chemlib

Release v1.0

Hari Ambethkar

Aug 25, 2022

CONTENTS

Chemlib is a pure Python library that supports a variety of functions pertaining to the vast field of chemistry. It is under active development and continually improved to include more and more features.

CHAPTER

ONE

INSTALLATION

Use the Python Package Installer (PyPI):

```
$ pip install -U chemlib
```

CHAPTER

TWO

CONTENTS

2.1 Core Data

2.1.1 Periodic Table

class chemlib.chemistry.**PeriodicTable**(**args: Any*, ***kwargs: Any*)

Bases:

Column Names

```
>>> list(chemlib.pte)   #Column names
['AtomicNumber', 'Element', 'Symbol', 'AtomicMass', 'Neutrons', 'Protons', 'Electrons',
↪'Period', 'Group', 'Phase', 'Radioactive', 'Natural', 'Metal', 'Nonmetal', 'Metalloid',
↪ 'Type', 'AtomicRadius', 'Electronegativity', 'FirstIonization', 'Density',
↪'MeltingPoint', 'BoilingPoint', 'Isotopes', 'Discoverer', 'Year', 'SpecificHeat',
↪'Shells', 'Valence', 'Config', 'MassNumber']
```

2.1.2 Elements

class chemlib.chemistry.**Element**(**args: Any*, ***kwargs: Any*)

Contains all the properties of the respective element:

```
>>> from chemlib import Element
>>> xenon = Element('Xe') #Instantiate with symbol of Element
>>> xenon.properties
{'AtomicNumber': 54.0, 'Element': 'Xenon', 'Symbol': 'Xe', 'AtomicMass': 131.293,
↪'Neutrons': 77.0, 'Protons': 54.0, 'Electrons': 54.0, 'Period': 5.0, 'Group': 18.0,
↪'Phase': 'gas', 'Radioactive': False, 'Natural': True, 'Metal': False, 'Nonmetal':␣
↪True, 'Metalloid': False, 'Type': 'Noble Gas', 'AtomicRadius': '1.2',
↪'Electronegativity': nan, 'FirstIonization': '12.1298', 'Density': '0.00589',
↪'MeltingPoint': '161.45', 'BoilingPoint': '165.03', 'Isotopes': 31.0, 'Discoverer':
↪'Ramsay and Travers', 'Year': '1898', 'SpecificHeat': '0.158', 'Shells': 5.0, 'Valence
↪': 8.0, 'Config': '[Kr] 4d10 5s2 5p6', 'MassNumber': 131.0}
>>> xenon.AtomicMass
131.293
>>> xenon.FirstIonization
'12.1298'
```

chemlib.chemistry.Element.**AtomicNumber: float**

chemlib.chemistry.Element.**Element: str = The name of the element**

chemlib.chemistry.Element.**Symbol: str = The symbol of the element**

chemlib.chemistry.Element.**AtomicMass: float**

chemlib.chemistry.Element.**Neutrons: float**

chemlib.chemistry.Element.**Protons: float**

chemlib.chemistry.Element.**Electrons: float**

chemlib.chemistry.Element.**Period: float**

chemlib.chemistry.Element.**Group: float**

chemlib.chemistry.Element.**Phase: str = The state of matter of the element at room temperature.**

chemlib.chemistry.Element.**Radioactive: boolean**

chemlib.chemistry.Element.**Natural: boolean**

chemlib.chemistry.Element.**Metal: boolean**

chemlib.chemistry.Element.**Nonmetal: boolean**

chemlib.chemistry.Element.**Metalloid: boolean**

chemlib.chemistry.Element.**Type: str**

chemlib.chemistry.Element.**AtomicRadius: str**

chemlib.chemistry.Element.**Electronegativity: str or NaN**

chemlib.chemistry.Element.**FirstIonization: str or NaN**

chemlib.chemistry.Element.**Density: float = The density in kg/L**

chemlib.chemistry.Element.**MeltingPoint: float**

chemlib.chemistry.Element.**BoilingPoint: float**

chemlib.chemistry.Element.**Isotopes: float**

chemlib.chemistry.Element.**Discoverer: str**

chemlib.chemistry.Element.**Year: str = Year of discovery**

chemlib.chemistry.Element.**SpecificHeat: float**

chemlib.chemistry.Element.**Shells: float**

chemlib.chemistry.Element.**Valence: float**

chemlib.chemistry.Element.**Config: float = Electron configuration**

chemlib.chemistry.Element.**MassNumber: float**

2.1.3 Other Constants

chemlib.**AVOGADROS_NUMBER: float = 6.02e+23**

Contains Avogaadro's Number, which relates the number of constituent particles in a sample with the amount of substance in that sample.

```
>>> import chemlib
>>> chemlib.AVOGADROS_NUMBER
6.02e+23
```

chemlib.**c: float = 2.998e+8**

The speed of light.

chemlib.**h: float = 6.626e-34**

Planck's constant.

chemlib.**R: float = 1.0974e+7**

Rydberg constant.

2.2 Chemical Compounds

2.2.1 Making a Compound

class chemlib.chemistry.**Compound**(**args: Any*, ***kwargs: Any*)

Instantiate a chemlib.Compound object with the formula of the compound.

```
>>> from chemlib import Compound
>>> water = Compound("H2O")
>>> water.formula
'H2O1'
```

chemlib.chemistry.Compound.**occurences: dict**

A dictionary containing the frequencies of the constituent elements in the compound.

```
>>> water.occurences
{'H': 2, 'O': 1}
```

2.2.2 Molar Mass

chemlib.chemistry.Compound.**molar_mass**(*self*)

Returns
The molar mass in (g/mol) of the compound

Return type
float

```
>>> water.molar_mass()
18.01
```

2.2.3 Percentage Composition by Mass

chemlib.chemistry.Compound.**percentage_by_mass**(*self*, *element*)

Get the percentage composition by mass of a certain element of the compound.

Parameters
element (*str*) – The constituent element of which the user wants to get percentage composition.

Returns
The percentage composition by mass of the element in the compound.

Return type
float

```
>>> water.percentage_by_mass('H') #Percent of Hydrogen in Compound
11.183
>>> water.percentage_by_mass('O') #Percent of Oxygen in Compound
88.834
```

2.2.4 Stoichiometry

chemlib.chemistry.Compound.**get_amounts**(*self*, ***kwargs*)

Get stoichiometric amounts of the compound given one measurement.

Parameters

- **compound_number** (*int*) – The chosen compound in the reaction by order of appearance.
- **kwargs** – The amount of the chosen compound (grams=, moles=, or molecules=)

Returns
The gram, mole, and molecule amounts of the compound.

Return type
dict

Raises

- **ValueError** – If the kwargs argument isn't either grams, moles, or molecules
- **ValueError** – if more than one argument is given under kwargs

Get the amount of moles and molecules of water given 2 grams of water.

```
>>> water.get_amounts(grams = 2)
{'Compound': 'H2O1', 'Grams': 2, 'Moles': 0.111, 'Molecules': 6.685e+22}
```

Get the amount of grams and molecules of water given 2 moles of water.

```
>>> water.get_amounts(moles = 2)
{'Compound': 'H2O1', 'Grams': 36.02, 'Moles': 2, 'Molecules': 1.204e+24}
```

Get the amount of moles and grams of water given 2e+24 molecules of water.

```
>>> water.get_amounts(molecules = 2e+24)
{'Compound': 'H2O1', 'Grams': 59.834, 'Moles': 3.3223, 'Molecules': 2e+24}
```

2.3 Empirical Formulae

2.3.1 EF by Percentage Composition

chemlib.chemistry.**empirical_formula_by_percent_comp**(***kwargs*)

Get the empirical formula given the percentage compositions of all elements in the compound.

Parameters
kwargs – The percentage compositions of elements in the compound (`<Element symbol> = <Percentage Composition> ...`)

Returns
The empirical formula of the compound.

Return type
chemlib.chemistry.Compound

Raises
ValueError – If the sums of the percentages is not equal to 100.

Get the empirical formula of a compound that is composed of 80.6% C, and 19.4% H by mass:

```
>>> from chemlib import empirical_formula_by_percent_comp as efbpc
>>> efbpc(C = 80.6, H = 19.4)
'C1H3'
```

2.3.2 Combustion Analysis

chemlib.chemistry.**combustion_analysis**(*CO2*, *H2O*)

Get the empirical formula of a hydrocarbon given the grams of CO2 and grams of H2O formed from its combustion.

Parameters

- **CO2** – The grams of carbon dioxide formed as a result of the combustion of the hydrocarbon.
- **H2O** – The grams of water formed as a result of the combustion of the hydrocarbon.

Returns
The empirical formula of the hydrocarbon.

Return type
str

A hydrocarbon fuel is fully combusted with 18.214 g of oxygen to yield 23.118 g of carbon dioxide and 4.729 g of water. Find the empirical formula for the hydrocarbon.

```
>>> from chemlib.chemistry import combustion_analysis
>>> combustion_analysis(23.118, 4.729)
'CH'
```

2.4 Chemical Reactions

2.4.1 Making a Reaction

class chemlib.chemistry.**Reaction**(**args: Any*, ***kwargs: Any*)

Instantiate a `chemlib.Reaction` object with a list of reactant Compounds and product Compounds.

```
>>> from chemlib import Compound, Reaction
>>> N2O5 = Compound("N2O5")
>>> H2O = Compound("H2O")
>>> HNO3 = Compound("HNO3")
>>> r = Reaction([N2O5, H2O], [HNO3])
```

classmethod chemlib.chemistry.Reaction.**by_formula**(*cls*, *formula: str*)

OR Instantiate a `chemlib.Reaction` object using a string to represent the formula:

In the formula string, place the formulae of reactants separated by + signs on the left side of the arrow `-->`. On the right side of the arrow, place the formulae of products separated by + signs:

```
>>> from chemlib import Reaction
>>> r = Reaction.by_formula("N2O5 + H2O --> HNO3")
```

chemlib.chemistry.Reaction.**formula: str**

```
>>> r.formula
'1N2O5 + 1H2O1 --> 1H1N1O3'
```

chemlib.chemistry.Reaction.**is_balanced: boolean**

```
>>> r.is_balanced
False
```

chemlib.chemistry.Reaction.**reactant_formulas: list**

```
>>> r.reactant_formulas
['N2O5', 'H2O1']
```

chemlib.chemistry.Reaction.**product_formulas: list**

```
>>> r.product_formulas
['H1N1O3']
```

2.4.2 Combustion Reactions

class chemlib.chemistry.**Combustion**(*compound*)

Inherits from chemlib.chemistry.Reaction

Makes a chemical reaction involving the combustion of one compound. Formula will be balanced.

```
>>> from chemlib import Compound, Combustion
>>> methane = Compound('CH4')
>>> c = Combustion(methane)
>>> c.formula
'1C1H4 + 2O2 --> 2H2O1 + 1C1O2'
>>> c.is_balanced
True
```

2.4.3 Balancing the Equation

chemlib.chemistry.Reaction.**balance**(*self*) → None

Balances the chemical equation using linear algebra. See Applications of Linear Algebra in Chemistry.

```
>>> r.balance()
>>> r.formula
'1N2O5 + 1H2O1 --> 2H1N1O3'
>>> r.is_balanced
True
```

2.4.4 Stoichiometry

chemlib.chemistry.Reaction.**get_amounts**(*self*, *compound_number*, ***kwargs*)

Get stoichiometric amounts of ALL compounds in the reaction given the amount of one compound.

Parameters

- **compound_number** (*int*) – The chosen compound in the reaction by order of appearance.
- **kwargs** – The amount of the chosen compound (grams=, moles=, or molecules=)

Returns
the amounts of each compound in the reaction

Return type
list of dicts

Raises

- **ValueError** – if the compound_number is less than 1 or greater than the number of compounds in the reaction
- **ValueError** – if more than one argument is given under kwargs

Best demonstrated by example:

```
>>> r.formula
'1N2O5 + 1H2O1 --> 2H1N1O3'
```

Get the amounts of ALL compounds in the above reaction given 5 grams of N_2O_5. It is the first compound in the reaction by order of appearance (left to right).

```
>>> r.get_amounts(1, grams=5)
[{'Compound': 'N2O5', 'Grams': 5, 'Moles': 0.0463, 'Molecules': 2.787e+22}, {'Compound':
↪'H2O1', 'Grams': 0.834, 'Moles': 0.0463, 'Molecules': 2.787e+22}, {'Compound': 'H1N1O3',
↪ 'Grams': 5.835, 'Moles': 0.0926, 'Molecules': 5.575e+22}]
```

Get the amounts of ALL compounds in the above reaction given 3.5 moles of HNO_3. It is the third compound in the reaction by order of appearance (left to right).

```
>>> r.get_amounts(3, moles=3.5)
[{'Compound': 'N2O5', 'Grams': 189.018, 'Moles': 1.75, 'Molecules': 1.054e+24}, {
↪'Compound': 'H2O1', 'Grams': 31.518, 'Moles': 1.75, 'Molecules': 1.054e+24}, {'Compound
↪': 'H1N1O3', 'Grams': 220.535, 'Moles': 3.5, 'Molecules': 2.107e+24}]
```

2.4.5 Limiting Reagent

chemlib.chemistry.Reaction.**limiting_reagent**(*self*, **args*, *mode='grams'*)

Get the limiting reagent (limiting reactant) in the chemical reaction.

Parameters

- **args** – The amounts of each reactant to use in the chemical reaction.
- **mode** (*str*) – The units of each amount in args. Default is grams, can also be moles or molecules.

Returns

The limiting reagent of the reaction.

Return type

chemlib.chemistry.Compound

Raises

- **TypeError** – If the number of args doesn't match the number of reactants in the reaction.
- **ValueError** – If the mode is not grams, moles, or molecules.

Find the limiting reagent of the reaction when using 50 grams of the first reactant (N_2O_5) and 80 grams of the second reactant (H_2O):

```
>>> lr = r.limiting_reagent(50, 80)
>>> lr.formula
'N2O5'
```

Find the limiting reagent of the reaction when using 3 moles of the first reactant (N_2O_5) and 1 mole of the second reactant (H_2O):

```
>>> lr = r.limiting_reagent(3, 1, mode = 'moles')
>>> lr.formula
'H2O1'
```

2.5 Aqueous Solutions

2.5.1 Acidity Calculation (pH, pOH)

chemlib.chemistry.**pH**(***kwargs*)

For any inputted pH, pOH, [H+], or [OH-], finds the corresponding values.

Parameters
kwargs – The value of the chosen input (pH=, pOH=, H=, or OH=)

Return type
dict

What is the pH, pOH and [OH-] given a [H+] of 1.07x10^-6 M?

```
>>> import chemlib
>>> chemlib.pH(H=1.07e-6)
{'H': 1.07e-06, 'pOH': 8.029, 'pH': 5.971, 'OH': 9.354e-09, 'acidity': 'acidic'}
```

What is the pH, pOH, and [H+] given a [OH-] of 2.06x10^-3 M?

```
>>> chemlib.pH(OH=2.06e-3)
{'OH': 0.002, 'H': 5e-12, 'pOH': 2.699, 'pH': 11.301, 'acidity': 'basic'}
```

What is the pOH, [H+], and [OH-] given a pH of 5.2?

```
>>> chemlib.pH(pH = 5.2)
{'pH': 5.2, 'OH': 1.585e-09, 'H': 6.309e-06, 'pOH': 8.8, 'acidity': 'acidic'}
```

2.5.2 Making a Solution

class chemlib.chemistry.**Solution**(*self*, *solute*, *molarity*)

Instantiate a `chemlib.Solution` object with the formula of the solute, and the molarity in mol/L.

param solute (str)
The formula of the solute without using subscripts OR a `chemlib.chemistry.Compound` object.

param molarity (float)
How many moles of solute per liter of solution

```
>>> from chemlib import Solution
>>> Solution('AgCl', 2)
<chemlib.chemistry.Solution object at 0x03F46370>
>>> s = Solution('AgCl', 2)
>>> s.molarity
2
```

classmethod chemlib.chemistry.Solution.**by_grams_per_liters**(*cls*, *solute*, *grams*, *liters*)

OR you can make a Solution with the solute, and grams per liter.

param str solute
The formula of the solute without using subscripts OR a `chemlib.chemistry.Compound` object.

param float grams
How many grams of solute

param float liters
How many liters of solution

```
>>> from chemlib import Solution
>>> s = Solution.by_grams_per_liters("NaCl", 10, 1)
>>> s.molarity
0.1711
```

2.5.3 Dilutions

chemlib.chemistry.Solution.**dilute**(*self*, *V1=None*, *M2=None*, *V2=None*, *inplace=False*) → dict

Using formula M1*V1 = M2*V2

param float V1
The starting volume of the solution. [Must be specified]

param float M2
The ending molarity after dilution.

param float V2
The ending volume after dilution

param bool inplace
You can set to true if the old molarity is to be replaced by the new molarity

return
The new volume and the new molarity.

rtype
dict

raises TypeError
if a starting volume is not specified

raises TypeError
if both M2 and V2 are specified

To find the dilution of 2.5 L of 0.25M NaCl to a 0.125M NaCl solution:

```
>>> from chemlib import Solution
>>> s = Solution("NaCl", 0.25)
>>> s.dilute(V1 = 2.5, M2 = 0.125)
{'Solute': 'Na1Cl1', 'Molarity': 0.125, 'Volume': 5.0}
```

2.6 Electrochemistry

2.6.1 Galvanic (Voltaic) Cells

class chemlib.electrochemistry.**Galvanic_Cell**(*self*, *electrode1: str*, *electrode2: str*)

Parameters

- **(str)** (*electrode2*) – The elemental composition of one of the electrodes of the galvanic cell.
- **(str)** – The elemental composition of the other electrode of the galvanic cell.

Raises

NotImplementedError – If either of the electrodes is invalid or its reduction potential is unknown.

Make a Galvanic Cell with Lead and Zinc electrodes:

```
>>> from chemlib import Galvanic_Cell
>>> g = Galvanic_Cell("Pb", "Zn")
>>>
```

chemlib.electrochemistry.Galvanic_Cell.**properties: dict**

A dictionary of the cell's properties:

```
>>> g.properties
{'Cell': 'Zn | Zn2+ || Pb2+ | Pb', 'Anode': 'Zn', 'Cathode': 'Pb', 'Cell Potential': 0.
↪63}
>>>
```

chemlib.electrochemistry.Galvanic_Cell.**cell_potential: float**

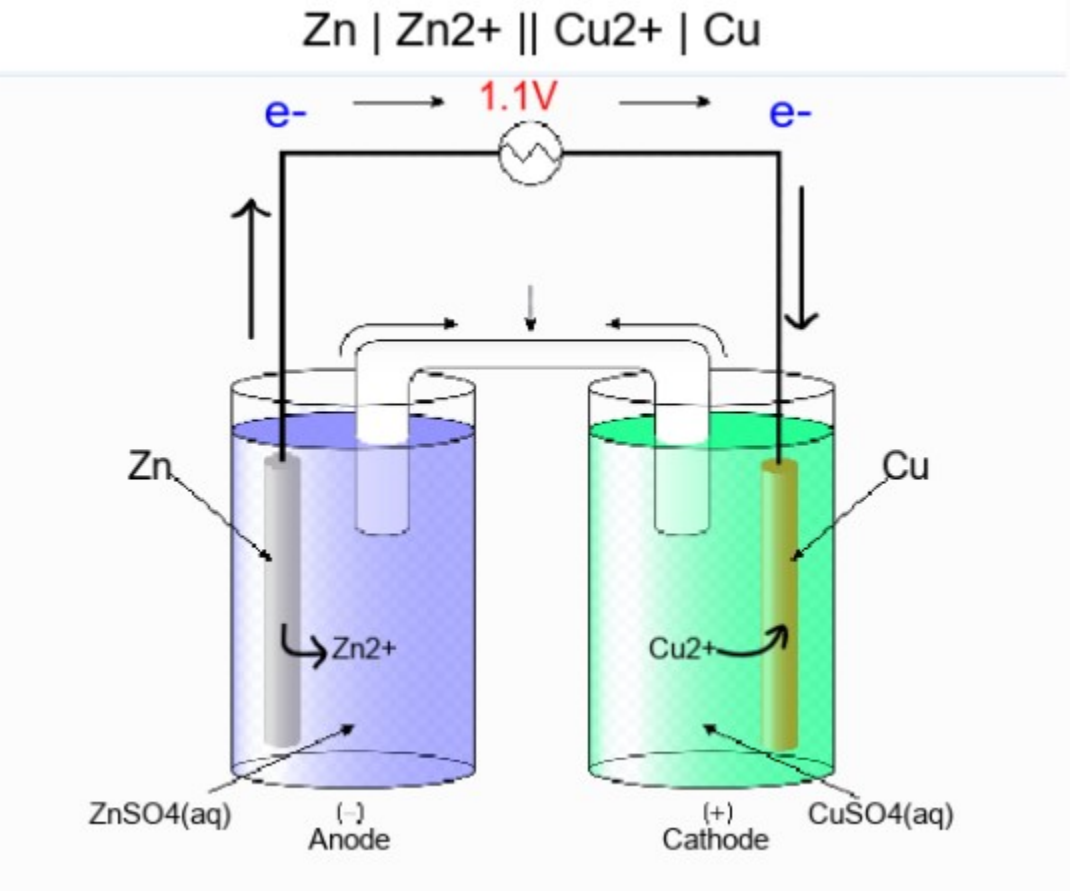
Access the cell potential of the galvanic cell:

```
>>> g.cell_potential
0.63
>>> g.E0
0.63
```

chemlib.electrochemistry.Galvanic_Cell.**diagram: PIL.Image**

The diagram of the galvanic cell is a `PIL.Image` object. To generate diagram:

```
>>> g.draw()
```



To save (as png file):

```
>>> g.diagram.save("filename.png")
>>>
```

2.6.2 Electrolysis

chemlib.electrochemistry.**electrolysis**(*element: str*, *n: int*, ***kwargs*) → dict:

Parameters

- **(str)** (*element*) – The symbol of a chemical element.
- **(int)** (*n*) – The moles of electrons transferred.
- **kwargs** – Provide two of the values from amps, seconds, and grams.

Raises

TypeError – If not only 2 of the parameters in kwargs are specified.

Example: Copper metal is purified by electrolysis. How much copper metal (in grams) could be produced from copper (ii) oxide by applying a current of 10.0 amps at the appropriate negative potential for 12.0 hours?

```
>>> from chemlib import electrolysis
>>> electrolysis('Cu', 2, amps = 10, seconds=12*60*60)
{'element': 'Cu', 'n': 2, 'seconds': 43200, 'amps': 10, 'grams': 142.25979167746283}
>>>
```

Example: How long would it take to electroplate a flute with 28.3 g of silver at a constant current of 2.0 amps using AgNO3?

```
>>> from chemlib import electrolysis
>>> electrolysis("Ag", 2, amps = 2, grams = 28.3)
{'element': 'Ag', 'n': 2, 'seconds': 25313.582341380206, 'amps': 2, 'grams': 28.3}
>>>
```

Example: How much current was used to produce 805 grams of Aluminum metal from Al2O3 in 24 hours?

```
>>> from chemlib import electrolysis
>>> electrolysis("Al", 3, grams = 805, seconds = 24*60*60)
{'element': 'Al', 'n': 3, 'seconds': 86400, 'amps': 99.95144010616133, 'grams': 805}
>>>
```

2.7 Quantum Mechanics

2.7.1 Electromagnetic Waves

class chemlib.quantum_mechanics.**Wave**(***kwargs*)

Makes a Wave object given either wavelength (in meters), frequency (in Hz), or energy (in J per photon), and calculates the aforementioned values.

param kwargs
The value of the known variable (wavelength=, frequency=, or energy=)

Determine the wavelength, frequency, and energy of a wave with frequency 2e+17 Hz:

```
>>> from chemlib import Wave
>>> w = Wave(frequency=2e+17)
>>> w.properties
{'wavelength': 1.499e-09, 'frequency': 2e+17, 'energy': 1.325e-16}
```

Determine the wavelength, frequency, and energy of a wave with wavelength 3e-9 m:

```
>>> w = Wave(wavelength=3e-9)
>>> w.properties
{'wavelength': 3e-09, 'frequency': 9.993e+16, 'energy': 6.622e-17}
```

Determine the wavelength, frequency, and energy of a wave with energy 3e-15 Joules per particle:

```
>>> from chemlib import Wave
>>> w = Wave(energy=3e-15)
>>> w.properties
{'wavelength': 4.387e-44, 'frequency': 4.528e+18, 'energy': 3e-15}
```

chemlib.quantum_mechanics.Wave.**properties:** **dict**

chemlib.quantum_mechanics.Wave.**wavelength:** **float**

chemlib.quantum_mechanics.Wave.**frequency:** **float**

chemlib.quantum_mechanics.Wave.**energy:** **float**

2.7.2 Electrons and Orbitals

chemlib.quantum_mechanics.**energy_of_hydrogen_orbital**(*n*) → float

Gets the energy of an electron in the nth orbital of the Hydrogen atom in Joules.

```
>>> from chemlib import energy_of_hydrogen_orbital
>>> energy_of_hydrogen_orbital(3)
-2.4221749394666667e-19
```

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

E

P

R