Uma Introdução à Linguagem Potigol

Leonardo Lucena

Esquenta SE4FP
2025

Potigol é ...

... Uma linguagem **moderna** para aprender a programar

Objetivos

- Poucos efeitos colaterais
- Expressividade
- Legibilidade
- Facilidade de escrever

Principais Características

Multiparadigma

- **Funcional**
- Orientada a Objetos
- Imperativa

Sintaxe Simples

Tipagem Estática

Inferência de Tipos

Palavras em Português

https://potigol.github.io

twitter.com/potigol

Metodologia de Desenvolvimento da Linguagem

- **Sintaxe**
  - Feedback dos alunos
  - Busca de uma forma natural para resolver problemas de programação
  - Meta: Sintaxe tão simples quanto Python e Ruby
- **Semântica**
  - Programação funcional em primeiro lugar
  - Algumas construções de programação imperativa e orientada a objetos quando necessário

Principais Influências

- Scala
- Ruby
- Haskell
- Portugol
- Pascal
- Python

Olá Mundo

```
# Olá Mundo
escreva "Olá Mundo!"

escreva "Qual o seu nome?"
nome = leia_texto
escreva "Olá, {nome}!"
```

https://codespaces.new/potigol/potigol-codespace?quickstart=1

Tipos Básicos

- **Inteiro**
  - `1, -2, 3, ...`
- **Real**
  - `1.5, -5.43, ...`
- **Lógico**
  - `verdadeiro, falso`
- **Texto**
  - `"Texto"`
- **Caractere**
  - `'c'`

Operações

```
1 + 1     # 2
3 - 1     # 2
4 * 3     # 12
5 / 2     # 2.5
5 div 3   # 1
5 mod 3   # 2

2 == 3    # falso
2 <> 3    # verdadeiro

verdadeiro e falso    # falso
verdadeiro ou falso   # verdadeiro
```

Tipagem

Tipagem Estática

```
a: Inteiro = 10
```

Tipagem Forte (mas não tão forte como em Python)

```
s = "abc" + 10   # válido em Potigol:  "abc10"
```

Inferência de Tipos

```
b = 20
```

Nulos não são permitidos

Variáveis (Valores)

Atribuição

```
val a: Inteiro = 10        # 'val' pode ser omitido
```

O tipo pode ser **inferido** (o valor atribuído à variável determina o tipo)

```
a = 10
```

Não pode haver uma nova atribuição à mesma variável

```
a = 10
a = a + 1  # erro de compilação
```

```
1 | a = 10
2 | a = a + 1

O valor 'a' já foi declarado antes.
Use outro nome.
linha: 2
```

Programação Funcional

- Funções como cidadãs de primeira classe
  - Funções
  - Funções Anônimas (lambda)
  - Funções de Alta ordem
  - Funções Recursivas
  - Tipo Função
- Ausência de Efeitos Colaterais
- Variáveis e valores imutáveis
- Listas
- Compreensão de Listas
- Casamento de Padrões (pattern matching)

Funções

Opcional

Opcional, pode ser inferido

```
def soma(x: Inteiro, y: Inteiro): Inteiro = x + y

soma(x, y: Inteiro) = x + y      # Forma compacta

soma(x, y: Inteiro)              # Multi-linha
  c = x + y
  retorne c
fim

soma(x, y: Inteiro)              # A última expressão é o retorno
  x + y
fim
```

Opcional

Funções são cidadãs de primeira classe

Funções são tratadas da mesma forma que valores dos demais tipos:

- Podem ser **anônimas**
  - `10` `# Inteiro`
  - `(x, y: Inteiro) => x + y` `# Função`
- Podem ser **atribuídas a variáveis**
  - `a = 10`
  - `f = soma`
  - `f = (x, y: Inteiro) => x + y`
- **Têm tipo**
  - `10` `# Inteiro`
  - `"ABC"` `# Texto`
  - `(x, y: Inteiro) => x + y` `# (Inteiro, Inteiro) => Inteiro`
  - `dobro(x: Inteiro) = 2.0 * x` `# Inteiro => Real`

Funções são cidadãs de primeira classe

Funções são tratadas da mesma forma que valores dos demais tipos:

- Podem ser **passadas como parâmetro** para outras funções
  - `k(10)`
  - `g(soma)`
  - `g((x, y: Inteiro) => x + y)`
- Podem ser o **Resultado de funções**
  - `m(a: Inteiro) = 2 * a`
  - `h(a: Inteiro) = se a mod 2 == 0 então soma senão mult fim`

Funções Anônimas

Definição

```
(x, y: Inteiro) => x + y
```

Uso

```
((x, y: Inteiro) => x + y)(2, 3)            # 5
soma = (x, y: Inteiro) => x + y
soma(2, 3)
```

Em alguns casos os **tipos podem ser inferidos**

```
[1,2,3,4,5].selecione((a: Inteiro) => a > 3)     # [4, 5]
[1,2,3,4,5].selecione(a => a > 3)                # [4, 5]
```

Funções de Alta ordem

Podem receber funções como parâmetro

```
é_par(x: Inteiro): Lógico = x mod 2 == 0
[1, 2, 3, 4, 5, 6].selecione(é_par)                    # [2, 4, 6]
f(x: Inteiro, g: Inteiro => Inteiro) = g(2 * x)
```

Podem devolver funções como resultado

```
h(x: Inteiro): Inteiro => Inteiro = (y: Inteiro) => x + y

h(3)(4)   # 7
```

Funções Recursivas

```
fatorial(x: Inteiro): Inteiro =
  se x <= 0 então
    1
  senão
    x * fatorial(x - 1)
  fim
```

Nas funções recursivas o tipo de retorno é obrigatório

Listas

Lista são **imutáveis**

```
lista = 1 :: 2 :: 3 :: 4 :: 5 :: []
numeros = [1, 2, 3, 4, 5]
pares: Lista[Inteiro] = [2, 4, 6, 8]

lista.cabeça                    # 1
lista.cauda                     # [2, 3, 4, 5]
nova_lista = 0 :: lista

primos = [2, 3, 5, 7, 11, 13]   # listas começam na posição 1
escreva primos[3]               # 5
a = leia_inteiros(" ")          # Lê números inteiros na mesma linha
```

Listas

Funções de alta ordem

- *mapeie:* *Mapeia uma lista em outra usando uma função*
- *selecione:* *Seleciona os elementos que satisfazem um predicado (T => Lógico)*
- *injete:* *Realiza uma operação (fold) com todos os elementos*

```
numeros = [1, 2, 3, 4, 5, 6]
numeros.mapeie(a => 3 * a)                # [3, 6, 9, 12, 15, 18]
       .selecione(a => a mod 2 == 0)      # [6, 12, 18]
       .injete(0)((a, b) => a + b)        # 36  ==  0 + 6 + 12 + 18
```

Tuplas

```
joão : (Texto, Inteiro) = ("João", 21)    # O tipo pode ser inferido
notas = ("João", 8, 9)                    # tipo: (Texto, Inteiro, Inteiro)

escreva joão.primeiro                     # "João"
escreva joão.segundo                      # 21

f(p: (Texto, Inteiro, Inteiro)) = p.segundo + p.terceiro
f(notas)                                  # 5
```

Compreensão de Listas

Permite definir listas

```
pares = para i de 1 até 6 gere i * 2 fim                    # [2, 4, 6, 8, 10, 12]

pitagoras = para x de 1 ate 15,
                y de x até 15,
                z de y + 1 até 15 se x^2 + y^2 == z^2 gere
                    (x, y, z)
            fim                         # [(3,4,5), (5,12,13), (6,8,10), (9,12,15)]

soma = para (x, y, z) em pitagoras gere x + y + z fim       # [12, 30, 24, 36]
```

Se

Se é uma expressão

```
a = se x mod 2 == 0
      então "par"
      senão "ímpar"
fim
```

Podemos encadear com senãose

```
b = se      x > 0 então "positivo"
    senãose x < 0 então "negativo"
                  senão "zero"
fim
```

Casamento de Padrões

Escolha é uma expressão: retorna um valor

```
x = 2
a = escolha x
  caso 0           =>  0
  caso n se n > 0 =>  1      # ‘se n > 0’ é uma guarda (condição)
  caso _           => -1
fim

soma(lista: Lista[Inteiro]): Inteiro = escolha lista
  caso []    => 0
  caso a::as => a + soma(as)
fim
```

Orientação a Objetos

Orientação a Objetos

```
tipo Pessoa                              # Classe
  nome: Texto                            # atributo
  nascimento: Inteiro
  ano_atual = 2025
  idade() = ano_atual - nascimento       # método
fim

joão = Pessoa("Joao", 2002)
escreva joão                        # Pessoa("Joao", 2002)
escreva joao.idade                  # 23
```

Orientação a Objetos com Casamento de Padrões

```
a = Pessoa("Joao", 2002)
msg = escolha a
  caso Pessoa("Joao", _)                => "Olá Joãozinho!"
  caso Pessoa(nome, ano) se ano <= 2000 => "{nome}, você nasceu no século passado!"
  caso Pessoa(nome, ano)                => "{nome}, voce tem {a.idade} anos!"
fim
```

Tipos Abstratos (Interfaces)

```
tipo abstrato Animal
  nome: Texto
fim

tipo Cachorro: Animal
  nome: Texto
  idade: Inteiro
fim

tipo Gato: Animal
  nome: Texto
  peso: Real
fim
```

```
a = Cachorro("Rex", 5)
b = Gato("Tom", 3.2)

ola(a: Animal) = escolha a
  caso Cachorro(nome, idade) =>
    "Eu sou {nome}, tenho {idade} anos."
  caso Gato(nome, peso) =>
    "Eu sou {nome}, peso {peso} kg."
  caso x => "Meu nome é {x.nome}."
fim

ola(a)   # Eu sou Rex, tenho 5 anos.
ola(b)   # "Eu sou {Tom}, peso {peso} kg."
```

Programação Imperativa

Programação Imperativa

- Efeitos colaterais
- Variáveis mutáveis
- Comandos
- Registros

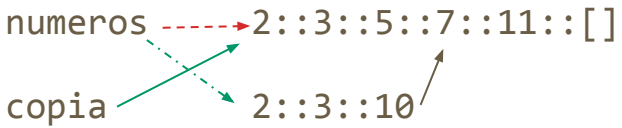
Variáveis mutáveis

Declaração e **atribuição** de valor a variáveis

```
var a := 10
a := a + 1
```

Listas (ainda são imutáveis)

```
var numeros := [2, 3, 5, 7, 11]
var copia := numeros
numeros[3] := 10
escreva numeros          # [2, 3, 10, 7, 11]
escreva copia            # [2, 3, 5, 7, 11]
```



Comandos

Se, Escolha e Para agora **são comandos**

```
se x mod 2 == 0 então
  escreva "par"
senão
  escreva "ímpar"
fim

para i de 1 até 5 faça
  escreva i
fim
```

```
escolha x
  caso 0          => escreva 0
  caso n se n > 0 =>
    var a := n + 1
    escreva "O próximo número é {a}."
  caso n          =>
    var a := n - 1
    escreva "O número anterior é {a}."
fim
```

Comando Enquanto

Depende de Efeitos colaterais

```
var a:= 15
enquanto a mod 7 <> 0 faça    # ‘a’ precisa mudar de valor para sair do laço
  se a mod 2 == 0 então
    a := a + 1
  senão
    a := 2 * (a - 3)
  fim
fim
escreva a
```

Site:

https://potigol.github.io/

Exemplos de programas:

https://potigol.github.io/beecrowd

Biblioteca de Jogos:

https://potigol.github.io/Jerimum/

Convite

- **Experimente** a linguagem Potigol e**compartilhe** suas experiências.
- Relatos de uso são muito bem-vindos!
- Dúvidas e sugestões:
  - leonardo.lucena@escolar.ifrn.edu.br
  - github.com/potigol

https://codespaces.new/potigol/potigol-codespace?quickstart=1

Exemplos

Olá Mundo!

```
# Funcional
escreva (nome => "Olá {nome}!")("Mundo")

# Imperativo
escreva "Olá Mundo!"

# Orientado a Objetos
tipo Ola
  nome: Texto
  saudação() = "Olá {nome}!"
fim
ola_mundo = Ola("Mundo")
escreva ola_mundo.saudação
```

Fibonacci

```
# Ineficiente
fib(n: Inteiro): Inteiro = n escolha
  caso 0 => 1
  caso 1 => 1
  caso a => fib(a-1) + fib(a-2)
fim
```

Quicksort

```
quicksort(nums: Lista[Inteiro]): Lista[Inteiro] = escolha nums
  caso [] => []
  caso pivot::resto =>
    esq = resto.selecione(_ <= pivot)
    dir = resto.selecione(_ > pivot)
    quicksort(esq) + [pivot] + quicksort(dir)
fim
```