PROGRAMAÇÃO FUNCIONAL

É um paradigma de programação como a orientação a objetos, por exemplo:

- Uma maneira de resolver problemas

Funções

- pequenas tarefas dentro de uma função
- abstrair um problema e isolar ele dentro da função
- não modificar dados fora dela
- pequenas e bem específicas no que fazem

Características principais da função

- um dado (ou mais) entra em uma função e um novo dado sai dessa função
- não alterar dados
- não guarda estado stateless

Linguagens funcionais

- javaScript (multiparadigma)
- PHP(multiparadigma)
- Elixir
- Haskell

Pontos Positivos

- Fácil manutenção
- Facil uso de testes(funções isoladas e consistentes)
- códigos mais confiáveis

Princípios

- Programação imperativa
- Programação declarativa

Dados

- Imutabilidade
- Stateless

Funções

- Independentes
- Puras
- Higher-order
- First-class
- composição

Programação Imperativa vs Declarativa

Programação imperativa

- O código é pensado e gerado em sequência
- O código é pensado como um passo-a-passo, como uma receita de bolo
- uma coisa depende da outra
- o estado de um dado é alterado constantemente causando mutações nas variáveis
- orientação a objetos é um tipo de paradigma imperativo

Exemplo:

```
let number = 2

function  square() {
   return number * number
}
number = square()
```

Programação declarativa

- o código é gerado para fazer algo, não importa como
- o que fazer e não como fazer
- não há necessidade de um passo a passo no código
- programação funcional é um tipo de paradigma declarativo

Exemplo:
const square = n => n * n

Imutabilidade

- Uma variável não vai variar
- se você precisar mudar uma variável, você não muda, você cria uma nova

Exemplo:
const cart = {
quantity: 2,
total: 200
}
não pode fazer isso: card.quantity = 3
e sim const newCart = {...cart, quantity:3

Stateless

- Não guarda estado
- a função só conhece dados entregues a ela
- a resposta não poderá variar

Exemplo:
const square = n => n * n;

Funções

Funções independentes

- deverá ter ao menos 1 argumento
- deverá retornar algo
- nada que acontecer lá dentro afeta o mundo externo, dados imutáveis, não guardar estado
- não faremos uso de loops (for, while), mas se houver necessidade de tal, usaremos recursão (a função chama ela mesma)

Exemplo:
const random = (number, Math) =>
Math.floor(Math.random() * number);
recursive
find the factorial of a number

```
const factorial = x =>{
   //if number is 0
        if( x === 0){
            return 1;
        }
        return x * factorial( x - 1);
}
```

Funções puras

- não deverá depender de nenhum dado externo a não ser o que foi passado como argumento
- não deverá sofrer nenhuma interferência do mundo externo a ela
- se o argumento é o mesmo, o retorno não podera ser diferente quando a função for chamada novamente

- não terá nenhum efeito colateral no seu codigo
- não irá modificar nenhum dado
- não irá guardar nenhum estado

Exemplo:

```
const calculateCircumference = function (pi, radius) {
   return pi * (radius + radius)
}
```

First-class function

- podem estar em qualquer lugar, inclusive, como parâmetro de outras funções
- a função poderá ser entendida como uma variável

Exemplo:

```
const sayMyName = () => console.log('Mayk')
const runFunction = fn => fn()

runFunction(sayMyName)
runFunction(() => console.log('discover'))

console.log(runFunction(Math.random))
```

Higher-order function

- funções que recebem funções como argumentos
- funções que poderão retornar outras funções

Exemplo:

```
const numbers = [2, 4, 8, 16]
const square = n => n * n
const squaredNumber = numbers.map(square)
```

exemplo de um retorno de função

```
const pause = wait => fn => setTimeout(fn, wait)
pause(600) (() => console.log('waiting 600ms') )
const wait200 = pause(200)
const wait1000 = pause(1000)
wait200 (() => console.log('waiting 200ms'))
wait1000 (() => console.log('waiting 1s'))
```

Composição de funções

- Um encadeamento de funções
- Uma função que retorna um dado e vai para outra função, que retorna um dado e vai para outra função, que retorna...

Exemplo:

```
const people = ['Rafa', 'Diego', 'Dani', 'Rod']
const upperCasePeopleThatStartsWithD = people
.filter(person => person.starsWith('D'))
.map(dperson => dperson.toUpperCase())
```