Programming in Chemistry

Author and notebook information

Author: Arun K. Sharma
Department of Physical Sciences, Wagner College, 1 Campus Road, Staten Island, NY 10301.
Email: aksharma@wagner.edu

This notebook provides examples of chemistry oriented programming exercises that have been used in the author's Introduction to Scientific Computing course at Wagner College. at Wagner College. The course provides students with an introduction to programming in the Wolfram language (Mathematica) and serves to fulfill the general education requirement of intensive technology in the newly established curriculum. Students develop competency in programming with examples that are drawn from the first-year Chemistry curriculum. They develop their problem-solving skills by writing small programs that reinforce and extend their understanding of chemical principles. These examples incorporate basic programming elements such as iteration, modular functions, arrays, and Boolean logic. Students also learn procedural and functional programming in the Wolfram language and engage in projects that utilize chemistry domain knowledge and programming elements learned in the course. This course has served as an efficient recruitment mechanism for the author's research group and the Chemistry program.

Notebook Setup

Examples

---

1. Nomenclature

Interactive chemical guessing game that asks for the molecular formula from the name of a random chemical chosen from the simpleChemicals list. User inputs the answer and if the answer is incorrect, the question is returned until the correct answer is supplied. All answers of chemicals that work are listed below.

```
formulaGuesser[chemical_String] := Module[
  {moleculeAnswer = MoleculeValue[Molecule[chemical], "MolecularFormulaString"]},
  While[InputString[StringJoin["What is the molecular formula for ",
      chemical, "?"]] ≠ moleculeAnswer, Return]]
```

```
formulaGuesser[RandomChoice[simpleChemicals]]
```

```
MoleculeValue[Molecule["Oxalate"], "MolecularFormula"]
```

Out[ ]= $C_2O_4^{2-}$

2. Molarity

Calculates molarity by supplying a chemical, mass, and volume amount in mL. The volume is converted into L in the function. This function works with the chemical list simpleChemicalsv2.

```
calcMolarity[chem_String, mass_, vol_] :=
  (Quantity[mass, "Grams"] / Interpreter["Chemical"][chem]["MolarMass"]) /
   UnitConvert[Quantity[vol, "Milliliters"], "Liters"]
```

Here, we calculate molarity for 18 g of each substance of all the chemicals present in the list simpleChemicalsv2

```
calcMolarity[#, 18, 1000] & /@ simpleChemicalsv2
```

Out[ ]= {0.17919 mol/L, 0.140721 mol/L, 0.22246 mol/L, 0.4937 mol/L, 0.18354 mol/L, 0.28566 mol/L, 0.19992 mol/L, 0.2193 mol/L, 0.18369 mol/L, 0.38287 mol/L, 0.89971 mol/L, 0.39109 mol/L, 0.66603 mol/L, 0.29112 mol/L, 0.5282 mol/L, 0.9992 mol/L, 0.5282 mol/L, 1.0569 mol/L, 0.3080 mol/L, 0.21427 mol/L, 0.21178 mol/L}

```
chem = RandomChoice[simpleChemicalsv2];
{chem, calcMolarity[chem, 20.5, 560.]}
```

Out[ ]= {Hydrocyanic acid, 1.35452 mol/L}

3. Molality

Calculate the molality by supplying a chemical identified as solute, mass of the solute(g) and mass of solvent (g). Works with simpleChemicalsv2.

```
calcMolality[chem_String, massSolute_, massSolvent_] :=
  (Quantity[massSolute, "Grams"] / Interpreter["Chemical"][chem]["MolarMass"]) /
   Quantity[massSolvent, "Kilograms"]
```

```
In[ ]:= calcMolality[#, 50, 2] & /@ simpleChemicalsv2
```

```
Out[ ]= {0.2489 mol/kg, 0.195446 mol/kg, 0.30898 mol/kg, 0.6857 mol/kg,
  0.2549 mol/kg, 0.39675 mol/kg, 0.27767 mol/kg, 0.3046 mol/kg,
  0.25512 mol/kg, 0.53177 mol/kg, 1.24960 mol/kg, 0.54318 mol/kg,
  0.9250 mol/kg, 0.4043 mol/kg, 0.7337 mol/kg, 1.3877 mol/kg, 0.7337 mol/kg,
  1.4679 mol/kg, 0.4278 mol/kg, 0.29760 mol/kg, 0.29414 mol/kg}
```

```
In[ ]:= chem = RandomChoice[simpleChemicalsv2];
        {chem, calcMolality[chem, 30., 1]}
```

```
Out[ ]= {Hydrochloric acid, 0.822865 mol/kg}
```

4. Mole Fraction

Calculate mole fraction by supplying chemicals and mass amounts. Supply the name and amount in grams for each of the two chemical species.

```
In[ ]:= calcMolFraction[chem1_String, mass1_, chem2_String, mass2_] :=
         Module[{mol1, mol2, molTotal, molFrac1, molFrac2},
          mol1 = Quantity[mass1, "Grams"] / Interpreter["Chemical"][chem1]["MolarMass"];
          mol2 = Quantity[mass2, "Grams"] / Interpreter["Chemical"][chem2]["MolarMass"];
          molTotal = mol1 + mol2;
          {molFrac1 = mol1 / molTotal, molFrac2 = mol2 / molTotal};
          {molFrac1 → chem1, molFrac2 → chem2}]
```

```
In[ ]:= calcMolFraction[RandomChoice[simpleChemicalsv2],
         20, RandomChoice[simpleChemicalsv2], 40]
```

```
Out[ ]= {0.3967 → Nitrous acid, 0.6033 → Boric Acid}
```

5. Partial Pressure

Calculate partial pressures for each gas in a mixture of two gases. The user supplies identity of each gas, amount of each gas, and the total pressure (mm Hg).

```
In[ ]:= calcPartialPressure [chem1_String ,
          mass1_ , chem2_String , mass2_ , totalPressure_ ] := Module[
          {mol1 , mol2 , molTotal , pressureTotal , molFrac1 , molFrac2 , pressure1 , pressure2},
          mol1 = mass1 / Interpreter ["Chemical "][chem1]["MolarMass "];
          mol2 = mass2 / Interpreter ["Chemical "][chem2]["MolarMass "];
          molTotal = mol1 + mol2 ;
          pressureTotal = Quantity [totalPressure , "MillimetersOfMercury "];
          {molFrac1 = mol1 / molTotal , molFrac2 = mol2 / molTotal };
          {molFrac1 → chem1 , molFrac2 → chem2};
          pressure1 = molFrac1 * pressureTotal ;
          pressure2 = molFrac2 * pressureTotal ;
          {pressure1 → chem1 , pressure2 → chem2}]

In[ ]:= calcPartialPressure [RandomChoice [simpleGases ], 20 , RandomChoice [simpleGases ], 35 , 40]

Out[ ]= { 10.003 mmHg → Ozone , 29.997 mmHg → Carbon Monoxide }
```

6. Ideal Gas Law

Calculate the number of moles using the ideal gas law equation: PV = nRT. Input variables are (1) pressure of the system (Torr), volume (Milliliters), and the temperature (Kelvin).

```
In[ ]:= calcIdealGasLawMoles [pressure_ , vol_ , temp_] :=
         Module[{rGasAtm},
          rGasAtm = Quantity [0.0820578 , ("Liters " "Atmospheres ")/("Kelvins " "Moles ")];
          (Quantity [pressure , "Torr"] * Quantity [vol , "Milliliters "]) /
                    (rGasAtm * Quantity [temp , "Kelvins "])]

In[ ]:= calcIdealGasLawMoles [500 , 750 , 300]

Out[ ]= 0.0200436 mol
```

7. Clausius-Clapeyron Equation

The Clausius-Clapeyron Equation and allows us to estimate the vapor pressure at another temperature, if the vapor pressure is known at some temperature, and if the enthalpy of vaporization is known.

$$\ln\left(\frac{P_1}{P_2}\right) = \frac{\Delta H_{vap}}{R}\left(\frac{1}{T_2} - \frac{1}{T_1}\right)$$

Compute ΔHvap from random pressures and temperatures. The user inputs pressures in Torr units (it

doesn't matter because it will cancel out) and temperatures in Kelvins. The gas constant is defined in the function. The function returns ΔH.

```
In[ ]:= calcCCEquation[pressure1_, temp1_, pressure2_, temp2_] :=
 Module[{p1, p2, rGasConstant, t1, t2},
  p1 = Quantity[pressure1, "Torr"];
  p2 = Quantity[pressure2, "Torr"];
  rGasConstant = Quantity[8.314, "Joules"/("Kelvins" "Moles")];
  t1 = Quantity[temp1, "Kelvins"];
  t2 = Quantity[temp2, "Kelvins"];
  UnitConvert[rGasConstant * Log[p1/p2] / (1/t2 - 1/t1), "Kilojoules"/"Moles"]]
```

The vapor pressures of ice at 268 K and 273 K are 2.965 and 4.560 torr respectively. Estimate the heat of sublimation of ice.

```
In[ ]:= calcCCEquation[2.965, 268, 4.560, 273]
Out[ ]= 52.3668 kJ/mol
```

Calculate ΔHvap for ethanol, given vapor pressure at 40 °C is 150 torr. The normal boiling point for ethanol is 78 °C.

```
In[ ]:= calcCCEquation[150, 313, 760, 351]
Out[ ]= 39.0042 kJ/mol
```

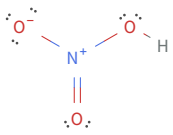
8. Lewis Dot Structures

A question and the answer are automatically generated asking for the Lewis dot structure from simple-Chemicalsv3 list.

```
In[ ]:= lewisStructureQuestion[] := Module[{chem1, chemAns},
   chem1 = RandomChoice[simpleChemicalsv3];
   chemAns = Entity["Chemical", chem1]["LewisDotStructureDiagram"];
   Print[StringJoin["Draw the Lewis dot structure for ", ToString[chem1], "."]];
   Print[chemAns]
  ]
```

```
In[ ]:= lewisStructureQuestion[]
```

```
Draw the Lewis dot structure for NitricAcid .
```



9. Periodic Trends

For loops to get the periodic trends of x number of elements. Atomic mass, Electronegativity, and atomic radius are shown here.

Atomic masses and radii of the first ten elements in the periodic table:

In[17]:=
```
For[i = 1, i ≤ 10, i++,
 Print[ElementData[i, "Abbreviation"], " ",
  ElementData[i, "AtomicMass"], " ", ElementData[i, "AtomicRadius"]]]
```

```
H  1.008 u  53. pm

He  4.002602 u  31. pm

Li  6.94 u  167. pm

Be  9.0121831 u  112. pm

B  10.81 u  87. pm

C  12.011 u  67. pm

N  14.007 u  56. pm

O  15.999 u  48. pm

F  18.998403163 u  42. pm

Ne  20.1797 u  38. pm
```

Electronegativity of elements in the second period:

In[ ]:=
```
For[i = 11, i ≤ 18, i++,
 Print[ElementData[i, "Abbreviation"], " ", ElementData[i, "Electronegativity"]]]
```

```
Na 0.93

Mg 1.31

Al 1.61

Si 1.90

P 2.19

S 2.58

Cl 3.16

Ar Missing[NotApplicable]
```

10. While loop to select elements based on specified

criterion.

Display atomic mass of elements that have an atomic mass less than 30 atomic mass units.

In[ ]:=
```
i = 1;
While[ElementData[i, "AtomicMass"] ≤ Quantity[30, "AtomicMassUnit"],
 Print[i, " ", ElementData[i, "Abbreviation"], " ", ElementData[i, "AtomicMass"]];
 i++]
```

```
1 H 1.008 u
2 He 4.002602 u
3 Li 6.94 u
4 Be 9.0121831 u
5 B 10.81 u
6 C 12.011 u
7 N 14.007 u
8 O 15.999 u
9 F 18.998403163 u
10 Ne 20.1797 u
11 Na 22.98976928 u
12 Mg 24.305 u
13 Al 26.9815385 u
14 Si 28.085 u
```

11. Mass to Moles Problems

A web deployable problem set for conversion of mass to moles for common chemicals. The form allows the user to input the number of problems to be generated and provides the answer for each of the generated problem.

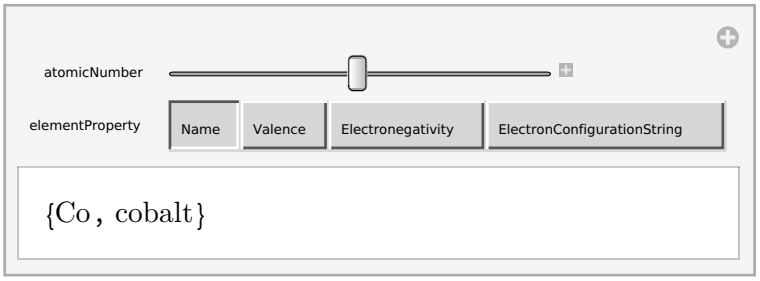
```
In[ ]:= chemicalsGenChem = {Entity["Chemical", "Water"],
          Entity["Chemical", "CarbonDioxide"], Entity["Chemical", "CarbonMonoxide"],
          Entity["Chemical", "Methane"], Entity["Chemical", "CopperIISulfate"],
          Entity["Chemical", "SodiumChloride"], Entity["Chemical", "PotassiumChloride"],
          Entity["Chemical", "AceticAcid"], Entity["Chemical", "NitricAcid"],
          Entity["Chemical", "IronIIOxide"], Entity["Chemical", "IronIIIOxide"],
          Entity["Chemical", "PotassiumSulfate"], Entity["Chemical", "MagnesiumChloride"],
          Entity["Chemical", "SodiumBicarbonate"], Entity["Chemical", "SulfuricAcid"],
          Entity["Chemical", "SodiumHypochlorite"], Entity["Chemical", "SodiumHydroxide"],
          Entity["Chemical", "CalciumCarbonate"], Entity["Chemical", "CalciumOxide"],
          Entity["Chemical", "MagnesiumSulfate"], Entity["Chemical", "HydrochloricAcid"],
          Entity["Chemical", "NitricAcid"], Entity["Chemical", "Ammonia"],
          Entity["Chemical", "PhosphoricAcid"], Entity["Chemical", "PotassiumDichromate"]};

In[ ]:= masstoMolesProblems[numProblems_Integer] := Module[{chemicalNames,
          chemicalAmounts, mmProblemStatements, mmSolutions, workingPrecision = 2},
        chemicalNames = RandomChoice[chemicalsGenChem, numProblems];
        chemicalAmounts = Table[RandomReal[{1, 5}, WorkingPrecision → workingPrecision] *
            RandomChoice[{mg, μg, ng, g}], numProblems];
        mmProblemStatements = Table[StringJoin["Convert ", ToString[chemicalAmounts[[i]]],
            " of ", ToString[EntityValue[chemicalNames[[i]], "Name"]],
            " to moles."], {i, 1, numProblems}];
        mmSolutions = chemicalAmounts / EntityValue[chemicalNames, "MolarMass"];
        TableForm[Transpose[{mmProblemStatements, mmSolutions}],
          TableHeadings → {Range[numProblems], {"Problem", "Solution"}}]
        ]

In[ ]:= CloudDeploy[FormFunction[{"problems" → Restricted["Integer", {1, 10}]},
          masstoMolesProblems[#problems] &, "CloudCDF", AppearanceRules → <|
            "Title" → "Mass to Moles Problems", "Description" → "Conversion problems"|>],
        Permissions → "Public", IncludeDefinitions → True]

Out[ ]= CloudObject[https://www.wolframcloud.com/obj/d7073f74-f0d5-4928-a8bc-6e07b9cd2de8]
```

12. Interactive displays

Use Manipulate to present information that the user can control.

In[ ]:=
```
Manipulate[{Style[ElementData[atomicNumber, "Abbreviation"],
    FontFamily → "Latin Modern Math", FontSize → 16],
   Style[ElementData[atomicNumber, elementProperty],
    FontFamily → "Latin Modern Math", FontSize → 16]},
  {atomicNumber, 1, 54, 1}, {elementProperty,
   {"Name", "Valence", "Electronegativity", "ElectronConfigurationString"}}]
```

Out[ ]=



In[ ]:=
```
Manipulate[ChemicalData[molecule, property], {molecule, simpleChemicalsv2},
 {property, {"BlackStructureDiagram", "MoleculePlot", "CompoundFormulaString"}}]
```

Out[ ]=