



Universidade Federal
de Campina Grande

PROPOSTA DE PROJETO

DISCIPLINA: PARADIGMAS DE LINGUAGEM DE PROGRAMAÇÃO
PROFESSOR: EVERTON LEANDRO GALDINO ALVES

ANALISADOR SINTÁTICO PARA SUBLINGUAGEM DE PYTHON

ARTHUR FÉLIX DE SIQUEIRA (123210218)
ARTHUR RODRIGUES MARINHO DA SILVA (123210353)
DANIEL PEREIRA DAMIÃO (123210982)
JOÃO VITOR MOITINHO BARBOSA (123210551)

**UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
CIÊNCIA DA COMPUTAÇÃO**

Campina Grande, PB - 12/07/2025

1. OBJETIVO

O presente documento é a especificação de um projeto de codificação de um analisador léxico e sintático para uma “sublinguagem” de Python a ser desenvolvido nas linguagens: Haskell e Prolog.

2. ESCOPO DA LINGUAGEM

A linguagem suportada é um subconjunto da linguagem Python com as seguintes construções:

Elemento	Exemplos
Identificadores	x, valor_1, soma123
Atribuição simples	x = 1, x = valor_1, x = 3 * 8
Tipos de dados: inteiro, float.	10, 10.8, 1.098, 001
Operadores	+, -, *, /.
Símbolos	=, ., (,).

2.1 IDENTIFICADORES

- Começam com uma letra (a-z, A-Z) ou sublinhado (_).
- Podem conter dígitos após o primeiro caracter.
- Case-sensitive.
- Não podem conter espaços, acentos ou caracteres especiais.
- Exemplos válidos: x, _valor, soma123.
- Exemplos inválidos: 1x, x!, Área.

2.2 TIPOS DE DADOS

- Inteiros: 5, 42, 1023, 000
- Float: 3.14, 0.01, 8.0, 00.0
- Restrição: inteiros com zero à esquerda são inválidos (01, 001)

2.3 ATRIBUIÇÕES SIMPLES

- Formato: `identificador = expressão`
- Exemplo válido:

```
x = 1
resultado = x + 3 * (5 - 1)
```

2.4 OPERAÇÕES SUPORTADAS

- Soma: +
- Subtração: -

- Multiplicação: *
- Divisão: /

2.4 SÍMBOLOS RECONHECIDOS

- =, ., (,).
- Observação: o . é considerado símbolo, não operador. Ele não representa acesso a atributos (o suporte a POO não está sendo considerado).

3. FUNCIONALIDADES DO SISTEMA

3.1 ENTRADA

- Arquivo com extensão .py contendo um ou mais comandos escritos na sublinguagem suportada.
- Segue exemplo de código de entrada válido (exemplo1.py):

```
x = 3 + 4 * (5 - 2)
y = 2 * x
```

3.2 PROCESSAMENTO

- Tokenização do código-fonte.
- Análise sintática dos tokens com base em gramática definida.
- Geração de uma Árvore Sintática Abstrata (AST).
- Observação: para a construção do AST a ordem de precedência aritmética segue como descrito na [documentação oficial do python](#).

3.3 SAÍDA

- Se válido: representação da AST. Segue exemplo de saída para exemplo1.py.

```
Assignment
├── Identifier: x
└── Expression:
    ├── Number: 3
    ├── Operator: +
    └── Expression:
        ├── Number: 4
        ├── Operator: *
        └── Parenthesis:
            ├── Number: 5
            ├── Operator: -
            └── Number: 2
```

```
Assignment
├── Identifier: y
└── Expression:
    ├── Number: 2
    ├── Operator: *
    └── Identifier: x
```

- Se inválido: mensagem de erro indicando linha, posição e breve descrição do problema identificado.

- Perceba o `exemplo2.py` para entrada inválida:

```
x = (4 + 2
y + 2
```

- Saída para `exemplo2.py`:

```
SyntaxError: '(' was never closed, line 1
NameError: name 'y' is not defined, line 2
```

4. INTERAÇÃO COM O USUÁRIO

A interação com o sistema será feita via linha de comando, com os seguintes comandos principais:

- Analisar um arquivo:

```
./analizador '/path/para/arquivo.py'
```

- Analisar um arquivo e salva a sua AST em `./ast_results/arquivo.txt`; se houver qualquer erro sintático o arquivo não será salvo, mas a mensagem de erro correspondente impressa no terminal:

```
./analizador '/path/para/arquivo.py' -s
```

- Visualizar comandos suportados:

```
./analizador --help
```

- Executar testes internos (salva em `./test/test1_log.txt` o código conteúdo para cada teste e sua respectiva árvore sintática ou mensagem de erro cabível):

```
./analizador -tests -s
```

- Nomeia os testes disponíveis:

```
./analizador -tests -l
```

- Executar teste interno com exibição de resposta no terminal (código de entrada + AST/ErrorMsg), caso o teste nomeado esteja no banco de testes:

```
./analizador -test exemplo_1
```

- Executar teste interno salvando resposta em arquivo de log:

```
./analizador -test exemplo_1 -s
```

- Mensagens de erro para interação com usuário mal sucedida:

```
Comando não identificado, tente './analizador --help' para
consultar os comandos válidos.
```

5. MENSAGENS DE ERRO:

Segue as mensagens de erro possíveis para o escopo:

- Exemplos `x = (1`, `x = 1 + (` :

```
SyntaxError: '(' was never closed, line 'n'
```

Perceba que `x = ()`, `x = (1)`, `x = 1 + ()`, são entradas verificadas como sintaticamente corretas. `x = 1 + (` contém um erro de tipo e não de sintaxe

- Exemplos `x =)`, `x = 1 +)`, `x = 1 + (1))` :

```
SyntaxError: unmatched ')', line 'n'
```

- Exemplos `x`, `x = y` :

```
NameError: name 'y' is not defined, line 'n'
```

- Exemplos `=`, `x =`, `= x`, `x = +`, `x = 1 +` :

```
SyntaxError: invalid syntax, line 'n'
```

- Exemplos `x = 001` :

```
SyntaxError: leading zeros in decimal integer literals are
not permitted, line 'n'
```