Introducción a la Programación Orientada a Objetos

¿Qué es la POO?

- La programación orientada a objetos (Abreviada P.O.O.) es un paradigma del desarrollo de software.
- **¿Qué son los paradigmas de la programación?**

  Un paradigma de programación es un estilo de desarrollo de programas.
- Son un modelo para resolver problemas en la programación. Los lenguajes de programación, necesariamente, se encuadran en uno o varios paradigmas a la vez a partir del tipo de órdenes que permiten implementar, algo que tiene una relación directa con su sintaxis.

Paradigmas de la programación

- Existen lenguajes que siguen uno, o algunos pocos paradigmas. Por ejemplo, C# y Java son lenguajes orientados a objetos.
- En java por ejemplo, absolutamente todo (salvo los tipos de datos primitivos) son objetos. En C# sucede algo similar.

- Luego hay otros lenguajes que admiten muchas formas de escribir código. A estos se les suele llamar lenguajes mutliparadigma.

Paradigmas en JavaScript

- Javascript es un lenguaje multiparadigma
- Podemos escribir código en javascript siguiendo distintos paradigmas, como la POO, la programación funcional, la programación reactiva, o la programación dirigida por eventos.

- **Python** es otro lenguaje multiparadigma.

Programación Orientada a Objetos

Es un paradigma el cual se basa en el concepto de clases y objetos.

- Este tipo de programación se utiliza para estructurar un programa en piezas simples y reutilizables de código (clases) para crear instancias individuales de objetos.
- Estos objetos se utilizan para la manipulación de datos y donde, por lo general, cada objeto ofrece una funcionalidad especial.
- Con esta forma de programar, podemos modelar objetos de la vida real, en nuestro código

Objetos

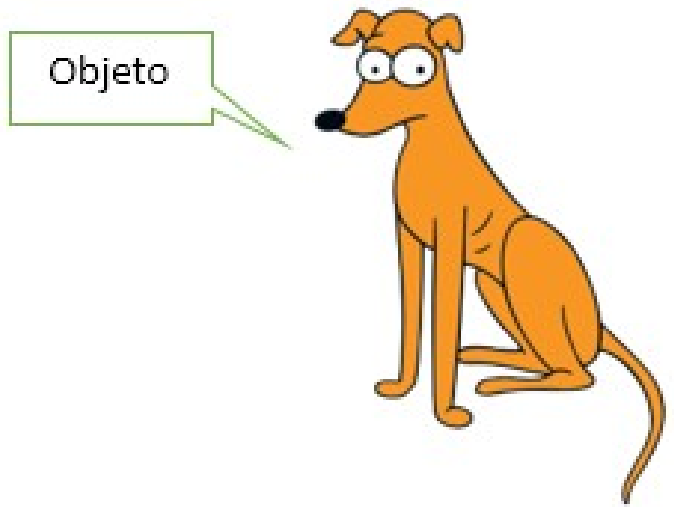
- Un objeto es una unidad dentro de un programa que tiene un estado, y un comportamiento. Es decir, tiene una serie de datos almacenados y tareas que realiza con esos datos en el tiempo de ejecución.
- Como vimos anteriormente, los objetos tiene propiedades y métodos.
- En lo que va del curso hemos trabajado con varios objetos, (console, window, Array, entre otros.)
- Array.forEach() es un método del objeto Array
- Array.length es una propiedad (o atributo) del objeto Array

Objetos

- Estos objetos son objetos que vienen con el lenguaje (javascript en este caso), y el programador solo interactúa con ellos.

- En cambio, nosotros podemos crear nuestros propios objetos (como ya lo hemos haciendo).
- Este es el pilar fundamental de este paradigma.

Representación de algo real en Objetos de POO



Atributos:	Métodos:
Nombre	Ladrando
Color	Oliendo
Raza	Buscando
Hambriento	Meneando la cola

El objeto de un Perro:

- Del ejemplo anterior tendríamos un objeto con las siguiente propiedades y metodos:
- Perro.nombre // aquí podemos guardar un String del nombre
- Perro.color
- Perro.raza
- Perro.hambriento // aquí podemos guardar un booleano
- Perro.ladrar()
- Perro.menearLaCola()

Clases

- En la POO, las clases son “plantillas” en donde definimos la estructura de nuestros objetos.
- Hasta el momento hemos creado objetos independientes y únicos, como el siguiente:
- const perro = {
    - nombre: “Tom”,
    - ladrar: () => { }
- };

Esta forma de declarar objetos no nos permite tener una “plantilla” reutilizable para crear varios objetos de este tipo.

Clases

- Si quisiera crear varios “perros”, puedo crear todos los objetos del tipo perro que quiera, a partir de mi plantilla, o **clase**
- A cada uno de estos objetos creados a partir de una clase, se les llama **instancias**.

- Hasta el momento vimos que podíamos crear nuestro objeto array, instanciando la clase Array.
- Esto lo hacíamos con el operador **new**:
- const unArray = new Array(“a”, “b”, “c”)

¿Como creo mis propias clases?

- Para crear una clase, usamos la palabra reservada class, seguida del nombre de la clase. Luego, debemos abrir un bloque, en el cual estará contenida toda la definición de la clase.
- Cada vez que vayamos a crear una clase, lo haremos con la primer letra de la palabra en mayúscula (esto es un estándar)

```
class Perro {
    // contenido de mi clase
}
```

¿Cómo defino las propiedades (o atributos) de mi clase?

- Para esto, vamos a utilizar un método especial llamado **constructor**.
- constructor(propiedad1, propiedad2, propiedadN) {

```
    this.propiedad1 = propiedad1;
    this.propiedad2 = propiedad2;
    this.propiedadN = propiedadN;
}
```

Operador this dentro de una clase:

- Al usar **this** dentro del constructor, le estamos indicando a javascript que vamos a hacer referencia a una propiedad de nuestra clase.
- Lo usaremos para definir y asignar valores a las propiedades de nuestra clase.
-

¿Cómo defino los métodos de mi clase?

- De la misma forma que definimos funciones, pero sin la necesidad de usar la palabra reservada function.

- ```
  class Perro {
      ladrar() {
          alert("Guau Guau!");
      }
  }
  ```
- De esta forma creamos un método llamado ladrar
  ```

Ejemplo completo de la clase Perro:

```
class Perro {

    // Constructor de mi clase Perro:
    constructor(nombre, color, raza, hambriento) {
        this.nombre = nombre;
        this.color = color;
        this.raza = raza;
        this.hambriento = hambriento;
    }
    // Métodos de mi clase:

    ladrar() {
        console.log("GUAU GUAU");
    }

    moverLaCola() {
        console.log("Moviendo la cola...");
    }

}
```

Creando objetos (o instancias de mi clase)

- Como ya vimos, usamos la palabra reservada **new**, que llamará al constructor de nuestra clase.
- Podemos pasarle como parámetros los datos de nuestro objeto, o podemos no pasarle ningún parámetro.
- Si no le pasamos parámetros al constructor, inicializará las propiedades de nuestro objeto en **undefinded**.

```
// Creo un objeto (o instancia de la clase)
const tom = new Perro("Tom", "Negro", "Caniche", false);

console.log(tom.nombre) // imprime "Tom"

// creo otro objeto:
const dog = new Perro();

console.log(dog.nombre) // imprime undefined
```

Llamando a los métodos de mi objeto y modificar valores:

```
// instancia de mi clase (o Objeto Perro)
const pluto = new Perro("Pluto", "Blanco", null, true);

// Llamo a un metodo de mi objeto:
pluto.ladrar();

// modifico una propiedad de mi perro:
pluto.hambriento = true;
```

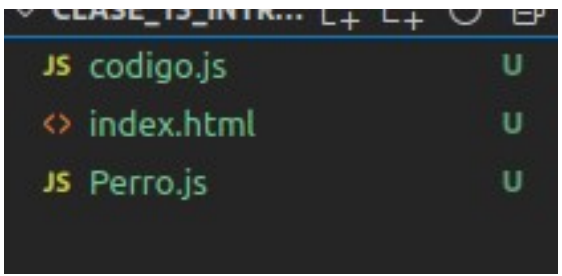
Otra forma de llamar al constructor

- En lugar de pasarle como parámetro los valores de las propiedades (en orden), puedo indicar el nombre de la propiedad.
- Esto me sirve si yo no se el orden de las propiedades declaradas en el constructor

```
const p = new Perro(nombre = "p", color = "rojo", raza = "ovejero", hambriento = true);
```

Organización de nuestro código

- Para tener un código mas organizado, crearemos un archivo .js por cada clase que definamos.
- Luego, tendremos la lógica de nuestro programa en otro archivo js.
- Lo haremos de la siguiente forma:



En codigo js tendremos la lógica

Perro.js tendrá la definición de nuestra clase

index.html será nuestro html (como siempre)

Incluir varios js en nuestro html:

- En este caso, como en codigo.js vamos a crear instancias de nuestra clase perro, entonces necesitamos importar antes nuestra clase Perro.
- Lo haremos de esta forma:

```
</head>
<body>
    <script src="Perro.js"></script>
    <script src="codigo.js"></script>
</body>
</html>
```

Ejemplo de la lógica de Codigo.js:

```
1
2   const miPerro = new Perro();
3   miPerro.nombre = prompt("Ingresar nombre del perro");
4   miPerro.color = prompt("Ingresar color del perro");
5   miPerro.hambriento = false;
6
7   if (! miPerro.hambriento) {
8       // si no esta hambriento, moverá la cola
9       miPerro.moverLaCola()
10  }
```

Algunas ventajas de la POO:

- Ayuda mucho en los sistemas grandes, ya que en vez de pensar en funciones, pensamos en relaciones o interacciones de los diferentes componentes del sistema.
- Nos permite abstraer nuestro código.
- Podemos reutilizar código.
- Sirve bastante a la hora de interactuar con bases de datos.
- Fácil de mantener.

Bibliografía consultada:

- https://i.pinimg.com/originals/a0/02/a2/a002a20d75a7ed6db201d5c2f4e1f715.jpg
- https://profile.es/blog/que-es-la-programacion-orientada-a-objetos/
- https://www.kikopalomares.com/blog/que-es-un-objeto-en-programacion
- http://cv.uoc.edu/annotation/cb7a0462407a23d1f3fc46cb1d4e01f8/645413/PID_00249622/PID_00249622.html
- https://i.pinimg.com/originals/cb/aa/89/cbaa89a3c6b9e92f8b0ef485cdfd53f5.png
- https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Working_with_Objects