# PLY Syntax Validator

## Python Lex-Yacc Implementation

A practical application of lexical analysis and parsing to validate lambda functions, list comprehensions, class declarations, function declarations, and dictionary method calls.

```
lambda x: x * 2
[x for x in data]
class MyClass: pass
```

Compiler Design & Parsing

# Project Goal and Scope

## PRIMARY GOAL

### OBJECTIVE

Build a functional syntax validator for a targeted subset of the Python language using PLY (Python Lex-Yacc).

### TECHNOLOGY

PLY is a powerful library for implementing lexical analysis and parsing tools in Python, separating tokenization from grammar validation.

## TARGET CONSTRUCTS

- → Lambda Functions
- → List Comprehensions
- → Class Declarations
- → Function Declarations
- → Dictionary Methods

### SCOPE LIMITATION

The validator focuses on syntactic structure only and does not handle complex features like indentation, nested blocks, or semantic validation.

# The Two Pillars of PLY: Lexer and Parser

LEXER

## Lexical Analysis

Reads the input code and converts it into a stream of tokens (e.g., NAME, LAMBDA, COLON). This is the first stage of parsing.

Defined in: python_lexer.py

PARSER

## Syntactic Analysis

Takes the tokens and checks if their sequence matches the defined grammar rules. This validates the structure of the code.

Defined in: python_parser.py

### PROCESS PIPELINE

Input Code

↓

Lexer (Tokenization)

↓

Token Stream

↓

Parser (Grammar Validation)

↓

Valid Syntax / Syntax Error

# Tokenizing Python: Recognizing Keywords and Symbols

| CATEGORY | EXAMPLE TOKENS | DESCRIPTION |
|---|---|---|
| **Keywords** | DEF, CLASS, LAMBDA, FOR, IN, IF, RETURN, PASS | Reserved words with special meaning in the grammar |
| **Operators** | ASSIGN (=), COLON (:), DOT (.), PLUS (+), MODULO (%) | Symbols used for operations and structure |
| **Punctuation** | LPAREN, RPAREN, LBRACKET, RBRACKET, COMMA | Symbols used to group or separate elements |
| **Identifiers** | NAME | Variable names, function names, and class names |

EXAMPLE TOKENIZATION

```
Input: my_func = lambda a, b: a + b

Tokens: NAME, ASSIGN, LAMBDA, NAME, COMMA, NAME, COLON, NAME, PLUS, NAME
```

# Validation in Action: Testing the Constructs

## LAMBDA FUNCTION TESTS

```
my_func = lambda a, b: a + b
```

✓ Valid Syntax

```
my_func = lambda a, b a + b
```

✗ Syntax Error

## LIST COMPREHENSION TESTS

```
[x*2 for x in data if x > 5]
```

✓ Valid Syntax

```
[x*2 x in data]
```

✗ Syntax Error

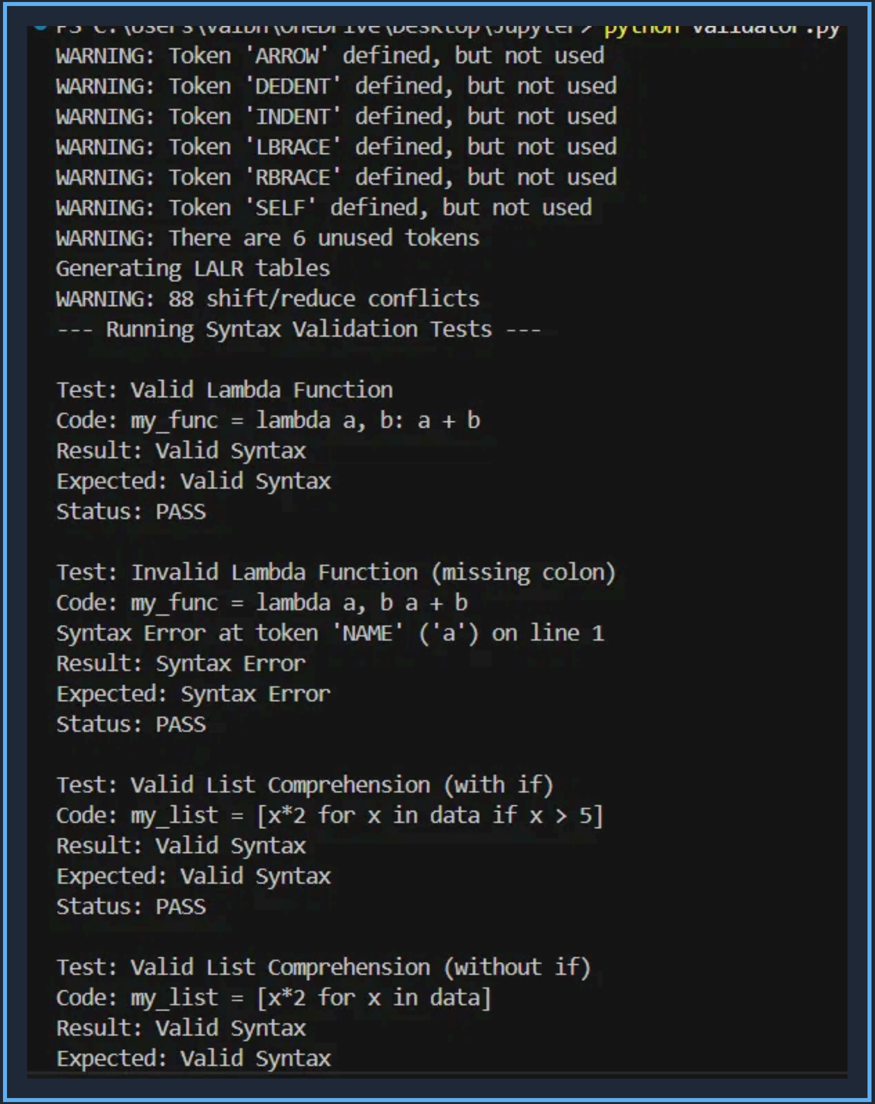
## CLASS DECLARATION TESTS

```
class MyClass: pass
```

✓ Valid Syntax

```
class MyClass pass
```

✗ Syntax Error



```
PS C:\Users\vaibh\OneDrive\Desktop\Jupyter> python validator.py
WARNING: Token 'ARROW' defined, but not used
WARNING: Token 'DEDENT' defined, but not used
WARNING: Token 'INDENT' defined, but not used
WARNING: Token 'LBRACE' defined, but not used
WARNING: Token 'RBRACE' defined, but not used
WARNING: Token 'SELF' defined, but not used
WARNING: There are 6 unused tokens
Generating LALR tables
WARNING: 88 shift/reduce conflicts
--- Running Syntax Validation Tests ---

Test: Valid Lambda Function
Code: my_func = lambda a, b: a + b
Result: Valid Syntax
Expected: Valid Syntax
Status: PASS

Test: Invalid Lambda Function (missing colon)
Code: my_func = lambda a, b a + b
Syntax Error at token 'NAME' ('a') on line 1
Result: Syntax Error
Expected: Syntax Error
Status: PASS

Test: Valid List Comprehension (with if)
Code: my_list = [x*2 for x in data if x > 5]
Result: Valid Syntax
Expected: Valid Syntax
Status: PASS

Test: Valid List Comprehension (without if)
Code: my_list = [x*2 for x in data]
Result: Valid Syntax
Expected: Valid Syntax
```

Actual validator test suite execution

# Project Success and Future Work

## PROJECT SUCCESS

### ✓ FUNCTIONAL VALIDATOR

Successfully implemented a PLY-based validator that accurately checks the syntax of all five specified Python constructs.

### ✓ COMPLETE TEST SUITE

Comprehensive test cases validate both correct acceptance of valid code and rejection of invalid syntax across all constructs.

### ✓ INTERACTIVE INTERFACE

Command-line interface enables real-time validation of user-provided code snippets with immediate feedback.

### LEARNING OUTCOME

Gained practical experience in compiler design principles, specifically lexical analysis, context-free grammars, and syntax validation using industry-standard tools.

## FUTURE ENHANCEMENTS

### → INDENTATION HANDLING

Integrate Python's indentation rules (`INDENT`/`DEDENT` tokens) to support multi-line blocks and nested structures.

### → GRAMMAR EXPANSION

Extend the grammar to cover additional Python features including loops, conditionals, exception handling, and dictionary/set literals.

### → SEMANTIC ANALYSIS

Implement semantic validation to check variable definitions, type consistency, and function signatures beyond structural syntax.

### → ERROR RECOVERY

Enhance error reporting with suggestions for common mistakes and automatic error recovery to continue parsing.

# Questions & Discussion

## Open Forum

Thank you for your attention. I welcome any questions you may have about the PLY syntax validator implementation, the design decisions, or the compiler design principles demonstrated in this project.

Let's explore the parsing process together