¿En qué paradigma de programación nos paramos?

> *"Lo importante no es escuchar lo que se dice, sino averiguar qué es lo que se piensa."*
>
> Juan Donoso Cortés — Escritor

Photo by Artem Sapegin on Unsplash

Imagina que estás por reunirte con amigos/as, por ejemplo, en un cafetería. Podrías buscar la dirección del lugar en Google Maps para saber cómo puedes llegar hasta allí. La aplicación ofrece distintos caminos según el medio de transporte que vayas a utilizar y calcula el tiempo que demorarías en hacerlo. Más rápida o más lentamente, todas las opciones llevarán al mismo lugar. El recorrido que elijas dependerá de tus preferencias y posibilidades.

En tu tarea como developer te encontrarás con este tipo de elecciones sobre qué camino elegir ante los problemas que se te presenten constantemente. ¡Podría decirse que los/as developers somos expertos/as resolvedores/as de problemas! Pero ¿en qué nos basamos para resolverlos?

En programación contamos con diferentes maneras para hallar la solución a los problemas. Para eso, ¡nos servimos de los denominados ¡paradigmas de programación!

Los paradigmas de programación son marcos conceptuales, es decir, **estilos definidos y consensuados para programar.** Nuestra elección va a depender del problema y de la solución que busquemos.

Uno de ellos es el paradigma de programación estructurado, que se utiliza históricamente en lenguajes conocidos como Pascal o Ad. Tal como su nombre lo indica, tiene una estructura que, para funcionar eficientemente, sigue una lógica, la cual será ejecutada paso a paso y leída en secuencia, de arriba hacia abajo.

Otro paradigma es el de la programación orientada a objetos (POO). PHP, Python o Java son algunos ejemplos de lenguajes concebidos bajo este paradigma. Cuando programamos orientado a objetos, lo hacemos mediante clases, objetos, propiedades y métodos. Según el artículo "Qué es la programación orientada a objetos", este paradigma responde a una lógica *"más cercana a como expresaríamos las cosas en la vida real que otros tipos de programación."* En POO creamos objetos que tengan "vida" propia. Por ejemplo, si tenemos que cocinar tendremos un objeto para cada ingrediente, un objeto cocina y por último un objeto persona. Cada objeto tendrá sus características así como también la indicación de las acciones que puede realizar.

Podríamos adelantarnos y decir que nos encontramos en el paradigma de programación orientada a eventos. Aquí, la estructura y la ejecución de las instrucciones van determinadas por los eventos o acciones que ocurren en el sistema, acciones de los/as usuarios/as o del propio sistema. Aquí la programación está asociada a la ocurrencia de algún evento como por ejemplo un click en el mouse, presionar una tecla cualquiera o almacenar un dato en una base de datos.

Hasta aquí cuentas con una cierta cantidad de conocimientos, y en esta bitácora ¡sumaremos algunos nuevos conceptos!

La programación y las cosas

Hector Ivan Patricio Moreno (líder en tecnología) en el capítulo 4 "Estructuras de Datos: Objetos y Arreglos" en GitHub dice, metafóricamente, que crear programas es como construir una casa. Gracias al paso por nuestras bitácoras anteriores, hasta el momento contamos con números, booleanos y cadenas que son los ladrillos para sentar nuestros cimientos, es decir, la estructura de datos. Pero necesitamos más materiales: ahora vamos dar un paso más para acercarnos a lo que hace falta para construir programas.

Para construir estructuras más complejas, los objetos que permiten agrupar valores, incluyendo otros objetos. ¡Conócelos!

Un objeto es una entidad que puede realizar acciones y tiene algunas características:

- **Las acciones que realiza son** métodos. Son exactamente iguales que las funciones y estarán disponibles dentro del objeto.
- **Las características individuales son** propiedades. Son iguales a las variables y, al igual que los métodos, vivirán dentro del objeto.

JavaScript es un lenguaje orientado a objetos —aunque a lo largo de la carrera también verás de qué manera influye en la orientación a eventos.

Como developers nuestro objetivo es identificar todas las propiedades y métodos en común que tienen los objetos independientemente de su valor.

Hagamos juntos/as el siguiente ejercicio para entender conceptualmente cómo trabajar con objetos:

1. Genera una lista con todas las **características** que crees pueden tener los aviones, como por ejemplo, la cantidad de asientos.
2. Genera una lista con todas las **acciones** que puede realizar un avión, como, por ejemplo, despegar.

Esta es nuestra lista de características:

- Asientos
- Bodega
- Peso
- Salidas de emergencia
- Longitud
- Alto
- Envergadura (Cuánto mide desde un ala a la otra)
- Motores
- Alerones
- Baños
- Velocidad máxima
- Tanque de combustible
- Pisos

Aquí, la lista de acciones:

- Despegar
- Aterrizar
- Frenar
- Acelerar
- Planear
- Flotar

¿Alguna de las características o acciones que enumeramos no está en tu lista?

¡No te preocupes! Esto es parte del análisis de requisitos que debes hacer antes de comenzar a programar. Saber cuál es el objetivo de tu desarrollo antes de comenzar es fundamental para que tu producto sea una solución funcional.

Ahora, como puedes ver en la lista de características (propiedades), no tenemos ningún valor. Simplemente estamos describiendo puntos en común que tienen los aviones.

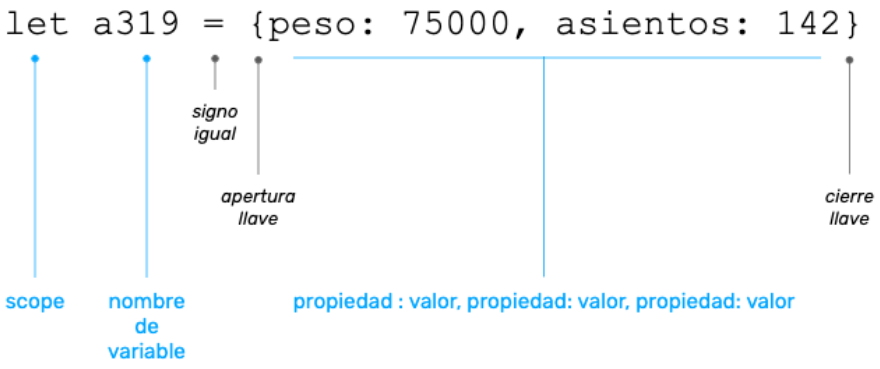
Si sumamos la lista de características a la de acciones, tendremos un esqueleto de aviones, algo que en programación llamamos clase. Una clase por sí sola no tiene valores. Las clases no darán el valor de una propiedad, pero estaremos seguros/as de que todos los objetos construidos a partir de esa clase contarán con esa propiedad.

Por ejemplo, un A319 (clase de avión) tiene un peso de 75.000 kilos (propiedad) mientras que un A340 pesa 380.000 kilos. Por ende, al contar con esta información, el próximo paso sería construir un objeto para cada avión a partir de la clase 'aviones'.

Componentes para nuestro primer objeto

JavaScript permite crear objetos de manera sencilla. Al utilizar las llaves { } puedes crear tu primer objeto, sin la necesidad de programar una clase.

La sintaxis que debes usar es la siguiente:



Es importante que sepas que las propiedades no son ningún tipo de dato, por lo tanto, no van entre comillas (al igual que las variables). Luego de los dos puntos (:) va el valor que tendrá la propiedad. Este sí es un tipo de dato por lo que no será lo mismo 75000 que "75000".

Para **acceder** a las propiedades lo haremos con punto:

```
console.log(a319.peso); //imprime 75000
```

Si necesitas **declarar** que un método trabaja de la misma manera, la forma será `propiedad : valor` siendo esta vez el valor un tipo de dato function:

```
let a340 = {peso: 380000, asientos: 359, msj_bienvenida :
function(){ return "Bienvenido a Airbus 380"} }
```

Finalmente, puedes **ejecutar** el método msj_bienvenida:

```
console.log(a340.msj_bienvenida());       //       imprime
"Bienvenido a Airbus 380"
```

Como puedes observar, el objeto `a340` tiene un método para dar un mensaje de bienvenida, mientras que el `a319` no lo tiene. Por lo tanto, si intentas ejecutar el saludo en el objeto `a319`, te producirá "`undefined`". Por suerte es solamente un saludo y no el método para aterrizar :-)

Para asegurarte de que todos los objetos tengan los mismos métodos, puedes crear una clase con todas las acciones disponibles y crear cada uno de los objetos a partir de ella.

¡Manos a la obra!

Aquí van 3 aclaraciones importantes antes de empezar:

- La palabra reservada para declarar una clase es `class`. ¡Chequea la documentación disponible!
- No es necesario que tenga parámetros en su definición, por lo tanto, no necesita los paréntesis habituales.
- Opcional, pero como práctica habitual en la POO, el nombre de la clase comienza con una letra mayúscula.

¡Ahora sí! Empecemos:

1. **Crea la clase**

```
class Aviones {


}
```

2. **Crea tu primer objeto** a partir de aviones: `a319 = new Aviones`

3. **Crea tu primera propiedad** y asígnale un valor: `a319.peso = 75000;`

Hasta aquí nada nuevo, ¿cierto? Comienza a crearle métodos a tu clase:

```
class Aviones {
     saludo(){
          return "Hola a todos";
     }
}

a319 = new Aviones
console.log(a319.saludo()) //Imprime "Hola a todos"
a340 = new Aviones
console.log(a340.saludo()) //Imprime "Hola a todos"
```

Es muy común tener que acceder a alguna propiedad desde adentro del objeto. Por ejemplo, si a tu saludo quisieras añadirle el nombre del avión, deberías acceder al nombre para concatenarlo en el saludo. La palabra reservada `this` te permitirá acceder a las propiedades en la definición de la clase:

```
class Aviones {
     saludo(){
          return "Bienvenidos al avión " + this.nombre;
```

```
    }
}

a319 = new Aviones
a319.nombre = "A319";
console.log(a319.saludo()); //Imprime "Bienvenidos al avión A319"
a340 = new Aviones
a340.nombre = "A340";
console.log(a340.saludo()); //Imprime "Bienvenidos al avión A340"
```

Observa el código fuente que te mostramos a continuación. Allí encontrarás cómo creamos una clase "Aviones" con las funcionalidades de acelerar y frenar. Luego, creamos 2 objetos. Uno para el avión A319 y otro para A340. Este último tiene más poder de aceleración y frenado y al ejecutar el método "acelerar", la velocidad del avión se incrementará más rápidamente.

Código Fuente

Cierre

Llegas a este encuentro con una posición clara sobre cómo vas a programar. No olvides revisar las tarjetas para profundizar tus conocimientos en objetos ya que te acompañarán el resto de la carrera. ¡Ya estás preparado/a para el siguiente!

¡Prepárate para el próximo encuentro!

Profundiza

Programación orientada a objetos

Constructores, Setters y Getters

Challenge

📝 Toma alguna de las propiedades y métodos que creaste en tu clase “Aviones” y genera un objeto a partir de la definición de tu clase.