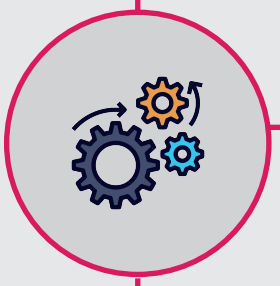
ECMA5

https://github.com/abecerraguz

PROGRAMACIÓN ORIENTADA A OBJETOS

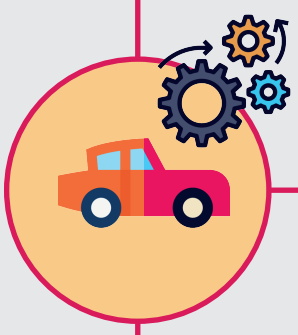


POO / PROCESO DE ABSTRACCION

Un objeto es una representación de algo en el mundo real.

- CASA
- AUTO
- BICICLETA
- CONJUNTO DE NUMEROS

- PROPIEDADES
- METODOS (FUNCIONES)



Función constructora para un objeto

```
function Auto(marca,color){
        this.nombre = nombre;
        this.color     = color
}
```

Instancias del objeto

```
var a1 = new Auto('Toyota' , 'Rojo')
var a2 = new Auto('Nissan' , 'Azul')
var a3 = new Auto('Fiat' , 'Verde')
```

Para agregar un metodo a un objeto

```
a1.sonar_bocina = function(){
        console.log('Beeeep')
}

a1.saludar = function(){
        console.log("Hola soy" +  this.marca )
}

a1.saludar() // Hola soy Toyota
```

ECMA5

https://github.com/abecerraguz

PROTOTIPOS

CONCEPTO DE PROTOTYPE

Los prototipos o prototype nos permiten agregar una propiedad o diversas funcionalidades a múltiples objetos. Por consiguiente, los prototipos son un mecanismo mediante el cual los objetos en JavaScript heredan características entre sí, es decir, un objeto puede tener diversas características pertenecientes a otro objeto, en otras palabras, los objetos en JavaScript se construyen en base a prototipos.

Función constructora para un objeto

```
function Usuario(nombre){
        this.nombre = nombre;
}
```

Instancias del objeto

```
var u1 = new Usuario('Juan')
var u2 = new Usuario('Jocelyn')
```

Para agregar un metodo a un objeto a traves de prototype

```
Usuario.prototype.saludar = function(){
        console.log( 'Hola soy el usuario' + this.nombre )
}
```

Propiedad a través de prototype

```
Usuario.prototype.edad = 30;
```

Salida por consola

```
usuario1.saludar();
usuario2.saludar();
// resultado
Hola soy el usuario Juan
Hola soy el usuario Jocelyn
```

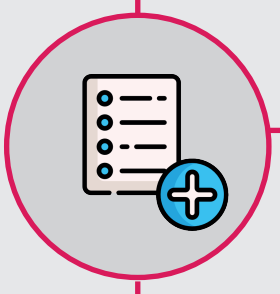
IMPORTANTE

Al igual que si crearamos la propiedad directamente sobre la función constructora, cuando asignamos un nuevo atributo con la propiedad “prototype”, este atributo quedará disponible para todos los objetos instanciados en base a la función constructora.

ECMA5

https://github.com/abecerraguz

Encapsulamiento



CONCEPTO DE PROTOTYPE

El principio de encapsulación nos enseña que debemos proteger los estados internos, es decir, las propiedades de los objetos. En donde su único propósito es ocultar el trabajo interno de los objetos del mundo exterior. En términos prácticos, este principio nos dice que debemos tener métodos para obtener y modificar cada estado, y que no deberíamos modificar los estados directamente. Este aislamiento hace que los datos (propiedades) del objeto sólo pueden gestionarse con las operaciones (métodos) definidos en ese objeto.

FORMAS DE ENCAPSULAMIENTO

Object.defineProperty(),
Object.freeze(),
WeakMap,
Closures y/o IIFE.

Instancias del objeto

```
function Estudiante(nombre){
    var getNombre = nombre;
    Object.defineProperty(this, "getNombre", {value: getNombre});
}
var estudiante1 = new Estudiante('Juan');
estudiante1.getNombre = "Jocelyn";
console.log(estudiante1.getNombre) // Juan
```

GETTER / SETTER

En el paradigma orientado a objetos, los **getters y setters** nos permiten tener accesos controlados a la información que se encuentra en los objetos.

EJEMPLO FUNCION CON GETTER / SETTER

```
function Vehiculos(marca) {
        var _marca = marca;
        Object.defineProperty(this, "_getMarca", {
                get: function () {
                        return _marca
                }
        });

        Object.defineProperty(this, "_setMarca", {
                set: function (marca) {
                        _marca = marca
                }
        });
}
```

Creación de metodos para obtener (get) o fijar (set) propiedad

```
Vehiculos.prototype.getMarca = function(){
        return this._getMarca;
};
```

```
Vehiculos.prototype.setMarca = function(marca){
        this._setMarca = marca;
};;
```

```
var v1 = new Vehiculos("Ford");
console.log(v1.getMarca()); // "Ford"

v1.setMarca("Kia");
console.log(v1.getMarca()); // "Kia"
```