**Análise Léxica – Python**

**Francisco Targa e Eduardo de Toledo**

Regra 1: STRING\_LITERAL

A regra léxica STRING\_LITERAL é responsável por reconhecer literais de strings no código Python e separá-los como tokens individuais para análise posterior pelo analisador sintático (parser).

**STRING\_LITERAL: '\'' (ESC | ~('\'' | '\\'))\* '\''**

**| '"' (ESC | ~('"' | '\\'))\* '"';**

**Entradas aceitas:**

String delimitada por aspas simples (''):

1. Entrada: 'Hello, world!'

Nesse caso, a string "Hello, world!" é delimitada por aspas simples, e não contém caracteres de escape.

String delimitada por aspas duplas (""):

1. Entrada: "Python is\nawesome!"

Nesse caso, a string "Python is\nawesome!" é delimitada por aspas duplas. A sequência \n é um caractere de escape que representa uma nova linha na string.

**Entradas rejeitadas:**

String não fechada:

Entrada: 'Hello, world!

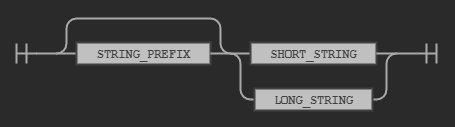
Nesse caso, a string é iniciada com uma aspa simples, mas não é fechada com outra aspa simples. Como resultado, a regra STRING\_LITERAL não conseguirá reconhecer essa string, pois não segue o formato de abertura e fechamento esperado.

String com caractere de escape inválido:

Entrada: "Invalid escape: \a"

Nesse caso, a string contém a sequência \a, que não é um caractere de escape válido em Python. A regra STRING\_LITERAL só reconhece sequências de escape válidas, como \n, \t, \\, \", entre outras. Portanto, essa string seria rejeitada.

**Diagrama de Sintaxe Equivalente (Railroad)**

****

Regra 2: DIGIT

A regra DIGIT é definida usando a notação de intervalo [0-9], o que significa que ela corresponde a qualquer caractere que seja um dígito numérico de 0 a 9. Essa regra é usada para identificar dígitos em várias partes do código Python, como números inteiros e números de ponto flutuante. Quando o analisador léxico encontra um dígito na entrada, ele o reconhece como um token DIGIT e o passa para o analisador sintático para posterior processamento.

**fragment DIGIT : [0-9];**

**Entradas aceitas:**

1. Número Inteiro:

Entrada: 12345

Nesse caso, todos os dígitos 1, 2, 3, 4 e 5 são reconhecidos como tokens DIGIT, e a sequência completa é interpretada como um número inteiro.

1. Número de Ponto Flutuante:

Entrada: 3.14

Os dígitos 3, 1 e 4 seriam reconhecidos como tokens DIGIT. Essa sequência pode ser usada para formar um número de ponto flutuante, por exemplo, 3.14.

**Entradas rejeitadas:**

Entrada 1: Número Negativo

x = -42

Neste exemplo, o sinal de menos - não é reconhecido pela regra DIGIT. A regra DIGIT reconhece apenas os dígitos numéricos de 0 a 9, mas não reconhece operadores como o sinal de menos.

Entrada 2: Número com Separador de Milhares

y = 1,000

Aqui, o número 1,000 contém uma vírgula como separador de milhares. A regra DIGIT não reconhece vírgulas ou outros caracteres que não sejam os dígitos numéricos de 0 a 9.

**Diagrama de Sintaxe Equivalente (Railroad)**

****

Regra 3: TYPE\_COMMENT

A regra TYPE\_COMMENT é responsável por reconhecer comentários relacionados a tipos (type hints) no código Python. Type hints são uma parte da sintaxe do Python que permitem especificar o tipo de uma variável ou retorno de função, embora o Python continue sendo uma linguagem de tipagem dinâmica.

**TYPE\_COMMENT**

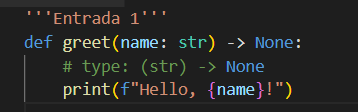
**: '#' WS? 'type:' ~[\r\n\f]\***

**;**

**Entradas aceitas:**

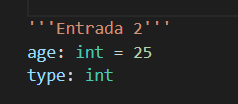
Entrada 1:

Type Hint em Função



Nesse exemplo, o comentário # type: (str) -> None está alinhado com a declaração da função e fornece informações sobre os tipos esperados dos argumentos e do tipo de retorno da função. Essa entrada seria reconhecida pela regra TYPE\_COMMENT.

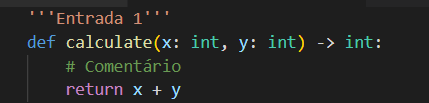
Entrada 2: Type Hint em Variável



Aqui, a variável age é declarada com um tipo de inteiro e, em seguida, o comentário # type: int é fornecido para indicar o tipo da variável. Isso também seria reconhecido pela regra TYPE\_COMMENT.

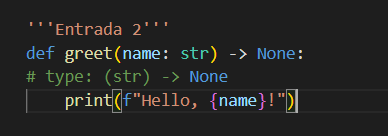
**Entradas rejeitadas:**

Entrada 1: Comentário sem Formato de Type Hint



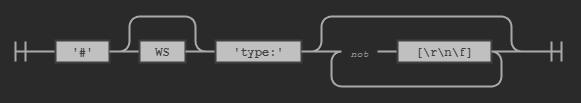
Neste exemplo, o comentário # Comentário não segue o formato de type hint esperado, que é # type: (...) -> .... Portanto, esse comentário não seria reconhecido pela regra TYPE\_COMMENT.

Entrada 2: Type Hint não Alinhado com Declaração



Nesse exemplo, o comentário # type: (str) -> None não está alinhado com a declaração da função. Os comentários de type hint devem estar alinhados para serem reconhecidos corretamente pela regra TYPE\_COMMENT. Portanto, essa entrada também seria rejeitada.

**Diagrama de Sintaxe Equivalente**

****