Universidad Galileo

Carrera: Técnico universitario en desarrollo Full Stack

Curso: Introducción ala nube.

Sede: Virtual

Programación Funcional

Nombre: Maidellin Suset Alvarado Cayax

Carné: 24011377

Sección: T

Fecha de entrega: 12/12/2024

Programación Funcional en JavaScript

La programación funcional (FP) es un paradigma de programación que se centra en funciones como bloques básicos de construcción y promueve un estilo declarativo. En este modelo las funciones pueden ser tratadas como ciudadanos de primera clase.

Las funciones puede:

- ✓ Asignar a variables.
- ✓ Pasar como argumentos a otras funciones.
- ✓ Devolver desde otras funciones.

La inmutabilidad: Son los datos que no se modifican. En lugar de cambiar un objeto o valor existente, se crean nuevas versiones con los cambios requeridos.

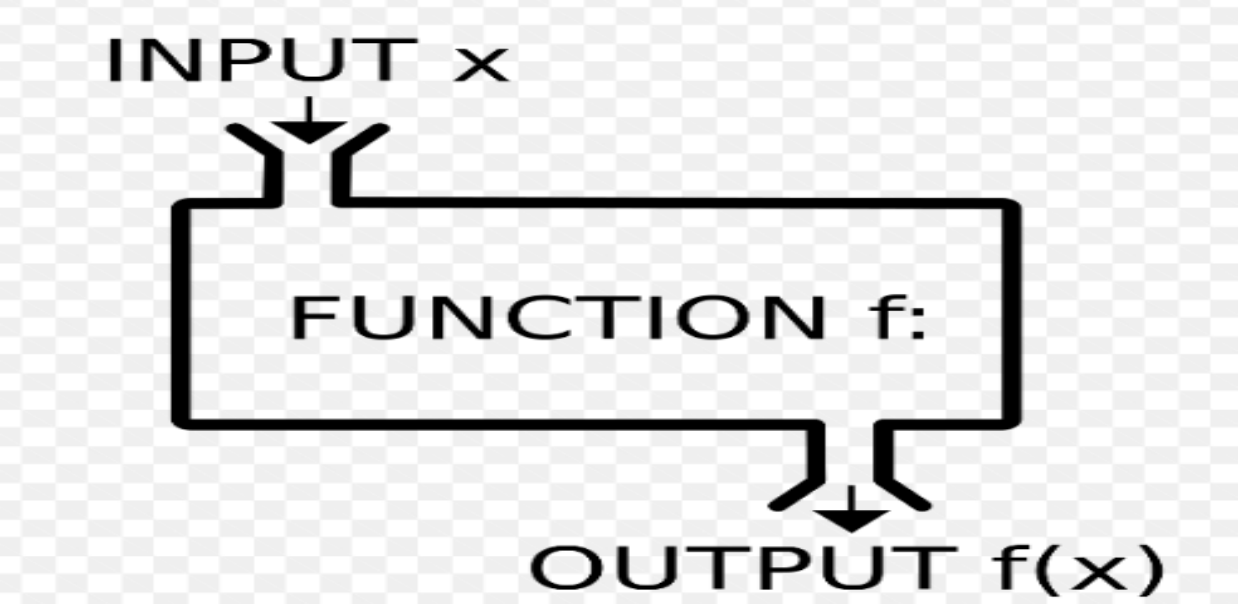
Un ejemplo clásico es una constante: Una vez definida, un valor no se puede cambiar.

La inmutabilidad nos ayuda reducir alteraciones en el código estados compartidos o variables modificadas inesperadamente. En este modelo se promueve la creación de funciones pequeñas que se pueden combinar para realizar tareas más complejas.

Como son inmutables, no hay conflictos en el acceso concurrente, lo que facilita la programación paralela.

En FP, usamos estructuras como List, Map, o frameworks que implementan versiones inmutables.

- *const numbers = [1, 2, 3];*
- *const newNumbers = numbers.map(num => num * 2); // [2, 4, 6]*



Lenguajes con soporte para FP:

- ✓ JavaScrip.
- ✓ Python.
- ✓ Scala.
- ✓ Clojure.

Funciones Puras en Programación Funcional (FP)

Son un tipo especial de función que respeta estrictamente ciertos principios para mantener un comportamiento predecible y confiable. Son más fáciles de razonar, ya que su salida depende únicamente de sus entradas.

En las funciones puras “No cambian datos” en lugar de modificar valores existentes, retornan nuevos valores.

EJEMPLO:

- *const square = (x) => x * x;*
- *console.log(square(3)); // Siempre devuelve 9*
- *console.log(square(3)); // Siempre devuelve 9*

Son ideales para la memorización, porque su salida siempre es la misma para los mismos argumentos.

Ausencia de Efectos Secundarios en Programación Funcional (FP)

El efecto secundario pasa cuando una función interactúa con algo fuera de su alcance local o cambia el estado del programa de alguna manera porque pasa por otra ruta o camino.

Pero la ausencia de efecto secundario promueve la pureza y la previsibilidad del código, lo que facilita su comprensión, prueba y mantenimiento.

En FP, se busca minimizar o eliminar estos efectos para mantener la pureza de las funciones.

Función con efecto secundario:

- *let count = 0;*
- *const increment = () => count++; // Modifica la variable global "count"*

Función sin efecto secundario:

- *const increment = (count) => count + 1; // No altera variables externas*

El no tener efectos secundarios más fáciles de probar, ya que no dependen del estado externo ni lo alteran. Nosotros como desarrolladores podemos aprovechar los beneficios de la FP sin sacrificar la funcionalidad necesaria para aplicaciones del mundo real.

Diferencias Entre Programación Funcional (FP) y Programación Orientada a Objetos (OOP)

La principal diferencia es que la programación funcional es adecuada para aplicaciones del lado del servidor, manipulación de datos y rastreo web, mientras que la programación orientada a objetos es el paradigma utilizado por los desarrolladores de software.

FP:

- ✓ Se enfoca en funciones puras y la transformación de datos.
- ✓ Utiliza principios matemáticos para resolver problemas, enfatizando la inmutabilidad, las funciones puras y la ausencia de efectos secundarios.
- ✓ Los datos y las funciones están separados.

OOP:

- ✓ Se centra en objetos que encapsulan datos y comportamientos relacionados.
- ✓ Los objetos son entidades que combinan estado (datos) y comportamiento (métodos).
- ✓ La interacción entre objetos es fundamental, utilizando conceptos como herencia, polimorfismo y encapsulación.

Característica	*PF*	*OPP*
Enfoque	*Funciones*	*Objetos*
Estado	*Inmutable*	*Mutable*
Objetivo	*Funciones puras, recursión*	*Clases, herencia, polimorfismo*

Lodash

Es una biblioteca de JavaScript que proporciona funciones para trabajar con arrays, objetos, cadenas, números y más. Puedes importar solo las funciones que necesitas, lo que ayuda a mantener tu código ligero y eficiente, es compatible con Node Js

```
const _ = require('lodash');
const numbers = [1, 2, 3, 4, 5];
const squaredNumbers = _.map(numbers, (n) => n * n);
console.log(squaredNumbers); // [1, 4, 9, 16, 25]
```

Ramda

Es una biblioteca práctica de programación funcional para JavaScript que se centra en un estilo de programación puramente funcional.

Las funciones de Ramda están automáticamente curried, lo que facilita la creación de nuevas funciones a partir de las existentes simplemente omitiendo los parámetros finales. Esto permite crear funciones más pequeñas y reutilizables.

```
const add = R.add;
const increment = add(1);
console.log(increment(2)); // 3
```