Programación Funcional

dev.f

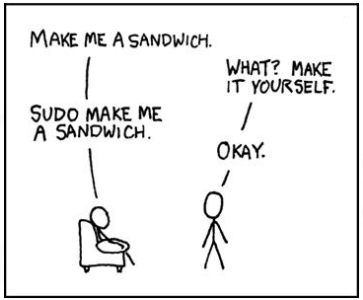
desarrollamos(personas);

¿Qué es ?

- Es un paradigma de programación.
- Es una forma de construir software a partir de solamente funciones
- La programación funcional es declarativa
- Tiende a ser más concisa, más predecible y más fácil de leer.
- Se compone de funciones puras (no utilizas librerías, funciones o procedimientos externos)

Programación Declarativa e Imperativa

Declarativa: Declaras una base de conocimiento, restricciones o afirmaciones para resolver el problema pero no dices como llegar a la solución.



Imperativa: Declaras un algoritmo y una serie de instrucciones o pasos para llegar a una solución.

dev.f

¿Que es una función Pura?

```
function sumaUnoAlNumero(numero) {
    return numero + 1;
}
```

No tiene efectos de estado que la afecten, lo que quiere decir que siempre que le pasemos un número A este dará como resultado un número B, como ejemplo si le damos como número un 3 este nos arrojará como número un 4.

¿Que es una función Impura?

```
function sumaNumeroRandom(numero) {
    return numero + Math.random()
}
```

Tiene efectos de estados porque un agente externo como es Math.random() está afectando el la fiabilidad con la que nos dará los resultados la función, en el sentido de que el número A que le demos no necesariamente nos dará siempre un mismo resultado B.

No hay mejor o peor elección todo depende de la complejidad del problema y que se necesita hacer

POO

- Dice que los datos (atributos) y las acciones (métodos) deben estar juntas en un solo “Objeto”.
- Dice que la Herencia y encapsulación hacen más fácil la forma de abstraer y asegurar el codigo
- Piensa en re-usabilidad a través de la herencia

VS

FP

- Dice que los datos y las acciones deben estar separadas ya que son distintivamente diferentes.
- Dice que separar métodos de atributos es mucho mejor ya que evita menores errores de lógica ya que se necesita un mayor nivel de abstracción
- Piensa en re-usabilidad a través de funciones pequeñas

Ejemplo

Digamos que dirigimos una empresa y acabamos de decidir otorgarle a todos los empleados un aumento de $ 10,000.00. ¿Cómo podríamos escribir un script para hacer este cambio?

Solucion con POO

```
class Empleado {
    constructor(nombre,salario){
        this.nombre = nombre;
        this.salario = salario;
    }
    cambiarSalario(aumento){
        this.salario = this.salario + aumento
    }
    descripcion(){
       return `El trabajador: ${this.nombre} gana ${this.salario}`
    }


}
```

Solucion con POO

```
var empleados  = [
    Empleado("Luis Torres",25000),
    Empleado("Maria Arriaga",42000)
]
```

Solucion con POO

```
empleados.forEach((empleado) =>{
    empleado.cambiarSalario(10000)
});
```

Solución con FP

```
var empleados = [
    ["Luis Torres",25000],
    ["Maria Arriaga",42000]
]
```

Solución con FP

```
function copiaEmpleados(empleados){
    let newEmpleados = new Array();
    empleados.forEach(empleado => newEmpleados.push([...empleado]));

    return newEmpleados

}

function cambiarSalario(empleados,cantidad){
    let copEmpleados = copiaEmpleados(empleados)
    copEmpleados.forEach((empleado) => {
        empleado[1] = empleado[1] + cantidad;
    })

    return copEmpleados
}
```

Solución con FP

```
var empleadoFelices = cambiarSalario(empleados,10000)

empleadoFelices.forEach((empleado) =>{

        console.log(`El empleado: ${empleado[0]} gana ${empleado[1]}`)
})
```

Funciones y Métodos de los Arrays

filter()

pop()

find()

push()

sort()

map()

forEach()

splice()

...

map()

Crea un nuevo array con los resultados de la llamada a la función indicada aplicados a cada uno de sus elementos.

```
var numbers = [1, 5, 10, 15];
var doubles = numbers.map(function(elem,index,arr) {
   return elem * 2;
});
// doubles is now [2, 10, 20, 30]
// numbers is still [1, 5, 10, 15]

var numbers2 = [1, 4, 9];
var roots = numbers2.map(Math.sqrt);
// roots is now [1, 2, 3]
// numbers is still [1, 4, 9]
```

filter()

Crea un nuevo array con todos los elementos que cumplan la condición implementada por la función dada.

```
var words = ['spray', 'limit', 'elite', 'exuberant',
             'destruction', 'present'];

const result = words.filter(word => word.length > 6);
const result2 = words.filter(function(elem,index,arr){
 return elem.length > 6;
})
var result3 = words.filter(word => word === 'elite');

console.log(result);
// expected output: Array ["exuberant", "destruction", "present"]
```

find()

Devuelve el valor del primer elemento del array que cumple la función de prueba proporcionada. En cualquier otro caso se devuelve **undefined**.

```
var array1 = [5, 12, 8, 130, 44];

var found = array1.find(function(element) {
  return element > 10;
});

console.log(found);
// expected output: 12
```

join()

Une todos los elementos de una matriz (o un objeto similar a una matriz) en una cadena y devuelve esta cadena.

```
var elements = ['Fire', 'Wind', 'Rain'];

console.log(elements.join());
// expected output: Fire,Wind,Rain

console.log(elements.join(''));
// expected output: FireWindRain

console.log(elements.join('-'));
// expected output: Fire-Wind-Rain
```

reverse()

Coloca al revés (inversamente) una matriz. El primer elemento pasa a ser el último y el último pasa a ser el primero.

```
var miMatriz = ['uno', 'dos', 'tres'];

console.log(miMatriz); // ['uno', 'dos', 'tres']

miMatriz.reverse();

console.log(miMatriz); // ['tres', 'dos', 'uno']
```