Desarrollo web en entorno cliente

UD – 4

Objetos en javascript

Contenidos

ÍNDICE

1. INTRODUCCIÓN

Javascript es un lenguaje que permite el uso de objetos. Muchas veces el término clase y objeto se confunden. Una definición podría ser que una clase define "***como es un objeto***" y que un objeto es la plasmación efectiva de ese objeto. A partir de una clase se pueden crear si se desea muchos objetos.

Para entenderlo ***un ejemplo***:
Supongamos tenemos la clase "casa". Esa clase define que atributos tiene una casa. Un ejemplo de esos atributos podría ser: dirección, numero habitaciones, metros cuadrados.

Ahora bien, cada objeto es una casa existente. Podemos tener por ejemplo dos objetos que surgen a partir de la clase "casa". Uno seria una casa con dirección "Avenida del puerto 1, Valencia", 3 habitaciones y 100m2 y otro una casa con dirección "Calle Colón 1, Valencia", 5 habitaciones y 200m2.
La clase definía como podían ser los objetos y los objetos en si son clases contextualizadas en algo concreto.

2. FUNCIONES

Hay varias formas de crear funciones en Javascript:

- Por **declaración** o por **expresión(***anónima, lambda***) o flecha(***Ecmas6***)**:

Definición	Descripción
function nombre (p_1,p_2....) { }	Crea una función mediante **declaración**
nombre = function (p_1,p_2....) { }	Crea una función mediante **expresión**
Nombre = () => { };	Crea una función mediante **flecha**
	return

******Funciones flecha*** *es igual a funciones por expresión pero sin usar la palabra reservada function.*

3. DEFINIR CLASES Y OBJETOS

3.1 Objeto Json

Un objeto JSON está rodeado de llaves {}. Siendo su estructura especificada como pares clave / valor.

```
ana = {
     nombre: "Ana Maria",                        // Propiedades
     edad:  40 ,
     cumplirAnos: function(incremento){          // Métodos
               this.edad = this.edad + incremento;
      }
     };

ana.cumplirAnos(1);
console.log(`</br> Nombre: ${agustin.nombre} y  edad : ${agustin.edad}`);
```

4. CLASES EN ESMAC 6

ES6 (entre otras novedades) incorpora una nueva forma de definir clases. Esto hará que los programadores que vienen de otros lenguajes se sientan mas cómodos y que podamos aplicar conceptos de la POO como la herencia.

La sintaxis de clases es muy similar a lenguajes como JAVA o C#:

```
class personaClass {
    constructor(nombre, edad) {
            this._nombre = nombre;
            this._edad = edad;
    }

    get edad() {
            return this._edad;
    }

    set edad(valor) {
            this._edad = valor;
    }

    cumplirAnos(incremento){
            this.edad = this.edad + incremento;
    }
    imprimirInfo() {
            document.write("Nombre" + this.nombre);
            document.write(", Edad " + this.edad);
    }

    static confirmacion() {
            alert("Enhorabuena esta funcionando la clase Persona");
    }
}

    var mama = new personaClass("Ana", 67);
    mama.cumplirAnos(1);
    mama.imprimirInfo();
    mama.edad;
    personaClass.confirmacion();
```

4.1 Constructor

El constructor es un método especial que inicializa una instancia de la clase. En JavaScript solo puede haber un constructor, no se admite la sobrecarga. Hemos visto en los ejemplos anteriores que usaremos **this** para asignar los valores contextuales y que podemos usar valores por defecto.

```
        constructor(nombre, edad) {
                this._nombre = nombre;
                this._edad = edad;
         }
```

4.2 Getter y Setter

Estos métodos de acceso nos permiten obtener y asignar valores a los atributos de los objetos. En JavaScript estos métodos se corresponden con atajos sintácticos sin funcionalidad adicional. Deberemos tener en cuenta que podemos caer en una referencia circular lo que daría lugar a un desbordamiento. El uso del guion bajo evitará estos errores.

```
        get edad() {
                return this._edad;
        }

        set edad(valor) {
                this._edad = valor;
        }
```

4.3 Herencia

Una vez que JS tiene clases, podemos esperar crear clases hijas a partir de otras, o como también se suele expresar: crear clases que extiendan a otras.

```
    class nieto extends personaClass {
        constructor(nombre, edad, cuidador) {
                super(nombre, edad);
                this.cuidador = cuidador;
        }

        imprimirInfo() {
                super.imprimirInfo();
                document.write(', Cuidador :' + this.cuidador);
        }


      var miguel = new nieto("Miguelito",6 , "Saly");
      miguel.imprimirInfo();
      miguel.cumplirAnos(1);

     document.write(", Edad:" + miguel.edad);
```

5. TRABAJANDO CON CLASES

5.1 ToString()

Al convertir un objeto a string (por ejemplo al concatenarlo con un String) se llama al método *.toString()* del mismo, que devuelve [object object]. Podemos sobrecargar este método para que devuelva lo que queramos:

```
class personaClass {
    ...
    toString() {
        return this.nombre
    }
}
let cpo = new personaClass('Carlos', 19);
console.log('Persona:' + cpo)          // imprime 'Persona: Carlos'
```

5.2 valueOf()

Al comparar objetos (con >, <, …) se usa el valor devuelto por el método *.toString()* pero si sobrecargamos el método ***.valueOf()*** será este el que se usará en comparaciones:

```
class personaClass {
    ...
    valueOf() {
        return this.edad
    }
}

let cpo = new personaClass('Carlos', 19)
let aat = new personaClass('Ana', 23)
console.log(cpo < aat)                  // imprime true ya que 19<23
```

5.3 Prototipos

Cada objeto tiene un prototipo del que hereda sus propiedades y métodos (es el equivalente a su clase, pero en realidad es un objeto que está instanciado). Si añadimos una propiedad o método al prototipo se añade a todos los objetos creados a partir de él lo que ahorra mucha memoria.

```
personaClass.prototype.getInfo2 = function() {
        return 'Estoy añadiendo una nueva función'
    }
}
let cpo = new personaClass('Carlos', 19)
console.log(cpo.getInfo2())     // imprime: 'Estoy añadiendo una nueva función'
```