Gramática da Linguagem Mini-Python

Discentes: Daniel Santos Lima, Davi Costa Martins e Rafael Barros Santos

// 1. Regra inicial

<program> ::= (<newline> | <stmt>)* <eof>

// Um programa MiniPython é uma sequência de statements (ou linhas em branco)

// terminando com o fim do arquivo (EOF). Não existe cabeçalho "program" nem ponto final.

// 2. Regras sintáticas (BNF/EBNF)

<stmt> ::= <simple_stmt> | <compound_stmt>

// Todo statement é simples (uma linha) ou composto (com bloco indentado).

<simple_stmt> ::= <small_stmt> <newline>

// Statements simples terminam com quebra de linha (sem ; no final).

<small_stmt> ::= <assignment-statement> | <print-statement>

// Apenas atribuição e print são statements simples suportados.

<assignment-statement> ::= <identifier> "=" <expression>

// Atribuição: identificador = expressão (sem atribuição múltipla nem +=, etc.).

<print-statement> ::= "print" <expression> {"," <expression>}

// Comando de saída: print expr1, expr2, ... (sem parênteses obrigatórios).

<compound_stmt> ::= <if-statement> | <while-statement>

// Estruturas de controle suportadas: apenas if-else e while.

<if-statement> ::= "if" <expression> ":" <suite> ["else" ":" <suite>]

// Comando condicional: if expressão : bloco [else : bloco].

// Blocos usam indentação (sem then/endif).

<while-statement> ::= "while" <expression> ":" <suite>

// Laço while: while expressão : bloco (sem else opcional).

<suite> ::= <simple_stmt> | <newline> INDENT <stmt>+ DEDENT

// Um bloco (suite) é:

// - um statement simples na mesma linha (one-liner)

// - ou nova linha + indentação + um ou mais statements + dedentação.

<expression> ::= <or-expression>

// Expressão geral (usada em condições e atribuições).

<or-expression> ::= <and-expression> {"or" <and-expression>}

// Operador lógico or (menor precedência).

<and-expression> ::= <not-expression> {"and" <not-expression>}

// Operador lógico and.

<not-expression> ::= "not" <not-expression> | <comparison>

// Operador lógico not (unário).

<comparison> ::= <simple-expression> {<relational-operator> <simple-expression>}

// Comparações (permite encadeamento como a < b < c).

<relational-operator> ::= "<" | ">" | "==" | ">=" | "<=" | "!="

// Operadores relacionais suportados.

<simple-expression> ::= <term> {("+" | "-") <term>}

// Expressão aritmética de nível + e -.

<term> ::= <factor> {("*" | "/") <factor>}

// Termo: multiplicação e divisão (sem % e //).

<factor> ::= ("+" | "-") <factor> | <atom>

// Fator: operadores unários + e - ou átomo.

<atom> ::= "(" <expression> ")" | <identifier> | <number> | <string-literal> | <call>

// Átomo: parênteses, variável, número, string ou chamada a built-in.

<call> ::= <builtin> "(" [<arglist>] ")"

// Chamada a funções built-in permitidas.

<builtin> ::= "input" | "int" | "float"

// Funções embutidas: input() → string, int() e float() para conversão.

<arglist> ::= <expression> {"," <expression>}

// Lista de argumentos para chamadas (separados por vírgula).

<identifier> ::= <letter> {<letter> | <digit> | "_"}

// Identificador: começa com letra ou _, seguido de letras, dígitos ou _.

<letter> ::= "A" | "B" | ... | "Z" | "a" | "b" | ... | "z" | "_"

// Letras maiúsculas, minúsculas e underscore.

<digit> ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

// Dígitos decimais.

<number> ::= <integer-number> | <real-number>

// Número inteiro ou ponto flutuante.

<integer-number> ::= <digit> {<digit>}

// Inteiro: sequência de dígitos (sem sinal aqui; sinal é tratado no factor).

<real-number> ::= <integer-number> "." [<digit> {<digit>}] | "." <digit> {<digit>}

// Real: com ponto decimal (ex: 3.14, .5, 42.).

<string-literal> ::= "" {<any-char-except-">} "" | "'" {<any-char-except-'} "'"

// String: delimitada por "..." ou '...' (sem escape complexo no escopo mínimo).

<newline> ::= NEWLINE

// Token de quebra de linha (gerenciado pelo lexer).

<eof> ::= EOF

// Fim do arquivo.

- Tokens léxicos importantes

-- IDENTIFIER   ::= [a-zA-Z_][a-zA-Z0-9_]*

-- INTEGER   ::= [0-9]+

-- FLOAT   ::= [0-9]* '.' [0-9]+ ([eE][+-]?[0-9]+)?

-- STRING_LITERAL ::= "" .*? "" | """ .*? """

-- INDENT ::= aumento consistente de espaços/tabs

-- DEDENT ::= redução consistente de espaços/tabs

// 3. Precedência e associatividade de operadores

// Ordem da menor para a maior precedência:

// 1) or

// 2) and

// 3) not

// 4) Operadores relacionais (< > == >= <= !=)

// 5) Operadores aritméticos binários + e -

// 6) Operadores aritméticos binários * e /

// 7) Operadores unários + e -

// 8) Parênteses (())

// Todos os operadores binários associam à esquerda.

// Exemplo: a - b - c é interpretado como (a - b) - c

// Exemplo: a * b + c é interpretado como (a * b) + c

// 4. Exemplos de programas VÁLIDOS

// 4.1. Exemplo 1 – Atribuição e Impressão Básica

```
x = 10
y = 5.5
z = "hello"
print x + y
print "O valor é:", x
```

// Demonstra atribuição de inteiros, floats e strings, e print com soma e múltiplos argumentos.

// 4.2. Exemplo 2 – Entrada e Conversão

```
x = int(input())

y = float(input())

print x, y
```

// Usa input() para ler do teclado, converte para int e float (built-ins obrigatórios), e imprime os valores.

// 4.3. Exemplo 3 – Estrutura Condicional (if-else)

```
if x > 0:
    print "positivo"
else:
    print "não positivo"
```

// Exemplo clássico de if-else com bloco indentado, condição relacional e strings.

// 4.4. Exemplo 4 – Estrutura de Repetição (while)

```
x = 0

while x < 10:
    x = x + 1
    print x
```

// Laço while com condição, incremento e print dentro do bloco indentado.

// 4.5. Exemplo 5 – Expressões Complexas

```
a = (10 + 5) * 2

b = not (a == 30 or a > 20)

if a != 30 and b:
    print "correto"
```

// Demonstra precedência de operadores (parênteses, *, +, ==, >, or, not, and, !=),

// atribuição, lógica e condicional.

// 5. Exemplos de programas INVÁLIDOS (com Erros de Sintaxe)

// 5.1. Inválido 1 – Falta de Dois-Pontos no if

```
if x > 0
    print "erro"
// Erro esperado: "Faltou ':' após a condição na linha X"
```

// 5.2. Inválido 2 – Indentação Inconsistente

```
if x > 0:
print "erro"  # Sem indentação
// Erro esperado: "Indentação esperada na linha X" (IndentationError)
```

// 5.3. Inválido 3 – Operador Não Suportado (ex: %)

```
x = 10 % 2
// Erro esperado: "Operador '%' não reconhecido" (token inválido no lexer/parser)
```

// 5.4. Inválido 4 – Chamada a Função Não Built-in

```
def soma(a, b):  # Funções não suportadas
    return a + b
// Erro esperado: "Palavra reservada 'def' não suportada" ou "Sintaxe inválida"
```

// 5.5. Inválido 5 – Atribuição Múltipla

```
x, y = 1, 2
// Erro esperado: "Atribuição múltipla não suportada" (não está na gramática)
```