = 0; } },
        "increment": { get : function () {this.counter++; } },
        "decrement": {  get : function () {this.counter--; } },
        "add": {  set : function (value) {this.counter += value; } },
        "subtract": {  set : function (value) {this.counter -= value; } }
});

// Play with counter:
obj.reset;
obj.add = 5;
obj.subtract = 1;
obj.increment;
obj.decrement;
document.getElementById("demo").innerHTML = obj.counter;
</script>
```

Ejer4. define el mismo getter y setter de los ejercicios 1 y 2, pero esta vez usando el método Object.defineProperties