Polynomial Evaluation Methods

This project implements polynomial evaluation using two different methods:

1. **Direct Substitution Method** (native method)
2. **Horner's Method** (optimized method)

Overview

Direct Substitution Method

The program evaluates polynomials of any degree using the direct substitution method, which involves:

1. Starting with the constant term
2. Adding each term step by step (x^1, x^2, x^3, etc.)
3. Displaying intermediate results at each step

Horner's Method

Horner's method is a more efficient algorithm that rearranges the polynomial to minimize multiplications:

1. $P(x) = a_0 + x(a_1 + x(a_2 + x(a_3 + ...)))$
2. Reduces the number of multiplications from $O(n^2)$ to $O(n)$
3. More numerically stable for high-degree polynomials

Features

- **Direct Substitution Method**: Implements the native method for polynomial evaluation (`polynomial.py`)
- **Horner's Method**: Implements the optimized Horner's method (`polynomial_horner.py`)
- **Step-by-step Evaluation**: Shows intermediate results (S0, S1, S2, etc.)
- **Interactive Mode**: Allows user input for custom polynomials
- **Comprehensive Testing**: Includes extensive test cases for both methods
- **Error Handling**: Validates input parameters
- **Flexible Parameters**: Both methods accept coefficients as individual numbers or lists

Example

For the polynomial $P(x) = -x^3 + 2x^2 + 5x - 7$ evaluated at $x = 2$:

Input:

```
Degree of the polynomial: 3
Value of x: 2
Value of constant term: -7
Coefficient of the x^1 term: 5
```

```
Coefficient of the x^2 term: 2
Coefficient of the x^3 term: -1
```

Output:

```
Desired Outputs
--------------------
S0 (value of the constant term) = -7
S1 (sum of the 2 lowest terms) = -7 + 5(2) = 3
S2 (sum of the 3 lowest terms) = 3 + 2(2^2) = 11
S3 (sum of the 4 lowest terms) = 11 + -1(2^3) = 3

P(x) = 3

Do you want to evaluate another polynomial? (y/n): y
```

Expected Terminal Output

Direct Substitution Method Output:

```
Polynomial Evaluation using Direct Substitution Method
==================================================

Degree of the polynomial: 3
Value of x: 2
Value of constant term: -7
Coefficient of the x^1 term: 5
Coefficient of the x^2 term: 2
Coefficient of the x^3 term: -1

Desired Outputs
--------------------
S0 (value of the constant term) = -7
S1 (sum of the 2 lowest terms) = -7 + 5(2^1) = 3
S2 (sum of the 3 lowest terms) = 3 + 2(2^2) = 11
S3 (sum of the 4 lowest terms) = 11 + -1(2^3) = 3

P(x) = 3

Do you want to evaluate another polynomial? (y/n):
```

Horner's Method Output:

```
Polynomial Evaluation using Horner's Method
==========================================
```

```
Degree of the polynomial: 3
Value of x: 2
Value of constant term: -7
Coefficient of the x^1 term: 5
Coefficient of the x^2 term: 2
Coefficient of the x^3 term: -1

Desired Outputs (Horner's Method)
------------------------------
S3 (highest degree coefficient) = -1
S2 (Horner step 2) = 0
S3 (Horner step 3) = 5
S4 (add constant term) = 3

P(x) = 3

Do you want to evaluate another polynomial? (y/n):
```

Important for Autograder:

- Follow the exact format shown above
- Include all step-by-step calculations
- Show intermediate results (S0, S1, S2, etc.)
- Display final result as "P(x) = [result]"
- Include the loop prompt for continuation

Running the Programs

Direct Substitution Method:

```
python polynomial.py
```

Horner's Method:

```
python polynomial_horner.py
```

Both programs will prompt for interactive input:

- Degree of polynomial
- Value of x
- Constant term
- Coefficients for each power of x

After each evaluation, you can choose to run another polynomial evaluation or exit.

Algorithms

Direct Substitution Method

The direct substitution method follows these steps:

1. **Initialize**: Start with the constant term (S0)
2. **Iterate**: For each power of x from 1 to degree:
    - Calculate the term value: coefficient × x^power
    - Add to previous sum
    - Display intermediate result
3. **Return**: Final polynomial value

Horner's Method

Horner's method is more efficient and follows these steps:

1. **Initialize**: Start with the highest degree coefficient
2. **Iterate**: For each coefficient from highest to lowest:
    - Multiply current result by x
    - Add the next coefficient
    - Display intermediate result
3. **Final Step**: Multiply by x and add constant term
4. **Return**: Final polynomial value

Example: For $P(x) = -x^3 + 2x^2 + 5x - 7$:

- Direct: $-7 + 5(2) + 2(2^2) + (-1)(2^3) = -7 + 10 + 8 - 8 = 3$
- Horner: $((-1)(2) + 2)(2) + 5)(2) - 7 = (0)(2) + 5)(2) - 7 = 10 - 7 = 3$

Test Coverage

The test suite includes:

- **Example polynomial**: Tests the main example ($P(x) = -x^3 + 2x^2 + 5x - 7$, $x = 2$)
- **Basic polynomial**: Tests simple linear polynomial ($P(x) = 2x + 3$, $x = 1$)
- **Error handling**: Tests incorrect coefficient count validation
- **Loop functionality**: Tests multiple polynomial evaluations
- **Interactive loop**: Tests user input simulation with 'y'/'n' choices
- **Number format flexibility**: Tests both integer and float number formats
- **Method verification**: Horner's method tests verify "Horner's Method" appears in output

Functions

`evaluate_polynomial(degree, x, constant_term, *coefficients)`

Evaluates a polynomial using direct substitution method.

Parameters:

- `degree` (int): Degree of the polynomial
- `x` (float): Value at which to evaluate

- `constant_term` (float): Constant term (coefficient of x^0)
- `*coefficients`: Individual coefficients for x^1, x^2, ..., x^degree

Returns:

- `float`: Final polynomial result

Features:

- Prints step-by-step evaluation process
- Handles both integer and float inputs
- **Flexible parameter handling**: Accepts coefficients in multiple ways

`evaluate_polynomial_horner(degree, x, constant_term, *coefficients)`

Evaluates a polynomial using Horner's method.

Parameters:

- `degree` (int): Degree of the polynomial
- `x` (float): Value at which to evaluate
- `constant_term` (float): Constant term (coefficient of x^0)
- `*coefficients`: Individual coefficients for x^1, x^2, ..., x^degree

Returns:

- `float`: Final polynomial result

Features:

- Prints step-by-step evaluation process using Horner's method
- Handles both integer and float inputs
- **Flexible parameter handling**: Accepts coefficients in multiple ways
- **More efficient**: O(n) multiplications vs $O(n^2)$ for direct substitution

Understanding `*coefficients`

The `*` operator in Python is called the **unpacking operator** (or splat operator). It makes the function flexible to accept coefficients in different ways:

Function Definition (`*coefficients`):

```
def evaluate_polynomial(degree, x, constant_term, *coefficients):
```

This means "collect all remaining arguments into a tuple called `coefficients`"

Usage Examples:

1. Individual Coefficients:

```
result = evaluate_polynomial(3, 2, -7, 5, 2, -1)
# Inside function: coefficients = (5, 2, -1)
```

2. List with Unpacking:

```
coeffs = [5, 2, -1]
result = evaluate_polynomial(3, 2, -7, *coeffs)
# The * unpacks the list: same as evaluate_polynomial(3, 2, -7, 5, 2,
-1)
```

3. Direct List (without unpacking):

```
result = evaluate_polynomial(3, 2, -7, [5, 2, -1])
# Inside function: coefficients = ([5, 2, -1],) - a tuple with one
element (the list)
```

Why Use `*coefficients`?

- **Flexibility**: Users can pass coefficients either as individual numbers or as a list
- **Clean API**: No need to worry about wrapping individual numbers in a list
- **Backward Compatible**: Existing code using lists still works with `*coeffs`

Error Handling

The program handles:

- Incorrect number of coefficients
- Invalid input types
- Edge cases (zero coefficients, degree 0 polynomials)