**Vitor Hugo Klein**

Ra: 2577895

1)

|  |  |
| --- | --- |
| **Swift** | |
| **Ano e criador:** | 2014 - Apple |
| **Objetivo:** | Criada para ser segura, rápida e moderna |
| **Metodologia de Implementação:** | Compilada |
| **Paradigma** | Funcional e imperativa |
| **Aplicações** | Desenvolvimento de apps iOS e macOS |
| **Curiosidades:** | Suporta inferência de tipos |

|  |  |
| --- | --- |
| **Elixir** | |
| **Ano e criador:** | 2011 - José Valim |
| **Objetivo:** | Criada para escalabilidade e tolerância a falhas |
| **Metodologia de Implementação:** | Interpretada. |
| **Paradigma** | Funcional e concorrente |
| **Aplicações** | Desenvolvimento Web e sistemas de alta disponibilidade |
| **Curiosidades:** | Usa a VM do Erlang |

|  |  |
| --- | --- |
| **Rust** | |
| **Ano e criador:** | 2010 - Mozilla |
| **Objetivo:** | Criada para segurança de memória e concorrência segura |
| **Metodologia de Implementação:** | Compilada |
| **Paradigma** | Funcional e imperativa |
| **Aplicações** | Desenvolvimento de sistemas, jogos e WebAssembly |
| **Curiosidades:** | Utiliza Ownership para evitar vazamento de memória |

|  |  |
| --- | --- |
| **Erlang** | |
| **Ano e criador:** | 1986 - Joe Armstrong |
| **Objetivo:** | Criada para sistemas distribuídos e tolerantes a falhas |
| **Metodologia de Implementação:** | Interpretada. |
| **Paradigma** | Funcional e concorrente |
| **Aplicações** | Aplicações de telecomunicações e sistemas distribuídos |
| **Curiosidades:** | Utilizada na Ericsson |

2)O paradigma funcional é um modelo de programação baseado na composição de funções e na avaliação de expressões, enfatizando a imutabilidade e a ausência de efeitos colaterais. Aqui estão os principais conceitos envolvidos:

**1. Cálculo Lambda**

O cálculo lambda é um sistema formal desenvolvido por Alonzo Church que serve como base teórica para a programação funcional. Ele descreve funções matematicamente utilizando apenas **funções anônimas** (funções sem nome) e aplicações de funções.

**Exemplo em JavaScript:**

const soma = (x) => (y) => x + y;

console.log(soma(2)(3)); // Saída: 5

Aqui, soma é uma função que retorna outra função, ilustrando a ideia de funções como cidadãos de primeira classe.

**2. Imutabilidade**

Imutabilidade significa que os dados não podem ser alterados depois de criados. Em vez de modificar valores existentes, novas versões dos dados são criadas.

**Exemplo em JavaScript:**

const lista = [1, 2, 3];

// Criando uma nova lista em vez de modificar a existente

const novaLista = [...lista, 4];

console.log(lista); // [1, 2, 3] (inalterada)

console.log(novaLista); // [1, 2, 3, 4] (nova estrutura)

Aqui, novaLista é uma nova referência, e lista permanece inalterada.

**3. Efeito Colateral**

Um efeito colateral ocorre quando uma função modifica algo fora do seu escopo (como alterar variáveis globais, modificar entradas de usuário ou interagir com o sistema de arquivos). Programação funcional evita efeitos colaterais para garantir previsibilidade.

**Exemplo de função com efeito colateral:**

let contador = 0;

function incrementar() {

contador++; // Modifica o estado global

}

incrementar();

console.log(contador); // Valor alterado

Isso não é desejável na programação funcional. Em vez disso, usamos funções puras:

const incrementar = (valor) => valor + 1;

console.log(incrementar(0)); // 1

Aqui, incrementar retorna um novo valor sem modificar variáveis externas.

**4. Funções de Primeira Ordem (ou Primeira Classe)**

Funções de primeira ordem são tratadas como qualquer outro valor, ou seja, podem ser atribuídas a variáveis, passadas como argumentos e retornadas de outras funções.

**Exemplo em JavaScript:**

const dobro = (x) => x \* 2;

const processar = (func, valor) => func(valor);

console.log(processar(dobro, 5)); // 10

Aqui, dobro é passado como argumento para processar, demonstrando que funções são tratadas como valores.

**Conclusão**

A programação funcional favorece um estilo declarativo e matematicamente sólido, garantindo código mais previsível, modular e menos propenso a bugs. Usando cálculo lambda, imutabilidade, ausência de efeitos colaterais e funções de primeira ordem, podemos escrever código mais expressivo e robusto.