Partial Derivative Calculator

Calculus III - Honors By Contract Project

Sadain Siddique

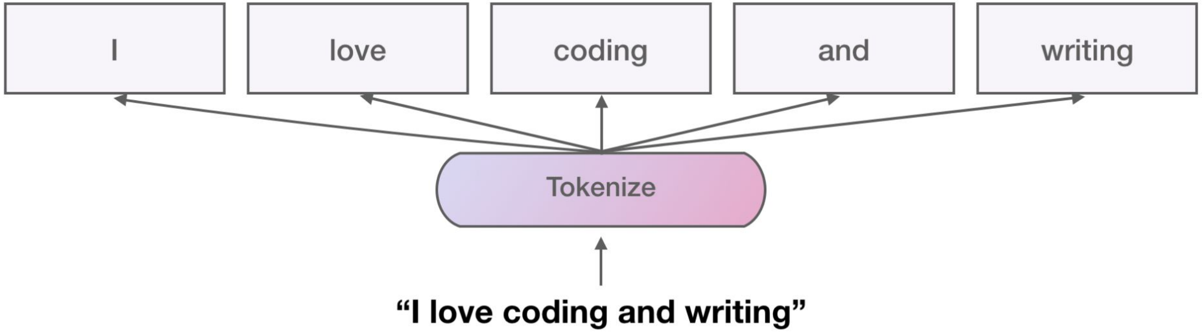
Importance of Partial Derivatives

- Derivative of multivariable function
- Allows us to define the gradient and directional derivative
- Real life example

Motivation

- Visualization and understanding
- Project-based learning
- Practice and reinforcement

Process - Tokenizer



Breaks a stream of text into tokens, usually by looking for whitespace (tabs, spaces, new lines).

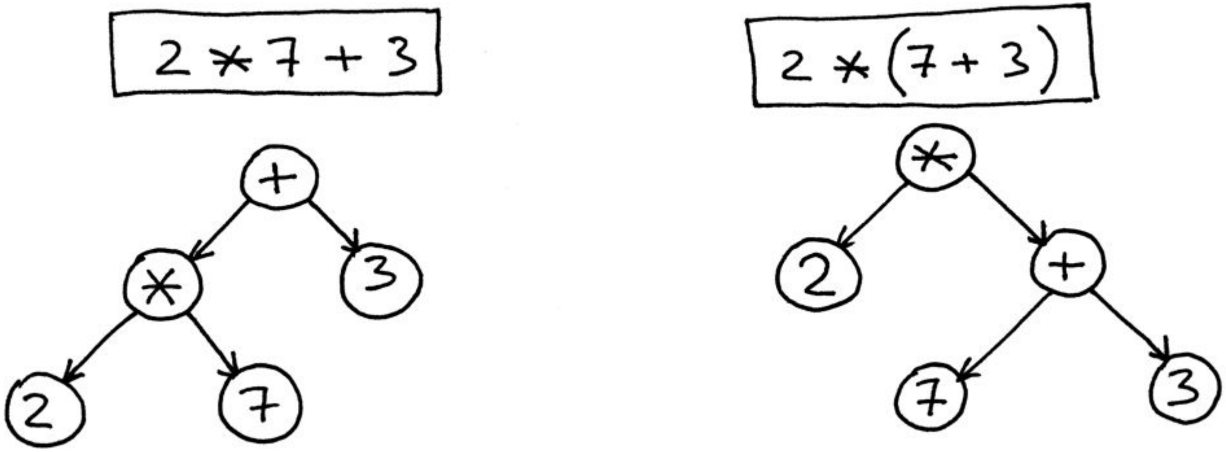
Process – Parser

Parsing token classifications

TOKEN	CLASS
x	identifier
+	addition operator
z	identifier
=	assignment operator
11	number

Takes the tokenized expression, verifies syntax, creates a structured representation (the AST) that can be further processed and evaluated.

Process - Abstract Syntax Tree



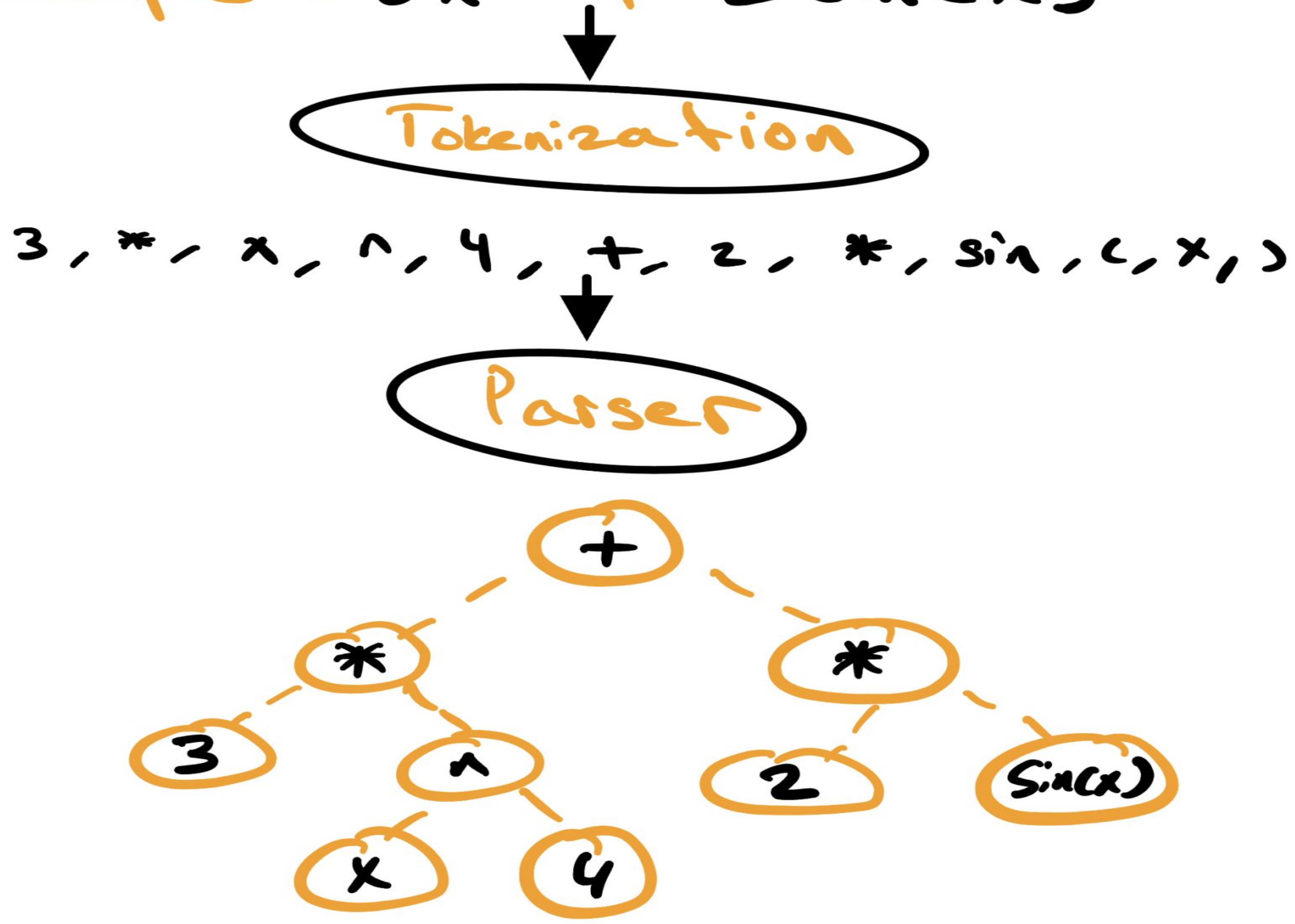
Organizes different components of an expression (numbers, variables, operations) into a branching tree format, where each node in the tree represents a part of the expression.

```java
private Operation getFunction(Token t) throws TokenizerException
{
    FunctionToken token = (FunctionToken) t;
    switch (token.getFunction().getName())
    {
        case "acos": return new Acos(getTree());
        case "asin": return new Asin(getTree());
        case "atan": return new Atan(getTree());
        case "log": return new Log(getTree());
        case "cos": return new Cos(getTree());
        case "sin": return new Sin(getTree());
        case "sqrt": return new Sqrt(getTree());
        case "tan": return new Tan(getTree());
        case "exp": return new Exp(getTree());
        case "abs": return new Abs(getTree());
        default: throw new TokenizerException("Function error");
    }
}
```

```java
private Operation getOperator(Token t) throws TokenizerException
{
    Operation right = getTree();
    Operation left = getTree();

    switch (((OperatorToken)t).getOperator().getSymbol())
    {
        case "+":   return new Addition(left, right);
        case "-":   return new Subtraction(left, right);
        case "*":   return new Product(left, right);
        case "/":   return new Division(left, right);
        case "^":   return new Pow(left, right);
        default: throw new TokenizerException("Function error");
    }
}
```

Example : $3x^{\wedge}4 + 2\sin(x)$



Derive each node

Term: $3 * x^{4}$

$3 \to 0$ constant rule

$x^{4} \to 4 * x^{3}$ power rule

$3x^{4} \to 12 * x^{3}$ constant & product rule

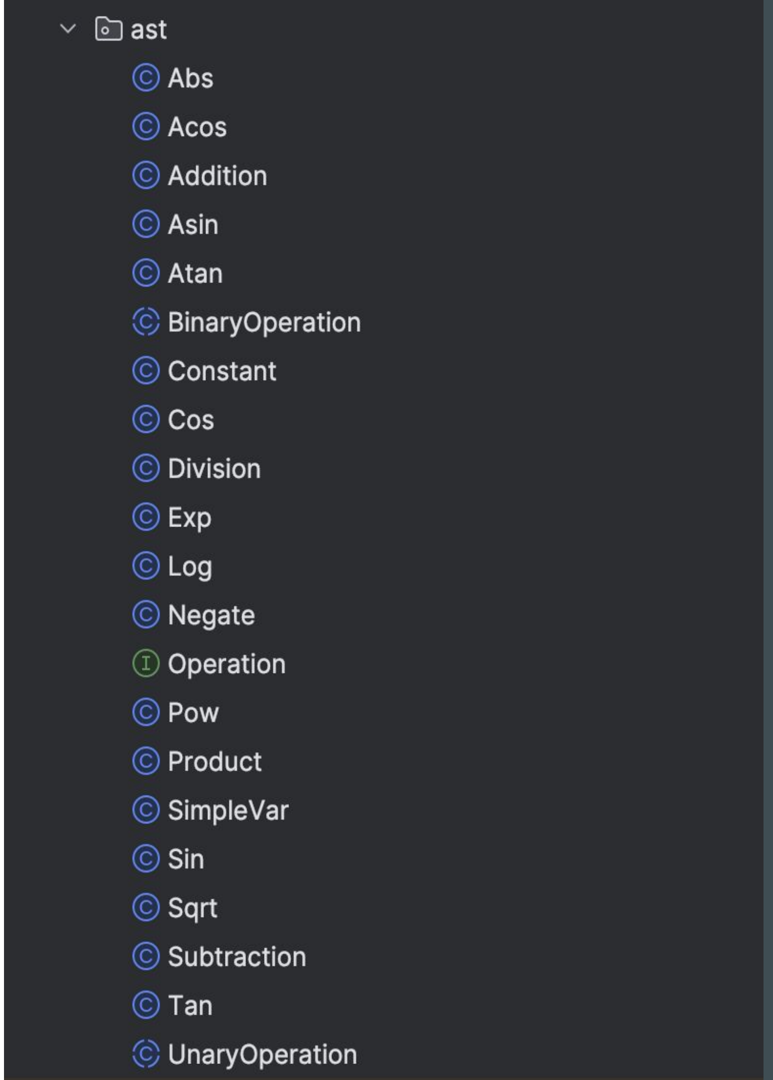
Term: $2 * \sin(x)$

$2 \to 0$ constant rule

$\sin(x) \to \cos(x)$ trig rule

$2 * \sin(x) \to 2 * \cos(x)$ constant & product rule

Derivative = sum of the derivative of both terms: $12 * x^{3} + 2 * \cos(x)$



Problem Solving

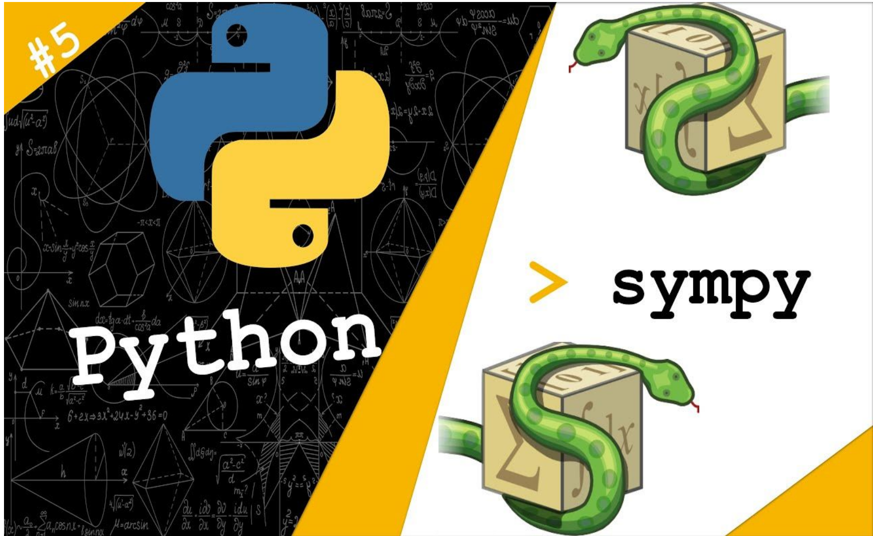
1. Complexity

2. Reliability and performance

3. Extensibility and Community Support

- SymPy is a Python library designed for symbolic mathematics

- Built-in support for recognizing and manipulating multiple symbolic variables.



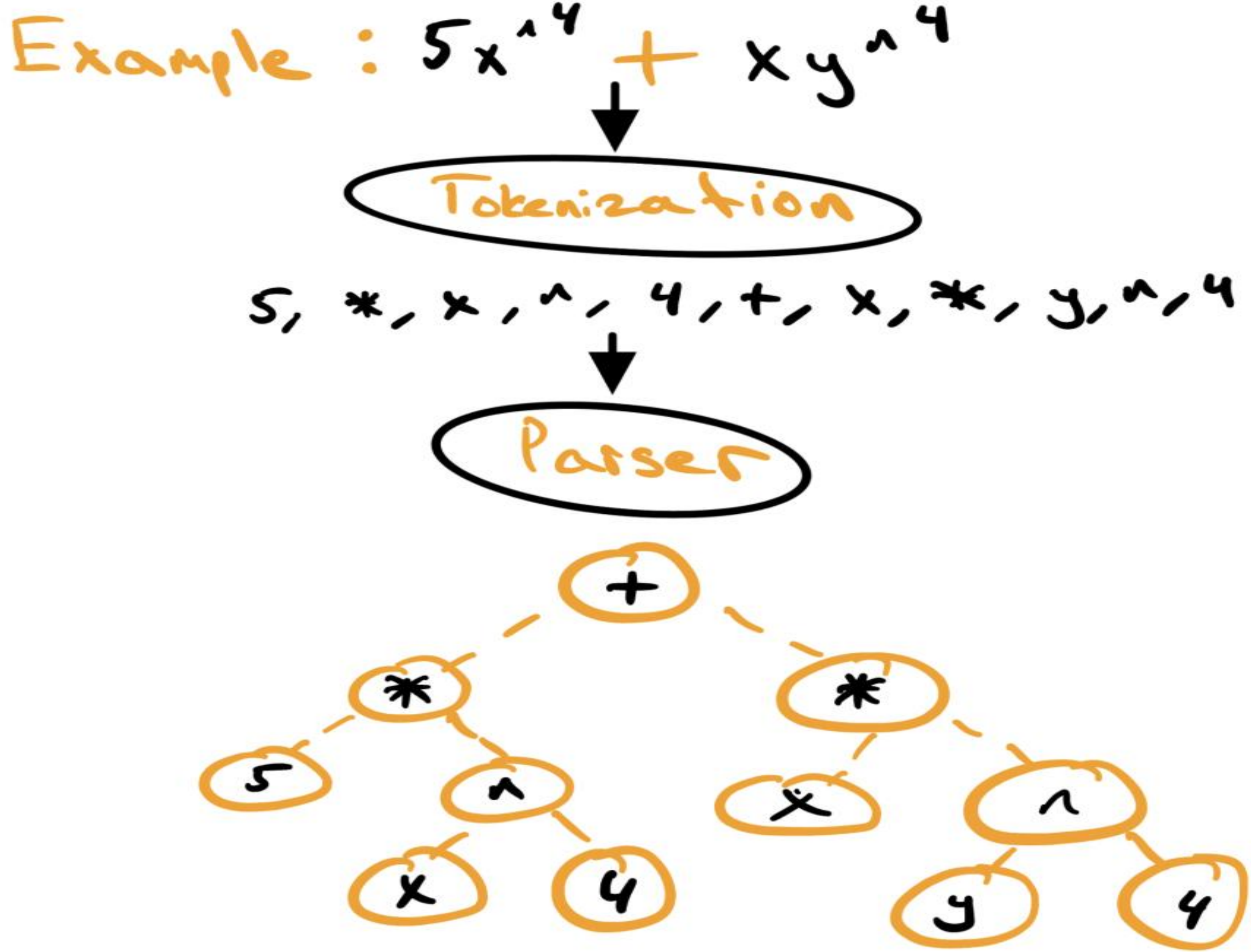
Graphical User Interface

- Easily shareable
- Web-based application so no system requirements or download required on user end.
- Handles capturing inputs, executing the Python code, and rendering the outputs automatically.

Gradio

All Platforms

Open-source Python package that allows you to quickly **build** a demo or web application for your machine learning model, API, or any Python function.


Example : 5x^4 + xy^4
Tokenization
5, *, x, ^, 4, +, x, *, y, ^, 4
Parser
+
*
*
5
^
x
^
x
4
y
4

Derive each Node (with respect to y)

Term: $5 * x^4$

$5 \to 0$ constant rule

$x^4 \to 0$ constant rule (x is now considered a constant)

$5x^4 \to 0$ constant & product rule

Term: xy^4

$x \to 0$ constant rule

$y^4 \to 4y^3$ power rule

$xy^4 \to 4xy^3$ constant & product rule

Derivative = sum of the derivative of both terms: $0 + 4xy^3 = 4xy^3$

Works Cited

Getting started#. Getting Started - pip documentation v24.0. (n.d.). https://pip.pypa.io/en/stable/getting-started/

YouTube. (2023, May 2). *What is an abstract syntax tree, with wealthfront engineer Spencer Miskoviak*. YouTube. https://www.youtube.com/watch?v=wINY109MG10

Lutkevich, B. (2022) *What is a parser? definition, types and examples*, *App Architecture*. Available at: https://www.techtarget.com/searchapparchitecture/definition/parser

Aydin, A. (2023) *3-Tokenization in NLP: The art of breaking down text data*, *Medium*. Available at: https://ayselaydin.medium.com/3-tokenization-in-nlp-the-art-of-breaking-down-text-data-5e1ba6786901 (Accessed: 03 May 2024).

SymPy (symbolic expressions on python) in one video | python # 5 (2019) *YouTube*. Available at: https://www.youtube.com/watch?app=desktop&v=kx2GzBeGPco (Accessed: 03 May 2024).

Spivak, R. (2015) *Let's build a simple interpreter. part 7: Abstract syntax trees*, *Ruslan's Blog*. Available at: https://ruslanspivak.com/lsbasi-part7/ (Accessed: 03 May 2024).

SymPy (symbolic expressions on python) in one video | python # 5 (2019) *YouTube*. Available at: https://www.youtube.com/watch?app=desktop&v=kx2GzBeGPco (Accessed: 03 May 2024).

Meurer A, Smith CP, Paprocki M, Čertík O, Kirpichev SB, Rocklin M, Kumar A, Ivanov S, Moore JK, Singh S, Rathnayake T, Vig S, Granger BE, Muller RP, Bonazzi F, Gupta H, Vats S, Johansson F, Pedregosa F, Curry MJ, Terrel AR, Roučka Š, Saboo A, Fernando I, Kulal S, Cimrman R, Scopatz A. (2017) SymPy: symbolic computing in Python. *PeerJ Computer Science* 3:e103 https://doi.org/10.7717/peerj-cs.103