This en métodos de objetos

CÁTEDRA: CONSTRUCCIÓN DE INTERFACES DE USUARIO

Para Pensar ...

¿ Qué texto se imprime en pantalla? Elegí una opción antes de probarlo en la compu

```
let nombre="Jose";

const persona = {
    nombre: 'Alicia',
    saludar: function() {
        console.log(`Hola, mi nombre es ${nombre}`);
    }
}

persona.saludar();
```

```
Hola, mi nombre es Alicia
>
```

ó

```
Hola, mi nombre es Jose
>
```

?

Respuesta

Ahora sí podes copiar el código y probarlo. El resultado que se muestra es:

Para Pensar …

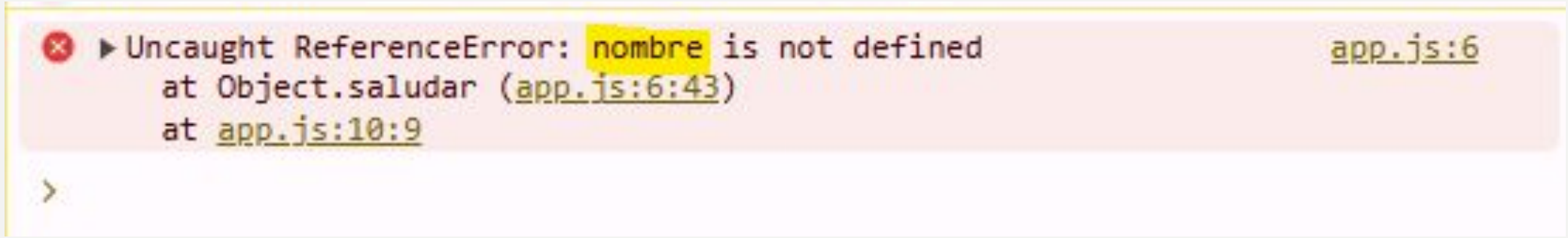
¿Y si sacamos la variable "const nombre='Jose'"? Dedicale unos minutos a pensarlo antes de probarlo, ¿que crees que se mostrará en pantalla?

```
const persona = {
    nombre: 'Alicia',
    saludar: function() {
        console.log(`Hola, mi nombre es ${nombre}`);
    }
}

persona.saludar();
```

Respuesta

Ahora sí podes copiar el código y probarlo. El resultado nos indica un error en el "nombre" ya que no lo encuentra definido:



¿Qué pasó?

En ambos casos cuando se ejecuta la funcion “saludar” se está intentando acceder a una variable ‘nombre’ dentro del scope actual. Algo asi:

```
persona.saludar() -> console.log(`Hola, mi nombre es ${nombre}`);
```

En el primer caso esa variable ‘nombre’ existe en el scope actual y tiene el valor “Jose”.
En cambio en el segundo caso no existe ninguna variable “nombre” que se pueda acceder y por eso devolvió error.

¿Como hacemos para acceder a la variable Nombre del Objeto?

This

Dentro de un método de objeto en JavaScript, **this** hace referencia al **objeto al que pertenece el método.**

```
const persona = {
    nombre: 'Alicia',
    saludar: function() {
        console.log(`Hola, mi nombre es ${this.nombre}`);
    }
}

persona.saludar();
```

Un método de objeto es simplemente una función que es una propiedad de un objeto. Cuando invocas este método, **this** se refiere al objeto que lo contiene.

Por qué this se refiere al objeto

Podemos imaginarlo de esta manera. Cuando invocamos a un método del objeto "persona" , ejemplo:

```
persona.saludar();
```

JavaScript establece **this** en el contexto de ejecución de **saludar**.

```
saludar: function() {
        // this = {nombre:"Alicia"};   (algo asi)
         console.log(`Hola, mi nombre es ${this.nombre}`);
    }
```

Así, dentro de **saludar**, **this** hace referencia a **persona** con todas sus propiedades

Constructores

CÁTEDRA: CONSTRUCCIÓN DE INTERFACES DE USUARIO

Formas de declarar Objetos...

1. Declaro la variable y le asigno un objeto que defino entre { } indicando cada una de sus propiedades y métodos:

```
const empleado1= {
    nombre:"Jose",
    apellido: "Perez",
    saludar: function() {
      console.log(`Hola soy ${this.nombre} ${this.apellido}`);
    }};
```

1. Declaro la varibale como tipo "objeto", y luego voy agregando las propiedades y métodos que quiero:

```
const empleado2= new Object();
empleado2.nombre = "Alicia";
empleado2["apellido"]= "Romero";   //otra opción con los corchetes
empleado2.funcion1 = function(){
    console.log(`Hola soy ${this.nombre} y ${this.apellido}`);
}
```

Formas de declarar Objetos...

¿Qué pasa si luego creo la persona3 y le cambio el nombre a las propiedades?

```
const persona3= {
    nomb:"Jose",
    ape: "Perez",
    saludar: function() {
      console.log(`Hola soy ${nombre} ${objeto1.apellido}`);
    }};
```

¿Y si necesito crear mas personas? Tengo que definir todas sus propiedades y metodos cada vez, y estar atento a los nombres que le pongo

Función Constructora...

La función constructora nos permite crear un "molde" del objeto. Definir la "forma" que va a tener mi objeto una sola vez:

```
function Empleado(nombre,edad,cargo){
    this.nombre=nombre;
    this.edad=edad;
    this.cargo=cargo;
    this.presentarse =  function(){
        console.log(`Hola , mi nombre es ${this.nombre} y tengo ${this.edad}. Mi rol es ${this.cargo} `);
    }}
```

Cuando quiera definir un empleado nuevo, voy a crear una "instancia" del objeto Empleado:

```
let empleado1 = new Empleado("Jose","47","Programador");
let empleado2 = new Empleado("Alicia","38","PO");
```

This y Constructores...

Cuando creamos una nueva instancia:

```
let empleado1 = new Empleado("Jose","47","Programador");
```

Podemos imaginarlo de esta manera, como que se crea el objeto this con los valores de cada propiedad. Por eso luego puedo accederlo como "this.propiedad":

```
function Empleado(nombre,edad,cargo){
    //this =  {nombre:"Jose",edad:47,cargo:"Programador"}
    this.nombre=nombre;
    this.edad=edad;
    this.cargo=cargo;
    this.presentar =  function(){
        console.log(`Hola , mi nombre es ${this.nombre} y tengo ${edad}. Mi rol es ${this.cargo} `);
    }
}
```

Clases

CÁTEDRA: CONSTRUCCIÓN DE INTERFACES DE USUARIO

Clases

Desde ES6 (2015) se implementó el uso de clases para la programación orientada a objetos. De esta forma nuestro objeto quedaría:

```
class Empleado{
    constructor(nombre,edad,cargo){
        this.nombre=nombre;
        this.edad=edad;
        this.cargo=cargo;
    }
}
```

Dentro de la clase se debe definir su **constructor**. El cual se invoca de la misma forma que antes:

```
let empleado2 = new Empleado("Alicia","38","PO");
```

Get y Set

Con el Get y Set definimos a qué se puede acceder de nuestro objeto. El Set nos permite "setear" nuevos valores, y el get nos permite obtener información del objeto:

```
class Empleado{
    constructor(nombre,edad,cargo){
        this.nombre=nombre;
        this.edad=edad;
        this.cargo=cargo;
    }
    get saludo(){
        return `Hola , mi nombre es ${this.nombre} y tengo ${this.edad}. Mi rol es ${this.cargo} `;
    }
    set nuevoCargo(nuevoCargo){
        this.cargo=nuevoCargo;
    }
}
```

Get y Set

Con el Set puedo asignar un nuevo valor al "cargo" de mi instancia "empleado1"

```
let empleado1 = new Empleado("Jose","47","Programador");
console.log(empleado1);
```

```
▸ Empleado {nombre: 'Jose', edad: '47', cargo: 'Programador'}
```

```
let empleado1 = new Empleado("Jose","47","Programador");
empleado1.nuevoCargo="Lider Tecnico"
console.log(empleado1);
```

```
▸ Empleado {nombre: 'Jose', edad: '47', cargo: 'Lider Tecnico'}
```

Get y Set

Con el Get puedo retornar información sobre la instancia de mi objeto:

```
let empleado1 = new Empleado("Jose","47","Programador");
console.log(empleado1.saludo);
```

```
Hola , mi nombre es Jose y tengo 47. Mi rol es Programador
```