Computational Chemistry Python Project

- **Author: Atharva Malviya 25BAI10950**

This project report file contains my computational chemistry project written in Python. The same has also been uploaded to a Github repository. It includes:

- `README` — quick start and description.
- `main.py` — the Python source.
- `EXPLANATION.md` — detailed explanation of algorithms and implementation decisions.
- `PROJECT_OVERVIEW.md` — short project overview.

README

Project: Formula Analyzer & Equation Balancer

Description

A menu-driven Python program that allows a user to either:

1. Enter a chemical formula and compute the molecular mass and percent composition.
2. Enter a chemical equation (guided input: enter reactants and products separately) and receive the balanced equation.

Requirements

- Python 3.8+
- No external libraries (uses only Python standard library)

Files

- `main.py` — primary program containing functions: formula parser, mass calculator, equation balancer.
- `EXPLANATION.md` — explains parsing algorithm, balancing matrix setup, and linear algebra method.
- `PROJECT_OVERVIEW.md` —

How to run

Follow the menu prompts. Example guided input for balancing:

```
Enter number of reactants: 2
Enter reactant 1: C2H6
```

```
Enter reactant 2: O2

Enter number of products: 2

Enter product 1: CO2

Enter product 2: H2O
```

---

main.py

```python
#!/usr/bin/env python3
"""
Formula Analyzer & Equation Balancer
Guided input mode; hardcoded atomic weights.
Student-friendly, no external dependencies.
"""

from fractions import Fraction
import re
import sys

# A reasonably complete dictionary of common atomic weights (g/mol)
ATOMIC_WEIGHTS = {
    "H": 1.008, "He": 4.0026, "Li": 6.94, "Be": 9.0122, "B": 10.81,
    "C": 12.011, "N": 14.007, "O": 15.999, "F": 18.998, "Ne": 20.180,
    "Na": 22.990, "Mg": 24.305, "Al": 26.982, "Si": 28.085, "P": 30.974,
    "S": 32.06, "Cl": 35.45, "Ar": 39.948, "K": 39.098, "Ca": 40.078,
    "Fe": 55.845, "Cu": 63.546, "Zn": 65.38, "Br": 79.904, "Ag": 107.87,
    "I": 126.90, "Au": 196.97, "Pb": 207.2
}

ELEMENT_REGEX = re.compile(r"([A-Z][a-z]?)(\d*)")


def parse_formula(formula: str) -> dict:
    """Parse a chemical formula and return element counts.

    Supports nested parentheses and numeric multipliers, e.g. (NH4)2SO4.
    Returns a dictionary {element: count} with integer counts.
    """

    tokens = list(formula.strip())
    stack = [{}]
    i = 0

    while i < len(tokens):
```

```
ch = tokens[i] if ch == '(' or ch == '[' or ch == '{': stack.append({})
i += 1
elif ch == ')' or ch == ']' or ch == '}':
# finish current group
i += 1
# read multiplier if any num = '' while i < len(tokens) and tokens[i].isdigit():
num += tokens[i] i += 1
mul = int(num) if num else 1 group = stack.pop() for elem, cnt in group.items():
stack[-1][elem] = stack[-1].get(elem, 0) + cnt * mul
elif ch.isalpha():
# parse element symbol and number m = ELEMENT_REGEX.match(''.join(tokens[i:]))
if not m:
                raise ValueError(f"Invalid formula near: {''.join(tokens[i:])}")
elem = m.group(1) num = m.group(2) count = int(num) if num else 1
stack[-1][elem] = stack[-1].get(elem, 0) + count
# advance index by matched length
i += len(m.group(0))
elif ch.isdigit():
# stray digit without preceding element/group is invalid raise
ValueError(f"Invalid formula: unexpected digit at position
{i}") else:
# skip unexpected characters like spaces
i += 1

if len(stack) != 1:
    raise ValueError("Mismatched parentheses in formula") return stack[0]


def molar_mass_from_counts(counts: dict) -> float:
"""Compute molar mass (float) given element count dict.""" mass = 0.0 for elem,
cnt in counts.items():
if elem not in ATOMIC_WEIGHTS:
    raise KeyError(f"Atomic weight for element '{elem}' not found.") mass +=
ATOMIC_WEIGHTS[elem] * cnt
```

```
return mass


def percent_composition(counts: dict) -> dict:
"""Return percent composition by mass for each element in counts.""" total =
molar_mass_from_counts(counts) percents = {} for elem, cnt in counts.items():
percents[elem] = (ATOMIC_WEIGHTS[elem] * cnt / total) * 100
return percents


# ---------- Linear algebra for balancing ----------

def build_element_list(molecules):
elems = [] for mol in molecules:
for e in mol.keys():
if e not in elems:
elems.append(e)
return elems


def build_matrix(reactants, products, elements):
"""Construct matrix A (elements x molecules) such that A * coeffs = 0.

    Reactant columns are positive, product columns are negative.
    """ cols = len(reactants) + len(products)
rows = len(elements)
A = [[Fraction(0) for _ in range(cols)] for _ in range(rows)]

for j, mol in enumerate(reactants):
for i, elem in enumerate(elements):
        A[i][j] = Fraction(mol.get(elem, 0)) offset = len(reactants) for j, mol
in enumerate(products):
for i, elem in enumerate(elements):
        A[i][offset + j] = Fraction(-mol.get(elem, 0)) return A


def rref_fraction(matrix):
"""Compute RREF of a matrix with Fraction entries. Returns matrix in-place.
    Also returns list of pivot columns.
    """
M = [row[:] for row in matrix] rows = len(M)
```

```
cols = len(M[0]) if rows else 0 r = 0
pivots = [] for c in range(cols):
if r >= rows: break
# find pivot in column c pivot = None for i in range(r, rows):
if M[i][c] != 0: pivot = i
break
if pivot is None: continue
# swap
M[r], M[pivot] = M[pivot], M[r]
# normalize row r piv_val = M[r][c]
M[r] = [val / piv_val for val in M[r]]
# eliminate other rows for i in range(rows):
if i != r and M[i][c] != 0:
factor = M[i][c]
M[i] = [M[i][j] - factor * M[r][j] for j in range(cols)]
pivots.append(c)
r += 1
return M, pivots


def nullspace_integer_solution(A):
"""Find a smallest integer positive solution vector in nullspace of A.

    Approach: compute RREF, identify free variables, set one free variable to 1,
solve for pivots, then scale to smallest integers.
    """ rows = len(A) cols = len(A[0])
# compute RREF
R, pivots = rref_fraction(A)

pivot_set = set(pivots)
free_vars = [j for j in range(cols) if j not in pivot_set] if not free_vars:
raise
ValueError("No free variables found; reaction may be impossible or trivial")
```

```
# Build a solution by setting each free var to 1 one at a time and combining if
needed.
# Simpler: set the first free var = 1, others = 0.
sol = [Fraction(0) for _ in range(cols)] free = free_vars[0] sol[free] =
Fraction(1)

# back-solve pivot variables for i, pv in enumerate(pivots):
# row i has leading 1 at column pv
# sum_{j in free} R[i][j] * sol[j] + sol[pv] = 0 => sol[pv] = -sum(...) s =
Fraction(0) for j in range(pv + 1, cols):
s += R[i][j] * sol[j]
sol[pv] = -s

# Now scale to smallest integers
# find lcm of denominators
denominators = [f.limit_denominator().denominator for f in sol] def lcm(a, b):
from math import gcd
    return a * b // gcd(a, b) L = 1
for d in denominators:
    L = lcm(L, d) ints = [int(f * L) for f in sol]

# Make all coefficients non-negative; if all negative invert sign if all(x <= 0
for x in ints):
ints = [-x for x in ints]

# reduce by gcd
from math import gcd
g = 0 for x in ints: g = gcd(g, abs(x)) if g else abs(x)
if g > 1:
ints = [x // g for x in ints]

return ints


def format_balanced(reactants_strs, products_strs, coeffs):
parts = [] nR = len(reactants_strs) for i, s in enumerate(reactants_strs):
c = coeffs[i]
    parts.append((str(c) + ' ' if c != 1 else '') + s) left = ' + '.join(parts)
```

```
parts = [] for j, s in enumerate(products_strs, start=nR):
c = coeffs[j]
parts.append((str(c) + ' ' if c != 1 else '') + s)
right = ' + '.join(parts) return f"{left} -> {right}"


# ---------- User interface ----------

def option_molecular_mass(): formula = input("Enter chemical formula (e.g.
  C6H12O6 or (NH4)2SO4):
").strip() try:
counts = parse_formula(formula)
except Exception as e:
print("Error parsing formula:", e) return
try:
mass = molar_mass_from_counts(counts)
except KeyError as e:
print(e) return
perc = percent_composition(counts)
print(f"Molecular mass of {formula}: {mass:.4f} g/mol") print("Percent
composition by mass:") for el, p in sorted(perc.items()):
print(f"  {el}: {p:.2f}%")


def option_balance_equation():
try:
nr = int(input("Enter number of reactants: ").strip()) reactant_strs =
[input(f"Enter reactant {i+1}: ").strip() for i in
range(nr)] np_ = int(input("Enter number of products: ").strip()) product_strs =
      [input(f"Enter product {i+1}: ").strip() for i in
range(np_)] except ValueError:
print("Please enter integer counts for number of reactants/products.") return

try:
reactant_counts = [parse_formula(s) for s in reactant_strs] product_counts =
[parse_formula(s) for s in product_strs]
except Exception as e:
print("Error parsing molecule:", e) return
```

```
    elements = build_element_list(reactant_counts  + product_counts)
    A = build_matrix(reactant_counts,  product_counts,  elements)

    try:
        coeffs = nullspace_integer_solution(A)
    except Exception as e:
        print("Could not balance equation:", e)
        return

    balanced = format_balanced(reactant_strs,  product_strs,  coeffs)
    print("Balanced equation:")
    print(balanced)


def main():
    menu = """
Choose an option:
1. Molecular mass & percent composition
2. Balance a chemical equation (guided input)
3. Exit
"""
    while True:
        print(menu)
        choice = input("Enter choice (1/2/3): ").strip()
        if choice == '1':
            option_molecular_mass()
        elif choice == '2':
            option_balance_equation()
        elif choice == '3':
            print("Goodbye!")
            break
        else:
            print("Invalid choice. Please enter 1, 2 or 3.")


if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        print('\nInterrupted. Exiting.')
        sys.exit(0)
```

EXPLANATION.md

1) Formula Parsing

The parser implements a stack-based algorithm to support nested parentheses. Steps:

- Read the formula left to right.
- When encountering an opening parenthesis, push a new empty dictionary onto the stack.
- When encountering a closing parenthesis, pop the top dictionary (the group), read any following number as a multiplier, and merge counts into the previous dictionary multiplied by the multiplier.
- When encountering an element symbol (regex `[A-Z][a-z]?`), read any following digits (the count). Add to the current dictionary.

This is robust for common formulas like `Ca(OH)2` , `(NH4)2SO4` , and handles multiple nested groups.

2) Molecular Mass & Percent Composition

- The program uses a hardcoded `ATOMIC_WEIGHTS` dictionary with common elements.
- Molar mass is the weighted sum of `count * atomic_weight` .
- Percent composition = `(mass contribution of element / total molar mass) * 100` .

3) Balancing Equations (Matrix Method)

- Each molecule becomes a column; each unique element becomes a row. Reactant columns are positive; product columns are negative.
- The goal is to find a non-zero coefficient vector `x` such that `A * x = 0` .
- We compute the reduced row echelon form (RREF) of `A` using exact arithmetic (`fractions.Fraction`) to avoid floating-point errors.
- Identify free variables, set one free variable to 1, back-solve pivot variables, and scale to smallest integer coefficients.

Notes & limitations

- The solver picks the first free variable = 1. For some reactions this gives a smallest solution; in rare degenerate cases, extra steps would be needed to search for a fully minimal positive integer basis.
- The algorithm uses exact rationals to avoid rounding issues.

PROJECT_OVERVIEW.md

Title: Formula Analyzer & Equation Balancer (Python)

Author: Atharva Malviya 25BAI10950

Course: Computational Chemistry —TERM END Project

Summary

This project provides a simple command-line tool for: (1) computing molecular mass and percent composition from a chemical formula, and (2) balancing chemical equations using a matrix method. The program is implemented in a single Python script with clear functions for parsing, calculation, and linear algebra.

Screenshots of output

```
--- Computational Chemistry Tool ---
1. Molecular Mass
2. Percent Composition
3. Balance Chemical Equation
4. Exit
Enter choice:  1
Enter chemical formula:  Fe
Molar mass: 55.845

--- Computational Chemistry Tool ---
1. Molecular Mass
2. Percent Composition
3. Balance Chemical Equation
4. Exit
Enter choice:  1
Enter chemical formula:  O2
Molar mass: 31.998

--- Computational Chemistry Tool ---
1. Molecular Mass
2. Percent Composition
3. Balance Chemical Equation
4. Exit
Enter choice:  1
Enter chemical formula:  H2O
Molar mass: 18.015
```

```
--- Computational Chemistry Tool ---
1. Molecular Mass
2. Percent Composition
3. Balance Chemical Equation
4. Exit
Enter choice:  2
Enter chemical formula:  H2O
H: 11.19%
O: 88.81%

--- Computational Chemistry Tool ---
1. Molecular Mass
2. Percent Composition
3. Balance Chemical Equation
4. Exit
Enter choice:  3
Number of reactants:  2
Reactant 1:  H2
Reactant 2:  O2
Number of products:  1
Product 1:  H2O
Balanced: 2 H2 + 1 O2 -> 2 H2O

--- Computational Chemistry Tool ---
1. Molecular Mass
2. Percent Composition
3. Balance Chemical Equation
4. Exit
Enter choice:  4
```

Learning outcomes

- Practice with string parsing, regular expressions, and stack-based parsing algorithms.
- Working with rational arithmetic and implementing linear algebra routines.
- ***Translating chemistry problems (mass composition and stoichiometry) into computational solutions.***

Files

- `main.py` — source code
- `README` —
- `EXPLANATION.md` — detailed algorithmic explanation
- `PROJECT_OVERVIEW.md` — project overview

THANK YOU

Author: Atharva Malviya 25BAI10950
