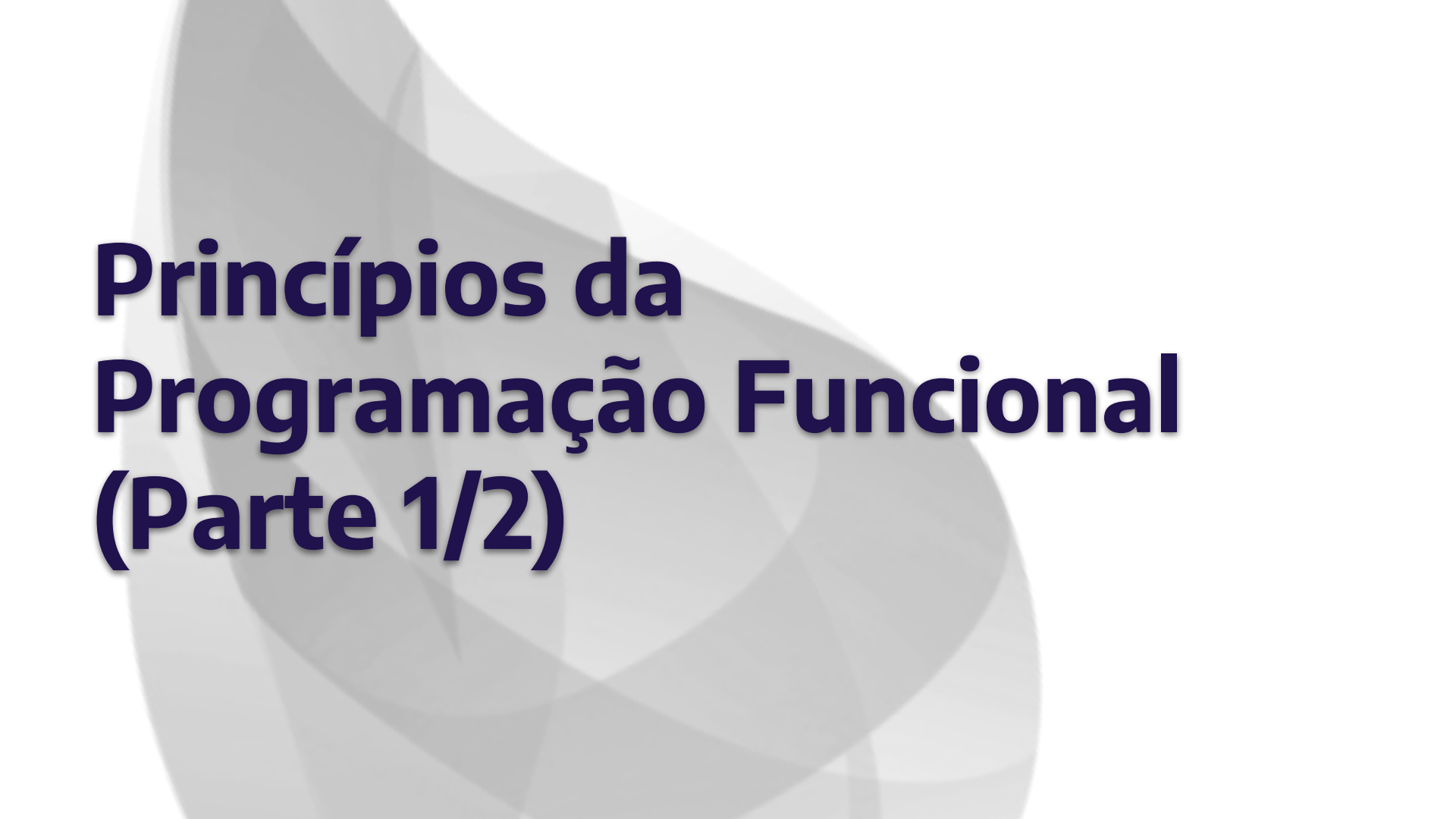




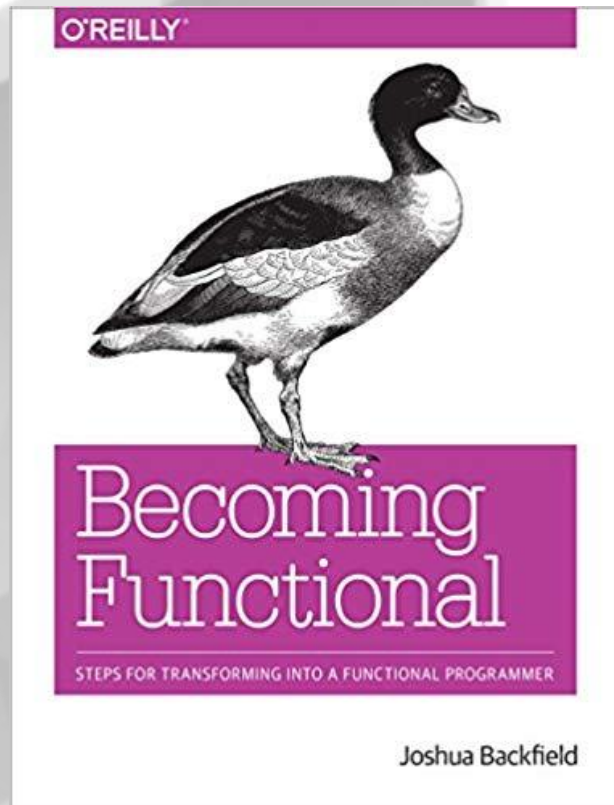
# **Programação Funcional com Elixir**



# **Princípios da Programação Funcional (Parte 1/2)**

# Princípios da Programação Funcional

- First-Class functions
- Pure functions
- Immutable variables
- Recursion
- Nonstrict evaluation
- Statements
- Pattern matching





# **First-Class Functions**

# First-Class Functions

- **Funções de primeira classe** são funções que podem aceitar outra função como um argumento ou mesmo retornar uma função.
- Ou seja, é a capacidade de criar funções e devolvê-las ou passá-las para outras funções.
- Isso é extremamente útil em reutilização de códigos e abstrações de código.

```
func A(){...}    func B(){...}    B = func {...}    func A(B){...}
```

# First-Class Functions

- Funções de primeira classe também são conhecidas como **Higher-Order Functions** (Função de Ordem Superior) ou ainda First-Class Citizens (Cidadãos de Primeira Classe)
- Lembre-se que quando falamos em **Lambda** é a possibilidade de criar uma função anônima, já uma função de primeira classe é a função que consegue receber funções anônimas como argumento/parâmetro, ou seja, receber lambdas.

The background of the slide features a series of overlapping, semi-transparent gray shapes that resemble stylized, flowing petals or abstract organic forms. These shapes are layered, creating a sense of depth and movement, primarily concentrated on the left side of the frame.

# Pure Functions

# Pure Functions

- **Funções puras** são funções que não têm efeitos colaterais.
- Efeitos colaterais são ações que uma função pode executar e que não estão contidas apenas na própria função.
- Um exemplo de função impura é quando passamos uma variável global e a transformamos diretamente dentro dessa função.



# Pure Functions

- **Funções puras** também são conhecidas por serem “indempotentes”, ou seja, a mesma entrada sempre gera a mesma saída.

# Pure Functions



The background of the slide features a series of overlapping, semi-transparent gray shapes that resemble stylized leaves or petals. These shapes are layered in a way that creates a sense of depth and movement, with some shapes appearing more prominent than others. The overall effect is a modern, minimalist aesthetic.

# **Immutable Variables**

# Immutable Variables

- **Variáveis imutáveis**, uma vez definidas, não podem ser alteradas.
- Embora a imutabilidade pareça muito difícil de fazer, dado o fato de que o estado deve mudar dentro de uma aplicação em algum momento, veremos como tratar isso mais adiante.

```
var minhaString = "abc"  
substituir(minhaString, "a", "x") //xbc  
minhaString //abc
```

The background of the slide features a series of overlapping, semi-transparent gray shapes that resemble stylized, curved petals or abstract organic forms. These shapes are layered, creating a sense of depth and movement, primarily concentrated on the left side of the frame.

# **Recursion**

# Recursion

- Em resumo, função recursiva é aquela que pode chamar a si mesma.
- A **Recursão** permite escrever algoritmos menores e mais concisos e operar observando apenas as entradas para nossas funções.
- Isso significa que a função estará preocupada apenas com a iteração atual e se deve continuar.

# Recursion

$$n! = \begin{cases} 1, & \text{se } n = 0 \text{ ou } n = 1 \\ n(n-1)!, & \text{se } n \geq 2 \end{cases}$$

```
funcao fatorial(n){  
    se (n==0 ou n==1)  
        retorne 1;  
  
    se (n > 1)  
        retorne n * fatorial(n-1);  
}
```

# Recursion

$$n! = \begin{cases} 1, & \text{se } n = 0 \text{ ou } n = 1 \\ n(n-1)!, & \text{se } n \geq 2 \end{cases}$$

```
int fatorial(int n)
{
    if (n >= 2)
        return n * fatorial(n - 1);
    else
        return 1;
}
```



# Recursion

$$n! = \begin{cases} 1, & \text{se } n = 0 \text{ ou } n = 1 \\ n(n-1)!, & \text{se } n \geq 2 \end{cases}$$

```
int fatorial(int n)
{
    if (n >= 2)
        return n * fatorial(n - 1);
    else
        return 1;
}
```

---

```
4! = 4 * 3!
          3! = 3 * 2!
                    2! = 2 * 1!
                              1! = 1 * 0!
                                                0! = 1
```

---

```
          1! = 1
                    2! = 2
                              3! = 6
                                      4! = 24
```

---