

```
1
2
3  Programação 'Funcional' {
4
5      |
6      [Aline Yuka,
7       Eduardo Albuquerque,
8       Vitor Tavares]
9
10
11
12  }
13
14
```

Sumário {

01 O que é Programação Funcional e suas Origens

02 Conceitos Fundamentais

03 Diferenças entre Programação Imperativa e Funcional

04 Linguagens com suporte ao paradigma

05 Utilidade do paradigma funcional na atualidade

06 Material e Exercícios

}

1
2 01 {
3
4
5
6
7
8
9
10
11
12
13
14

[0 que é Programação
Funcional e suas
Origens]

}

1 O que é {

2
3 var1 = "Paradigma que trata a computação como avaliação de
4 funções matemáticas."

5
6 var2 = "Enfatiza imutabilidade, funções puras e composição
7 funcional."

8
9 var3 = "Ausência de efeitos colaterais: funções sempre
10 retornam o mesmo resultado para os mesmos inputs."

11
12 var4 = "Abordagem declarativa: foco no "o que fazer", não no
13 "como fazer"."

14 }

História {

```
base = "Em 1930 Alonzo Church cria o Cálculo Lambda, base  
teórica da programação funcional."
```

```
creation(base){
```

```
v1950 = "John McCarthy cria o LISP, que introduz funções  
como dados."
```

```
v1970 = "Surgem linguagens como ML e Haskell (puramente  
funcional e tipagem forte)."
```

```
}
```

```
}
```

1
2 02 {
3
4
5
6
7

8 [Conceitos Fundamentais]
9
10
11

12 }
13
14

Basico{

função_pura = "Funções puras não causam efeitos colaterais e sempre retornam o mesmo resultado para os mesmos inputs."

sum a b = a + b

imutabilidade = "Dados não são modificados, mas sim recriados com novas versões."

addElement x xs = x : xs

ordem_superior = "Aceitam ou retornam outras funções."

map (\x → x * 2) xs

}

Conceitos{

```
ausencia_efeitos_colaterais = "A interação com o mundo  
externo é controlada (ex: tipo IO em Haskell)."
```

```
square x = x * x
```

```
main = do
```

```
    input ← getLine
```

```
    print (square (read input))
```

```
}
```


Conceitos{

```
avaliacao_preguicosa = "Cálculos só são feitos quando  
necessários."
```

```
recursao = "Utilizada no lugar de loops."
```

```
factorial n = n * factorial (n - 1)
```

```
composicao_funcoes = "Combina funções pequenas em funções  
maiores."
```

```
doubleAndIncrement = increment . double
```

```
}
```

Funções Básicas{

map = "Aplica uma função a cada item da lista."

```
map (\x → x * 2) [1,2,3] -- [2,4,6]
```

filter = "Filtra elementos com base em um predicado."

```
filter (\x → x mod 2 == 0) [1..5] -- [2,4]
```

fold = "Reduz uma lista a um único valor."

```
foldl (+) 0 [1,2,3,4] -- 10
```

}

1
2 03 {
3
4
5
6
7
8
9
10
11
12
13
14

[Diferenças entre
Programação Imperativa e
Funcional]

}

```
1  Diferenças {
2      imperativa = "Foco em como fazer: sequência de instruções,
3      controle de fluxo e variáveis mutáveis."
4
5      funcional = "Foco no que calcular: composição de funções
6      puras, imutabilidade e ausência de efeitos colaterais."
7  }
8  Vantagens{
9      efeitos_controlados = "Efeitos colaterais são isolados com IO,
10     facilitando o rastreamento."
11
12     concorrencia = "Imutabilidade evita race conditions e melhora
13     escalabilidade."
14     modularidade = "Funções reutilizáveis facilitam manutenção e
15     evolução do código."
```

1 Imperativo {

```
2  
3 bool isPrime(int n) {  
4     if (n ≤ 1) return false;  
5     for (int i = 2; i < n; i++) {  
6         if (n % i == 0) return false;  
7     }  
8     return true;  
9 }  
10  
11 }  
12  
13  
14 }
```

Funcional {

```
isPrime :: Int → Bool  
isPrime n  
    | n ≤ 1 = False  
    | otherwise = not (any (\x → n  
        `mod` x == 0) [2..n-1])  
  
}
```

1
2 04 {
3
4
5
6
7
8
9
10
11
12
13
14

[Linguagens com suporte ao
paradigma]

}

1 Linguagens{

```
2   linguagens_puramente_funcionais = "Seguem rigorosamente os
3   princípios da programação funcional com imutabilidade e funções
4   puras como norma."
5   [Haskell, Elm, PureScript, Elixir]
6   linguagens_fortemente_funcionais = "Permitem mesclar paradigmas e
7   incentivam o uso de funções puras, imutabilidade e composição
8   funcional."
9   [Scala, F#, Clojure, OCaml]
10  linguagens_parcialmente_funcionais = "Linguagens tradicionais que
11  incluem construções funcionais, como funções de ordem superior e
12  coleções imutáveis."
13  [JavaScript, Python, Ruby, Rust]
14 }
```

1
2 05 {
3
4
5
6
7
8
9
10
11
12
13
14

[Utilidade do paradigma
funcional na atualidade]

}

Utilização {

```
utilidade_paradigma_funcional = "A programação funcional é  
amplamente aplicada em áreas que exigem robustez,  
paralelismo, testabilidade e manutenibilidade."  
  
desenvolvimento_web = "Frameworks como React.js e Redux  
utilizam imutabilidade e funções puras."  
  
aplicacoes_dados = "Linguagens como Haskell são ideais para  
construção de pipelines de transformação de dados."  
  
sistemas_distribuidos = "Programação funcional oferece  
segurança em manipulação concorrente de dados."  
}
```

Utilização {

computacao_concorrente = "Imutabilidade permite computação paralela sem mecanismos complexos de sincronização."

aplicacoes_financeiras = "Linguagens como Haskell e F# são usadas para modelagem de ativos e avaliação de riscos."

inteligencia_artificial = "Facilidade na manipulação de dados em tarefas de machine learning e NLP."

telecomunicacoes = "Linguagens funcionais são aplicadas em roteamento de pacotes e sistemas tolerantes a falhas."

}

1
2 06 {
3
4
5
6
7

8 [Material e Exercícios]
9
10
11

12 }
13
14

```
1 E-Book e PDF{
```

```
2
```

```
3
```

```
4
```

```
5
```

```
    e_book="https://app.lumi.education/run/OTIE06"
```

```
6
```

```
7
```

```
    pdf="https://github.com/heliokamakawa/psa_2025/blob/main/YET/Programa%C3%A7%C3%A3o%20Funcional/O_Paradigma_Funcional_no_Desenvolvimento_de_Software.pdf"
```

```
8
```

```
9
```

```
10
```

```
11
```

```
12
```

```
13
```

```
14
```

```
    }
```

thanks

```
1  
2  
3 Obrigado 'Pela Atenção' {  
4  
5  
6  
7  
8 [Perguntas ??]  
9  
10  
11  
12 }  
13  
14
```