Nomes: Matheus Baron Lauritzen e Gustavo Baron Lauritzen.

Paradigma Funcional

Um paradigma de programação diferente dos paradigmas imperativos mencionados em sala de aula é o Paradigma Funcional. Ele é baseado na avaliação de funções matemáticas e na utilização de funções puras, que não possuem efeitos colaterais e não dependem de variáveis mutáveis. Em vez disso, as funções funcionais retornam um valor calculado com base em seus argumentos de entrada, sem modificar o estado do programa.

Alguns dos principais conceitos do paradigma funcional incluem:

- Funções puras: funções que não possuem efeitos colaterais e não dependem de variáveis mutáveis.
- Imutabilidade: dados que não podem ser modificados após serem criados.
- Expressões e avaliação: programas são compostos de expressões que são avaliadas para produzir um resultado.
- Recursão: a capacidade de uma função chamar a si mesma para resolver um problema.
- Funções de primeira classe: funções que podem ser passadas como argumentos para outras funções e retornadas como resultados.
- High-order functions: funções que operam sobre outras funções, como as funções de mapa e redução.

Sua Origem:

O paradigma funcional tem suas raízes na teoria matemática da computação, particularmente na teoria dos conjuntos e na lógica matemática. O uso de funções para descrever algoritmos remonta a Gottfried Wilhelm Leibniz, que criou um cálculo simbólico para a lógica baseado em funções em 1680.

No entanto, o paradigma funcional como o conhecemos hoje teve seu início na década de 1930 com o trabalho de Alonzo Church e seu aluno Stephen Kleene na Universidade de Princeton. Eles desenvolveram a Teoria da Computabilidade Lambda, uma teoria matemática que descreve a computação usando funções. Em 1958, John McCarthy usou a Teoria da Computabilidade Lambda para criar a linguagem de programação Lisp, que é uma das linguagens funcionais mais antigas ainda em uso.

A popularidade do paradigma funcional cresceu na década de 1980 com o desenvolvimento de linguagens funcionais como ML, Miranda e Haskell. O desenvolvimento de linguagens funcionais modernas como F# e Scala, que combinam elementos de programação funcional e orientada a objetos, continuam a expandir a aplicação do paradigma funcional na indústria e na academia.

Evolução:

Esse paradigma evoluiu ao longo dos anos, passando por várias fases e influenciando outras áreas da ciência da computação. A seguir, estão algumas das principais etapas da evolução do paradigma funcional:

- Lambda cálculo: O lambda cálculo é uma teoria matemática desenvolvida por Alonzo Church na década de 1930, que descreve a computação em termos de funções. Ele é considerado o

precursor do paradigma funcional e serviu como base para o desenvolvimento de linguagens de programação funcionais posteriores.

- Lisp: Lisp foi a primeira linguagem de programação funcional de uso geral, criada por John McCarthy em 1958. Ela foi baseada no lambda cálculo e introduziu conceitos como listas e recursão em linguagens de programação. Lisp teve uma influência significativa no desenvolvimento de outras linguagens de programação, incluindo Scheme e Clojure;
- Programação funcional pura: Na década de 1970, uma nova abordagem para a programação funcional surgiu, que se concentrava em funções puras, que não possuem efeitos colaterais ou estado mutável. Isso tornou a programação funcional mais fácil de entender e testar. Exemplos de linguagens de programação que se concentraram na programação funcional pura incluem Haskell e Clean;
- Tipagem estática: A tipagem estática é uma abordagem para a programação funcional que se concentra na verificação de tipos em tempo de compilação. Isso torna mais fácil detectar erros no código antes de executá-lo e ajuda a melhorar a segurança e a confiabilidade do programa. Exemplos de linguagens de programação que usam tipagem estática incluem Haskell, OCaml e F#;
- Programação funcional reativa: A programação funcional reativa é uma abordagem que se concentra na reatividade do programa, ou seja, em como o programa reage a eventos ou mudanças no ambiente. Isso é especialmente útil para aplicativos que precisam responder a mudanças em tempo real, como jogos ou aplicativos da web. Exemplos de linguagens de programação que suportam programação funcional reativa incluem Elm e ReactiveX.

Em resumo, o paradigma funcional evoluiu ao longo dos anos, passando por várias fases e influenciando outras áreas da ciência da computação. A partir do lambda cálculo, ele se desenvolveu em linguagens de programação de uso geral, como Lisp, e depois evoluiu para se concentrar em funções puras, tipagem estática e programação funcional reativa. Essas abordagens tornaram a programação funcional mais fácil de entender, testar e aplicar em diversos tipos de aplicativos.

Linguagens Específicas para esse paradigma:

Alguns exemplos de linguagens que têm somente o Paradigma Funcional como base:

- Haskell: é uma linguagem de programação funcional pura, que segue a teoria das categorias e do lambda cálculo. Ela foi criada na década de 1990 e é uma linguagem tipada estática;
- Lisp: é a primeira linguagem de programação funcional de uso geral, criada em 1958 por John McCarthy. Ela é baseada no lambda cálculo e introduziu conceitos como listas e recursão em linguagens de programação;
- Clojure: é uma linguagem de programação funcional que roda na JVM (Java Virtual Machine). Ela é baseada em Lisp e usa a tipagem dinâmica.

Linguagens Multiparadigmas para esse paradigma:

Cinco exemplos de linguagens Multiparadigmas que incluem o paradigma funcional:

- JavaScript: é uma linguagem de programação amplamente usada para desenvolvimento de aplicativos da web. Ela inclui recursos do paradigma funcional, como funções de primeira classe, funções anônimas e closures;
- Python: é uma linguagem de programação de alto nível que suporta múltiplos paradigmas, incluindo programação orientada a objetos e programação funcional. Python inclui recursos como funções lambda e map, filter e reduce para trabalhar com listas;

- Ruby: é outra linguagem de programação de alto nível que suporta múltiplos paradigmas, incluindo programação orientada a objetos e programação funcional. Ruby inclui recursos como blocos e lambdas para trabalhar com coleções;
- ML: é uma família de linguagens de programação funcionais que incluem Standard ML, OCaml e F#. Elas são tipadas estáticas e possuem suporte para programação orientada a objetos;
- Swift: é uma linguagem de programação que foi desenvolvida pela Apple para desenvolvimento de aplicativos para iOS e MacOS. Ela suporta programação orientada a objetos e programação funcional. Swift inclui recursos como funções de primeira classe, funções anônimas, lambdas e closures.

Trecho de código para exemplo:

Exemplo de código em F# que recebe uma lista de inteiros e retorna outra lista de inteiros com o dobro de cada elemento:

```
let doubleList xs = List.map (fun x -> x * 2) xs


let myList = [1; 2; 3; 4; 5]

let doubledList = doubleList myList

printfn "%A" doubledList // [2; 4; 6; 8; 10]
```

No exemplo, a função "List.map" é uma função de ordem superior que recebe uma função como argumento e aplica essa função a cada elemento da lista, retornando uma nova lista com os resultados. Isso demonstra o conceito do paradigma funcional em prática, onde funções são usadas como valores de primeira classe e os programas são construídos a partir de composições de funções.

Comparação entre o Paradigma Funcional e o Orientado a Objetos:

O paradigma funcional se concentra em funções puras e dados imutáveis. Isso significa que as funções não têm efeitos colaterais e não modificam os dados de entrada. Em vez disso, as funções recebem os dados como entrada, processam os dados e retornam um resultado, sem modificar o estado do programa. O foco principal do paradigma funcional é escrever funções que são modulares, reutilizáveis e fáceis de entender e testar.

Por outro lado, o paradigma orientado a objetos se concentra em objetos com estado interno mutável e métodos que operam nesses objetos. Nesse paradigma, os dados e as operações são agrupados em objetos, que têm um estado interno e podem ser modificados por meio de métodos. O foco principal do paradigma orientado a objetos é escrever classes que encapsulam dados e comportamentos relacionados e fornecem uma interface para interagir com esses objetos.

Outrossim, é importante salientar a maneira de como eles lidam com a herança. No paradigma orientado a objetos, a herança é uma técnica comum de reutilização de código, que permite que as classes herdem propriedades e métodos de outras classes. Por outro lado, no paradigma funcional, a composição é uma técnica mais comum, em que funções são combinadas para criar novas funções, em vez de herdar comportamentos de outras funções.

Ademais, no paradigma orientado a objetos o encapsulamento é uma técnica de proteção de dados, em que as propriedades e métodos são encapsulados em uma classe e só podem ser acessados por meio de métodos públicos. Por outro lado, no paradigma funcional, os dados são frequentemente passados para as funções como argumentos, e as funções são projetadas para operar nesses dados de maneira transparente, sem se preocupar com sua fonte ou tipo.

Em suma, o paradigma funcional se concentra em funções puras e dados imutáveis, enquanto o paradigma orientado a objetos se concentra em objetos com estado interno mutável e métodos que operam nesses objetos. Ambas as abordagens têm suas forças e fraquezas e são adequadas para diferentes tipos de problemas de programação. O programador deve escolher o paradigma mais apropriado para resolver o problema em pauta, considerando as características do problema e as vantagens e desvantagens de cada paradigma.