Potigol: Linguagem de Programação moderna para iniciantes

Características:

- Projetada para ser usada por alunos iniciantes;
- Sintaxe Simples;
- Tipagem estática e forte com inferência de tipos;
- Palavras-chave em português;
- Multiparadigma;
- Estímulo ao paradigma funcional: valores imutáveis, casamento de padrões, funções como valores;
- Linguagem brasileira;

A Linguagem

A linguagem ***Potigol***, assim como a maioria das linguagens, possui variáveis, estrutura de controle: `se, senão, senão se, escolha`; estrutura de repetição: `para, enquanto;` funções, tipos de dados; além de conceitos de programação funcional como: *filter, fold, map,* expressões lambdas, list comprehension, funções de alta ordem (*High Order Function*), *Currying*, Recursão em cauda otimizada e casamento de padrões.

Palavras reservadas

tipos: Inteiro, Real, Texto, Lógico, Logico
métodos: inverta, cabeça, ordene, Lista, Matriz, Cubo, inteiro, arredonde, texto, real, tamanho, posição, posiçao, posicão, posicao, contém, contem, maiúsculo, maiusculo, minúsculo, minusculo, inverta, divida, lista, cabeça, cabeca, cauda, último, ultimo, pegue, descarte, selecione, mapeie, descarte_enquanto, pegue_enquanto, ordene, junte, insira, remova, mutável, mutavel, imutável, imutavel, vazia, injete, primeiro, segundo, terceiro, quarto, quinto, sexto, sétimo, setimo, oitavo, nono, décimo, decimo.
funções: leia_inteiro, leia_inteiros, leia_real, leia_reais, leia_texto, leia_textos, sen, cos, tg, aleatório.
palavras: var, se, senão, senãose, senao, senaose, então, entao, enquanto, faça, faca, de, em, e, escolha, tipo, até, passo, gere, retorne, use.

declaração de objeto imutável

```
x = 10                  # Declaração de um valor fixo (não pode ser alterado)
y, z = 20               # Mais de uma variável recebe o mesmo valor y = 20 e z = 20
a, b, c = 1, 2, 3       # Declaração paralela: a = 1, b = 2 e c = 3
```

declaração de objeto mutável

```
var y := 10             # Declaração de uma variável alterável
y := y + 2              # Atribuição de um valor a uma variável
var a, b, c := 1, 2, 3 # Declaração paralela: var a := 1, var b := 2 e var c := 3
a, b, c := b, a, 4      # Atribuição paralela: a := 2, b := 1 e c := 4
```

Tipos Básicos

Tipo	Valores
Inteiro	-4, 0, 5, ...
Real	-7.23, 0.0, 5.25, ...
Texto	"texto", "ola", "mundo", ...
Lógico	verdadeiro e falso

Caractere	'a', '4', '&', ...

Operações Aritméticas

```
5 + 3         # Soma: 8
5 - 3         # Subtração: 2
5 * 3         # Multiplicação: 15
5 / 3         # Divisão real: 1.66667
5 div 3       # Divisão inteira: 1
5 mod 3       # Resto da divisão: 2
```

Operações Lógicas e Relacionais

```
                        # Valores lógicos: verdadeiro, falso
verdadeiro e falso                  # e lógico              : falso
verdadeiro ou falso                 # ou lógico             : verdadeiro
não verdadeiro                      # não lógico            : falso

2 == 3                              # teste de igualdade    : falso
2 <> 3                              # teste de desigualdade : verdadeiro
2 < 3                               # menor                 : verdadeiro
2 <= 3                              # menor ou igual        : verdadeiro
2 > 3                               # maior                 : falso
2 >= 3                              # maior ou igual        : falso
```

Entrada de dados

```
a = leia_inteiro                # lê um número inteiro do teclado
b = leia_real                   # lê um número real do teclado
c = leia_texto                  # lê um texto do teclado
x, y = leia_inteiro             # lê 2 inteiros, o mesmo que x = leia_inteiro, y = leia_inteiro
números = leia_inteiros(5)      # lê um lista de 5 números inteiros, um por linha
números = leia_inteiros(",")    # lê uma lista de números inteiros separados por vírgula
```

Saída de dados

```
escreva "Olá Mundo"   # Escreve e passa para a próxima linha
imprima "Olá "        # Escreve e continua na mesma linha
escreva "Mundo"

nome = "Mundo"
escreva "Olá {nome}!"  # "Olá Mundo!" # use chave para imprimir o conteúdo de uma variável
```

Estrutura de decisão: Se, senão, senãose

```
x = leia_inteiro

# se ... então ... fim
se x > 5 então
  escreva "Maior do que cinco."
fim

# se ... então ... senão ... fim
se x > 5 então
  escreva "Maior do que cinco."
senão
  escreva "Menor ou igual a cinco."
```

```
fim

se verdadeiro então                # escreva "verdadeiro"
  escreva "verdadeiro"
senão
  escreva "falso"
fim

se falso então                     # escreva "falso"
  escreva "verdadeiro"
senão
  escreva "falso"
fim

# se ... então ... senãose ... senão ... fim
se x > 8 então
  escreva "Maior do que oito."
senãose x > 6 então
  escreva "Maior do que seis."
senãose x > 4 então
  escreva "Maior do que quatro."
senãose x > 2 então
  escreva "Maior do que dois."
senão
  escreva "Menor ou igual a dois."
fim

# usando se como uma expressão
a = se x mod 2 == 0 então "par" senão "ímpar" fim

maior = se a >= b e a >= c então a senãose b > c então b senão c fim
```

Estrutura de decisão: escolha

```
x = leia_inteiro
escolha x
  caso 1 => escreva "Um"               # se x == 1
  caso 2 => escreva "Dois"             # se x <> 1 e x == 2
  caso 3 => escreva "Três"             # se x <> 1 e x <> 2 e x == 3
  caso _ => escreva "Outro valor"      # se x <> 1 e x <> 2 e x <> 3
fim

# escolha com condições
escolha x
  caso n se n < 0        => escreva "{n} é negativo"
  caso n se n mod 2 == 0 => escreva "{n} é par"
  caso n                 => escreva "{n} é ímpar"
fim

# usando escolha como uma expressão
é_zero = escolha x
  caso 0 => verdadeiro
  caso _ => falso
fim
```

```
sinal = escolha x               # escolha retorna um número: -1, 0 ou 1
  caso n se n < 0 => -1
  caso n se n > 0 =>  1
  caso _          =>  0
fim
```

Estrutura de Repetição: para

```
para i de 1 até 10 faça            # escreve os números de 1 a 10
  escreva i
fim

var soma := 0
para i de 1 até 10 faça            # soma os números de 1 a 10
  soma := soma + i
fim

escreva "A soma é {soma}."
para i de 1 até 10 passo 2 faça    # escreve os números ímpares de 1 a 10
  escreva i
fim

# Para decrescente
para i de 10 até 1 passo -1 faça   # escreve os números de 10 a 1
  escreva i
fim

# Para com mais de um gerador
para i de 1 até 4,
     j de 1 até 3 faça             # escreve a tabuada {1..4} x {1..3}
  escreva "{i} * {j} == {i * j}"
fim

# Para com listas
cores = ["azul", vermelho", "verde"]
para cor em cores faça
  escreva cor
fim

# Para gerando uma lista
numeros = para i de 1 até 5 gere i fim             # [1, 2, 3, 4, 5]

pares = para i de 1 até 10 se i mod 2 == 0 gere i  # [2, 4, 5, 6, 8, 10]
```

Estrutura de Repetição: enquanto

```
var i := 0
enquanto i<=10 faça  # Escreve os números de 1 a 10
  escreva i
  i := i + 1
fim
```

Funções

```
soma(x: Inteiro, y: Inteiro) = x + y    # Declaração de função em uma linha

soma(x, y: Inteiro) = x + y             # Agrupando parâmetros do mesmo tipo

rep(a: Texto, n: Inteiro) = a * n       # Funções com parâmetros de tipos diferentes

a, b = leia_inteiro
c = soma(a, b)                          # Aplicando a função
escreva "{a} + {b} = {c}"

soma(x, y: Inteiro): Inteiro = x + y    # O tipo de retorno pode ser definido explicitamente

soma(x, y: Inteiro)                     # Declaração de função com corpo
  c = x + y
  retorne c                             # A última linha tem o valor de retorno
fim

soma(x, y: Inteiro)                     # Declaração de função com corpo
  c = x + y
  c                                     # A palavra 'retorne' é opcional
fim

fatorial(n: Inteiro): Inteiro           # Função recursiva (tipo de retorno é obrigatório)
  se n <= 1 então
    1
  senão
    n * fatorial(n - 1)
  fim
fim
a = leia_inteiro
escreva "Fatorial de {a} é {fatorial(a)}"

f(a: Inteiro)
  g(b: Inteiro) = b * 2                 # Função interna
  retorne g(a) + 3
fim
```

Tipos

```
Número (Inteiro e Real)
12345.qual_tipo                 # "Inteiro"
12345.real                      # 12345.0
12345.texto                     # "12345"
97.caractere                    # 'a'
12345 formato "%8d"             # "   12345"

12345.678.qual_tipo             # "Real"
12345.678.inteiro               # 12345
12345.678.texto                 # "12345.678"
12345.678.arredonde             # 12346
12345.678.arredonde(2)          # 12345.68
123.45 formato "%.1f"           # "123.4"

12345.678.piso                  # 12345.0 (arredonda para baixo)
12345.678.teto                  # 12346.0 (arredonda para cima)
12345.678.inteiro               # 12345
```

Texto

```
"abc".qual_tipo                  # "Texto"
"123".inteiro                    # 123
"12abc3".inteiro                 # 12
"abc".inteiro                    # 0
"abc"[2]                         # 'b' (caractere na posição 2)

"12.3".real                      # 12.3
"12a.3".real                     # 12.0
"abc".real                       # 0.0

"ab" + "cd"                      # "abcd"  (concatenação)
"abcb" - "bd"                    # "acb"   (subtração)

"abc".tamanho                    # 3
"abc".posição('b')               # 2 (posição de 'b' em "abc")
"abc".posição('d')               # 0
"abc".contém('a')                # verdadeiro (testa de 'a' está em "abc")
"abc".contém('d')                # falso

"Abc".maiúsculo                  # "ABC"
"Abc".minúsculo                  # "abc"
"Abc".inverta                    # "cbA"
"cab".ordene                     # "abc"
"abc".junte("-")                 # "a-b-c"
"abc".junte("[", ", ", "]")      # "[a, b, c]"

"Um texto".divida                # ["Um", "texto"]
"Um texto".divida("t")           # ["Um ", "ex", "o"]
"Um texto".lista                 # ['U', 'm', ' ', 't', 'e', 'x', 't', 'o']

"abc".cabeça                     # 'a'  (primeiro caractere de "abc")
"abc".cauda                      # "bc" ("abc" sem o primeiro caractere)
"abc".último                     # 'c'  (último caractere de "abc")
"abcde".pegue(3)                 # "abc" (primeiros 3 caracteres)
"abcde".descarte(3)              # "de"  (sem os primeiros 3 caracteres)

"abcb".selecione(letra => letra<>'c')          # "abb" ("abcb" sem 'c')
"abc".injete(0)((x,y) => x + y)                # 294   (97 + 98 + 99)
"abc".injete("")((x,y) => x + "-" + y)         # "-a-b-c"

"abcb".descarte_enquanto(letra => letra<>'c')  # "cb" (descarte caracteres antes de 'c')
"abcb".pegue_enquanto(letra => letra<'c')      # "ab" (pegue caracteres antes de 'c')

x = "abc".remova(2)              # x = "ac"  (remove o caractere na posição 2)
y = "abc".insira(3, 'd')         # y = "abdc" (insere 'd' na posição 2)
z = "abc".insira(3, "def")       # z = "abdefc" (insere "def" na posição 2)
```

Lista

```
[2, 4, 6, 8, 10]                    # lista literal
2 :: [4, 6, 8, 10]                  # [2, 4, 6, 8, 10]
[2, 4, 6, 8, 10].tamanho            # 5
[2, 4, 6, 8, 10].cabeça             # 2
[2, 4, 6, 8, 10].cauda              # [4, 6, 8, 10]
[2, 4, 6, 8, 10].último             # 10
[2, 4, 6, 8, 10].pegue(2)           # [2, 4]
[2, 4, 6, 8, 10].descarte(2)           # [6, 8, 10]
```

```
[2, 4, 6, 8, 10].inverta                 # [10, 8, 6, 4, 2]
[2, 6, 8, 10, 4].ordene                  # [2, 4, 6, 8, 10]
[2, 4, 6] + [8, 10]                      # [2, 4, 6, 8, 10]
[2, 4, 6].junte                          # "246"
[2, 4, 6].junte(", ")                    # "2, 4, 6"
[2, 4, 6].junte("[", ", ", "]")          # "[2, 4, 6]"

a = [2, 4, 6, 8, 10]
a[3]                                     # 6
a.posição(6)                             # 3
a.posição(12)                            # 0
a.contém(6)                              # verdadeiro
a.contém(12)                             # falso
a.remova(4)                              # [2, 4, 6, 10]
a.insira(3,5)                            # [2, 4, 5, 6, 8, 10]

Lista.imutável(5, 0)                     # [0, 0, 0, 0, 0]
Lista.vazia[Inteiro]                     # []   - Lista vazia de inteiros

# Matrizes e Cubos
a = [[1, 2], [3, 4]]                     # Matriz 2x2
a[2]                                     # [3, 4]
a[2][1]                                  # 3
b = Matriz.imutável(2, 2, 0)             # b == [[0, 0], [0, 0]]
c = Cubo.imutável(2, 2, 2, "-")          # c == [[["-", "-"],["-", "-"]],[["-", "-"],["-", "-"]]]
c[1][2][1]                               # "-"

# Listas mutáveis
a = Lista.mutável(5, 0)                      # [0, 0, 0, 0, 0].mutável
a[3] := 5                                    # a == [0, 0, 5, 0, 0].mutável

# Funções de alta-ordem
[2, 4, 6, 8, 10].selecione(n => n mod 4 == 0)  # [4, 8]
[2, 4, 6, 8, 10].mapeie(n => n div 2)          # [1, 2, 3, 4, 5]
[2, 4, 6].injete(0)((a, b) => a + b)           # 0 + 2 + 4 + 6 == 12
[2, 4, 6].injete((a, b: Inteiro) => a + b)     # 2 + 4 + 6 == 12
[2, 4, 6, 2, 4].pegue_enquanto(n => n < 6)     # [2, 4]
[2, 4, 6, 2, 4].descarte_enquanto(n => n < 6)  # [6, 2, 4]
```