Mathematical Expression Analyzer

A complete Python implementation for analyzing, parsing, and comparing mathematical expressions with support for polynomial normalization and equivalence checking.

Features

Supported Expression Elements

- **Numbers**: Integer constants (positive and negative)
- **Variables**: Single-letter variables (x-z, X-Z)
- **Operators**:
  - Arithmetic: `+`, `-`, `*`, `/`
  - Exponentiation: `^` (with special handling for powers 2 and 3)
- **Functions**: `sin`, `cos`, `tan`, `ln`, `sqrt`
- **Implicit Multiplication**: `2x`, `x(y+1)`, `xy`, etc.
- **Parentheses**: Full support for grouping

Core Capabilities

1. **Lexical Analysis (Lexer)**
   - Tokenizes mathematical expressions
   - Automatically detects and inserts implicit multiplication
   - Handles multi-character functions and operators
   - **Important**: Function calls must include parentheses (e.g., `sin(x)`). Writing `sinx` will be parsed as `s*i*n*x`
2. **Syntax Analysis (Parser)**
   - Determines whether an expression is valid. Invalid cases include:
     1. Unary minus without proper parentheses (`--x`, `x^-2`, etc.)
     2. Mismatched parentheses (`((x)`, `(x))`, etc.)
     3. Invalid operator usage (`x ++ 2`, `()`, etc.)
   - Recursive descent parser with proper operator precedence
   - Precedence order:
     1. Functions (sin, cos, tan, ln, sqrt)
     2. Exponentiation (^)
     3. Multiplication and Division (*, /)
     4. Addition and Subtraction / Unary minus (+, -)
3. **Abstract Syntax Tree (AST)**
   - Comprehensive node types for all expression elements
   - Pretty-printing tree structure for debugging
4. **Polynomial Normalization**
   - Converts expressions into canonical polynomial form
   - Supports expansion of (expr)^2 and (expr)^3
   - Supports simplification of (expr)^0 or (expr)^1

- Treats non-expandable operations (/, functions, arbitrary powers) as atomic expressions
- Applies associativity and commutativity rules to simplify

5. **Equivalence Checking**
   - Determines if two expressions are mathematically equivalent (functional equivalence)
   - **Equivalence criterion**: Two expressions are equivalent if they produce the same result for all variable values
   - **Implementation strategy**:
     1. Polynomial normalization
     2. Structural equality checking (considering commutativity)
     3. Rational form simplification: converts to numerator/denominator form and uses cross-multiplication
   - Supports recognition of:
     - Additive commutativity: `1 + x` ≡ `x + 1`
     - Multiplicative commutativity: `x * y` ≡ `y * x`
     - Associativity (both addition and multiplication)
     - Distributivity and expansion: `(x + 1)^2` ≡ `x^2 + 2x + 1`
     - Rational equivalence: `1 - 1/x` ≡ `(x-1)/x`

Running

```
# Run from the project root directory
python main.py
```

This launches the interactive mode. After successful execution, the terminal will display the following interface:

```
======================================================================
  MATHEMATICAL EXPRESSION ANALYZER - INTERACTIVE MODE
======================================================================

Please select an option:
  1. Analyze single expression (Lexer -> Parser -> Polynomial)
  2. Check equality of two expressions
  3. Exit
----------------------------------------------------------------------
Enter your choice (1-3)
```

AST Visualization

Generate visual AST diagrams using Graphviz:

```
# Visualize a single expression
python visualize.py "x^2+2*x+1"

# Specify output filename
python visualize.py "x^2+2*x+1" -o my_expression.png

# Specify output format (png/pdf/svg/jpg)
python visualize.py "sin(x+y)" -f svg
```

```
# Custom title
python visualize.py "x+1" -t "My Custom Title"

# No title
python visualize.py "x+1" --no-title

# Compare two expressions side by side
python visualize.py "x+1" "1+x" --compare

# Run built-in examples
python visualize.py --examples

# View help
python visualize.py -h
```

> **Note**: Requires `graphviz` Python package and Graphviz system installation. See Dependencies.

Testing

```
# Run from the project root directory
python test.py
```

This will run all test cases in `test_cases.json` (129 cases in total) and generate a detailed test result log file.

Project Structure

```
Mathematical-Expression-Analyzer/
├── ast_nodes.py        # AST node definitions
├── lexer.py            # Lexical analyzer
├── parser.py           # Syntax analyzer
├── polynomial.py       # Polynomial normalization
├── equality.py         # Equivalence checking
├── visualize.py        # AST visualization tool
├── main.py             # Interactive mode entry point
├── test.py             # Automated test runner
├── test_cases.json     # Test case definitions
├── images/             # Generated AST visualizations(Some samples have already
been in)
└── README.md           # This file
```

Limitations

The implementation does NOT handle:

- Arbitrary algebraic identities (e.g., `sin²(x) + cos²(x) = 1`)
- Complex simplifications (e.g., `(1/x) * x = 1`)
- Non-constant exponents (powers other than 2 or 3)
- Automatic differentiation or symbolic integration
- Taylor series expansion

Dependencies

Required

None! The core functionality uses only Python standard library:

- `abc` - Abstract base classes
- `enum` - Enum types
- `typing` - Type hints

Optional (for AST Visualization)

To use the `visualize.py` tool, install the following:

```
# Python package
pip install graphviz

# System installation
# macOS:
brew install graphviz

# Ubuntu/Debian:
sudo apt-get install graphviz

# Windows:
# Download from https://graphviz.org/download/
# Add Graphviz bin directory to your PATH
```