08 Prove Assignment: Dictionaries

Purpose

Prove that you can write a Python program that uses a dictionary and lists.

Assignment

During this assignment, you will write and test the remaining parts of the molar mass calculator that you started writing in the previous lesson's prove milestone.

Helpful Documentation

- The prove milestone of the previous lesson explains how a molar mass calculator should work.
- The prepare content for this lesson explains how to create and use a dictionary in a Python program.
- The prepare content for lesson 5 explains how to use `pytest`, `assert`, and `approx` to automatically verify that functions are correct. It also contains an example test function and links to additional documentation about `pytest`.

Steps

Do the following:

1. Change the compound list that is in your `make_periodic_table` function to a compound dictionary. Each item in the dictionary should have a chemical symbol as the key and the chemical name and atomic mass in a list as the value, like this:

```
    periodic_table_dict = {
        # symbol: [name, atomic_mass]
        "Ac": ["Actinium", 227],
        "Ag": ["Silver", 107.8682],
        "Al": ["Aluminum", 26.9815386],
          ⋮
    }
```

2. Copy and paste the following Python code into your `chemistry.py` program. Be certain not to paste the code inside an existing function.

```
class FormulaError(ValueError):
    """FormulaError is the type of error that
    parse_formula will raise if a formula is invalid.
    """


def parse_formula(formula, periodic_table_dict):
    """Convert a chemical formula for a molecule into a compound list
    that stores the quantity of atoms of each element in the molecule.
    For example, this function will convert "H2O" to [["H", 2], ["O", 1]]
    and "PO4H2(CH2)12CH3" to [["P", 1], ["O", 4], ["H", 29], ["C", 13]]

    Parameters
        formula: a string that contains a chemical formula
        periodic_table_dict: the compound dictionary returned
            from make_periodic_table
```

```
    Return: a compound list that contains chemical symbols and
        quantities like this [["Fe", 2], ["O", 3]]
    """
    if not isinstance(formula, str):
        raise TypeError("wrong data type for formula; "
            f"formula is a {type(formula)} but must be a string")
    if not isinstance(periodic_table_dict, dict):
        raise TypeError("wrong data type for periodic_table_dict; "
            f"periodic_table_dict is a {type(periodic_table_dict)} "
            "but must be a dictionary")

    def parse_quant(formula, index):
        quant = 1
        if index < len(formula) and formula[index].isdecimal():
            start = index
            index += 1
            while index < len(formula) and formula[index].isdecimal():
                index += 1
            quant = int(formula[start:index])
        return quant, index

    def get_quant(elems, symbol):
        return 0 if symbol not in elems else elems[symbol]

    def parse_r(formula, index, level):
        start_index = index
        start_level = level
        elem_dict = {}
        while index < len(formula):
            ch = formula[index]
            if ch == "(":
                group_dict, index = parse_r(formula, index+1, level+1)
                quant, index = parse_quant(formula, index)
                for symbol in group_dict:
                    prev = get_quant(elem_dict, symbol)
                    elem_dict[symbol] = prev + group_dict[symbol] * quant
            elif ch.isalpha():
                symbol = formula[index:index+2]
                if symbol in periodic_table_dict:
                    index += 2
                else:
                    symbol = formula[index:index+1]
                    if symbol in periodic_table_dict:
                        index += 1
                    else:
                        raise FormulaError(
                            "invalid formula, unknown element symbol:",
                            formula, index)
                quant, index = parse_quant(formula, index)
                prev = get_quant(elem_dict, symbol)
                elem_dict[symbol] = prev + quant
            elif ch == ")":
                if level == 0:
                    raise FormulaError(
                        "invalid formula, unmatched close parenthesis:",
                        formula, index)
                level -= 1
                index += 1
                break
            else:
                if ch.isdecimal():
                    # Decimal digit not preceded by an
                    # element symbol or close parenthesis
                    message = "invalid formula:"
                else:
                    # Illegal character: [^()0-9a-zA-Z]
                    message = "invalid formula, illegal character:"
                raise FormulaError(message, formula, index)
        if level > 0 and level >= start_level:
            raise FormulaError(
                "invalid formula, unmatched open parenthesis:",
                formula, start_index - 1)
        return elem_dict, index
```

```
    # Return the compound list of element symbols and
    # quantities. Each element in the compound list
    # will be a list in this form: ["symbol", quantity]
    elem_dict, _ = parse_r(formula, 0, 0)
    return list(elem_dict.items())


# These are the indexes of the
# elements in the periodic table.
NAME_INDEX = 0
ATOMIC_MASS_INDEX = 1


def compute_molar_mass(symbol_quantity_list, periodic_table_dict):
    """Compute and return the total molar mass of all the
    elements listed in symbol_quantity_list.

    Parameters
        symbol_quantity_list is a compound list. Each small
            list in symbol_quantity_list has this form:
            ["symbol", quantity].
        periodic_table_dict is the compound dictionary returned
            from make_periodic_table.
        Return: the total molar mass of all the elements in
            symbol_quantity_list.

    For example, if symbol_quantity_list is [["H", 2], ["O", 1]],
    this function will calculate and return
    atomic_mass("H") * 2 + atomic_mass("O") * 1
    1.00794 * 2 + 15.9994 * 1
    18.01528
    """
    # For each list in the compound symbol_quantity_list:
        # Split the list into symbol and quantity.
        # Get the atomic mass for the symbol from the dictionary.
        # Multiply the atomic mass by the quantity.
        # Add the product into the total mass.
    # Return the total mass.
    return
```

The code that you pasted includes a function named `parse_formula` that converts a chemical formula for a molecule, such as "C13H16N2O2" (melatonin), into a compound list, such as [["C", 13], ["H", 16], ["N", 2], ["O", 2]]. In the code that you pasted, this compound list is known as a *symbol_quantity_list* because it contains the symbols of chemical elements and the quantity of each element that appears in a chemical formula.

3. The code that you pasted also includes the header and documentation string for a function named `compute_molar_mass`. Read the docstring and comments in the `compute_molar_mass` function and write the code for that function. Note that you can complete the `compute_molar_mass` function by writing ten or fewer lines of code.

4. Modify the `main` function in your `chemistry.py` program so that it does the following:

```
def main():
    # Get a chemical formula for a molecule from the user.

    # Get the mass of a chemical sample in grams from the user.

    # Call the make_periodic_table function and
    # store the periodic table in a variable.

    # Call the parse_formula function to convert the
    # chemical formula given by the user to a compound
    # list that stores element symbols and the quantity
    # of atoms of each element in the molecule.

    # Call the compute_molar_mass function to compute the
    # molar mass of the molecule from the compound list.
```

```
    # Compute the number of moles in the sample.

    # Print the molar mass.

    # Print the number of moles.
```

Testing Procedure

Verify that your program works correctly by following each step in this testing procedure:

1. Download the [test_chemistry_2.py]() Python file and save it in the same folder where you saved your `chemistry.py` program. Run the `test_chemistry_2.py` file and ensure that all three of the test functions pass. If any of the test functions don't pass, there is a mistake in your `chemistry.py` program. Read the output from `pytest`, fix the mistake, and run the `test_chemistry_2.py` file again until the test functions pass.

```
> python test_chemistry_2.py
====================== test session starts =======================
platform win32--Python 3.8.6, pytest-6.1.2, py-1.9.0, pluggy-0.13.
rootdir: C:\Users\cse111\lesson07
collected 3 items

test_chemistry_2.py::test_make_periodic_table PASSED        [ 33%]
test_chemistry_2.py::test_parse_formula PASSED              [ 66%]
test_chemistry_2.py::test_compute_molar_mass PASSED         [100%]

======================= 3 passed in 0.17s ========================
```

2. Run your finished `chemistry.py` program. Enter the user input shown below. Ensure that your program prints correct output.

```
> python chemistry.py
Enter the molecular formula of the sample: C13H18O2
Enter the mass in grams of the sample: 5.04
206.28082 grams/mole
0.02443 moles
```

Exceeding the Requirements

If you wish to exceed the requirements of this assignment, here are a few suggestions for additional features that you could add to your program. If you wish, you can add different features to your program. However, you don't have to add any additional features to your program because exceeding the requirements of this assignment is optional.

- Add a dictionary that contains known chemical formulas and their names. For example:

```
    known_molecules_dict = {
        "Al2O3": "aluminum oxide",
        "CH3OH": "methanol",
        "C2H6O": "ethanol",
        "C2H5OH": "ethanol",
        "C3H8O": "isopropyl alcohol",
        "C3H8": "propane",
        "C4H10": "butane",
        "C6H6": "benzene",
        "C6H14": "hexane",
        "C8H18": "octane",
        "CH3(CH2)6CH3": "octane",
        "C13H18O2": "ibuprofen",
        "C13H16N2O2": "melatonin",
        "Fe2O3": "iron oxide",
        "FeS2": "iron pyrite",
```

```
        "H2O": "water"
    }
```

Then write a function named `get_formula_name` with the following header and documentation string.

```
def get_formula_name(formula, known_molecules_dict):
    """Try to find formula in the known_molecules_dict.
    If formula is in the known_molecules_dict, return
    the name of the chemical formula; otherwise return
    "unknown compound".
    """
```

Call the `get_formula_name` function from your `main` function and print the compound name for the user to see with the other output.

- Add the atomic number for each element to the compound dictionary of elements. The atomic number of an element is the number of protons in the nucleus of that element. Write a function named `sum_protons` with the following header and documentation string.

```
def sum_protons(symbol_quantity_list, periodic_table_dict):
    """Compute and return the total number of protons in
    all the elements listed in symbol_quantity_list.
    """
```

Call the `sum_protons` function from your `main` function and print the number of protons for the user to see with the other output.

Submission

To submit your program, return to I-Learn and do these two things:

1. Upload your `chemistry.py` file for feedback.
2. Add a submission comment that specifies the grading category that best describes your program along with a one or two sentence justification for your choice. The grading criteria are:
    - 1. Some attempt made
    - 2. Developing but significantly deficient
    - 3. Slightly deficient
    - 4. Meets requirements
    - 5. Exceeds requirements