Programación con JavaScript I

Objetos y arreglos

Introducción

Para JavaScript, prácticamente todo lo que se utiliza es un objeto, es por ello por lo que es uno de los conceptos más importantes de este lenguaje. En esta experiencia educativa aprenderás que existen algunas diferencias con respecto a los objetos de otros lenguajes, así que estos se revisarán partiendo de los conceptos más básicos para que te vayas familiarizando con el tema.

En este tema también aprenderás las características, declaración, uso y manipulación de los *arrays*, unos objetos que son como las listas, pero que son mucho más potentes y que podrás utilizar dentro de tus programas, ya verás que te facilitarán mucho el desarrollo.

Explicación

Objetos

Existen dos grandes tipos de datos en JavaScript: los primitivos y los objetos. Algunas de las características de los primeros es que hay seis tipos definidos: string, number, bigint, boolean, undefined y symbol. Los primitivos son inmutables, es decir, puedes modificar una copia o instancia, pero no se puede modificar el valor original.

Los objetos, según los define Kantor (2022), son la estructura de datos fundamental de JavaScript. Intuitivamente, un objeto representa una tabla que relaciona cadenas con valores, o sea, los objetos permiten agrupar valores para construir estructuras más complejas (valores primitivos u otros objetos). Un objeto es una colección desordenada de propiedades, cada una de estas tiene un nombre y un valor.

En otros lenguajes se utiliza la palabra new para crear un objeto, y aunque en JavaScript también se puede hacer, esta notación se pospondrá hasta que se revise el tema de programación orientada a objetos. En JavaScript es preferible utilizar la notación literal, esto es, se utiliza una forma abreviada para crear objetos sin la necesidad de utilizar la palabra new, así como en el siguiente ejemplo:

```
const objeto={}; //Esto es un objeto vacío
```

El anterior sería un ejemplo de un objeto vacío. Un mejor ejemplo sería el objeto jugador:

```
const jugador={
  nombre:"Ryu" ,
  poder:10,
  vida:99,
};
```

Todo objeto tiene **propiedades**, las cuales se pueden ver representadas por estas variables. Debido a que dichas variables están dentro del objeto, se puede decir que están encapsuladas, por lo que se puede acceder a ellas de dos maneras:

Usando la notación con puntos (preferida):

```
console. log ( jugador.nombre) ; // muestra "Ryu"
console. log ( jugador.life) ; // muestra "99"
```

Explicación

O a través de la notación con corchetes:

```
console. log ( jugador.["nombre"] ) ; // muestra "Ryu"
console. log ( jugador ["life"]) ; // muestra 99
```

Cualquiera de las notaciones es correcta, por lo que puedes utilizar la que más te guste o con la que más te acomodes. Sin embargo, es importante mencionar que la notación con corchetes en otros lenguajes se conoce como diccionarios o arrays asociativos.

Una vez que se tiene definido un objeto, se le puede modificar agregando propiedades. Por ejemplo, se toma como base el objeto jugador que se creó anteriormente, pero ahora se va a crear de diferente manera, es decir, primero se crea el objeto y después se pueden agregar las propiedades:

```
const jugador={};

jugador.nombre="Ryu";
jugador.poder=10;
jugador.vida=99;
```

Otra característica de los objetos es que puedes poner una función (o una variable que contiene una función definida), con esto se tendría lo que se conoce como método de un objeto, por ejemplo:

```
const usuario ={
nombre: "Ryu",
hablar: function(){return "Hola";}
};


usuario.nombre; //Esta es una variable(propiedad) que devuelve "Ryu"
usuario.hablar(); //Esta es una función(método), que se ejecuta y devuelve "Hola"
```

Explicación

Cuando se genera una variable tipo "**objeto**", o variable del tipo object, esta hereda una serie de métodos que existen para ese tipo de variables. Por ejemplo, el método .toString(), el cual intenta convertir la información del objeto en una cadena (string). Se puede representar más gráficamente con el siguiente ejemplo:

```
const objeto={};
objeto.toString(); //Devuelve "[object Object]" (representación textual de un objeto genérico)
```

Observa que en ningún momento se ha declarado el método toString, no obstante, no muestra un error porque ese método fue heredado.

Cuando se declara una variable de un determinado tipo de datos, también será implícitamente del tipo object, por lo tanto, esa variable tiene lo siguiente:

- **Métodos que se le implementen por parte del programador.**
- **Métodos heredados de su tipo de dato definido explícitamente.**
- **Métodos heredados desde el tipo object.**

Por ejemplo:

```
const numero= 45.2;
numero.toString(); //Devuelve "45.2" (Método de variables de tipo object)
numero.toLocaleString(); //Devuelve "45.2" (Método de variables de tipo object)
numero.toFixed(3); //Devuelve "45.200" (Método de variables de tipo numérico)
```

En el cual se realizan las siguientes acciones:

- **Se define una variable de tipo numérico.**
- **Se ejecuta el método .toString() para mostrar el contenido como texto.**
- **Con el método .toLocaleString() se presenta igual, pero toma las variables locales del sistema.**
- **Con el método .toFixed se le da un formato numérico, pero con tres decimales.**

Explicación

Es importante mencionar que se pueden redefinir algunas funciones heredadas para que tengan un comportamiento diferente al esperado, por ejemplo, el .toString() se puede redefinir de la siguiente manera:

```
const jugador={
nombre:"Ryu", //Nombre del jugador
vida:4, //Cantidad de vida actual
totalVida: 6, //Maximo de vida posible
toString: function(){
return `${this.nombre} (${this.vida}/${this.totalVida})`;
}
};
```

En este to.String() se está concatenando una cadena a un objeto:

```
console.log("Mi jugador favorito es "+ jugador); //"Mi jugador favorito es Ryu (4/6)"
```

Como el objeto jugador no es del tipo string, se llama intrínsecamente al método .toString() que lo convierte en cadena para presentarlo en pantalla.

Luego de haber revisado las características de los primitivos y los objetos, es necesario comprender que las principales diferencias entre unos y otros es que los primitivos siempre pasan por valor, mientras que los objetos se copian por referencia. Como lo explica Flanagan (2020), los tipos primitivos son inmutables, mientras que, por otro lado, lo único inmutable en un objeto es su referencia, es decir, que sí se puede modificar su valor.

A continuación, se expone un ejemplo que te ayudará a comprender de mejor manera estas afirmaciones:

Cuando copias un objeto de tipo primitivo:

```
let animal = 'perro';
let mascota = animal;
```

Cuando se intenta cambiar el valor de la variable animal, observa lo que sucede con la variable mascota:

```
animal = 'dinosaurio';
console.log(mascota); //la salida es perro
```

Explicación

Lo anterior ocurre debido a que cuando se trabaja con tipos primitivos, la asignación que se ha realizado al principio es por valor y no por referencia. Esto es, si bien es cierto que se puede asignar el valor de una variable a otra, ambas son independientes una de la otra.

Ahora, observa el comportamiento cuando se copia un objeto:

```
let animal = {
especie: 'perro'
};


let mascota = animal;


animal.especie = 'dinosaurio';
console.log mascota.especie); //la salida es "dinosaurio"
```

El comportamiento anterior se presenta debido a que la variable mascota está apuntando al mismo objeto que la variable animal, esto es, la asignación se realizó por referencia. Dicho en otras palabras, se tiene un solo objeto con dos referencias.

Arreglos

Haverbeke (2018) define un arreglo (array) como una lista de elementos entre corchetes, que es utilizado en todos los lenguajes de programación:

```
let arregloDeNumeros = [2, 3, 5, 7, 11];
let arregloDeCadenas = ["mango","manzana","uva","pera"];
```

Como se puede ver en los ejemplos, los elementos del arreglo están entre corchetes y separados por una coma (,), pueden ser elementos de cualquier tipo y puede haber más de un tipo en un solo arreglo.

```
const ArrayTipoMixto = [100, true, 'Frontend', {}];
```

Explicación

Para acceder a los elementos que se encuentran dentro de un arreglo se utiliza el índice del elemento. El índice comienza en 0. Los siguientes son ejemplos para acceder a algunos elementos del arreglo anterior:

```
let primero = ArrayTipoMixto[0]; //100
let ultimo = ArrayTipoMixto[ArrayTipoMixto.length -1]; //{}
```

La longitud del arreglo está determinada por el número de elementos, por tanto, la longitud del ejemplo anterior es igual a 4. En el ejemplo anterior, para obtener el último elemento se utilizó el método **length** y el valor obtenido es 4, pero como los índices comienzan en 0, se le tiene que restar 1 para obtener el valor del elemento con posición 4, pero con índice 3.

En algunas ocasiones no solo se requiere obtener el valor de una variable en determinada posición, sino que se necesitan obtener todos los valores del arreglo. Para ello se puede utilizar un ciclo for, por ejemplo:

```
const numeros = [1, 2, 3, 4, 5];

for (i = 0; i < numeros.length; i++) {
console.log(numeros[i]);
}
//1
//2
//3
//4
//5
```

Otro método para recorrer un arreglo es el método forEach (), el cual requiere de una función (*callback* o anónima):

```
numeros.forEach(function(numero) {
console.log(numero);
});
```

Explicación

El tercer método es recorrer el arreglo con una función de flecha y se vería de la siguiente manera:

```
numeros.forEach(numero => console.log(numero));
```

Existen algunos métodos que modifican a los arreglos. El siguiente arreglo se utiliza como ejemplo:

```
let fruta = ["mango", "manzana", "uva", "pera"];
```

- *Push*: sirve para añadir un elemento a un arreglo.

```
let nuevaLongitud = fruta.push('naranja'); // Añade "naranja" al final
console.log(fruta);
// ["mango","manzana","uva","pera","naranja"]
```

- *Shift:* se utiliza para eliminar el primer elemento.

```
let primer = fruta.shift(); //Elimina "mango" del inicio
console.log(fruta);
// ["manzana","uva","pera","naranja"]
```

- *Pop:* sirve para eliminar el último elemento.

```
let ultimo = fruta.pop(); // Elimina "naranja" del final
console.log(fruta);
// ["manzana","uva","pera"]
```

Explicación

- *Unshift:* se utiliza para añadir un elemento al inicio del arreglo.

```
let principio = fruta.unshift('fresa'); // Añade "fresa" al principio
console.log(fruta);
// ["fresa","manzana","uva","pera"]
```

- *Splice:* sirve para eliminar un elemento mediante su posición.

```
let elementoEliminado = fruta.splice(1, 1);
console.log(fruta);
// ["fresa","uva","pera"]
```

Cierre

En este tema aprendiste que todos los objetos en JavaScript tienen sus propias características y funciones, así como su diferencia con otros tipos de datos. En este lenguaje todo es un objeto, por tanto, se aplican todos los conceptos relacionados a los objetos como la herencia. Aprendiste, además, que los arreglos solo son agrupaciones de elementos de distintos tipos y que existen varias maneras para recorrerlos y manipularlos.

Ambos temas son de suma importancia en tu formación y son pilares en tu camino hacia convertirte en un programador.

Referencias bibliográficas

- Flanagan, D. (2020). *JavaScript: the Definitive Guide* (7a ed.). Estados Unidos: O'Reilly Media, Inc.
- Haverbeke, M. (2018). *ELOQUENT JAVASCRIPT* (3ª ed.). Estados Unidos: No Starch Press.
- Kantor, I. (s.f.). *El lenguaje JavaScript.* Recuperado de https://es.javascript.info/js

Para saber más

Lecturas

- LenguajeJS. (s.f.). *¿Que son los objetos?* Recuperado de https://lenguajejs.com/javascript/objetos/que-son/
- MDN. (s.f.). *Array.* Recuperado de https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array

Videos

- Emprinnos. (2021, 7 de enero). *{Valor vs Referencia} Fundamentos para programación con JavaScript [Archivo de video].* Recuperado de https://www.youtube.com/watch?v=7N3nDVIBmrE
- Jonmircha. (2020, 21 de febrero). *Curso JavaScript: 11. Arreglos (Arrays) - #jonmircha* [Archivo de video]. Recuperado de https://www.youtube.com/watch?v=o3c8cW73weQ
- Jonmircha. (2020, 24 de febrero). *Curso JavaScript: 12. Objetos - #jonmircha* [Archivo de video]. Recuperado de https://www.youtube.com/watch?v=4xig5UPRC00

Checkpoints

Asegúrate de:

- Desarrollar código, creando nuevas formas de usar objetos, para practicar los conocimientos que vas adquiriendo.
- Comprender la diferencia entre paso por valor y paso por referencia de los valores.
- Comprender la diferencia entre un tipo de dato primitivo y de un objeto.
- Identificar las operaciones que se pueden realizar con los arreglos.

Requerimientos técnicos

- Computadora con acceso a Internet.
- Editor de texto.
- Permisos de administrador y/o Git instalado previamente.

Prework

- Leer detenidamente y comprender el material explicado en el tema.
- Practicar todos los ejemplos que se describen en el tema.
- Revisar cada uno de los recursos adicionales que se proponen en este tema.