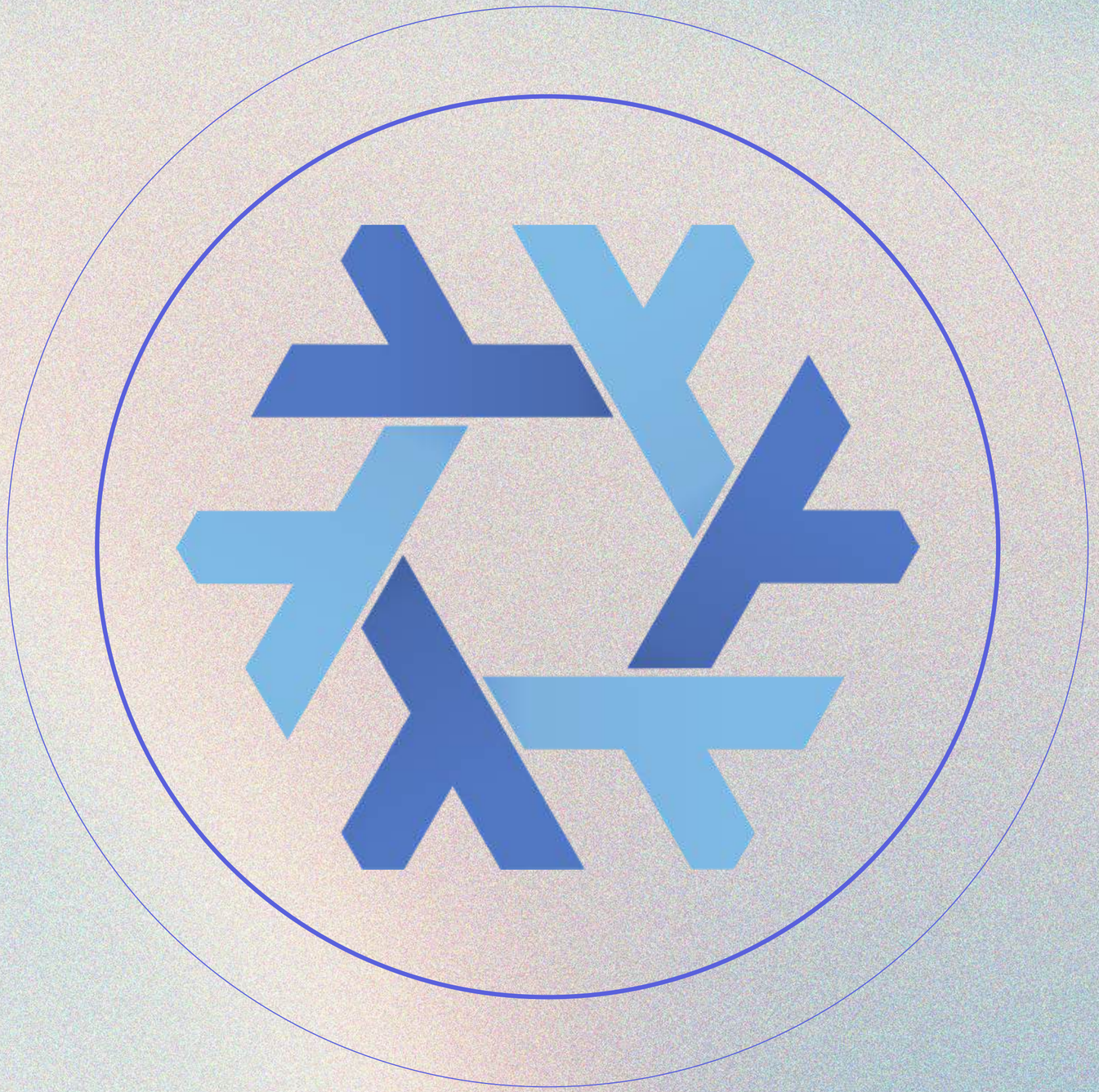
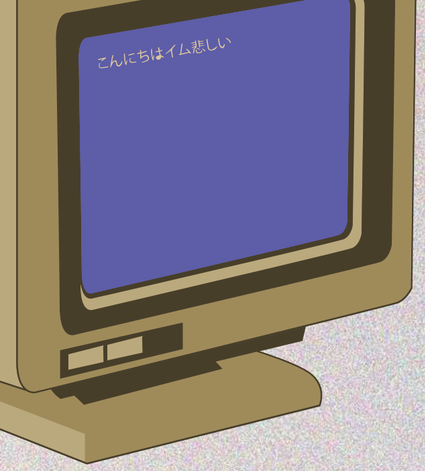
Teoria da Computação

LINGUAGEM FUNCIONAL

NIX

MATHEUS HENRIQUE DA COSTA
YAN BALBINO NOGUEIRA

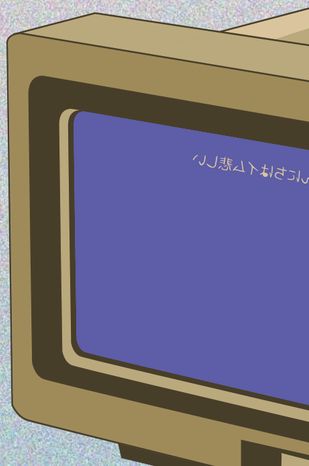


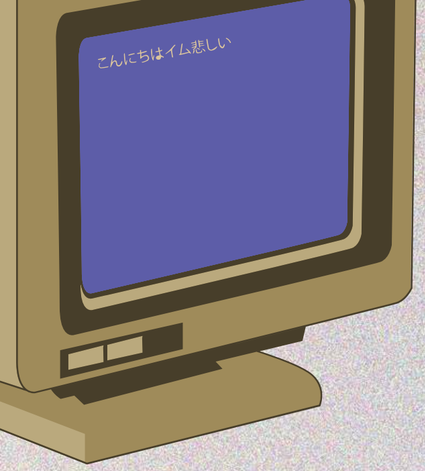


HISTÓRICO

A linguagem Nix foi criada como parte de um esforço acadêmico para resolver problemas clássicos de gerenciamento de pacotes em sistemas operacionais.

Seu criador, Eelco Dolstra, desenvolveu o conceito central da linguagem e do Nix Package Manager em sua tese de doutorado na Universidade Técnica de Delft, nos Países Baixos, no início dos anos 2000.



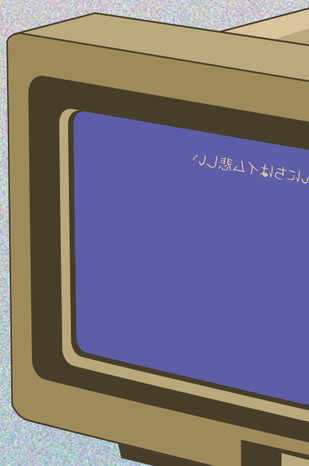


HISTÓRICO

A linguagem Nix, embora não seja de uso geral, é uma DSL (Domain-Specific Language) altamente expressiva voltada para a descrição de pacotes, ambientes e configurações de sistema.

Sua origem acadêmica confere à linguagem fundamentos sólidos nas áreas de ***semântica formal***, ***teoria da programação funcional*** — todos temas caros à Teoria da Computação.

Com o tempo, a linguagem evoluiu junto com o NixOS, um sistema operacional inteiramente construído com Nix, onde tudo — do kernel ao ambiente de desktop — é definido com código funcional



APLICABILIDADE

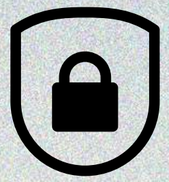
Construção de pacotes de software de forma determinística

```nix
# shell.nix
{ pkgs ? import <nixpkgs> {} }:

pkgs.mkShell {
  buildInputs = [ pkgs.nodejs pkgs.git ];
}
```

NO NET

NO HOST

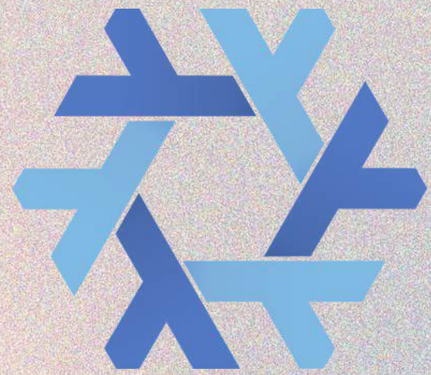


ISOLATED

DECLARATIVE

Criação de ambientes de desenvolvimento imutáveis

NIX-SHELL

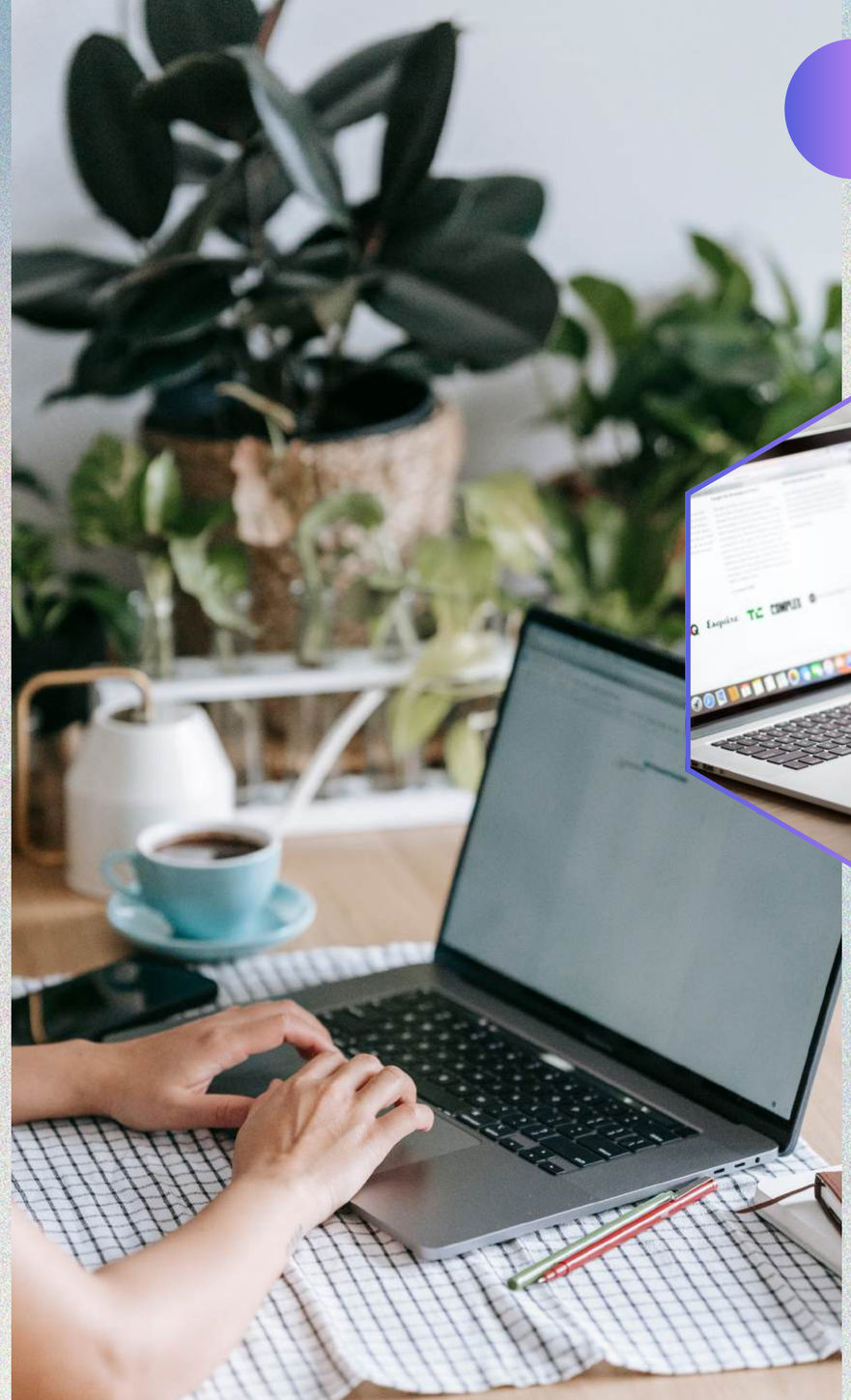


APLICABILIDADE

Gerenciamento declarativo de configurações de sistema

```nix
# configuration.nix
{
  environment.systemPackages = [
    pkgs.git
    pkgs.firefox
    pkgs.nodejs
  ];

  services.nginx.enable = true;
}
```



VANTAGENS

Pureza Funcional & Determinismo

- Sem efeitos colaterais
- Mesmo input, mesmo output
- Builds confiáveis e previsíveis

Interpretação sob demanda

- Avaliação preguiçosa: carrega apenas o que está sendo avaliado
- Não compila; interpreta a expressão e gera uma descrição do que construir

Controle Declarativo

- Tudo como código: do pacote ao servidor
- Fácil versionar, auditar e automatizar

Teoria da Computação

DESVANTAGENS

Curva de Aprendizado Íngreme

- Baseada em paradigmas como programação funcional pura e avaliação preguiçosa
- Difícil para quem vem de linguagens imperativas (Python, JavaScript, C++)

Documentação Dispersa

- Recursos espalhados entre blogs, wikis e repositórios
- Aprendizado autodidata pode ser frustrante

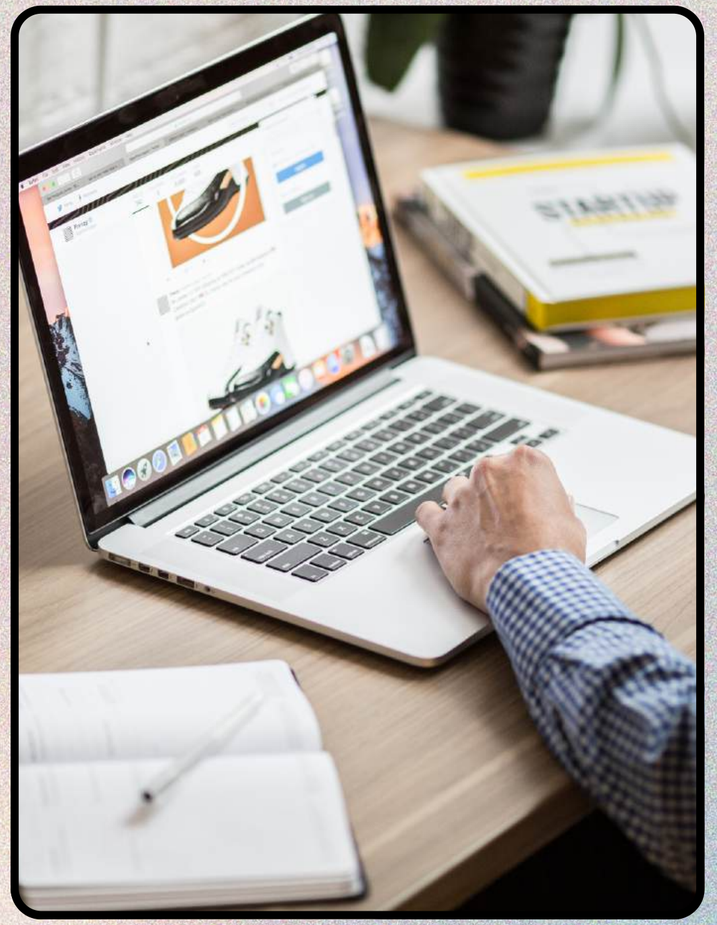
Ferramentas em Evolução

- Ecossistema ativo e inovador
- Mudanças frequentes podem causar quebras e exigir manutenção

CARACTERÍSTICAS GERAIS

- **EDIÇÃO**
  - Extensão: .nix
  - Recursos com plugins:
    - Autocompletar (nixpkgs)
    - Verificação de erros
- **COMPILAÇÃO**
  - Tipo: Interpretada (avaliação preguiçosa)
  - Funcionamento:
  - Nix interpreta .nix e gera derivações (build plans)
  - Ferramentas:
    - nix-instantiate → avalia arquivos
    - nix build → constrói pacotes/ambientes
    - nix eval → avalia expressões
    - nix repl → modo interativo
- **EXECUÇÃO**
  - Avaliada pelo motor Nix (em C++)
  - Pode gerar Strings, listas, conjuntos, derivações (receitas de build) e ambientes interativos (nix-shell, nix develop)

EXEMPLOS



EXEMPLOS

Função lambda → n + 1

```nix
# Função para incrementar um
  x: x + 1;
```

nix eval --expr 'import (./inc.nix) 5

OU

```nix
# Função para incrementar
y = x: x + 1;
```

nix eval --expr 'import (./inc.nix).y 5

LAMBDA ABSTRAÇÃO

$$\lambda x.\, E$$

$$Exemplo: \lambda x.\, x + 1$$

LAMBDA APLICAÇÃO

$$(\lambda x.\, E)\, A$$

$$Exemplo: (\lambda x.\, x + 1)5$$

EXEMPLOS

```
# Função soma recursiva
sum = let
  sum = n: if n == 0 then 0 else n + sum (n - 1);
in
  sum;
```

IMPLEMENTAÇÃO

Linguagem turing decidível de duplo balanceamento L = {w | w => (a^n b^n)}}.

```
# Arquivo: doubleBalance.nix

# Algoritmo para processar uma linguagem turing decidível de duplo balanceamento
# L = { w | w => (a^n b^n)}.
# Avalia se, para uma dada string de entrada,
# há uma sequência de 'a's seguida por uma sequência de 'b's
# em que 'a's e 'b's têm a mesma quantidade.

# ACEITE: quantidade de 'a's == quantidade de 'b's (incluindo 0)
# REJEITE: quantidade de 'a's != quantidade de 'b's
# || ordem incorreta || caracteres indesejados
```

IMPLEMENTAÇÃO

```
let
  lib = import <nixpkgs/lib>;

  # Entrada: uma string str
  # Saída: true se str ∈ L (ou seja, está na forma a^n b^n), false caso contrário.
  isABBalanced = str:
    let
      chars = lib.stringToCharacters str; # Transforma a string em uma lista de caracteres

      # Função recursiva: conta quantos 'a's consecutivos existem no início da lista
      countPrefixAs = list:
        if list == [] then 0                                    # Caso base: lista vazia: retorna 0
        else if lib.head list == "a" then                       # Se o primeiro elemento é 'a'
          1 + countPrefixAs (lib.tail list)                     # Conta 1 e continua recursiv. com o resto da lista
        else 0;                                                 # Se não for 'a', para de contar

      # Função recursiva: conta quantos 'b's aparecem *depois* dos 'a's
      countSuffixBs = list:
        if list == [] then 0                                    # Lista vazia: retorna 0
        else if lib.head list == "a" then 0                     # Encontrou 'a' depois dos 'a's iniciais → rejeita
        else if lib.head list == "b" then                       # Se for 'b', conta 1 e continua
          1 + countSuffixBs (lib.tail list)
        else 0;                                                 # Qualquer outro caractere: ignora/termina contagem
```

IMPLEMENTAÇÃO

```
34        # prefixAs: número de 'a's consecutivos no início da string
35        prefixAs = countPrefixAs chars;
36
37        # suffixBs: número de 'b's consecutivos depois dos 'a's
38        suffixBs = countSuffixBs (lib.drop prefixAs chars);
39
40        # totalLength: número total de caracteres da string
41        totalLength = lib.length chars;
42      in
43        # A string é válida se:
44        # 1. A quantidade de 'a's e 'b's equivalem ao comprimento total da string
45        # 2. A quantidade de 'a's e 'b's é igual
46        (prefixAs + suffixBs == totalLength) && (prefixAs == suffixBs);
47
```

IMPLEMENTAÇÃO

```
47
48  in
49
50  # Testes para a função isABBalanced com diferentes strings
51  {
52    test1 = isABBalanced "aabbghgdhsfskhlkf";   # false — contém letras além de 'a' e 'b'
53    test2 = isABBalanced "a2abb3";              # false — contém números
54    test3 = isABBalanced "aaabb";               # false — número de 'a's ≠ número de 'b's
55    test4 = isABBalanced "bbaa";                # false — 'b's aparecem antes dos 'a's
56    test5 = isABBalanced "aaabbb";              # true  — válido: 3 'a's seguidos por 3 'b's
57    test6 = isABBalanced "aaaaaabbbbbb";        # true  — válido: 6 'a's e 6 'b's
58    test7 = isABBalanced "ab";                  # true  — válido: 1 'a', 1 'b'
59    test8 = isABBalanced "";                    # true  — string vazia é aceita (n = 0)
60  }
```

```
yabo@desktopYab:~/Projects/teoriaComputacao/teoria-nix$ nxe '(import ./doubleBalance.nix)'
{ test1 = false; test2 = false; test3 = false; test4 = false; test5 = true; test6 = true; test7 = true; test8 = true; }
yabo@desktopYab:~/Projects/teoriaComputacao/teoria-nix$
```

OBRIGADO!

MATHEUS HENRIQUE DA COSTA

YAN BALBINO NOGUEIRA