Programación con JavaScript I

Programación funcional

Introducción

La **programación funcional**, al igual que la programación orientada a objetos, es un paradigma de la programación, es decir, una forma distinta de ver el código al momento de crear aplicaciones, en donde las funciones se vuelven las protagonistas.

En esta experiencia educativa aprenderás que los paradigmas de programación no son excluyentes, sino que pueden combinarse. La programación funcional no es un concepto nuevo, sin embargo, en la actualidad ha tomado mucha relevancia, pues permite al desarrollador concentrarse en la forma en que se van a hacer las cosas y no en cómo deberían hacerlas.

A continuación, aprenderás algunos conceptos y buenas prácticas en este paradigma de programación.

Explicación

Programación funcional

Un paradigma de la programación es la manera en que se puede afrontar un problema, el cual cuenta con sus propias reglas e instrucciones para ayudar al desarrollador a escribir mejor su código. Eso no quiere decir que un paradigma sea mejor que otro, solamente se dice que cada paradigma te puede ayudar a resolver un problema específico.

Los siguientes son algunos de los tipos de paradigmas de programación que existen:

- Programación imperativa o procedimental: basada en dar instrucciones de cómo hacer las cosas (receta de cocina).
- Programación orientada a objetos: basada en el imperativo, pero encapsula elementos denominados objetos.
- Programación orientada a eventos: la ejecución de los programas va determinada por los sucesos que ocurran en el sistema.
- Programación declarativa: basada en describir el problema, declarando propiedades y reglas que deben cumplirse.
  - Programación funcional: basada en la definición de los predicados y es de corte más matemático.
  - Programación lógica: basada en la definición de relaciones lógicas, está representada por Prolog.
- Programación multiparadigma: se refiere al uso de dos o más paradigmas dentro de un programa.
- Programación reactiva: se basa en la declaración de objetos emisores de eventos asíncronos y objetos que se “suscriben” a los primeros.

La programación funcional es un paradigma de programación declarativa. Un ejemplo de este tipo de programación sería SQL o HTML:

Codigo SQL	Codigo HTML
SELECT * FROM clientes;	

En estos ejemplos no se indica cómo se hace el select o cómo imprimir el div, simplemente se le indica a la aplicación correspondiente lo que va a hacer sin explicarle cómo se debe hacer.

Para ilustrar gráficamente, ve el siguiente ejemplo en el que se desea implementar una función que recibe dos arreglos (**a** y **b**) y debe retornar un nuevo arreglo con todos aquellos elementos de **a** que se encuentren en **b**:

Explicación

En programación orientada a objetos quedaría así:

```
function interseccion(a,b){
    var resultado = [];

    for (var i = 0;  i < a.length; ++) {
        for (var j = 0; j < b.length; j++) {
            if (a [ i ] === b[ j ]) {
                resultado.push(a [ i ]);
                break;
            }
        }
    }
    return resultado.sort ( );
}
```

Utilizando una sintaxis moderna y en programación funcional quedaría de la siguiente

```
const intersection = (a, b) => a. filter (value => b. index0f (value) > -1) .sort ( );
```

Como menciona Flanagan (2020), JavaScript no es un lenguaje de programación funcional como lo pudiera ser Lisp o Haskell, pero el hecho de que JavaScript pueda manipular funciones como objetos significa que se pueden usar técnicas de programación funcional en este lenguaje.

Algunos de los principales beneficios de la programación funcional son la legibilidad, pues se puede eliminar el código repetitivo que no aporta valor al producto, lo que te permitirá obtener un código menos propenso al error, permite realizar test unitarios de manera más fácil, reduce la complejidad, permite que el código sea más modular y sencillo de entender, el código generado es más confiable y permite convertir en predecible su comportamiento.

Cuando se comienza a desarrollar en JavaScript, la mayoría de los desarrolladores ignora que muchas de las funciones que se necesitan ya están integradas en el lenguaje. Esta es una de las razones por las que muchos, cuando ya tienen conocimiento y dominio del lenguaje, prefieren la programación funcional.

Explicación

Antes de continuar profundizando en la PF, es necesario que sepas que en JavaScript existen cuatro tipos de funciones:

Funciones de primera clase: son aquellas que tienen las siguientes características:

- Se pueden asignar a variables.
- Se pueden pasar como parámetro de otras funciones (callbacks).
- Se pueden retornar desde otras funciones.

Por tanto, se puede deducir que para JavaScript todas las funciones pertenecen a este tipo.

```
const hola = () => 4
  console. log("Hola Mundo"); // Hola Mundo
};
```

Funciones callback: pasan a otras funciones como argumento. Estas no se pueden invocar desde fuera, ya que son llamadas cuando la función principal está en marcha. Por ejemplo:

```
["A", "B", "C"] .forEach((e,i) => console.log("i=" , i, "list=", e));
```

En este ejemplo se le está pasando una función callback a la función forEach(), la cual se va a ejecutar para cada elemento del arreglo y en cada iteración los parámetros tendrán un valor específico.

Funciones de orden superior: reciben una función como argumento o devuelven una función.

Métodos de manejo de arreglos, como lo son **.map(), .filter()** y **.reduce()**, se prestan particularmente bien como ejemplos de orden superior, dado que iteran sobre el arreglo y llaman a la función que recibieron para cada uno de los elementos del arreglo. Los siguientes podrían ser ejemplos de funciones de orden superior:

Explicación

```
const arreglo = [1, 2, 3];

const mapeo = arreglo.map(function(elemento){
    return elemento + 1;
}); // mapeo es [2, 3, 4]

const reducido = arreglo.reduce(function(elem1, elem2){
    return elem1 + elem2;
}); // reducido es 6

const filtrado = arreglo.filter(function(elemento){
    return elemento !== 1;
}); // filtrado is [2, 3]

console.log(mapeo);
console.log(reducido);
console.log(filtrado);
```

Funciones asíncronas: normalmente se utilizan cuando se requiere que algo se ejecute en otro momento como resultado de una petición o un evento.

```
const sumar=(x,y) => console.log(x + y);
setTimeout(()=>sumar(2,2),1000);
sumar(4,4);
```

Considerando este ejemplo, primero se genera 8 e instantes después el 4, que son los resultados de sumar(4,4) y sumar(2,2) respectivamente.

Para poder comenzar el desarrollo con la programación funcional, Perez (2021) explica que solo hay que seguir dos reglas:

1. Utilizar funciones puras y aisladas.
2. Evitar estado cambiante (inmutabilidad).

Se consideran una **función pura** cuando:

- Dada la misma entrada (argumentos), siempre devolverá la misma salida (transparencia referencial).
- No tiene ningún efecto secundario observable.

La transparencia referencial se refiere a que el comportamiento de las funciones puras depende solamente de los argumentos pasados a la función, esto es, si se ingresan los mismos datos, la función siempre debe producir el mismo valor de salida. Observa el siguiente ejemplo de funciones puras e impuras:

Explicación

```
/Impura
const tiempo = () => Date.now();
tiempo();//1664171354245
tiempo();//1664171354251

//Pura
const suma= (a,b)=> (a+b);
suma(2,5);// =>7
suma(2,5);// =>7
```

La función tiempo retorna un valor distinto para cada invocación, por otro lado, la función suma siempre regresará el número 7 cuando se le llama con (2,5).

Si se quisiera crear alguna prueba con la función tiempo, sería muy difícil poder predecir el valor de retorno, por tanto, el valor no es verificable y la prueba fallaría.

Respecto al segundo punto, cuando se habla de efectos secundarios, se hace referencia a que no afecta un estado fuera de su ámbito o alcance. Dicho en otras palabras, la función no puede depender de variables ni tampoco mutar las variables que se encuentran declaradas fuera del cuerpo de la función.

```
var variable_global=5

const funcion = (x)=>{
   variable_global = 10; //Cambio de valor
   return x + variable_global;
}

funcion(3); //13
```

Según Escobar (2020), la **inmutabilidad** establece que los objetos no deberían cambiar una vez que se crean, lo que implicaría siempre usar const en variables y crear un nuevo objeto cada vez que se requiera modificarlas. En JavaScript, todas las cadenas son inmutables y esto se debe a que todos los métodos que se pueden aplicar a una cadena necesariamente regresan nuevas cadenas. Por ejemplo:

Explicación

```
const cadena = "Este string es inmutable"
const cadena1 = cadena.slice(5, 11)
const cadena2 = cadena1.toUpperCase()

console.log(cadena) // "Este string es inmutable"
console.log(cadena1) // "string"
console.log(cadena2) // "STRING"
```

Por otro lado, los arreglos son mutables, incluso cuentan con métodos que los pueden modificar *(push, shift, splice)*. Sin embargo, también se puede modificar un arreglo de modo inmutable. Por ejemplo:

```
const arreglo = { a: 1, b: 2, c: 3 }
const nuevoArreglo = { ...arreglo, c: 4, d: 5}
// nuevoArreglo queda { a: 1, b: 2, c: 4, d: 5}
```

Cierre

En esta experiencia educativa aprendiste que la programación funcional, al igual que la POO y la programación procedimental, es un paradigma de programación y que, a diferencia de las anteriores, la PF es declarativa y como tal delega el control de flujo a las funciones.

Es importante señalar que, aunque al principio no lo parezca o te parezca extraño, la programación funcional genera código más conciso y expresivo, fácil de mantener y sirve para atender problemas actuales como la paralelización y cómputo distribuido, entre otros.

Finalmente, aprendiste que, en lo posible, debes evitar reasignar valores a las variables y que, si necesitas hacerlo, lo mejor es crear nuevos objetos.

Referencias bibliográficas

- Flanagan, D. (2020). *JavaScript: the Definitive Guide* (7a ed.). Estados Unidos: O'Reilly Media, Inc.
- Perez, C. (2021). *¿Qué es la programación funcional? Una guía de Javascript para principiantes.* Recuperado de https://www.freecodecamp.org/espanol/news/que-es-la-programacion-funcional-una-guia-de-javascript-para-principiantes/
- Escobar, G. (2020). *Programación funcional.* Recuperado de https://guias.makeitreal.camp/javascript-ii/programacion-funcional

Para saber más

Lecturas

- Delgado, L. (2021). *Programación funcional en JavaScript Explicado.* Recuperado de https://www.freecodecamp.org/espanol/news/programacion-funcional-en-javascript-explicado/#:~:text=La%20programaci%C3%B3n%20funcional%20es%20un,principal%20de%20la%20programaci%C3%B3n%20funcional.

- Montero, L. (2017). *Introducción a programación funcional en JavaScript - Parte 1.* Recuperado de https://medium.com/laboratoria-developers/introducci%C3%B3n-a-la-programaci%C3%B3n-funcional-en-javascript-parte-1-e0b1d0b2142e

- Montero, L. (2018). *Introducción a la programación funcional en JavaScript - Parte 2: Funciones Puras.* Recuperado de https://medium.com/@lupomontero/introducci%C3%B3n-a-la-programaci%C3%B3n-funcional-en-javascript-parte-2-funciones-puras-b99e08c2895d

- Montero, L. (2018). *Introducción a la programación funcional en JavaScript - Parte 3: Composición.* Recuperado de https://medium.com/laboratoria-developers/introducci%C3%B3n-a-la-programaci%C3%B3n-funcional-en-javascript-parte-3-composici%C3%B3n-f82ac871dcfb

Videos

- Nabenik. (2020, 18 de mayo). *Programación funcional en JavaScript* [Archivo de video]. Recuperado de https://www.youtube.com/watch?v=5epCHMnLuZk

- HolaMundo. (2019, 19 de octubre). *Promesas, programación funcional en javascript parte 9* [Archivo de video]. Recuperado de https://www.youtube.com/playlist?list=PLSnadb41DsdKMddToNitoXrgHK7CEbUki

Checkpoints

Asegúrate de:

- Comprender los conceptos básicos de la programación funcional en JavaScript.
- Asociar los conocimientos de funciones, principalmente las funciones flecha, para implementar la programación funcional de forma más sencilla.
- Practicar el uso de las funciones señaladas en el contenido de esta experiencia educativa para que pongas a prueba las características multiparadigmáticas de JavaScript.

Requerimientos técnicos

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

Prework

- Haber realizado la actividad guiada del tema 5, de preferencia durante la sesión.
- Leer detenidamente y comprender el material explicado en el tema.
- Practicar todos los ejemplos que se describen previamente en el tema.
- Revisar cada uno de los recursos adicionales que se proponen en este tema.