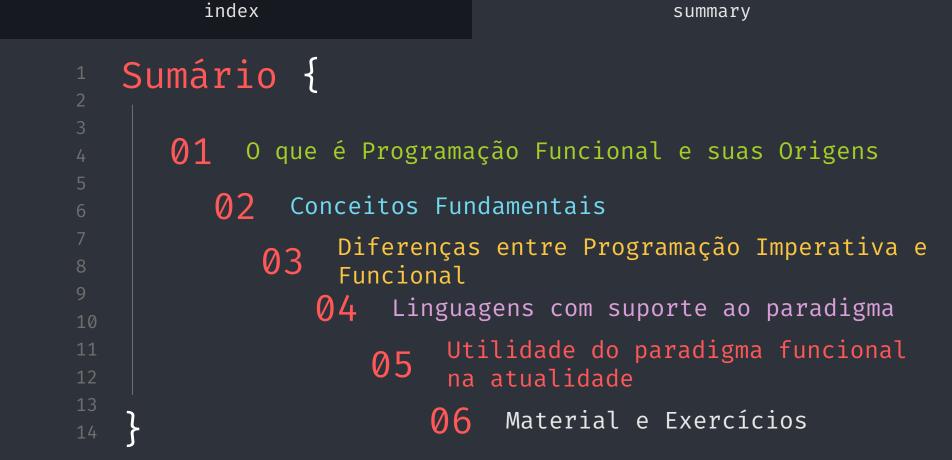
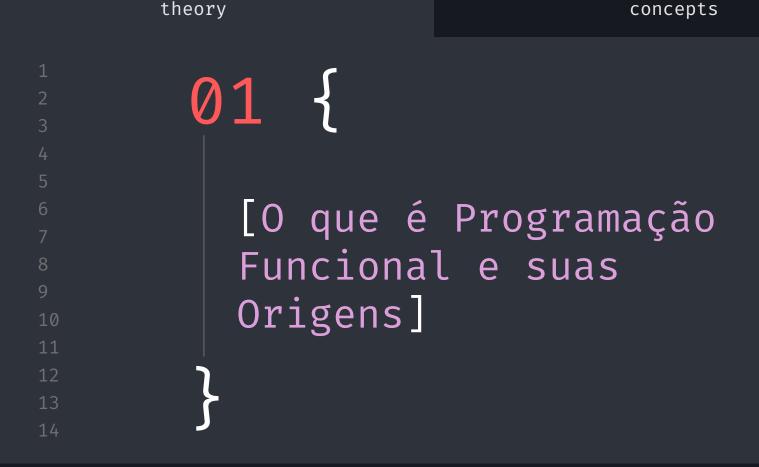
```
[Aline Yuka,
Eduardo Albuquerque,
Vitor Tavares]
```





```
1 0 que é {
   var1 = "Paradigma que trata a computação como avaliação de
   funções matemáticas."
   var2 = "Enfatiza imutabilidade, funções puras e composição
   funcional."
   var3 = "Ausência de efeitos colaterais: funções sempre
   retornam o mesmo resultado para os mesmos inputs."
   var4 = "Abordagem declarativa: foco no "o que fazer", não no
   "como fazer"."
```

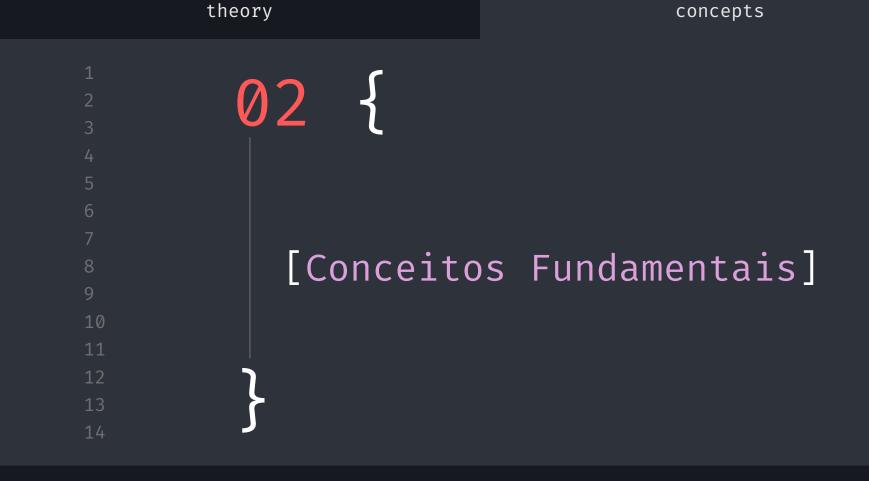
concepts

theory

```
theory concepts

1 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)."
```



theory

concepts

```
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 \times x = x : x = x
   ordem superior = "Aceitam ou retornam outras funções."
      map (\x \rightarrow x * 2) xs
```

theory

concepts

```
<sup>1</sup> 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))
```

concepts

1 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
```

theory concepts

```
Funções Básicas
    map = "Aplica uma função a cada item da lista."
       map (x \rightarrow x * 2) [1,2,3] - [2,4,6]
    filter = "Filtra elementos com base em um predicado."
       filter (x \rightarrow x \mod 2 = 0) [1..5] -- [2.4]
    fold = "Reduz uma lista a um único valor."
       foldl (+) 0 [1,2,3,4] -- 10
```

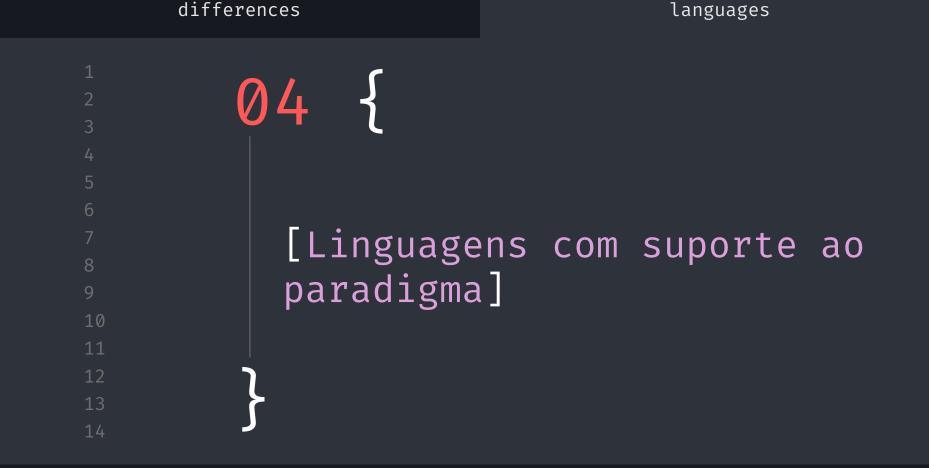


```
differences
```

languages

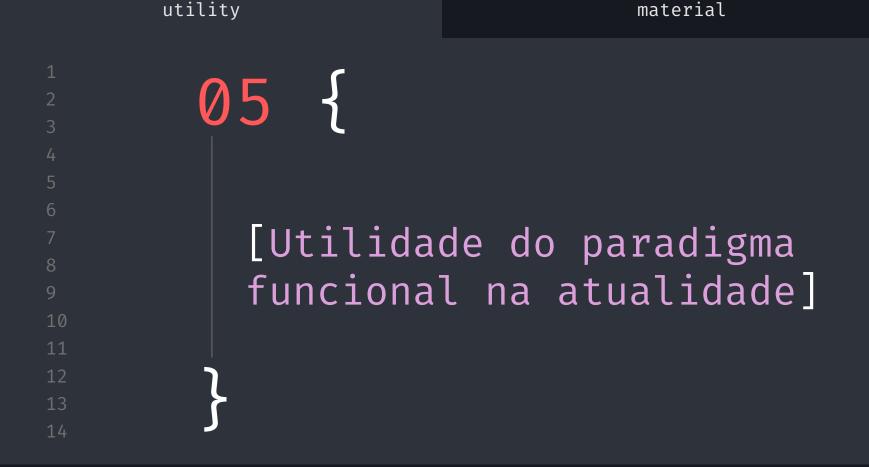
```
1 Diferenças {
    imperativa = "Foco em como fazer: sequência de instruções,
    controle de fluxo e variáveis mutáveis."
    funcional = "Foco no que calcular: composição de funções
    puras, imutabilidade e ausência de efeitos colaterais."
  Vantagens{
    efeitos controlados = "Efeitos colaterais são isolados com IO,
    facilitando o rastreio."
    concorrencia = "Imutabilidade evita race conditions e melhora
   escalabilidade."
    modularidade = "Funções reutilizáveis facilitam manutenção e
    evolução do código.
```

```
1Imperativo {
                                         Funcional {
   bool isPrime(int n) {
     if (n \le 1) return false;
                                          isPrime :: Int \rightarrow Bool
     for (int i = 2; i < n; i \leftrightarrow ) {
                                           isPrime n
       if (n \% i = \emptyset) return false;
                                               n \leq 1 = False
                                               otherwise = not (any (\xspacex \rightarrow n
                                               mod x = 0) [2..n-1]
     return true;
```



1 Linguagens{

```
linguagens puramente funcionais = "Seguem rigorosamente os
princípios da programação funcional com imutabilidade e funções
puras como norma."
linguagens fortemente funcionais = "Permitem mesclar paradigmas e
incentivam o uso de funções puras, imutabilidade e composição
funcional."
[Scala, F#, Clojure, OCaml]
linguagens parcialmente funcionais = "Linguagens tradicionais que
incluem construções funcionais, como funções de ordem superior e
coleções imutáveis."
```



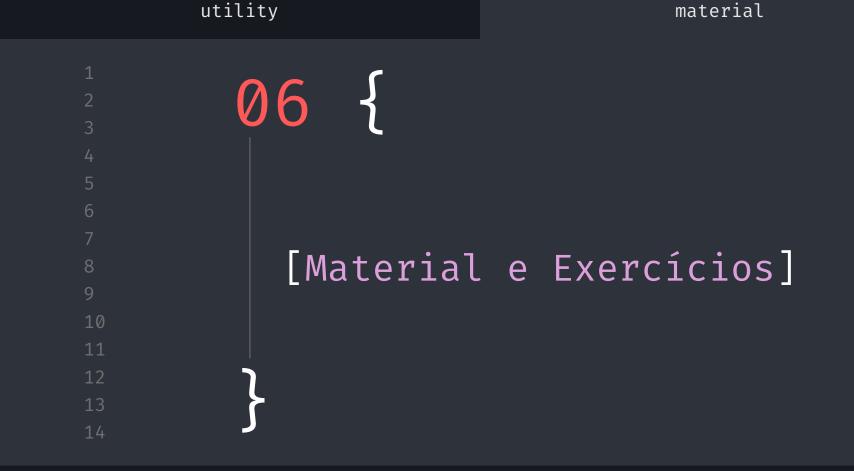
utility material

```
1 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."
14
```

Programação Funcional

utility material

```
1 Utilização {
   computação 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."
```



```
utility
                                              material
<sub>1</sub> E-Book e PDF{
     e_book="https://app.lumi.education/run/OTIEO6"
     pdf="https://github.com/heliokamakawa/psa_2025/blob/ma
     in/YET/Programa%C3%A7%C3%A3o%20Funcional/O_Paradigma_
     Funcional no Desenvolvimento de Software.pdf"
```

```
Obrigado 'Pela Atenção' {
       Perguntas ??
```