
chemistry-tools

Release 1.0.1

Python tools for analysis of chemical compounds.

Dominic Davis-Foster

Apr 18, 2024

Contents

1	Installation	1
1.1	from PyPI	1
1.2	from Anaconda	1
1.3	from GitHub	1
I	API Reference	3
2	chemistry_tools.elements	5
2.1	References	5
2.2	Examples	5
2.3	alkali_metals	6
2.4	alkaline_earth_metals	6
2.5	transition_metals	6
2.6	triels	7
2.7	tetrels	8
2.8	pnictogens	8
2.9	chalcogens	8
2.10	halogens	9
2.11	noble_gases	9
2.12	lanthanides	9
2.13	actinides	10
2.14	classes	11
3	chemistry_tools.formulae	21
3.1	chemistry_tools.formulae.composition	21
3.2	chemistry_tools.formulae.compound	23
3.3	chemistry_tools.formulae.dataarray	25
3.4	chemistry_tools.formulae.formula	28
3.5	chemistry_tools.formulae.html	37
3.6	chemistry_tools.formulae.iso_dist	38
3.7	chemistry_tools.formulae.latex	42
3.8	chemistry_tools.formulae.parser	43
3.9	chemistry_tools.formulae.species	44
3.10	chemistry_tools.formulae.unicode	47
3.11	chemistry_tools.formulae.utils	48
4	chemistry_tools.pubchem	49
4.1	chemistry_tools.pubchem.atom	51
4.2	chemistry_tools.pubchem.bond	53
4.3	chemistry_tools.pubchem.compound	55
4.4	chemistry_tools.pubchem.description	59

4.5	<code>chemistry_tools.pubchem.enums</code>	61
4.6	<code>chemistry_tools.pubchem.errors</code>	63
4.7	<code>chemistry_tools.pubchem.full_record</code>	64
4.8	<code>chemistry_tools.pubchem.images</code>	65
4.9	<code>chemistry_tools.pubchem.lookup</code>	66
4.10	<code>chemistry_tools.pubchem.properties</code>	66
4.11	<code>chemistry_tools.pubchem.pug_rest</code>	70
4.12	<code>chemistry_tools.pubchem.synonyms</code>	71
4.13	<code>chemistry_tools.pubchem.utils</code>	73
5	<code>chemistry_tools.cache</code>	75
5.1	<code>cache</code>	75
5.2	<code>cache_dir</code>	75
5.3	<code>cached_requests</code>	75
5.4	<code>clear_cache</code>	75
6	<code>chemistry_tools.cas</code>	77
6.1	<code>cas_int_to_string</code>	77
6.2	<code>cas_string_to_int</code>	77
6.3	<code>check_cas_number</code>	77
7	<code>chemistry_tools.constants</code>	79
7.1	<code>Constant</code>	79
7.2	<code>atomic_mass_constant</code>	80
7.3	<code>avogadro_number</code>	80
7.4	<code>boltzmann_constant</code>	80
7.5	<code>electron_radius</code>	80
7.6	<code>faraday_constant</code>	81
7.7	<code>molar_gas_constant</code>	81
7.8	<code>neutron_mass</code>	81
7.9	<code>plancks_constant</code>	81
7.10	<code>prefixes</code>	81
7.11	<code>speed_of_light</code>	81
7.12	<code>vacuum_permittivity</code>	81
8	<code>chemistry_tools.names</code>	83
8.1	<code>cas_from_iupac_name</code>	83
8.2	<code>get_IUPAC_parts</code>	83
8.3	<code>get_IUPAC_sort_order</code>	84
8.4	<code>get_sorted_parts</code>	84
8.5	<code>iupac_name_from_cas</code>	84
8.6	<code>multiplier_regex</code>	84
8.7	<code>re_strings</code>	84
8.8	<code>sort_IUPAC_names</code>	84
8.9	<code>sort_array_by_name</code>	85
8.10	<code>sort_dataframe_by_name</code>	85
9	<code>chemistry_tools.spectrum_similarity</code>	87
9.1	<code>SpectrumSimilarity</code>	87
9.2	<code>create_array</code>	88
9.3	<code>normalize</code>	88
9.4	<code>spectrum_similarity</code>	88
10	<code>chemistry_tools.units</code>	91
10.1	<code>SI_base_registry</code>	91

10.2	allclose	92
10.3	as_latex	92
10.4	cm3	92
10.5	compare_equality	92
10.6	dimension_codes	92
10.7	dm	93
10.8	dm3	93
10.9	format_si_units	93
10.10	format_string	93
10.11	kilogray	93
10.12	kilojoule	93
10.13	m3	93
10.14	m_math_space	94
10.15	micromole	94
10.16	molal	94
10.17	nanomolar	94
10.18	nanomole	94
10.19	per100eV	94
10.20	perMolar_perSecond	94
10.21	umol_per_J	94
II	Contributing	95
11	Contributing	97
11.1	Coding style	97
11.2	Automated tests	97
11.3	Type Annotations	97
11.4	Build documentation locally	98
12	Downloading source code	99
12.1	Building from source	100
13	License	101
	Python Module Index	105
	Index	107

Installation

1.1 from PyPI

```
$ python3 -m pip install chemistry_tools --user
```

1.2 from Anaconda

First add the required channels

```
$ conda config --add channels https://conda.anaconda.org/conda-forge  
$ conda config --add channels https://conda.anaconda.org/domdfcoding
```

Then install

```
$ conda install chemistry_tools
```

1.3 from GitHub

```
$ python3 -m pip install git+https://github.com/domdfcoding/chemistry_tools@master --user
```


Part I

API Reference

`chemistry_tools.elements`

Properties of the chemical elements.

Each chemical element is represented as an object instance. Physicochemical and descriptive properties of the elements are stored as instance attributes.

Originally created by [Christoph Gohlke](#). Licensed under the BSD 3-Clause license

2.1 References

1. <https://www.nist.gov/pml/atomic-weights-and-isotopic-compositions-relative-atomic-mass>
2. <https://en.wikipedia.org/wiki/{element.name}>

2.2 Examples

```
>>> from chemistry_tools.elements import ELEMENTS
>>> ele = ELEMENTS['C']
>>> ele.number
6
>>> ele.symbol
'C'
>>> ele.name
'Carbon'
>>> ele.description[:21]
'Carbon is a member of'
>>> ele.eleconfig
'[He] 2s2 2p2'
>>> ele.eleconfig_dict
{(1, 's'): 2, (2, 's'): 2, (2, 'p'): 2}
>>> str(ELEMENTS[6])
'Carbon'
>>> len(ELEMENTS)
109
>>> sum(ele.mass for ele in ELEMENTS)
14693.181589001004
>>> for ele in ELEMENTS:
...     ele.validate()
```

2.3 alkali_metals

Group 1: Alkali Metals in the Periodic Table.

Li
Element representing Lithium

Na
Element representing Sodium

K
Element representing Potassium

Rb
Element representing Rubidium

Cs
Element representing Caesium

Fr
Element representing Francium

2.4 alkaline_earth_metals

Group 2: Alkaline Earth Metals in the Periodic Table.

Be
Element representing Beryllium

Mg
Element representing Magnesium

Ca
Element representing Calcium

Sr
Element representing Strontium

Ba
Element representing Barium

Ra
Element representing Radium

2.5 transition_metals

Transition Metals block in the Periodic Table.

Sc
Element representing Scandium

Ti
Element representing Titanium

V
Element representing Vanadium

Cr
Element representing Chromium

Mn
Element representing Manganese

Fe
Element representing Iron

Co
Element representing Cobalt

Ni
Element representing Nickel

Cu
Element representing Copper

Zn
Element representing Zinc

Y
Element representing Yttrium

Zr
Element representing Zirconium

Nb <i>Element</i> representing Niobium	Ir <i>Element</i> representing Iridium
Mo <i>Element</i> representing Molybdenum	Pt <i>Element</i> representing Platinum
Tc <i>Element</i> representing Technetium	Au <i>Element</i> representing Gold
Ru <i>Element</i> representing Ruthenium	Hg <i>Element</i> representing Mercury
Rh <i>Element</i> representing Rhodium	Rf <i>Element</i> representing Rutherfordium
Pd <i>Element</i> representing Palladium	Db <i>Element</i> representing Dubnium
Ag <i>Element</i> representing Silver	Sg <i>Element</i> representing Seaborgium
Cd <i>Element</i> representing Cadmium	Bh <i>Element</i> representing Bohrium
Hf <i>Element</i> representing Hafnium	Hs <i>Element</i> representing Hassium
Ta <i>Element</i> representing Tantalum	Mt <i>Element</i> representing Meitnerium
W <i>Element</i> representing Tungsten	Ds <i>Element</i> representing Darmstadtium
Re <i>Element</i> representing Rhenium	Rg <i>Element</i> representing Roentgenium
Os <i>Element</i> representing Osmium	Cn <i>Element</i> representing Roentgenium

2.6 triels

Group 13: Triels (or boron group) in the Periodic Table.

B <i>Element</i> representing Boron	Ga <i>Element</i> representing Gallium
Al <i>Element</i> representing Aluminium	In <i>Element</i> representing Indium

Tl
Element representing Thallium

Nh
Element representing Nihonium

2.7 tetrels

Group 14: Tetrels, carbon group, crystallogens or adamantogens in the Periodic Table.

C
Element representing Carbon

Sn
Element representing Tin

Si
Element representing Silicon

Pb
Element representing Lead

Ge
Element representing Germanium

Fl
Element representing Flerovium

2.8 pnictogens

Group 15: Pnictogens in the Periodic Table.

N
Element representing Nitrogen

Sb
Element representing Antimony

P
Element representing Phosphorus

Bi
Element representing Bismuth

As
Element representing Arsenic

Mc
Element representing Moscovium

2.9 chalcogens

Group 16: Chalcogens in the Periodic Table.

O
Element representing Oxygen

Te
Element representing Tellurium

S
Element representing Sulfur

Po
Element representing Polonium

Se
Element representing Selenium

Lv
Element representing Livermorium

2.10 halogens

Group 17: Halogens in the Periodic Table.

F
Element representing Fluorine

Cl
Element representing Chlorine

Br
Element representing Bromine

I
Element representing Iodine

At
Element representing Astatine

Ts
Element representing Tennessine

2.11 noble_gases

Group 18: Noble Gases in the Periodic Table.

He
Element representing Helium

Ne
Element representing Neon

Ar
Element representing Argon

Kr
Element representing Krypton

Xe
Element representing Xenon

Rn
Element representing Radon

Og
Element representing Oganesson

2.12 lanthanides

Lanthanides (or lanthanoids) in the Periodic Table.

La
Element representing Lanthanum

Ce
Element representing Cerium

Pr
Element representing Praseodymium

Nd
Element representing Neodymium

Pm
Element representing Promethium

Sm
Element representing Samarium

Eu
Element representing Europium

Gd
Element representing Gadolinium

Tb
Element representing Terbium

Dy
Element representing Dysprosium

Ho
Element representing Holmium

Er
Element representing Erbium

Tm
Element representing Thulium

Yb
Element representing Ytterbium

Lu
Element representing Lutetium

2.13 actinides

Actinides (or actinoids) in the Periodic Table.

Ac
Element representing Actinium

Th
Element representing Thorium

Pa
Element representing Protactinium

U
Element representing Uranium

Np
Element representing Neptunium

Pu
Element representing Plutonium

Am
Element representing Americium

Cm
Element representing Curium

Bk
Element representing Berkelium

Cf
Element representing Californium

Es
Element representing Einsteinium

Fm
Element representing Fermium

Md
Element representing Mendelevium

No
Element representing Nobelium

Lr
Element representing Lawrencium

2.14 classes

Provides classes to model period table elements.

Classes:

<i>Element</i> (number, symbol, name[, group, ...])	Chemical element.
<i>Elements</i> (*elements)	Ordered dict of Elements with lookup by number, symbol, and name.
<i>HeavyHydrogen</i> (number, symbol, name[, group, ...])	Subclass of <i>Element</i> to handle the Heavy Hydrogen isotopes Deuterium and Tritium.
<i>Isotope</i> ([mass, abundance, massnumber])	Isotope massnumber, relative atomic mass, and abundance.

Data:

<i>IsotopeDict</i>	Type alias for isotope dictionaries.
--------------------	--------------------------------------

```
class Element (number, symbol, name, group=0, period=0, block="", series=0, mass=0.0, eleneg=0.0,
               eleaffin=0.0, covrad=0.0, atmrad=0.0, vdwrad=0.0, tboil=0.0, tmelt=0.0, density=0.0,
               eleconfig="", oxistates="", ionenergy=None, isotopes=None, description="")
```

Bases: Dictable

Chemical element.

Parameters

- **number** (*int*) – The atomic number of the element.
- **symbol** (*str*) – The chemical symbol of the element.
- **name** (*str*) – The name of the element in English.
- **group** (*int*) – The number of electrons in the element. Default 0.
- **period** (*int*) – The number of protons in the element. Default 0.
- **block** (*str*) – The group of the element in the periodic table. Default ' '.
- **series** (*int*) – The Period of the element in the periodic table. Default 0.
- **mass** (*float*) – The relative atomic mass. Default 0.0.
- **eleneg** (*float*) – The Electronegativity (Pauling scale). Default 0.0.
- **eleaffin** (*float*) – The electron affinity in eV. Default 0.0.
- **covrad** (*float*) – The Covalent radius in Angstrom. Default 0.0.
- **atmrad** (*float*) – The Atomic radius in Angstrom. Default 0.0.
- **vdwrad** (*float*) – The Van der Waals radius in Angstrom. Default 0.0.
- **tboil** (*float*) – The boiling temperature in K. Default 0.0.
- **tmelt** (*float*) – The melting temperature in K. Default 0.0.
- **density** (*float*) – The density at 295K in g/cm³ respectively g/L. Default 0.0.
- **eleconfig** (*str*) – The Ground state electron configuration. Default ' '.
- **oxistates** (*str*) – The oxidation states. Default ' '.

- **ionenergy** (Optional[Tuple]) – The ionization energies in eV. Default `None`.
- **isotopes** (Optional[Dict[int, Union[Isotope, Tuple[float, float]]]]) – The Isotopic composition. A mapping of isotope mass numbers to *Isotope* objects. Default `None`.
- **description** (str) – A description of the element. Default `' '`.

Methods:

<code>__repr__()</code>	Return a string representation of the <i>Element</i> .
<code>__str__()</code>	Return <code>str(self)</code> .
<code>validate()</code>	Check consistency of the data.

Attributes:

<i>atmrad</i>	The Atomic radius in Angstrom.
<i>block</i>	The Block of the element in the periodic table.
<i>covrad</i>	The Covalent radius in Angstrom.
<i>density</i>	The density at 295K in g/cm ³ respectively g/L.
<i>description</i>	A description of the element.
<i>eleaffin</i>	The electron affinity in eV.
<i>eleconfig</i>	The Ground state electron configuration.
<i>eleconfig_dict</i>	The ground state electron configuration.
<i>electrons</i>	The number of electrons in the element.
<i>eleneg</i>	The Electronegativity (Pauling scale).
<i>elshells</i>	The number of electrons per shell as tuple.
<i>exactmass</i>	The relative atomic mass calculated from the isotopic composition.
<i>group</i>	The group of the element in the periodic table.
<i>ionenergy</i>	The ionization energies in eV.
<i>isotopes</i>	The Isotopic composition.
<i>mass</i>	The relative atomic mass.
<i>molecular_weight</i>	The relative atomic mass.
<i>name</i>	The name of the element in English.
<i>neutrons</i>	The number of neutrons in the most abundant natural stable isotope.
<i>nominalmass</i>	The mass number of the most abundant natural stable isotope.
<i>number</i>	The atomic number of the element.
<i>oxistates</i>	The oxidation states.
<i>period</i>	The Period of the element in the periodic table.
<i>protons</i>	The number of protons in the element.
<i>series</i>	Index to chemical series.
<i>symbol</i>	The chemical symbol of the element.
<i>tboil</i>	The boiling temperature in K.
<i>tmelt</i>	The melting temperature in K.
<i>vdwrad</i>	The Van der Waals radius in Angstrom.

`__repr__()`
Return a string representation of the *Element*.

Return type `str`

__str__()

Return `str(self)`.

Return type `str`

property atmrad

The Atomic radius in Angstrom.

Return type `float`

property block

The Block of the element in the periodic table.

Return type `str`

property covrad

The Covalent radius in Angstrom.

Return type `float`

property density

The density at 295K in g/cm³ respectively g/L.

Return type `float`

property description

A description of the element.

Return type `str`

property eleaffin

The electron affinity in eV.

Return type `float`

property eleconfig

The Ground state electron configuration.

Return type `str`

property eleconfig_dict

The ground state electron configuration.

Mapping of Tuple(shell, subshell): electrons.

Return type `Dict[Tuple, int]`

property electrons

The number of electrons in the element.

Return type `int`

property eleneg

The Electronegativity (Pauling scale).

Return type `float`

property eleshells

The number of electrons per shell as tuple.

Return type `Tuple[int, ...]`

property exactmass

The relative atomic mass calculated from the isotopic composition.

Return type `float`

property group

The group of the element in the periodic table.

Return type `int`

property ionenergy

The ionization energies in eV.

Return type `Tuple`

property isotopes

The Isotopic composition.

- **keys:** isotope mass number
- **values:** Isotope(relative atomic mass, abundance)

Return type `Dict[int, Isotope]`

property mass

The relative atomic mass.

Ratio of the average mass of atoms.

Return type `float`

property molecular_weight

The relative atomic mass.

Ratio of the average mass of atoms.

Return type `float`

property name

The name of the element in English.

Return type `str`

property neutrons

The number of neutrons in the most abundant natural stable isotope.

Return type `int`

property nominalmass

The mass number of the most abundant natural stable isotope.

Return type `int`

property number

The atomic number of the element.

Return type `int`

property oxistates

The oxidation states.

Return type `str`

property period

The Period of the element in the periodic table.

Return type `int`

property protons

The number of protons in the element.

Return type `int`

property series

Index to chemical series.

Return type `int`

property symbol

The chemical symbol of the element.

Return type `str`

property tboil

The boiling temperature in K.

Return type `float`

property tmelt

The melting temperature in K.

Return type `float`

validate()

Check consistency of the data.

Raises `ValueError` – If there are any validation issues.

property vdwradius

The Van der Waals radius in Angstrom.

Return type `float`

class **Elements** (**elements*)

Bases: `Iterable[Element]`

Ordered dict of Elements with lookup by number, symbol, and name.

Parameters ***elements** (*Element*) – The elements to add to the dictionary.

Methods:

<code>__contains__(item)</code>	Return key in self.
<code>__getitem__(key)</code>	Return self[key].
<code>__iter__()</code>	Returns an iterator over the elements, in order.
<code>__len__()</code>	Returns the number of elements.
<code>__repr__()</code>	Return a string representation of the <i>Elements</i> .
<code>__str__()</code>	Return <code>str(self)</code> .
<code>add_alternate_spelling(element, spelling)</code>	Adds an alternate spelling for an element.
<code>split_isotope(string)</code>	Returns the symbol and mass number for the isotope represented by string.

Attributes:

<code>lower_names</code>	The names of the elements, all in lowercase.
<code>names</code>	The names of the elements.
<code>symbols</code>	The symbols of the elements.

`__contains__(item)`

Return key in self.

Return type `bool`

`__getitem__(key)`

Return self[key].

Parameters **key** – If a string, return the *Element* with that name or symbol. If a number, return the element with that atomic number.

Overloads

- `__getitem__(key: slice) -> List[Element]`
- `__getitem__(key: Union[str, int, float]) -> Element`

`__iter__()`

Returns an iterator over the elements, in order.

Return type `Iterator[Element]`

`__len__()`

Returns the number of elements.

Return type `int`

`__repr__()`

Return a string representation of the *Elements*.

Return type `str`

__str__()

Return `str(self)`.

Return type `str`

add_alternate_spelling (*element*, *spelling*)

Adds an alternate spelling for an element.

Parameters

- **element** (*Element*)
- **spelling** (`str`)

property lower_names

The names of the elements, all in lowercase.

Return type `List[str]`

property names

The names of the elements.

Return type `List[str]`

split_isotope (*string*)

Returns the symbol and mass number for the isotope represented by *string*.

Valid isotopes include ' [C12] ', ' C[12] ' and ' [12C] '.

Parameters **string** (`str`)

Return type `Tuple[str, int]`

Returns Tuple representing the element and the isotope number.

property symbols

The symbols of the elements.

Return type `List[str]`

class HeavyHydrogen (*number*, *symbol*, *name*, *group*=0, *period*=0, *block*="", *series*=0, *mass*=0.0, *eleneg*=0.0, *eleaffin*=0.0, *covrad*=0.0, *atmrad*=0.0, *vdwrad*=0.0, *tboil*=0.0, *tmelt*=0.0, *density*=0.0, *eleconfig*="", *oxistates*="", *ionenergy*=None, *isotopes*=None, *description*="")

Bases: *Element*

Subclass of *Element* to handle the Heavy Hydrogen isotopes Deuterium and Tritium.

Chemical element.

Parameters

- **number** (`int`) – The atomic number of the element.
- **symbol** (`str`) – The chemical symbol of the element.
- **name** (`str`) – The name of the element in English.
- **group** (`int`) – The number of electrons in the element. Default 0.
- **period** (`int`) – The number of protons in the element. Default 0.

- **block** (*str*) – The group of the element in the periodic table. Default ' '.
- **series** (*int*) – The Period of the element in the periodic table. Default 0.
- **mass** (*float*) – The relative atomic mass. Default 0.0.
- **eleneg** (*float*) – The Electronegativity (Pauling scale). Default 0.0.
- **eleaffin** (*float*) – The electron affinity in eV. Default 0.0.
- **covrad** (*float*) – The Covalent radius in Angstrom. Default 0.0.
- **atmrad** (*float*) – The Atomic radius in Angstrom. Default 0.0.
- **vdwrad** (*float*) – The Van der Waals radius in Angstrom. Default 0.0.
- **tboil** (*float*) – The boiling temperature in K. Default 0.0.
- **tmelt** (*float*) – The melting temperature in K. Default 0.0.
- **density** (*float*) – The density at 295K in g/cm³ respectively g/L. Default 0.0.
- **eleconfig** (*str*) – The Ground state electron configuration. Default ' '.
- **oxistates** (*str*) – The oxidation states. Default ' '.
- **ionenergy** (*Optional[Tuple]*) – The ionization energies in eV. Default *None*.
- **isotopes** (*Optional[Dict[int, Union[Isotope, Tuple[float, float]]]]*) – The Isotopic composition. A mapping of isotope mass numbers to *Isotope* objects. Default *None*.
- **description** (*str*) – A description of the element. Default ' '.

Attributes:

<i>as_isotope</i>	Return the isotope in H[X] format.
<i>nominalmass</i>	Return mass number of most abundant natural stable isotope.

property as_isotope

Return the isotope in H[X] format.

Return type *str***property nominalmass**

Return mass number of most abundant natural stable isotope.

Return type *int***class Isotope** (*mass=0.0, abundance=1.0, massnumber=0*)Bases: *Dictable*

Isotope massnumber, relative atomic mass, and abundance.

Parameters

- **mass** (*float*) – The mass of the isotope. Default 0.0.
- **abundance** (*float*) – The natural abundance of the isotope. Default 1.0.
- **massnumber** (*int*) – The mass number of the isotope. Default 0.

Methods:

<code>__repr__()</code>	Return a string representation of the <i>Isotope</i> .
<code>__str__()</code>	Return <code>str(self)</code> .

Attributes:

<code>abundance</code>	The natural abundance of the isotope.
<code>mass</code>	The mass of the isotope.
<code>massnumber</code>	The mass number of the isotope.

`__repr__()`
Return a string representation of the *Isotope*.

Return type `str`

`__str__()`
Return `str(self)`.

Return type `str`

property abundance
The natural abundance of the isotope.

Return type `float`

property mass
The mass of the isotope.

Return type `float`

property massnumber
The mass number of the isotope.

Return type `int`

IsotopeDict

Type alias for isotope dictionaries.

Alias of `Dict[int, Union[Isotope, Tuple[float, float]]]`

chemistry_tools.formulae

Parse formulae into a Python object.

Attention: This package has the following additional requirements:

```
cawdrey>=0.5.0
mathematical>=0.5.1
pyparsing>=2.4.6
tabulate>=0.8.9
```

These can be installed as follows:

```
$ python -m pip install chemistry-tools[formulae]
```

3.1 chemistry_tools.formulae.composition

Elemental composition of a *Formula*.

Classes:

<i>Composition</i> (formula)	Class to represent the elemental composition of a <i>Formula</i> .
<i>CompositionSort</i> (value)	Lookup for sorting elemental composition output.

class *Composition* (formula)

Bases: *DataArray*

Class to represent the elemental composition of a *Formula*.

Parameters **formula** (*Formula*) – A *Formula* object to create the composition for

Methods:

<i>__str__</i> ()	Return <i>str</i> (self).
<i>as_array</i> ([sort_by, reverse])	Returns the elemental composition as a list of lists.

Attributes:

<i>n_elements</i>	The number of elements in the composition.
<i>total_mass</i>	The total mass of the composition.

`__str__()`

Return `str(self)`.

Return type `str`

as_array (*sort_by*=<CompositionSort.symbol: 'symbol'>, *reverse*=False)

Returns the elemental composition as a list of lists.

Parameters

- **sort_by** (CompositionSort) – The column to sort by. Default <CompositionSort.symbol: 'symbol'>.
- **reverse** (bool) – Whether the isotopologues should be sorted in reverse order. Default False.

Return type `List[List[Any]]`

property n_elements

The number of elements in the composition.

Return type `int`

property total_mass

The total mass of the composition.

Return type `float`

enum CompositionSort (*value*)

Bases: `enum.Enum`

Lookup for sorting elemental composition output.

Valid values are as follows:

`symbol = <CompositionSort.symbol: 'symbol'>`

`count = <CompositionSort.count: 'count'>`

`rel_mass = <CompositionSort.rel_mass: 'rel_mass'>`

`mass_fraction = <CompositionSort.mass_fraction: 'mass_fraction'>`

3.2 chemistry_tools.formulae.compound

Parse formulae into a Python object.

Classes:

<code>Compound(name[, formula, data, latex_name, ...])</code>	Class representing a chemical compound.
---	---

```
class Compound(name, formula=None, data=None, latex_name=None, unicode_name=None,
                html_name=None)
```

Bases: `Dicttable`

Class representing a chemical compound.

Parameters

- **name** (`str`) – The name of the compound
- **formula** (`Optional[Formula]`) – The chemical formula of the compound. If `None` this is generated from the name. Default `None`.
- **data** (`Optional[Dict]`) – Free form dictionary. Default `None`.
- **latex_name** (`Optional[str]`) – Default `None`.
- **unicode_name** (`Optional[str]`) – Default `None`.
- **html_name** (`Optional[str]`) – Default `None`.

data could be simple such as { 'mp': 0, 'bp': 100 } or considerably more involved, e.g.:

```
{
  'diffusion_coefficient': {
    'water': lambda T: 2.1*m**2/s/K*(T - 273.15*K),
  }
}
```

Methods:

<code>__eq__(other)</code>	Return <code>self == other</code> .
<code>__repr__()</code>	Return a string representation of the <code>Compound</code> .
<code>__str__()</code>	Return <code>str(self)</code> .
<code>molar_mass()</code>	Returns the molar mass (with units) of the substance.

Attributes:

<code>charge</code>	The charge of the compound.
<code>mass</code>	The mass of the compound.

```
__eq__(other)
    Return self == other.
```

Return type `bool`

__repr__()
Return a string representation of the *Compound*.

Return type `str`

__str__()
Return `str(self)`.

Return type `str`

property charge
The charge of the compound.

Return type `int`

property mass
The mass of the compound.

Return type `float`

molar_mass()
Returns the molar mass (with units) of the substance.

Example:

```
>>> nh4p = Compound('NH4+')
>>> import quantities
>>> nh4p.molar_mass(quantities)
array(18.0384511...) * g/mol
```

Return type `Quantity`

3.3 chemistry_tools.formulae.dataarray

Provides a base class which can output data as a `pandas.DataFrame`, to CSV, or as a pretty-printed table in a variety of formats.

class DataArray (*formula, data*)

Bases: `FrozenOrderedDict`

A class which can output data as a `pandas.DataFrame`, to CSV, or as a pretty-printed table in a variety of formats.

To use this class it must first be subclassed. Subclasses must implement `as_array()` which handles the conversion of the data to a list of lists of values.

Parameters

- **formula** (`str`) – The formula in hill notation
- **data** (`Dict`) – A dictionary of data to add to the internal `FrozenOrderedDict`

Attributes:

`__class_getitem__`

Methods:

<code>__contains__(key)</code>	Return <code>key</code> in <code>self</code> .
<code>__eq__(other)</code>	Return <code>self == other</code> .
<code>__getitem__(key)</code>	Return <code>self[key]</code> .
<code>__iter__()</code>	Iterates over the dictionary's keys.
<code>__len__()</code>	Returns the number of keys in the dictionary.
<code>__repr__()</code>	Return a string representation of the <code>DataArray</code> .
<code>__str__()</code>	Return <code>str(self)</code> .
<code>as_array(sort_by[, reverse])</code>	Must be implemented in subclasses to hand the conversion of the data to a list of lists of values.
<code>as_csv(*args[, sep])</code>	Returns the data as a CSV formatted string.
<code>as_dataframe(*args, **kwargs)</code>	Returns the isotope distribution data as a <code>pandas.DataFrame</code> .
<code>as_table(*args, **kwargs)</code>	Returns the isotope distribution data as a table using <code>tabulate</code> .
<code>copy(*args, **kwargs)</code>	Return a copy of the <code>FrozenOrderedDict</code> .
<code>fromkeys(iterable[, value])</code>	Create a new dictionary with keys from <code>iterable</code> and values set to <code>value</code> .
<code>get(k[, default])</code>	Return the value for <code>k</code> if <code>k</code> is in the dictionary, else <code>default</code> .
<code>items()</code>	Returns a set-like object providing a view on the <code>FrozenOrderedDict</code> 's items.
<code>keys()</code>	Returns a set-like object providing a view on the <code>FrozenOrderedDict</code> 's keys.
<code>values()</code>	Returns an object providing a view on the <code>FrozenOrderedDict</code> 's values.

`__class_getitem__` = <bound method GenericAlias of <class 'chemistry_tools.formulae.dataarray.DataArray'>>

Type: `MethodType`

__contains__ (*key*)
Return *key* in *self*.

Parameters *key* (*object*)

Return type *bool*

__eq__ (*other*)
Return *self* == *other*.

Return type *bool*

__getitem__ (*key*)
Return *self*[*key*].

Parameters *key* (~KT)

Return type ~VT

__iter__ ()
Iterates over the dictionary's keys.

Return type *Iterator*[~KT]

__len__ ()
Returns the number of keys in the dictionary.

Return type *int*

__repr__ ()
Return a string representation of the *dataArray*.

Return type *str*

__str__ ()
Return *str*(*self*).

Return type *str*

abstract as_array (*sort_by*, *reverse=False*)
Must be implemented in subclasses to hand the conversion of the data to a list of lists of values.

Parameters

- **sort_by** (*Any*)
- **reverse** (*bool*) – Default *False*.

Return type *List*[*List*[*Any*]]

as_csv (**args*, *sep=','*, ***kwargs*)
Returns the data as a CSV formatted string.

Parameters

- ***args** – Arguments passed to *as_array* ().
- **sep** (*str*) – The separator for the CSV data. Default *' , '*.

- ****kwargs** – Additional keyword arguments passed to `as_array()`.

Return type `str`

as_dataframe (**args*, ***kwargs*)

Returns the isotope distribution data as a `pandas.DataFrame`.

Any arguments taken by `as_array()` can also be used here.

Return type `DataFrame`

as_table (**args*, ***kwargs*)

Returns the isotope distribution data as a table using `tabulate`.

Any arguments taken by `as_array()` can also be used here.

Additionally, any valid keyword argument for `tabulate.tabulate()` can be used.

Return type `str`

copy (**args*, ***kwargs*)

Return a copy of the `FrozenOrderedDict`.

Parameters

- **args**
- **kwargs**

classmethod fromkeys (*iterable*, *value=None*)

Create a new dictionary with keys from *iterable* and values set to *value*.

Return type `FrozenBase[~KT, ~VT]`

get (*k*, *default=None*)

Return the value for *k* if *k* is in the dictionary, else *default*.

Parameters

- **k** – The key to return the value for.
- **default** – The value to return if key is not in the dictionary. Default `None`.

items ()

Returns a set-like object providing a view on the `FrozenOrderedDict`'s items.

Return type `AbstractSet[Tuple[~KT, ~VT]]`

keys ()

Returns a set-like object providing a view on the `FrozenOrderedDict`'s keys.

Return type `AbstractSet[~KT]`

values ()

Returns an object providing a view on the `FrozenOrderedDict`'s values.

Return type `ValuesView[~VT]`

3.4 chemistry_tools.formulae.formula

Parse formulae into a Python object.

Data:

<code>F</code>	Invariant <code>TypeVar</code> bound to <code>chemistry_tools.formulae.formula.Formula</code> .
----------------	---

Classes:

<code>Formula([composition, charge])</code>	A <code>Formula</code> object stores a chemical composition of a compound.
---	--

```
F = TypeVar(F, bound=Formula)
```

```
    Type: TypeVar
```

```
    Invariant TypeVar bound to chemistry_tools.formulae.formula.Formula.
```

```
class Formula(composition=None, charge=0)
```

```
    Bases: defaultdict, Counter
```

A `Formula` object stores a chemical composition of a compound. It is based on `dict`, with the symbols of chemical elements as keys and the values equal to the number of atoms of the corresponding element in the compound.

Parameters

- **composition** (`Optional[Dict[str, int]]`) – A `Formula` object with the elemental composition of a substance, or a `dict` representing the same. If `None` an empty object is created. Default `None`.
- **charge** (`int`) – Default 0.

Methods:

<code>__add__(other)</code>	Return <code>self + value</code> .
<code>__eq__(other)</code>	Return <code>self == other</code> .
<code>__iadd__(other)</code>	Inplace add from another counter, keeping only positive counts.
<code>__imul__(other)</code>	
	rtype <code>Formula</code>
<code>__isub__(other)</code>	Inplace subtract counter, but keep only results with positive counts.
<code>__mul__(other)</code>	Return <code>self * value</code> .
<code>__radd__(other)</code>	Return <code>value + self</code> .
<code>__repr__()</code>	Return a string representation of the <code>Formula</code> .
<code>__rmul__(other)</code>	Return <code>value * self</code> .
<code>__rsub__(other)</code>	Return <code>value - self</code> .
<code>__setitem__(key, value)</code>	Set <code>self[key]</code> to <code>value</code> .
<code>__str__()</code>	Return <code>str(self)</code> .
<code>__sub__(other)</code>	Return <code>value - self</code> .

continues on next page

Table 11 – continued from previous page

<code>copy()</code>	Returns a copy of the <i>Formula</i> .
<code>from_kwargs(*[, charge])</code>	Create a new <i>Formula</i> object from keyword arguments representing the elements in the compound.
<code>from_mass_fractions(fractions[, charge, ...])</code>	Create a new <i>Formula</i> object from elemental mass fractions by parsing a string.
<code>from_string(formula[, charge])</code>	Create a new <i>Formula</i> object by parsing a string.
<code>get_mz([average, charge])</code>	Calculate the average mass:charge ratio (<i>m/z</i>) of a <i>Formula</i> .
<code>isotope_distribution()</code>	Returns an <i>IsotopeDistribution</i> object representing the distribution of the isotopologues of the formula.
<code>iter_isotopologues([report_abundance, ...])</code>	Iterate over possible isotopic states of the molecule.
<code>most_probable_isotopic_composition([...])</code>	Calculate the most probable isotopic composition of a molecule/ion.

Attributes:

<code>average_mass</code>	Calculate the average mass of a <i>Formula</i> .
<code>average_mz</code>	The average mass to charge ratio of the formula.
<code>composition</code>	A <i>Composition</i> object representing the elemental composition of the Formula.
<code>elements</code>	A list of the element symbols in the formula.
<code>empirical_formula</code>	Returns the empirical formula in Hill notation.
<code>exact_mass</code>	Calculate the monoisotopic mass of a <i>Formula</i> .
<code>hill_formula</code>	Returns the formula in Hill notation.
<code>isotopic_composition_abundance</code>	Calculate the relative abundance of the current isotopic composition of this molecule.
<code>mass</code>	Calculate the average mass of a <i>Formula</i> .
<code>monoisotopic_mass</code>	Calculate the monoisotopic mass of a <i>Formula</i> .
<code>mz</code>	The mass to charge ratio of the formula.
<code>n_atoms</code>	Return the number of atoms in the formula.
<code>n_elements</code>	Return the number of elements in the formula.
<code>no_isotope_hill_formula</code>	Returns formula in Hill notation, without any isotopes specified.
<code>nominal_mass</code>	Calculate the monoisotopic mass of a <i>Formula</i> .

`__add__(other)`
Return self + value.

`__eq__(other)`
Return self == other.

Return type `bool`

`__iadd__`(*other*)

Inplace add from another counter, keeping only positive counts.

```
>>> c = Counter('abbb')
>>> c += Counter('bcc')
>>> c
Counter({'b': 4, 'c': 2, 'a': 1})
```

Return type *Formula*

`__imul__`(*other*)

Return type *Formula*

`__isub__`(*other*)

Inplace subtract counter, but keep only results with positive counts.

```
>>> c = Counter('abbbc')
>>> c -= Counter('bccd')
>>> c
Counter({'b': 2, 'a': 1})
```

Return type *Formula*

`__mul__`(*other*)

Return `self * value`.

Return type *Formula*

`__radd__`(*other*)

Return `value + self`.

`__repr__`()

Return a string representation of the *Formula*.

Return type `str`

`__rmul__`(*other*)

Return `value * self`.

`__rsub__`(*other*)

Return `value - self`.

`__setitem__`(*key*, *value*)

Set `self[key]` to *value*.

`__str__`()

Return `str(self)`.

Return type `str`

__sub__(*other*)
Return value - self.

property average_mass
Calculate the average mass of a *Formula*.
Note that mass is not averaged for elements with specified isotopes.
Return type `float`

property average_mz
The average mass to charge ratio of the formula.
Return type `float`

property composition
A *Composition* object representing the elemental composition of the Formula.
Return type *Composition*

copy()
Returns a copy of the *Formula*.
Return type *~F*

property elements
A list of the element symbols in the formula.
Return type `List[str]`

property empirical_formula
Returns the empirical formula in Hill notation.
The empirical formula has the simplest whole number ratio of atoms of each element present in the formula.

Examples:

```
>>> Formula.from_string('H2O').empirical_formula
'H2O'
>>> Formula.from_string('S4').empirical_formula
'S'
>>> Formula.from_string('C6H12O6').empirical_formula
'CH2O'
```

Return type `str`

property exact_mass
Calculate the monoisotopic mass of a *Formula*. If any isotopes are already present in the formula, the mass of these will be preserved
Return type `float`

classmethod `from_kwargs` (*, *charge=0*, ***kwargs*)

Create a new *Formula* object from keyword arguments representing the elements in the compound.

Parameters `charge` (*int*) – Default 0.

Return type *~F*

classmethod `from_mass_fractions` (*fractions*, *charge=0*, *maxcount=10*, *precision=0.0001*)

Create a new *Formula* object from elemental mass fractions by parsing a string.

Note: Isotopes cannot (currently) be parsed using this method

Parameters

- **fractions** (`Dict[str, float]`) – A dictionary of elements and mass fractions
- **charge** (`int`) – Default 0.
- **maxcount** (`int`) – Default 10.
- **precision** (`float`) – Default 0.0001.

Examples:

```
>>> Formula.from_mass_fractions({'H': 0.112, 'O': 0.888})
'H2O'
>>> Formula.from_mass_fractions({'D': 0.2, 'O': 0.8})
'O[2H]2'
>>> Formula.from_mass_fractions({'H': 8.97, 'C': 59.39, 'O': 31.64})
'C5H9O2'
>>> Formula.from_mass_fractions({'O': 0.26, '30Si': 0.74})
'O2[30Si]3'
```

Return type *Formula*

classmethod `from_string` (*formula*, *charge=0*)

Create a new *Formula* object by parsing a string.

Note: Isotopes cannot (currently) be parsed using this method

Parameters

- **formula** (`str`) – A string with a chemical formula
- **charge** (`int`) – Default 0.

Return type *~F*

get_mz (*average=True*, *charge=None*)

Calculate the average mass:charge ratio (*m/z*) of a *Formula*.

Parameters

- **average** (`bool`) – If `True` then the average *m/z* is calculated. Note that the mass is not averaged for elements with specified isotopes. Default `True`.
- **charge** (`Optional[int]`) – The charge of the compound. If `None` then the existing charge of the *Formula* is used. Default `None`.

Return type `float`

property hill_formula

Returns the formula in Hill notation.

Example:

```
>>> Formula.from_string('BrC2H5').hill_formula
'C2H5Br'
>>> Formula.from_string('HBr').hill_formula
'BrH'
>>> Formula.from_string('[ (CH3)3Si2]2NNa').hill_formula
'C6H18NNaSi4'
```

Return type `str`

isotope_distribution()

Returns an *IsotopeDistribution* object representing the distribution of the isotopologues of the formula.

Return type *IsotopeDistribution*

property isotopic_composition_abundance

Calculate the relative abundance of the current isotopic composition of this molecule.

Return type `float`

Returns The relative abundance of the current isotopic composition.

iter_isotopologues (*report_abundance=False, elements_with_isotopes=None, isotope_threshold=0.0005, overall_threshold=0*)

Iterate over possible isotopic states of the molecule.

The space of possible isotopic compositions is restrained by parameters *elements_with_isotopes*, *isotope_threshold*, *overall_threshold*.

Parameters

- **report_abundance** (`bool`) – If `True`, the output will contain 2-tuples: (*composition*, *abundance*). Otherwise, only compositions are yielded. Default `False`.
- **elements_with_isotopes** (*Optional*[*Sequence*[`str`]]) – A set of elements to be considered in isotopic distributions (by default, every element has an isotopic distribution). Default `None`.
- **isotope_threshold** (`float`) – The threshold abundance of a specific isotope to be considered. Default `0.0005`.
- **overall_threshold** (`float`) – The threshold abundance of the calculated isotopic composition. Default `0`.

Return type *Iterator*

Returns Iterator over possible isotopic compositions.

property mass

Calculate the average mass of a *Formula*.

Note that mass is not averaged for elements with specified isotopes.

Return type `float`

property monoisotopic_mass

Calculate the monoisotopic mass of a *Formula*. If any isotopes are already present in the formula, the mass of these will be preserved

Return type `float`

most_probable_isotopic_composition (*elements_with_isotopes=None*)

Calculate the most probable isotopic composition of a molecule/ion.

For each element, only two most abundant isotopes are considered. Any isotopes already in the Formula will be changed to the most abundant isotope

Parameters *elements_with_isotopes* (`Optional[Sequence[str]]`) – A set of elements to be considered in isotopic distribution (by default, every element has an isotopic distribution). Default `None`.

Return type `Tuple[Formula, float]`

Returns A tuple with the most probable isotopic composition and its relative abundance.

property mz

The mass to charge ratio of the formula.

Return type `float`

property n_atoms

Return the number of atoms in the formula.

Example:

```
>>> Formula.from_string('CH3COOH').n_atoms
8
```

Return type `int`

property n_elements

Return the number of elements in the formula.

Return type `int`

Example:

```
>>> Formula.from_string('CH3COOH').n_elements
3
```

property no_isotope_hill_formula

Returns formula in Hill notation, without any isotopes specified.

Example:

```
>>> Formula.from_string('BrC2H5').no_isotope_hill_formula
'C2H5Br'
>>> Formula.from_string('HBr').no_isotope_hill_formula
'BrH'
>>> Formula.from_string('[ (CH3)3Si2]2NNa').no_isotope_hill_formula
'C6H18NNaSi4'
```

Return type `str`

property nominal_mass

Calculate the monoisotopic mass of a *Formula*. If any isotopes are already present in the formula, the mass of these will be preserved

Return type `float`

3.5 chemistry_tools.formulae.html

Functions and constants for converting formulae to HTML.

Functions:

<code>html_subscript(val)</code>	Returns the HTML subscript of the given value.
<code>html_superscript(val)</code>	Returns the HTML superscript of the given value.
<code>string_to_html(formula[, prefixes, infixes, ...])</code>	Convert formula string to HTML string representation.

`html_subscript (val)`

Returns the HTML subscript of the given value.

Parameters `val` (`Union[str, float]`) – The value to superscript

Return type `str`

`html_superscript (val)`

Returns the HTML superscript of the given value.

Parameters `val` (`Union[str, float]`) – The value to subscript

Return type `str`

`string_to_html (formula, prefixes=None, infixes=None, suffixes=('(s)', '(l)', '(g)', '(aq)'))`

Convert formula string to HTML string representation.

Examples:

```
>>> string_to_html("NH4+")
'NH<sub>4</sub><sup>+</sup>'
>>> string_to_html("Fe(CN)6+2")
'Fe(CN)<sub>6</sub><sup>2+</sup>'
>>> string_to_html("Fe(CN)6+2(aq)")
'Fe(CN)<sub>6</sub><sup>2+</sup>(aq)'
>>> string_to_html(".NHO-(aq)")
'&sdot;NHO<sup>-</sup>(aq)'
>>> string_to_html("alpha-FeOOH(s)")
'&alpha;-FeOOH(s)'
```

Parameters

- **formula** (`str`) – Chemical formula, e.g. 'H2O', 'Fe+3', 'Cl-'
- **prefixes** (`Optional[Dict[str, str]]`) – Mapping of prefixes to their HTML equivalents. Default greek letters and .
- **infixes** (`Optional[Dict[str, str]]`) – Mapping of infixes to their HTML equivalents. Default `None`.
- **suffixes** (`Sequence[str]`) – Suffixes to keep. Default ('(s)', '(l)', '(g)', '(aq)').

Return type `str`

Returns The HTML representation of the formula

3.6 chemistry_tools.formulae.iso_dist

Isotope Distributions.

Classes:

<code>IsoDistSort(value)</code>	Lookup for sorting isotope distribution output.
<code>IsotopeDistribution(formula)</code>	An isotope distribution.

enum IsoDistSort (value)

Bases: `enum_tools.custom_enums.IntEnum`

Lookup for sorting isotope distribution output.

Member Type `int`

Valid values are as follows:

Formula = `<IsoDistSort.Formula: 0>`

Sort the isotope distribution by the formulae.

Mass = `<IsoDistSort.Mass: 1>`

Sort the isotope distribution by the masses.

Abundance = `<IsoDistSort.Abundance: 2>`

Sort the isotope distribution by the abundances.

Relative_Abundance = `<IsoDistSort.Relative_Abundance: 3>`

Sort the isotope distribution by the relative abundances.

class IsotopeDistribution (formula)

Bases: `DataArray`

An isotope distribution.

Parameters `formula` (`Formula`) – A `Formula` object to create the distribution for

Each composition can be accessed with their hill formulae like a dictionary (e.g. `iso_dict['H[1]2O[16]']`)

Attributes:

<code>__class_getitem__</code>

Methods:

<code>__contains__(key)</code>	Return key in self.
<code>__eq__(other)</code>	Return self == other.
<code>__getitem__(key)</code>	Return self[key].
<code>__iter__()</code>	Iterates over the dictionary's keys.
<code>__len__()</code>	Returns the number of keys in the dictionary.
<code>__repr__()</code>	Return a string representation of the <code>DataArray</code> .
<code>__str__()</code>	Return <code>str(self)</code> .

continues on next page

Table 16 – continued from previous page

<code>as_array([sort_by, reverse, format_percentage])</code>	Returns the isotope distribution data as a list of lists.
<code>as_csv(*args[, sep])</code>	Returns the data as a CSV formatted string.
<code>as_dataframe(*args, **kwargs)</code>	Returns the isotope distribution data as a <code>pandas.DataFrame</code> .
<code>as_table(*args, **kwargs)</code>	Returns the isotope distribution data as a table using <code>tabulate</code> .
<code>copy(*args, **kwargs)</code>	Return a copy of the <code>FrozenOrderedDict</code> .
<code>fromkeys(iterable[, value])</code>	Create a new dictionary with keys from iterable and values set to value.
<code>get(k[, default])</code>	Return the value for k if k is in the dictionary, else default.
<code>items()</code>	Returns a set-like object providing a view on the <code>FrozenOrderedDict</code> 's items.
<code>keys()</code>	Returns a set-like object providing a view on the <code>FrozenOrderedDict</code> 's keys.
<code>values()</code>	Returns an object providing a view on the <code>FrozenOrderedDict</code> 's values.

`__class_getitem__` = <bound method `GenericAlias` of <class 'chemistry_tools.formulae.iso.
Type: `MethodType`

`__contains__(key)`
 Return key in self.

Parameters `key` (`object`)

Return type `bool`

`__eq__(other)`
 Return self == other.

Return type `bool`

`__getitem__(key)`
 Return self[key].

Parameters `key` (`~KT`)

Return type `~VT`

`__iter__()`
 Iterates over the dictionary's keys.

Return type `Iterator[~KT]`

`__len__()`
 Returns the number of keys in the dictionary.

Return type `int`

`__repr__()`
 Return a string representation of the `DataArray`.

Return type `str`

`__str__()`

Return `str(self)`.

Return type `str`

as_array (*sort_by*=<IsoDistSort.Formula: 0>, *reverse*=False, *format_percentage*=True)

Returns the isotope distribution data as a list of lists.

Parameters

- **sort_by** (`Union[int, IsoDistSort]`) – The column to sort by. Default <IsoDistSort.Formula: 0>.
- **reverse** (`bool`) – Whether the isotopologues should be sorted in reverse order. Default `False`.
- **format_percentage** (`bool`) – Whether the abundances should be formatted as percentages or not. Default `True`.

Return type `List[List[Any]]`

as_csv (**args*, *sep*=' ', ***kwargs*)

Returns the data as a CSV formatted string.

Parameters

- ***args** – Arguments passed to `as_array()`.
- **sep** (`str`) – The separator for the CSV data. Default ' '.
- ****kwargs** – Additional keyword arguments passed to `as_array()`.

Return type `str`

as_dataframe (**args*, ***kwargs*)

Returns the isotope distribution data as a `pandas.DataFrame`.

Any arguments taken by `as_array()` can also be used here.

Return type `DataFrame`

as_table (**args*, ***kwargs*)

Returns the isotope distribution data as a table using `tabulate`.

Any arguments taken by `as_array()` can also be used here.

Additionally, any valid keyword argument for `tabulate.tabulate()` can be used.

Return type `str`

copy (**args*, ***kwargs*)

Return a copy of the `FrozenOrderedDict`.

Parameters

- **args**
- **kwargs**

classmethod fromkeys (*iterable*, *value=None*)

Create a new dictionary with keys from iterable and values set to value.

Return type `FrozenBase[~KT, ~VT]`

get (*k*, *default=None*)

Return the value for *k* if *k* is in the dictionary, else *default*.

Parameters

- **k** – The key to return the value for.
- **default** – The value to return if key is not in the dictionary. Default `None`.

items ()

Returns a set-like object providing a view on the `FrozenOrderedDict`'s items.

Return type `AbstractSet[Tuple[~KT, ~VT]]`

keys ()

Returns a set-like object providing a view on the `FrozenOrderedDict`'s keys.

Return type `AbstractSet[~KT]`

values ()

Returns an object providing a view on the `FrozenOrderedDict`'s values.

Return type `ValuesView[~VT]`

3.7 chemistry_tools.formulae.latex

Functions and constants for converting formulae to LaTeX.

Functions:

<code>latex_subscript(val)</code>	Returns the LaTeX subscript of the given value.
<code>latex_superscript(val)</code>	Returns the LaTeX superscript of the given value.
<code>string_to_latex(formula[, prefixes, ...])</code>	Convert a formula string to its LaTeX representation.

latex_subscript (*val*)

Returns the LaTeX subscript of the given value.

Parameters **val** (`Union[str, float]`) – The value to superscript

Return type `str`

latex_superscript (*val*)

Returns the LaTeX superscript of the given value.

Parameters **val** (`Union[str, float]`) – The value to subscript

Return type `str`

string_to_latex (*formula*, *prefixes=None*, *infixes=None*, *suffixes=('', 'l', 'g', 'aq')*)

Convert a formula string to its LaTeX representation.

Examples:

```
>>> string_to_latex('NH4+')
'NH_{4}^{+}'
>>> string_to_latex('Fe(CN)6+2')
'Fe(CN)_{6}^{2+}'
>>> string_to_latex('Fe(CN)6+2(aq)')
'Fe(CN)_{6}^{2+}(aq)'
>>> string_to_latex('.NHO-(aq)')
'^{\bullet} NHO^{-}(aq)'
>>> string_to_latex('alpha-FeOOH(s)')
'\alpha-FeOOH(s)'
```

Parameters

- **formula** (`str`) – Chemical formula, e.g. 'H2O', 'Fe+3', 'Cl-'.
- **prefixes** (`Optional[Dict[str, str]]`) – Mapping of prefixes to their LaTeX equivalents. Default greek letters and ..
- **infixes** (`Optional[Dict[str, str]]`) – Mapping of infixes to their LaTeX equivalents. Default `None`.
- **suffixes** (`Sequence[str]`) – Suffixes to keep. Default `(' (s) ', ' (l) ', ' (g) ', ' (aq) ')`.

Return type `str`

Returns The LaTeX representation of the formula.

3.8 chemistry_tools.formulae.parser

Functions and parsing formulae.

Functions:

<code>mass_from_composition</code> (composition[, charge])	Calculates molecular mass, in atomic mass units, from atomic weights.
<code>string_to_composition</code> (formula[, prefixes, ...])	Parse composition of formula representing a chemical formula.

mass_from_composition (composition, charge=0)

Calculates molecular mass, in atomic mass units, from atomic weights.

Note: Atomic number 0 denotes charge or “net electron deficiency”

Example:

```
>>> f'{mass_from_composition({0: -1, "H": 1, 8: 1}):.2f}'
'17.01'
```

Parameters

- **composition** (Mapping[Union[str, int], int]) – Mapping of str or int (element symbol or atomic number) to int (coefficient)
- **charge** (int) – The charge of the composition. Default 0.

Return type float

string_to_composition (formula, prefixes=None, suffixes=('(s)', '(l)', '(g)', '(aq)'))

Parse composition of formula representing a chemical formula.

Examples:

```
>>> string_to_composition('NH4+') == {0: 1, "H": 4, "N": 1}
True
>>> string_to_composition('.NHO-(aq)') == {0: -1, "H": 1, "N": 1, "O": 1}
True
>>> string_to_composition('Na2CO3.7H2O') == {"Na": 2, "C": 1, "O": 10, "H": 14}
True
```

Parameters

- **formula** (str) – Chemical formula, e.g. 'H2O', 'Fe+3', 'Cl-'
- **prefixes** (Optional[Iterable[str]]) – Prefixes to ignore, e.g. ('.', 'alpha-'). Default None.
- **suffixes** (Sequence[str]) – Suffixes to ignore. Default ('(s)', '(l)', '(g)', '(aq)').

Return type Dict[int, int]

Returns The composition, as a dictionary mapping atomic number -> multiplicity. “Atomic number” 0 represents net charge.

3.9 chemistry_tools.formulae.species

Class to represent a formula with phase information (e.g. solid, liquid, gas, or aqueous).

Data:

<i>S</i>	Invariant <code>TypeVar</code> bound to <code>chemistry_tools.formulae.species.Species</code> .
----------	---

Classes:

<i>Species</i> ([composition, charge, phase])	Formula with phase information (e.g.
---	--------------------------------------

```
S = TypeVar(S, bound=Species)
Type: TypeVar
Invariant TypeVar bound to chemistry_tools.formulae.species.Species.
```

```
class Species (composition=None, charge=0, phase=None)
Bases: Formula
Formula with phase information (e.g. solid, liquid, gas, or aqueous).
Species extends Formula with the new attribute phase
```

Parameters

- **composition** (Optional[Dict[str, int]]) – A *Formula* object with the elemental composition of a substance, or a dict representing the same. If *None* an empty object is created. Default *None*.
- **charge** (int) – Default 0.
- **phase** (Optional[Literal['s', 'l', 'g', 'aq']]) – Either 's', 'l', 'g', or 'aq'. *None* represents an unknown phase. Default *None*.

Methods:

<code>__eq__(other)</code>	Returns <code>self == other</code> .
<code>copy()</code>	Returns a copy of the <i>Species</i> .
<code>from_kwargs(*[, charge, phase])</code>	Create a new <i>Species</i> object from keyword arguments representing the elements in the compound.
<code>from_string(formula[, charge, phase])</code>	Create a new <i>Species</i> object by parsing a string.

Attributes:

<code>empirical_formula</code>	Returns the empirical formula in Hill notation.
<code>hill_formula</code>	Returns the formula in Hill notation.
<code>phase</code>	The phase of the species (e.g.

`__eq__`(*other*)
Returns `self == other`.

Return type `bool`

`copy`()
Returns a copy of the *Species*.

Return type *~S*

`property empirical_formula`
Returns the empirical formula in Hill notation.

The empirical formula has the simplest whole number ratio of atoms of each element present in the formula.

Examples:

```
>>> Formula.from_string('H2O').empirical_formula
'H2O'
>>> Formula.from_string('S4').empirical_formula
'S'
>>> Formula.from_string('C6H12O6').empirical_formula
'CH2O'
```

Return type `str`

`classmethod from_kwargs`(***, *charge=0*, *phase=None*, ***kwargs*)
Create a new *Species* object from keyword arguments representing the elements in the compound.

Parameters

- **`charge`** (`int`) – The charge of the compound. Default 0.
- **`phase`** (`Optional[Literal['s', 'l', 'g', 'aq']]`) – The phase of the compound (e.g. 's' for solid). Default `None`.

Return type *~S*

`classmethod from_string`(*formula*, *charge=0*, *phase=None*)
Create a new *Species* object by parsing a string.

Note: Isotopes cannot (currently) be parsed using this method

Parameters

- **`formula`** (`str`) – A string with a chemical formula
- **`phase`** (`Optional[Literal['s', 'l', 'g', 'aq']]`) – Either 's', 'l', 'g', or 'aq'. `None` represents an unknown phase. Default `None`.
- **`charge`** (`int`) – Default 0.

Return type *~S*

Examples:

```
>>> water = Species.from_string('H2O')
>>> water.phase
None
>>> NaCl = Species.from_string('NaCl(s)')
>>> NaCl.phase
s
>>> Hg_l = Species.from_string('Hg(l)')
>>> Hg_l.phase
l
>>> CO2g = Species.from_string('CO2(g)')
>>> CO2g.phase
g
>>> CO2aq = Species.from_string('CO2(aq)')
>>> CO2aq.phase
aq
```

property hill_formula

Returns the formula in Hill notation.

Examples:

```
>>> Species.from_string('BrC2H5').hill_formula
'C2H5Br'
>>> Species.from_string('HBr').hill_formula
'BrH'
>>> Species.from_string('[ (CH3)3Si2]2NNa').hill_formula
'C6H18NNaSi4'
```

Return type `str`

phase

Type: `Optional[Literal['s', 'l', 'g', 'aq']]`

The phase of the species (e.g. solid, liquid, gas, or aqueous). `None` represents an unknown phase.

3.10 chemistry_tools.formulae.unicode

Functions and constants for converting formulae to unicode.

Functions:

<code>string_to_unicode(formula[, prefixes, ...])</code>	Convert the given formula string to a unicode string representation.
<code>unicode_subscript(val)</code>	Returns the Unicode subscript of the given value.
<code>unicode_superscript(val)</code>	Returns the Unicode superscript of the given value.

string_to_unicode (*formula*, *prefixes=None*, *infixes=None*, *suffixes=('('s)', '(l)', '(g)', '(aq)')*)

Convert the given formula string to a unicode string representation.

Examples:

```
>>> string_to_unicode('NH4+')
'NH4+'
>>> string_to_unicode('Fe(CN)6+2')
'Fe(CN)62+'
>>> string_to_unicode('Fe(CN)6+2(aq)')
'Fe(CN)62+(aq)'
>>> string_to_unicode('.NHO-(aq)')
'.NHO-(aq)'
>>> string_to_unicode('alpha-FeOOH(s)')
'α-FeOOH(s)'
```

Parameters

- **formula** (*str*) – Chemical formula, e.g. 'H2O', 'Fe+3', 'Cl-'
- **prefixes** (*Optional[Dict[str, str]]*) – Mapping of prefixes to their Unicode equivalents. Default greek letters and .
- **infixes** (*Optional[Dict[str, str]]*) – Mapping of infixes to their Unicode equivalents. Default *None*.
- **suffixes** (*Sequence[str]*) – Suffixes to keep. Default ('(s)', '(l)', '(g)', '(aq)').

Return type *str*

Returns The Unicode representation of the formula.

unicode_subscript (*val*)

Returns the Unicode subscript of the given value.

Parameters *val* (*Union[str, float]*) – The value to superscript

Return type *str*

unicode_superscript (*val*)

Returns the Unicode superscript of the given value.

Parameters *val* (*Union[str, float]*) – The value to subscript

Return type *str*

3.11 chemistry_tools.formulae.utils

General utility functions.

Data:

<i>GROUPS</i>	Common chemical groups
---------------	------------------------

Functions:

<i>hill_order</i> (symbols)	Returns an iterator over the given element symbols in order of Hill notation.
<i>split_isotope</i> (string)	Returns the symbol and mass number for the isotope represented by <i>string</i> .

```
GROUPS = {'Abu': 'C4H7NO', 'Acet': 'C2H3O', 'Acm': 'C3H6NO', 'Adao': 'C10H15O', 'Aib':
Type: Dict[str, str]
```

Common chemical groups

hill_order (*symbols*)

Returns an iterator over the given element symbols in order of Hill notation.

Example:

```
>>> for i in hill_order("H", "C[12]", "O"): print(i, end=' ')
CHO
```

Return type `Iterator[str]`

split_isotope (*string*)

Returns the symbol and mass number for the isotope represented by *string*.

Valid isotopes include ' [C12] ', 'C [12] ' and ' [12C] '.

Parameters *string* (`str`)

Return type `Tuple[str, int]`

Returns Tuple representing the element and the isotope number.

chemistry_tools.pubchem

This module provides a wrapper around the PubChem PUG_REST API.

Data for compounds can be accessed using the `pubchem.lookup.get_compounds` function.

The following table lists the various properties that can be obtained from the PubChem API:

Property	Description
MolecularFormula	Molecular formula.
MolecularWeight	The molecular weight is the sum of all atomic weights of the constituent atoms in a compound, measured in g/mol. In the absence of explicit isotope labelling, averaged natural abundance is assumed. If an atom bears an explicit isotope label, 100% isotopic purity is assumed at this location.
CanonicalSMILES	Canonical SMILES (Simplified Molecular Input Line Entry System) string. It is a unique SMILES string of a compound, generated by a “canonicalization” algorithm.
IsomericSMILES	Isomeric SMILES string. It is a SMILES string with stereochemical and isotopic specifications.
InChI	Standard IUPAC International Chemical Identifier (InChI). It does not allow for user selectable options in dealing with the stereochemistry and tautomer layers of the InChI string.
InChIKey	Hashed version of the full standard InChI, consisting of 27 characters.
IUPACName	Chemical name systematically determined according to the IUPAC nomenclatures.
XLogP	Computationally generated octanol-water partition coefficient or distribution coefficient. XLogP is used as a measure of hydrophilicity or hydrophobicity of a molecule.
ExactMass	The mass of the most likely isotopic composition for a single molecule, corresponding to the most intense ion/molecule peak in a mass spectrum.
MonoisotopicMass	The mass of a molecule, calculated using the mass of the most abundant isotope of each element.
TPSA	Topological polar surface area, computed by the algorithm described in the paper by Ertl et al.
Complexity	The molecular complexity rating of a compound, computed using the Bertz/Hendrickson/Ihlenfeldt formula.
Charge	The total (or net) charge of a molecule.
HBondDonorCount	Number of hydrogen-bond donors in the structure.
HBondAcceptorCount	Number of hydrogen-bond acceptors in the structure.
RotatableBondCount	Number of rotatable bonds.
HeavyAtomCount	Number of non-hydrogen atoms.
IsotopeAtomCount	Number of atoms with enriched isotope(s)

continues on next page

Table 1 – continued from previous page

Property	Description
AtomStereoCount	Total number of atoms with tetrahedral (sp ³) stereo [e.g., (R)- or (S)-configuration]
DefinedAtomStereoCount	Number of atoms with defined tetrahedral (sp ³) stereo.
UndefinedAtomStereoCount	Number of atoms with undefined tetrahedral (sp ³) stereo.
BondStereoCount	Total number of bonds with planar (sp ²) stereo [e.g., (E)- or (Z)-configuration].
DefinedBondStereoCount	Number of atoms with defined planar (sp ²) stereo.
UndefinedBondStereoCount	Number of atoms with undefined planar (sp ²) stereo.
CovalentUnitCount	Number of covalently bound units.
Volume3D	Analytic volume of the first diverse conformer (default conformer) for a compound.
XStericQuadrupole3D	The x component of the quadrupole moment (Q _x) of the first diverse conformer (default conformer) for a compound.
YStericQuadrupole3D	The y component of the quadrupole moment (Q _y) of the first diverse conformer (default conformer) for a compound.
ZStericQuadrupole3D	The z component of the quadrupole moment (Q _z) of the first diverse conformer (default conformer) for a compound.
FeatureCount3D	Total number of 3D features (the sum of FeatureAcceptorCount3D, FeatureDonorCount3D, FeatureAnionCount3D, FeatureCationCount3D, FeatureRingCount3D and FeatureHydrophobeCount3D)
FeatureAcceptorCount3D	Number of hydrogen-bond acceptors of a conformer.
FeatureDonorCount3D	Number of hydrogen-bond donors of a conformer.
FeatureAnionCount3D	Number of anionic centers (at pH 7) of a conformer.
FeatureCationCount3D	Number of cationic centers (at pH 7) of a conformer.
FeatureRingCount3D	Number of rings of a conformer.
FeatureHydrophobeCount3D	Number of hydrophobes of a conformer.
ConformerModelRMSD3D	Conformer sampling RMSD in Å.
EffectiveRotorCount3D	Total number of 3D features (the sum of FeatureAcceptorCount3D, FeatureDonorCount3D, FeatureAnionCount3D, FeatureCationCount3D, FeatureRingCount3D and FeatureHydrophobeCount3D)
ConformerCount3D	The number of conformers in the conformer model for a compound.
Fingerprint2D	Base64-encoded PubChem Substructure Fingerprint of a molecule.

Attention: This package has the following additional requirements:

```
cawdrey>=0.1.7
mathematical>=0.1.13
pillow>=7.0.0
pyparsing>=2.4.6
tabulate>=0.8.9
```

These can be installed as follows:

```
$ python -m pip install chemistry-tools[pubchem]
```


4.1 chemistry_tools.pubchem.atom

Represents an atom in a *Compound*.

Classes:

<code>Atom(aid, number[, x, y, z, charge])</code>	Class to represent an atom in a <i>Compound</i> .
---	---

Functions:

<code>parse_atoms(atoms_dict[, coords_dict])</code>	Parse atoms from the given dictionary.
---	--

class `Atom` (*aid*, *number*, *x=None*, *y=None*, *z=None*, *charge=0*)

Bases: `object`

Class to represent an atom in a *Compound*.

Parameters

- **aid** (`int`) – The Atom ID within the owning Compound.
- **number** (`int`) – The Atomic number for this atom.
- **x** (`Optional[float]`) – The x coordinate for this atom. Default `None`.
- **y** (`Optional[float]`) – The y coordinate for this atom. Default `None`.
- **z** (`Optional[float]`) – The z coordinate for this atom. Will be `None` in 2D Compound records. Default `None`.
- **charge** (`int`) – Formal charge on atom. Default 0.

Methods:

<code>__eq__(other)</code>	Return <code>self == other</code> .
<code>__repr__()</code>	Return a string representation of the <i>Atom</i> .
<code>set_coordinates(x, y[, z])</code>	Set all coordinate dimensions at once.
<code>to_dict()</code>	Return a dictionary containing Atom data.

Attributes:

<code>coordinate_type</code>	Returns whether this atom has 2D or 3D coordinates.
<code>element</code>	The element symbol for this atom.

`__eq__(other)`
Return `self == other`.

Return type `bool`

`__repr__()`
Return a string representation of the *Atom*.

Return type `str`

property coordinate_type

Returns whether this atom has 2D or 3D coordinates.

Return type `str`

property element

The element symbol for this atom.

Return type `str`

set_coordinates (*x*, *y*, *z=None*)

Set all coordinate dimensions at once.

to_dict ()

Return a dictionary containing Atom data.

Return type `Dict[str, Any]`

parse_atoms (*atoms_dict*, *coords_dict=None*)

Parse atoms from the given dictionary.

Parameters

- **atoms_dict** (`Dict[str, Any]`)
- **coords_dict** (`Optional[Dict]`) – Default `None`.

Return type `Dict[FrozenSet[int], Atom]`

4.2 chemistry_tools.pubchem.bond

Represents a bond between atoms in a *Compound*.

Classes:

<code>Bond(aid1, aid2[, order, style])</code>	Class to represent a bond between two atoms in a <i>Compound</i> .
<code>BondType(value)</code>	Enumeration of possible bond types.

Functions:

<code>parse_bonds(bonds_dict[, coords_dict])</code>	Parse bonds from the given dictionary.
---	--

class Bond (*aid1, aid2, order=<BondType.SINGLE: 1>, style=None*)

Bases: `object`

Class to represent a bond between two atoms in a *Compound*.

Parameters

- **aid1** (`int`) – ID of the begin atom of this bond
- **aid2** (`int`) – ID of the end atom of this bond
- **order** (`Union[int, BondType]`) – Bond order. Default `<BondType.SINGLE: 1>`.
- **style** – Bond style annotation. Default `None`.

Methods:

<code>__eq__(other)</code>	Return <code>self == other</code> .
<code>__repr__()</code>	Return a string representation of the <i>Bond</i> .
<code>to_dict()</code>	Return a dictionary containing bond data.

`__eq__(other)`

Return `self == other`.

Return type `bool`

`__repr__()`

Return a string representation of the *Bond*.

Return type `str`

`to_dict()`

Return a dictionary containing bond data.

Return type `Dict[str, Any]`

enum `BondType` (*value*)

Bases: `enum_tools.custom_enums.IntEnum`

Enumeration of possible bond types.

Member Type `int`

Valid values are as follows:

`SINGLE = <BondType.SINGLE: 1>`

`DOUBLE = <BondType.DOUBLE: 2>`

`TRIPLE = <BondType.TRIPLE: 3>`

`QUADRUPLE = <BondType.QUADRUPLE: 4>`

`DATIVE = <BondType.DATIVE: 5>`

`COMPLEX = <BondType.COMPLEX: 6>`

`IONIC = <BondType.IONIC: 7>`

`UNKNOWN = <BondType.UNKNOWN: 255>`

parse_bonds (*bonds_dict*, *coords_dict=None*)

Parse bonds from the given dictionary.

Parameters

- **bonds_dict** (`Dict[str, Any]`)
- **coords_dict** (`Optional[Dict]`) – Default `None`.

Return type `Dict[FrozenSet[int], Bond]`

4.3 chemistry_tools.pubchem.compound

Represents a chemical compound.

Data:

<code>C</code>	Invariant <code>TypeVar</code> bound to <code>chemistry_tools.pubchem.compound.Compound</code> .
----------------	--

Classes:

<code>Compound</code> (title, CID, description, **_)	Represents a single record from the PubChem Compound database.
--	--

Functions:

<code>compounds_to_frame</code> (compounds)	Construct a <code>DataFrame</code> from a list of <code>Compound</code> objects.
---	--

C = TypeVar (C, bound=Compound)

Type: `TypeVar`

Invariant `TypeVar` bound to `chemistry_tools.pubchem.compound.Compound`.

class Compound (title, CID, description, **_)

Bases: `Dictable`

Represents a single record from the PubChem Compound database.

The PubChem Compound database is constructed from the Substance database using a standardization and deduplication process. Each Compound is uniquely identified by a CID.

Parameters

- **title** (`str`) – The title of the compound record (usually the name of the compound)
- **CID** (`int`)
- **description** (`str`)

Methods:

<code>__repr__</code> ()	Return a string representation of the <code>Compound</code> .
<code>from_cid</code> (cid[, record_type])	Returns the Compound objects for the compound with the given CID.
<code>get_iupac_name</code> ([type_])	Return the IUPAC name of this compound.
<code>get_properties</code> (properties)	Returns the requested properties for the Compound.
<code>get_property</code> (prop)	Get a single property for the compound.
<code>precache</code> ()	Precache all properties for this compound.
<code>to_series</code> ()	Return a pandas <code>Series</code> containing Compound data.

Attributes:

<code>atoms</code>	List of <code>Atoms</code> in this Compound.
<code>bonds</code>	List of <code>Bonds</code> between <code>Atoms</code> in this Compound.

continues on next page

Table 13 – continued from previous page

<i>cactvs_fingerprint</i>	PubChem CACTVS fingerprint.
<i>canonical_smiles</i>	Canonical SMILES, with no stereochemistry information.
<i>canonicalized</i>	Whether the compound is canonicalized.
<i>charge</i>	The charge of the compound.
<i>cid</i>	Returns the ID of this compound.
<i>coordinate_type</i>	The coordinate type of this compound.
<i>elements</i>	List of element symbols for atoms in this Compound.
<i>fingerprint</i>	Raw padded and hex-encoded fingerprint, as returned by the PUG REST API.
<i>has_full_record</i>	Returns whether this compound has a full record available.
<i>iupac_name</i>	The preferred IUPAC name of this compound.
<i>molecular_formula</i>	Molecular formula.
<i>molecular_mass</i>	Molecular Weight.
<i>molecular_weight</i>	Molecular Weight.
<i>smiles</i>	Canonical SMILES, with no stereochemistry information.
<i>synonyms</i>	Returns a list of synonyms for the Compound.
<i>systematic_name</i>	The systematic IUPAC name of this compound.

__repr__()

Return a string representation of the *Compound*.

Return type `str`

property atoms

List of *Atoms* in this Compound.

Return type `List[Atom]`

property bonds

List of *Bonds* between *Atoms* in this Compound.

Return type `List[Bond]`

property cactvs_fingerprint

PubChem CACTVS fingerprint.

Each bit in the fingerprint represents the presence or absence of one of 881 chemical substructures.

See also: `ftp://ftp.ncbi.nlm.nih.gov/pubchem/specifications/pubchem_fingerprints.txt`

Return type `Optional[str]`

property canonical_smiles

Canonical SMILES, with no stereochemistry information.

Return type `str`

property canonicalized

Whether the compound is canonicalized.

Return type `bool`

property charge

The charge of the compound.

Return type `int`

property cid

Returns the ID of this compound.

Return type `int`

property coordinate_type

The coordinate type of this compound.

Return type `Optional[str]`

property elements

List of element symbols for atoms in this Compound.

Return type `List[str]`

property fingerprint

Raw padded and hex-encoded fingerprint, as returned by the PUG REST API.

Return type `Optional[str]`

classmethod from_cid (*cid*, *record_type*='2d')

Returns the Compound objects for the compound with the given CID.

Return type `Compound`

get_iupac_name (*type_*='Systematic')

Return the IUPAC name of this compound.

Parameters *type_* (`str`) – The type of IUPAC name. Default 'Systematic'.

Return type `Optional[str]`

get_properties (*properties*)

Returns the requested properties for the Compound.

Parameters *properties* (`Union[Sequence[str], str]`) – The properties to retrieve for the compound. Can be either a comma-separated string or a list. See [the table at the start of this chapter](#) for a list of valid properties.

Return type `Dict[str, Any]`

Returns Dictionary mapping the property names to their values

get_property (*prop*)

Get a single property for the compound.

Parameters *prop* (`str`) – The property to retrieve for the compound. See [the table at the start of this chapter](#) for a list of valid properties.

Return type `Any`

property has_full_record

Returns whether this compound has a full record available.

Return type `bool`

property iupac_name

The preferred IUPAC name of this compound.

Return type `Optional[str]`

property molecular_formula

Molecular formula.

Return type `Formula`

property molecular_mass

Molecular Weight.

Return type `float`

property molecular_weight

Molecular Weight.

Return type `float`

precache()

Precache all properties for this compound.

property smiles

Canonical SMILES, with no stereochemistry information.

Return type `str`

property synonyms

Returns a list of synonyms for the Compound.

Return type `Optional[List[str]]`

property systematic_name

The systematic IUPAC name of this compound.

Return type `Optional[str]`

to_series()

Return a pandas `Series` containing Compound data.

Return type `Series`

compounds_to_frame(compounds)

Construct a `DataFrame` from a list of `Compound` objects.

Parameters `compounds` (`Union[Compound, List[Compound]]`)

Return type `DataFrame`

4.4 chemistry_tools.pubchem.description

Functions to access the name and description of compounds in the PubChem database.

Functions:

<code>get_common_name(name)</code>	Returns the common name for the compound with the given name.
<code>get_compound_id(name)</code>	Returns the compound ID (CID) for the compound with the given name.
<code>get_description(name)</code>	Returns the description compound with the given name.
<code>get_iupac_name(name)</code>	Returns the systematic IUPAC name for the compound with the given name.
<code>parse_description(description_data)</code>	Parse raw data from the description endpoint of the REST API.
<code>rest_get_description(identifier[, namespace])</code>	Obtains the description for the given compound from the PubChem REST API.

`get_common_name(name)`

Returns the common name for the compound with the given name.

Parameters `name` (`str`)

Return type `str`

`get_compound_id(name)`

Returns the compound ID (CID) for the compound with the given name.

Parameters `name` (`str`)

Return type `str`

`get_description(name)`

Returns the description compound with the given name.

Parameters `name` (`str`)

Return type `str`

`get_iupac_name(name)`

Returns the systematic IUPAC name for the compound with the given name.

Parameters `name` (`str`)

Return type `str`

`parse_description(description_data)`

Parse raw data from the description endpoint of the REST API.

Parameters `description_data` (`Dict[str, Any]`)

Return type `List[Dict]`

Returns A list of dictionaries containing the CID, Title and Description for each compound

rest_get_description (*identifier*, *namespace*=<PubChemNamespace.Name: 'name'>, ***kwargs*)

Obtains the description for the given compound from the PubChem REST API.

Parameters

- **identifier** (`Union[str, int, Sequence[Union[str, int]]]`) – Identifiers (e.g. name, CID) for the compound to look up. When using the CID namespace data for multiple compounds can be retrieved at once by supplying either a comma-separated string or a list.
- **namespace** (`Union[PubChemNamespace, str]`) – The type of identifier to look up. Valid values are in `PubChemNamespace`. Default <PubChemNamespace.Name: 'name'>.
- **kwargs** – Optional arguments that `json.loads` takes.

Raises `ValueError` – If the response body does not contain valid JSON.

Return type `Dict[str, Any]`

Returns Parsed JSON data

4.5 chemistry_tools.pubchem.enums

Enumerations.

Classes:

<i>CoordinateType</i> (value)	Enumeration of valid values for the coordinate type.
<i>PubChemFormats</i> (value)	Enumeration of supported formats for the PubChem REST API.
<i>PubChemNamespace</i> (value)	Enumeration of possible values for the PubChem namespace.

enum `CoordinateType` (value)

Bases: `enum_tools.custom_enums.IntEnum`

Enumeration of valid values for the coordinate type.

Member Type `int`

Valid values are as follows:

`TWO_D = <CoordinateType.TWO_D: 1>`

`THREE_D = <CoordinateType.THREE_D: 2>`

`SUBMITTED = <CoordinateType.SUBMITTED: 3>`

`EXPERIMENTAL = <CoordinateType.EXPERIMENTAL: 4>`

`COMPUTED = <CoordinateType.COMPUTED: 5>`

`STANDARDIZED = <CoordinateType.STANDARDIZED: 6>`

`AUGMENTED = <CoordinateType.AUGMENTED: 7>`

`ALIGNED = <CoordinateType.ALIGNED: 8>`

`COMPACT = <CoordinateType.COMPACT: 9>`

`UNITS_ANGSTROMS = <CoordinateType.UNITS_ANGSTROMS: 10>`

`UNITS_NANOMETERS = <CoordinateType.UNITS_NANOMETERS: 11>`

`UNITS_PIXEL = <CoordinateType.UNITS_PIXEL: 12>`

`UNITS_POINTS = <CoordinateType.UNITS_POINTS: 13>`

`UNITS_STDBONDS = <CoordinateType.UNITS_STDBONDS: 14>`

`UNITS_UNKNOWN = <CoordinateType.UNITS_UNKNOWN: 255>`

The `Enum` and its members also have the following methods:

classmethod `is_valid_value` (*value*)

Returns whether the value is a valid member of this `enum.Enum`.

Parameters `value` (*Any*)

Return type `bool`

enum `PubChemFormats` (*value*)

Bases: `enum_tools.custom_enums.StrEnum`

Enumeration of supported formats for the PubChem REST API.

Member Type `str`

Valid values are as follows:

JSON = `<PubChemFormats.JSON: 'JSON'>`
JSON Format

XML = `<PubChemFormats.XML: 'XML'>`
XML Format

CSV = `<PubChemFormats.CSV: 'CSV'>`
CSV Format

PNG = `<PubChemFormats.PNG: 'PNG'>`
PNG Format

The `Enum` and its members also have the following methods:

classmethod `is_valid_value` (*value*)

Returns whether the value is a valid member of this `enum.Enum`.

Parameters `value` (*Any*)

Return type `bool`

enum `PubChemNamespace` (*value*)

Bases: `enum_tools.custom_enums.StrEnum`

Enumeration of possible values for the PubChem namespace.

Member Type `str`

Valid values are as follows:

CID = `<PubChemNamespace.CID: 'cid'>`
PubChem Compound ID

Name = `<PubChemNamespace.Name: 'name'>`
Compound Name

SMILES = `<PubChemNamespace.SMILES: 'smiles'>`
SMILES String

INCHIKEY = <PubChemNamespace.INCHIKEY: 'inchikey'>

InChI Key

The `Enum` and its members also have the following methods:

classmethod `is_valid_value` (*value*)

Returns whether the value is a valid member of this `enum.Enum`.

Parameters `value` (*Any*)

Return type `bool`

4.6 chemistry_tools.pubchem.errors

Error handling.

Exceptions:

<code>BadRequestError([msg])</code>	Request is improperly formed (syntax error in the URL, POST body, etc.).
<code>HTTPTimeoutError([msg])</code>	The request timed out, from server overload or too broad a request.
<code>MethodNotAllowedError([msg])</code>	Request not allowed (such as invalid MIME type in the HTTP Accept header).
<code>NotFoundError([msg])</code>	The input record was not found (e.g.
<code>PubChemHTTPError(e)</code>	Generic error class to handle all HTTP error codes.
<code>ResponseParseError</code>	PubChem response is uninterpretable.
<code>ServerError([msg])</code>	Some problem on the server side (such as a database server down, etc.).
<code>TimeoutError</code>	alias of <code>chemistry_tools.pubchem.errors.HTTPTimeoutError</code>
<code>UnimplementedError([msg])</code>	The requested operation has not (yet) been implemented by the server.

Data:

<code>HTTP_ERROR_CODES</code>	Numerical list of HTTP status codes considered to be errors.
-------------------------------	--

exception `BadRequestError` (*msg='Request is improperly formed'*)

Bases: `chemistry_tools.pubchem.errors.PubChemHTTPError`

Request is improperly formed (syntax error in the URL, POST body, etc.).

exception `HTTPTimeoutError` (*msg='The request timed out'*)

Bases: `chemistry_tools.pubchem.errors.PubChemHTTPError`

The request timed out, from server overload or too broad a request.

Changed in version 0.4.0: Renamed from `TimeoutErrpr`

HTTP_ERROR_CODES = [400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622, 623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 671, 672, 673, 674, 675, 676, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 698, 699, 700, 701, 702, 703, 704, 705, 706, 707, 708, 709, 710, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, 730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 740, 741, 742, 743, 744, 745, 746, 747, 748, 749, 750, 751, 752, 753, 754, 755, 756, 757, 758, 759, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785, 786, 787, 788, 789, 790, 791, 792, 793, 794, 795, 796, 797, 798, 799, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816, 817, 818, 819, 820, 821, 822, 823, 824, 825, 826, 827, 828, 829, 830, 831, 832, 833, 834, 835, 836, 837, 838, 839, 840, 841, 842, 843, 844, 845, 846, 847, 848, 849, 850, 851, 852, 853, 854, 855, 856, 857, 858, 859, 860, 861, 862, 863, 864, 865, 866, 867, 868, 869, 870, 871, 872, 873, 874, 875, 876, 877, 878, 879, 880, 881, 882, 883, 884, 885, 886, 887, 888, 889, 890, 891, 892, 893, 894, 895, 896, 897, 898, 899, 900, 901, 902, 903, 904, 905, 906, 907, 908, 909, 910, 911, 912, 913, 914, 915, 916, 917, 918, 919, 920, 921, 922, 923, 924, 925, 926, 927, 928, 929, 930, 931, 932, 933, 934, 935, 936, 937, 938, 939, 940, 941, 942, 943, 944, 945, 946, 947, 948, 949, 950, 951, 952, 953, 954, 955, 956, 957, 958, 959, 960, 961, 962, 963, 964, 965, 966, 967, 968, 969, 970, 971, 972, 973, 974, 975, 976, 977, 978, 979, 980, 981, 982, 983, 984, 985, 986, 987, 988, 989, 990, 991, 992, 993, 994, 995, 996, 997, 998, 999, 1000]

Type: `list`

Numerical list of HTTP status codes considered to be errors.

exception MethodNotAllowedError (*msg='Request not allowed'*)

Bases: `chemistry_tools.pubchem.errors.PubChemHTTPError`

Request not allowed (such as invalid MIME type in the HTTP Accept header).

exception NotFoundError (*msg='The input record was not found'*)

Bases: `chemistry_tools.pubchem.errors.PubChemHTTPError`

The input record was not found (e.g. invalid CID).

exception PubChemHTTPError (*e*)

Bases: `Exception`

Generic error class to handle all HTTP error codes.

`__str__()`

Return `str(self)`.

Return type `str`

exception ResponseParseError

Bases: `Exception`

PubChem response is uninterpretable.

exception ServerError (*msg='Some problem on the server side'*)

Bases: `chemistry_tools.pubchem.errors.PubChemHTTPError`

Some problem on the server side (such as a database server down, etc.).

TimeoutError

alias of `chemistry_tools.pubchem.errors.HTTPTimeoutError`

exception UnimplementedError (*msg='The requested operation has not been implemented'*)

Bases: `chemistry_tools.pubchem.errors.PubChemHTTPError`

The requested operation has not (yet) been implemented by the server.

4.7 chemistry_tools.pubchem.full_record

Functions for access the complete set of data held by PubChem for a compound.

Functions:

<code>parse_full_record(record)</code>	Parse the complete PubChem record for a compound.
<code>rest_get_full_record(identifier[, ...])</code>	Obtains the full record for the given compound from the PubChem REST API.

parse_full_record(*record*)

Parse the complete PubChem record for a compound.

Parameters *record* (*Dict*)

Return type *List*[*Dict*]

rest_get_full_record(*identifier*, *namespace*=<PubChemNamespace.Name: 'name'>, *record_type*='2d', ***kwargs*)

Obtains the full record for the given compound from the PubChem REST API.

Parameters

- **identifier** (*Union*[*str*, *int*, *Sequence*[*Union*[*str*, *int*]]]) – Identifiers (e.g. name, CID) for the compound to look up. When using the CID namespace data for multiple compounds can be retrieved at once by supplying either a comma-separated string or a list.
- **namespace** (*Union*[*PubChemNamespace*, *str*]) – The type of identifier to look up. Valid values are in *PubChemNamespace*. Default <PubChemNamespace.Name: 'name'>.
- **record_type** (*str*) – Default '2d'.
- **kwargs** – Optional arguments that *json.loads* takes.

Raises *ValueError* – If the response body does not contain valid JSON.

Return type *Dict*

Returns Parsed JSON data

4.8 chemistry_tools.pubchem.images

Functions for handling images.

Functions:

<i>get_structure_image</i> (<i>identifier</i> [, <i>namespace</i> , ...])	Returns an image of the structure of the compound with the given name.
--	--

get_structure_image(*identifier*, *namespace*=<PubChemNamespace.Name: 'name'>, *width*=300, *height*=300)

Returns an image of the structure of the compound with the given name.

Parameters

- **identifier** (*Union*[*str*, *int*, *Sequence*[*Union*[*str*, *int*]]]) – Identifiers (e.g. name, CID) for the compound to look up. When using the CID namespace data for multiple compounds can be retrieved at once by supplying either a comma-separated string or a list.
- **namespace** (*Union*[*PubChemNamespace*, *str*]) – The type of identifier to look up. Valid values are in *PubChemNamespace*. Default <PubChemNamespace.Name: 'name'>.
- **width** (*int*) – The image width in pixels. Default 300.
- **height** (*int*) – The image height in pixels. Default 300.

Return type *Image*

Returns Pillow Image data

4.9 chemistry_tools.pubchem.lookup

Lookup properties for compound by name or CAS number.

Functions:

<code>get_compounds(identifier[, namespace])</code>	Returns a list of Compound objects for compounds that match the search criteria.
---	--

get_compounds (*identifier*, *namespace*=<PubChemNamespace.Name: 'name'>)

Returns a list of Compound objects for compounds that match the search criteria.

As more than one compound may be identified the results are returned in a list.

Parameters

- **identifier** (Union[str, int, Sequence[Union[str, int]]]) – Identifiers (e.g. name, CID) for the compound to look up. When using the CID namespace data for multiple compounds can be retrieved at once by supplying either a comma-separated string or a list.
- **namespace** (Union[PubChemNamespace, str]) – The type of identifier to look up. Valid values are in *PubChemNamespace*. Default <PubChemNamespace.Name: 'name'>.

Return type List[Compound]

4.10 chemistry_tools.pubchem.properties

Functions and classes to access properties of compounds in the PubChem database.

Data:

<code>PROPERTY_MAP</code>	Allows properties to optionally be specified as underscore_separated, consistent with Compound attributes
<code>valid_properties</code>	Properties for PubChem REST API

Classes:

<code>PropData(name, description, type, attr_name)</code>	Metadata about a property.
<code>PubChemProperty(label[, name, value, dtype, ...])</code>	Represents a property parsed from the full PubChem record.

Functions:

<code>force_valid_properties(properties)</code>	Coerce <i>properties</i> into a list of strings and exclude any invalid properties, or raise a <i>ValueError</i> if that is not possible.
<code>get_properties(identifier[, properties, ...])</code>	Returns the requested properties for the compound with the given identifier.
<code>get_property(identifier[, property, namespace])</code>	Returns the requested property for the compound with the given identifier.

continues on next page

Table 23 – continued from previous page

<code>parse_properties(property_data)</code>	Parse raw data from the <code>property</code> endpoint of the REST API.
<code>rest_get_properties(identifier[, namespace, ...])</code>	Returns the properties for the compound with the given identifier in the desired format.
<code>rest_get_properties_json(identifier[, ...])</code>	Returns the properties for the compound with the given identifier as a dictionary.

```
PROPERTY_MAP = {'atom_stereo_count': 'AtomStereoCount', 'bond_stereo_count': 'BondStereoCount'}
Type: Dict[str, str]
```

Allows properties to optionally be specified as underscore_separated, consistent with Compound attributes

namedtuple PropData (*name, description, type, attr_name*)

Bases: `NamedTuple`

Metadata about a property.

Fields

- 0) **name** (`str`) – The name of the property.
- 1) **description** (`str`) – The description of the property.
- 2) **type** (`Callable`) – The type of the property.
- 3) **attr_name** (`str`) – The Python attribute name of the property in a `chemistry_tools.pubchem.compound.Compound`.

`__repr__` ()

Return a nicely formatted representation string

namedtuple PubChemProperty (*label, name=None, value=None, dtype=None, source=None*)

Bases: `NamedTuple`

Represents a property parsed from the full PubChem record.

Fields

- 0) **label** (`str`) – The label of the property.
- 1) **name** (`Optional[str]`) – The name of the property.
- 2) **value** (`Any`) – The property's value.
- 3) **dtype** (`Callable`) – The data type property's value.
- 4) **source** (`Dict`) – Dictionary of property sources.

static `__new__` (*cls, label, name=None, value=None, dtype=None, source=None*)

Create new instance of `__BasePubChemProperty`(label, name, value, dtype, source)

force_valid_properties (*properties*)

Coerce properties into a list of strings and exclude any invalid properties, or raise a `ValueError` if that is not possible.

Parameters `properties` (`Union[str, Iterable[str]]`)

Return type `List[str]`

get_properties (*identifier*, *properties*="", *namespace*=<PubChemNamespace.Name: 'name'>, *as_dataframe*=False)

Returns the requested properties for the compound with the given identifier. As more than one compound may be identified the results are returned in a list.

Parameters

- **identifier** (Union[str, int, Sequence[Union[str, int]]]) – Identifiers (e.g. name, CID) for the compound to look up. When using the CID namespace data for multiple compounds can be retrieved at once by supplying either a comma-separated string or a list.
- **properties** (Union[Sequence[str], str]) – The properties to retrieve for the compound. Can be either a comma-separated string or a list. See [the table at the start of this chapter](#) for a list of valid properties. Default ''.
- **namespace** (Union[PubChemNamespace, str]) – The type of identifier to look up. Valid values are in [PubChemNamespace](#). Default <PubChemNamespace.Name: 'name'>.
- **as_dataframe** (bool) – Automatically extract the properties into a pandas `DataFrame`. Default `False`.

Raises

- **ValueError** – If the response body does not contain valid JSON.
- **NotFoundError** – If the compound with the requested identifier was not found in PubChem.

Return type Union[List[Dict[str, Any]], DataFrame]

Returns List of dictionaries mapping properties to values

get_property (*identifier*, *property*="", *namespace*=<PubChemNamespace.Name: 'name'>)

Returns the requested property for the compound with the given identifier.

This convenience function only allows for a single property to be accessed at once, and for only a single compound. If you require multiple properties and/or properties for multiple compounds use [chemistry_tools.pubchem.properties.get_properties](#), which helps reduce the burden on the PubChem servers.

Parameters

- **identifier** (Union[str, int, Sequence[Union[str, int]]]) – Identifiers (e.g. name, CID) for the compound to look up.
- **properties** – The properties to retrieve for the compound. Can be either a comma-separated string or a list. See [the table at the start of this chapter](#) for a list of valid properties.
- **namespace** (Union[PubChemNamespace, str]) – The type of identifier to look up. Valid values are in [PubChemNamespace](#). Default <PubChemNamespace.Name: 'name'>.

Raises

- **ValueError** – If the response body does not contain valid JSON.
- **NotFoundError** – If the compound with the requested identifier was not found in PubChem.

Return type Any

Returns The requested property. Type depends on the property requested.

parse_properties (*property_data*)

Parse raw data from the `property` endpoint of the REST API.

Parameters `property_data` (`Dict`)

Return type `List[Dict]`

Returns A list of dictionaries mapping the properties to values for each compound

rest_get_properties (*identifier*, *namespace*=<*PubChemNamespace.Name*: 'name'>, *properties*="", *format_*=<*PubChemFormats.CSV*: 'CSV'>)

Returns the properties for the compound with the given identifier in the desired format.

Parameters

- **identifier** (`Union[str, int, Sequence[Union[str, int]]]`) – Identifiers (e.g. name, CID) for the compound to look up. When using the CID namespace data for multiple compounds can be retrieved at once by supplying either a comma-separated string or a list.
- **namespace** – The type of identifier to look up. Valid values are in *PubChemNamespace*. Default <*PubChemNamespace.Name*: 'name'>.
- **properties** (`Union[Sequence[str], str]`) – The properties to retrieve for the compound. Can be either a comma-separated string or a list. See *the table at the start of this chapter* for a list of valid properties. Default ''.
- **format_** (`Union[PubChemFormats, str]`) – The format to obtain the data in. Default <*PubChemFormats.CSV*: 'CSV'>.

Return type `str`

rest_get_properties_json (*identifier*, *namespace*=<*PubChemNamespace.Name*: 'name'>, *properties*="", ***kwargs*)

Returns the properties for the compound with the given identifier as a dictionary.

Parameters

- **identifier** (`Union[str, int, Sequence[Union[str, int]]]`) – Identifiers (e.g. name, CID) for the compound to look up. When using the CID namespace data for multiple compounds can be retrieved at once by supplying either a comma-separated string or a list.
- **namespace** (`Union[str, PubChemNamespace]`) – The type of identifier to look up. Valid values are in *PubChemNamespace*. Default <*PubChemNamespace.Name*: 'name'>.
- **properties** (`Union[Sequence[str], str]`) – The properties to retrieve for the compound. Can be either a comma-separated string or a list. See *the table at the start of this chapter* for a list of valid properties. Default ''.
- **kwargs** – Optional arguments that `json.loads` takes.

Raises `ValueError` – If the response body does not contain valid JSON.

Return type `Dict`

Returns Parsed JSON data

valid_properties = {'AtomStereoCount': <class 'int'>, 'BondStereoCount': <class 'int'>,
Type: `Dict[str, Callable]`

Properties for PubChem REST API

4.11 chemistry_tools.pubchem.pug_rest

Functions for interacting with PubChem PUG_REST API.

Functions:

<code>async_get(identifier[, namespace, ...])</code>	Request wrapper that automatically handles asynchronous requests.
<code>do_rest_get(namespace, identifier[, ...])</code>	Responsible for performing the actual GET request.
<code>get_full_json(cid)</code>	Returns the full JSON record for the compound with the given ID.
<code>request(identifier[, namespace, operation, ...])</code>	Construct API request from parameters and return the response.

async_get (*identifier*, *namespace*='cid', *operation*=None, *output*='JSON', *searchtype*=None, ***kwargs*)

Request wrapper that automatically handles asynchronous requests.

Parameters

- **identifier** – Identifiers (e.g. name, CID) for the compounds to look up. When using the CID namespace data for multiple compounds can be retrieved at once by supplying either a comma-separated string or a list.
- **namespace** (`Union[PubChemNamespace, str]`) – The type of identifier to look up. Valid values are in `PubChemNamespace`. Default 'cid'.
- **operation** – Default None.
- **output** – Default 'JSON'.
- **searchtype** – Default None.
- ****kwargs** – Keyword parameters passed along with the GET request.

Return type `bytes`

do_rest_get (*namespace*, *identifier*, *format_*=<PubChemFormats.JSON: 'JSON'>, *domain*=None, *record_type*='2d', *png_width*=300, *png_height*=300)

Responsible for performing the actual GET request.

Parameters

- **namespace** (`Union[PubChemNamespace, str]`) – The type of identifier to look up. Valid values are in `PubChemNamespace`.
- **identifier** (`Union[str, int, Sequence[Union[str, int]]]`) – Identifiers (e.g. name, CID) for the compounds to look up. When using the CID namespace data for multiple compounds can be retrieved at once by supplying either a comma-separated string or a list.
- **format_** (`Union[PubChemFormats, str]`) – The file format to retrieve the data in. Valid values are in `PubChemFormats`, plus 'PNG'. Default <PubChemFormats.JSON: 'JSON'>.
- **domain** (`Optional[str]`) – Default None.
- **record_type** (`str`) – Default '2d'.
- **png_width** (`int`) – Default 300.
- **png_height** (`int`) – Default 300.

Return type `Response`

get_full_json (*cid*)

Returns the full JSON record for the compound with the given ID.

Parameters *cid* (`Union[str, int]`)

Return type `str`

request (*identifier*, *namespace*='cid', *operation*=None, *output*='JSON', *searchtype*=None, ***kwargs*)

Construct API request from parameters and return the response.

Full specification at http://pubchem.ncbi.nlm.nih.gov/pug_rest/PUG_REST.html

Parameters

- **identifier** – Identifiers (e.g. name, CID) for the compounds to look up. When using the CID namespace data for multiple compounds can be retrieved at once by supplying either a comma-separated string or a list.
- **namespace** (`Union[PubChemNamespace, str]`) – The type of identifier to look up. Valid values are in `PubChemNamespace`. Default 'cid'.
- **operation** – Default None.
- **output** (`Union[PubChemFormats, str]`) – Default 'JSON'.
- **searchtype** – Default None.
- ****kwargs** – Keyword parameters passed along with the GET request.

Return type `Response`

4.12 chemistry_tools.pubchem.synonyms

Functions for obtaining the synonyms of a compound from the PubChem database.

Classes:

<code>Synonyms</code> (<i>initlist</i>)	Contains a list of synonyms for a compound.
---	---

Functions:

<code>get_synonyms</code> (<i>identifier</i> [, <i>namespace</i>])	Returns a list of synonyms for the compound with the given identifier.
<code>rest_get_synonyms</code> (<i>identifier</i> [, <i>namespace</i>])	Get the list of synonyms for the given compound.

class `Synonyms` (*initlist*)

Bases: `List[str]`

Contains a list of synonyms for a compound.

Parameters *initlist* – The content to initialise the list with.

Methods:

<code>__contains__</code> (<i>synonym</i>)	Return synonym in self.
<code>append</code> (<i>synonym</i>)	Append synonym to the end of the list.

__contains__ (*synonym*)

Return *synonym* in self.

The comparison treats hyphens and underscores as whitespace.

Parameters *synonym*

Return type `bool`

append (*synonym*)

Append *synonym* to the end of the list.

Parameters *synonym* (`str`)

get_synonyms (*identifier*, *namespace*=<*PubChemNamespace.Name*: 'name'>)

Returns a list of synonyms for the compound with the given identifier. As more than one compound may be identified the results are returned in a list.

Parameters

- **identifier** (`Union[str, int, Sequence[Union[str, int]]]`) – Identifiers (e.g. name, CID) for the compound to look up. When using the CID namespace data for multiple compounds can be retrieved at once by supplying either a comma-separated string or a list.
- **namespace** (`Union[PubChemNamespace, str]`) – The type of identifier to look up. Valid values are in *PubChemNamespace*. Default <*PubChemNamespace.Name*: 'name'>.

Return type `List[Dict]`

Returns List of dictionaries containing the CID and a list of synonyms for the compounds.

rest_get_synonyms (*identifier*, *namespace*=<*PubChemNamespace.Name*: 'name'>, ****kwargs**)

Get the list of synonyms for the given compound.

Parameters

- **identifier** (`Union[str, int, Sequence[Union[str, int]]]`) – Identifiers (e.g. name, CID) for the compound to look up. When using the CID namespace data for multiple compounds can be retrieved at once by supplying either a comma-separated string or a list.
- **namespace** (`Union[PubChemNamespace, str]`) – The type of identifier to look up. Valid values are in *PubChemNamespace*. Default <*PubChemNamespace.Name*: 'name'>.
- **kwargs** – Optional arguments that `json.loads` takes.

Raises `ValueError` – If the response body does not contain valid JSON.

Return type `Dict`

Returns Parsed JSON data.

4.13 chemistry_tools.pubchem.utils

General utility functions.

Functions:

<code>format_string(stringwithmarkup)</code>	Convert a PubChem formatted string into an HTML formatted string.
--	---

format_string (*stringwithmarkup*)

Convert a PubChem formatted string into an HTML formatted string.

Parameters `stringwithmarkup` (`Dict[str, Any]`)

Return type `str`

`chemistry_tools.cache`

Cache for HTTP requests.

Data:

<code>cache</code>	The cache object.
<code>cache_dir</code>	The cache directory
<code>cached_requests</code>	Instance of <code>requests.Session</code> with a rate limit of 5 requests per second and a 28 day on-disk cache.

Functions:

<code>clear_cache()</code>	Clear the cache.
----------------------------	------------------

`cache = <apeye.rate_limiter.HTTPCache object>`

 Type: `HTTPCache`

 The cache object.

`cache_dir`

 Type: `PosixPathPlus`

 The cache directory

`cached_requests`

 Type: `Session`

 Instance of `requests.Session` with a rate limit of 5 requests per second and a 28 day on-disk cache.

`clear_cache()`

 Clear the cache.

chemistry_tools.cas

Functions for working with CAS registry numbers.

Functions:

<code>cas_int_to_string(cas_no)</code>	Converts an integer CAS registry number to a hyphenated string.
<code>cas_string_to_int(cas_no)</code>	Converts a hyphenated string CAS registry number to an integer.
<code>check_cas_number(cas_no)</code>	Checks the CAS registry number to ensure the check digit is valid with respect to the rest of the number.

cas_int_to_string (*cas_no*)

Converts an integer CAS registry number to a hyphenated string.

Parameters `cas_no` (*int*)

Return type *str*

cas_string_to_int (*cas_no*)

Converts a hyphenated string CAS registry number to an integer.

Parameters `cas_no`

Raises `ValueError` – If the CAS registry number is invalid.

check_cas_number (*cas_no*)

Checks the CAS registry number to ensure the check digit is valid with respect to the rest of the number.

If the CAS registry number is valid 0 is returned. If there is a problem the difference between the computed check digit and that given as part of the CAS registry number is returned.

Parameters `cas_no` (*int*)

Return type *int*

chemistry_tools.constants

Scientific constants.

Classes:

<i>Constant</i> (name, value, unit[, symbol])	Represents a scientific constant.
---	-----------------------------------

Data:

<i>atomic_mass_constant</i>	The atomic mass constant.
<i>avogadro_number</i>	Avogadro's constant (Avogadro's number)
<i>boltzmann_constant</i>	Boltzmann constant
<i>electron_radius</i>	Electron Radius
<i>faraday_constant</i>	Faraday constant
<i>molar_gas_constant</i>	Molar gas constant
<i>neutron_mass</i>	Neutron mass
<i>plancks_constant</i>	Planck's constant
<i>prefixes</i>	Numerical IUPAC prefixes (e.g.
<i>speed_of_light</i>	The speed of light in a vacuum.
<i>vacuum_permittivity</i>	Vacuum permittivity

class Constant (name, value, unit, symbol=None)

Bases: tuple

Represents a scientific constant.

Methods:

<i>__float__</i> ()	Returns the constant as a float (without the unit).
<i>__int__</i> ()	Returns the constant as an integer (without the unit).
<i>__repr__</i> ()	Return a nicely formatted representation string
<i>as_quantity</i> ()	Returns the constant as a quantities.quantity.Quantity object.

Attributes:

<i>name</i>	The name of the constant.
<i>symbol</i>	An optional symbol for the constant.
<i>unit</i>	The constant's unit.
<i>value</i>	The value of the constant.

`__float__()`

Returns the constant as a float (without the unit).

Return type `float`

`__int__()`

Returns the constant as an integer (without the unit).

Return type `int`

`__repr__()`

Return a nicely formatted representation string

`as_quantity()`

Returns the constant as a `quantities.quantity.Quantity` object.

Return type `Quantity`

`name`

Type: `str`

The name of the constant.

`symbol`

Type: `Optional[str]`

An optional symbol for the constant. Default `None`.

`unit`

Type: `Quantity`

The constant's unit.

`value`

Type: `float`

The value of the constant.

`atomic_mass_constant`

Type: `float`

The atomic mass constant.

`avogadro_number`

Type: `Constant`

Avogadro's constant (Avogadro's number)

`boltzmann_constant`

Type: `Constant`

Boltzmann constant

`electron_radius`

Type: `Constant`

Electron Radius

faraday_constant

Type: *Constant*

Faraday constant

molar_gas_constant

Type: *Constant*

Molar gas constant

neutron_mass

Type: *Constant*

Neutron mass

plancks_constant

Type: *Constant*

Planck's constant

prefixes = {1: 'mono', 2: 'di', 3: 'tri', 4: 'tetra', 5: 'penta', 6: 'hexa', 7: 'hep

Type: *Dict[int, str]*

Numerical IUPAC prefixes (e.g. **mono-**).

speed_of_light

Type: *Constant*

The speed of light in a vacuum.

vacuum_permittivity

Type: *Constant*

Vacuum permittivity

chemistry_tools.names

Functions for working with IUPAC names for chemicals.

Functions:

<code>cas_from_iupac_name(iupac_name)</code>	Returns the corresponding CAS registry number for the given IUPAC name.
<code>get_IUPAC_parts(string)</code>	Splits an IUPAC name for a compound into its constituent parts.
<code>get_IUPAC_sort_order(iupac_names)</code>	Returns the order the given IUPAC names should be sorted in.
<code>get_sorted_parts(iupac_names)</code>	Returns the constituent parts of the IUPAC names sorted into order.
<code>iupac_name_from_cas(cas_number)</code>	Returns the corresponding IUPAC name for the given CAS registry number.
<code>sort_IUPAC_names(iupac_names)</code>	Sort a list of IUPAC names into order.
<code>sort_array_by_name(array[, name_col, reverse])</code>	Sort a list of lists by the IUPAC name in each row.
<code>sort_dataframe_by_name(df, name_col[, reverse])</code>	Sorts a <code>pandas.DataFrame</code> by the IUPAC name in each row.

Data:

<code>multiplier_regex</code>	Regular expression to match “multiple” prefixes such as mono- .
<code>re_strings</code>	List of regular expressions to decompose an IUPAC name.

cas_from_iupac_name (*iupac_name*)

Returns the corresponding CAS registry number for the given IUPAC name.

Parameters `iupac_name` (`str`) – The IUPAC name to search.

Return type `str`

Returns The CAS registry number.

get_IUPAC_parts (*string*)

Splits an IUPAC name for a compound into its constituent parts.

Parameters `string` (`str`) – The IUPAC name to split.

Return type `List[str]`

Returns A list of constituent parts.

get_IUPAC_sort_order (*iupac_names*)

Returns the order the given IUPAC names should be sorted in.

Useful when sorting arrays containing data in addition to the name. e.g.

```
>>> sort_order = get_IUPAC_sort_order([row[0] for row in data])
>>> sorted_data = sorted(data, key=lambda row: sort_order[row[0]])
```

where row[0] would be the name of the compound

Parameters *iupac_names* (Sequence[str]) – The list of IUPAC names to sort.

Return type Dict[str, int]

Returns Dictionary mapping the IUPAC names to the order in which they should be sorted.

get_sorted_parts (*iupac_names*)

Returns the constituent parts of the IUPAC names sorted into order.

The parts returned are in reverse order (i.e. 'diphenylamine' becomes ['amine', 'phenyl', 'di']).

Parameters *iupac_names* (Sequence[str])

Return type List[List[str]]

iupac_name_from_cas (*cas_number*)

Returns the corresponding IUPAC name for the given CAS registry number.

Parameters *cas_number* (str) – The cas number to search

Return type str

Returns The IUPAC name

multiplier_regex

Type: Pattern

Regular expression to match “multiple” prefixes such as **mono-**.

Pat- tern	(mono) * (di) * (tri) * (tetra) * (penta) * (hexa) * (hepta) * (octa) * (nona) * (deca) * (undeca) * (penta)
----------------------	--

```
re_strings = [re.compile('((\\d+), ?)+(\\d+)-'), re.compile('(mono)*(di)*(tri)*(tetra)*(penta)')]
```

Type: List[Pattern]

List of regular expressions to decompose an IUPAC name.

sort_IUPAC_names (*iupac_names*)

Sort a list of IUPAC names into order.

Parameters *iupac_names* (Sequence[str]) – The list of IUPAC names to sort

Return type List[str]

Returns The list of sorted IUPAC names.

sort_array_by_name (*array*, *name_col*=0, *reverse*=False)

Sort a list of lists by the IUPAC name in each row.

Parameters

- **array** (`List[List[Any]]`)
- **name_col** (`int`) – The index of the column containing the IUPAC names. Default 0.
- **reverse** (`bool`) – Whether the names should be sorted in reverse order. Default is `False`, which sorts from A-Z.

Return type `List[List[Any]]`

Returns The sorted array

sort_dataframe_by_name (*df*, *name_col*, *reverse*=False)

Sorts a `pandas.DataFrame` by the IUPAC name in each row.

Parameters

- **df** (`DataFrame`)
- **name_col** (`str`) – The name of the column containing the IUPAC names
- **reverse** (`bool`) – Whether the names should be sorted in reverse order. Default is `False`, which sorts from A-Z

Return type `DataFrame`

Returns The sorted `DataFrame`

chemistry_tools.spectrum_similarity

Mass spectrum similarity calculations.

Classes:

<code>SpectrumSimilarity(spec_top, spec_bottom[, ...])</code>	Calculate the similarity score for two mass spectra.
---	--

Functions:

<code>create_array(intensities, mz)</code>	Create a <code>numpy.ndarray</code> , in a format appropriate for <code>SpectrumSimilarity</code> , from a list of intensities and a list of m/z values.
<code>normalize(row, max_val)</code>	Returns the normalised intensity for each rows of a <code>pandas.DataFrame</code> .
<code>spectrum_similarity(spec_top, spec_bottom[, ...])</code>	Calculate the similarity score for two mass spectra.

class `SpectrumSimilarity` (*spec_top, spec_bottom, b=1, xlim=(50, 1200)*)

Calculate the similarity score for two mass spectra.

Parameters

- **spec_top** (`ndarray`) – Array containing the experimental spectrum's peak list with the m/z values in the first column and corresponding intensities in the second
- **spec_bottom** (`ndarray`) – Array containing the reference spectrum's peak list with the m/z values in the first column and corresponding intensities in the second
- **b** (`float`) – numeric value specifying the baseline threshold for peak identification. Expressed as a percent of the maximum intensity. Default 1.
- **xlim** (`Tuple[int, int]`) – tuple of length 2, defining the beginning and ending values of the x-axis. Default (50, 1200).

New in version 1.0.0.

Methods:

<code>plot([top_label, bottom_label, filter])</code>	Plot the mass spectra head to tail.
<code>print_alignment()</code>	Print the dataframe giving aligned peaks in the top and bottom spectra.
<code>score()</code>	Returns the similarity score.

plot (*top_label=None, bottom_label=None, filter=False*)

Plot the mass spectra head to tail.

Parameters

- **top_label** (*Optional[str]*) – string to label the top spectrum. Default `None`.
- **bottom_label** (*Optional[str]*) – string to label the bottom spectrum. Default `None`.

print_alignment ()

Print the dataframe giving aligned peaks in the top and bottom spectra.

score ()

Returns the similarity score.

Return type `Tuple[float, float]`

create_array (*intensities, mz*)

Create a `numpy.ndarray`, in a format appropriate for `SpectrumSimilarity`, from a list of intensities and a list of *m/z* values.

Parameters

- **intensities** (*Sequence[float]*) – List of intensities
- **mz** (*Sequence[float]*) – List of *m/z* values.

Return type `ndarray`

normalize (*row, max_val*)

Returns the normalised intensity for each rows of a `pandas.DataFrame`.

Parameters

- **row** (*Union[Mapping, Series]*)
- **max_val** (*Union[float, str]*)

Return type `float`

spectrum_similarity (*spec_top, spec_bottom, t=0.25, b=10, top_label=None, bottom_label=None, xlim=(50, 1200), x_threshold=0, print_alignment=False, print_graphic=True, output_list=False*)

Calculate the similarity score for two mass spectra.

Attention: The `SpectrumSimilarity` class is recommended over this function.

Parameters

- **spec_top** (*ndarray*) – Array containing the experimental spectrum's peak list with the *m/z* values in the first column and corresponding intensities in the second
- **spec_bottom** (*ndarray*) – Array containing the reference spectrum's peak list with the *m/z* values in the first column and corresponding intensities in the second
- **t** (*float*) – numeric value specifying the tolerance used to align the *m/z* values of the two spectra. Default `0.25`.

- **b** (`float`) – numeric value specifying the baseline threshold for peak identification. Expressed as a percent of the maximum intensity. Default 10.
- **top_label** (`Optional[str]`) – string to label the top spectrum. Default `None`.
- **bottom_label** (`Optional[str]`) – string to label the bottom spectrum. Default `None`.
- **xlim** (`Tuple[int, int]`) – tuple of length 2, defining the beginning and ending values of the x-axis. Default (50, 1200).
- **x_threshold** (`float`) – Default 0.
- **print_alignment** (`bool`) – whether the intensities should be printed. Default `False`.
- **print_graphic** (`bool`) – Default `True`.
- **output_list** (`bool`) – whether the intensities should be returned as a third element of the tuple. Default `False`.

Return type `Union[Tuple[float, float], Tuple[float, float, DataFrame]]`

chemistry_tools.units

Functions for handling SI units.

Data:

<code>SI_base_registry</code>	Mapping of SI measurements to their units.
<code>cm3</code>	Square centimetre
<code>dimension_codes</code>	Mapping of dimension names to symbols.
<code>dm</code>	Decimetre
<code>dm3</code>	Square decimetre
<code>kilogray</code>	Kilogray
<code>kilojoule</code>	Kilojoule
<code>m3</code>	Square metre
<code>m_math_space</code>	A medium mathematical space, `` ` / ``\u205f.
<code>micromole</code>	Micromole
<code>molal</code>	Molal (moles per kilogram)
<code>nanomolar</code>	Nanomolar
<code>nanomole</code>	Nanomole
<code>per100eV</code>	Per 100 electronVolts.
<code>perMolar_perSecond</code>	Per Molar per second.
<code>umol_per_J</code>	Micro mole per joule.

Functions:

<code>allclose(a, b[, rtol, atol])</code>	Analogous to <code>numpy.allclose()</code> .
<code>as_latex(quant)</code>	Returns the LaTeX representation of the unit of a quantity.
<code>compare_equality(a, b)</code>	Returns <code>True</code> if two arguments are equal.
<code>format_si_units(value, *units)</code>	Returns the given value, followed by the given units, and separated by a medium mathematical space.
<code>format_string(value[, precision, tex])</code>	Formats a scalar with unit as two strings.

```
SI_base_registry = {'amount': UnitSubstance('mole', 'mol'), 'current': UnitCurrent('ampe
Type: dict
```

Mapping of SI measurements to their units.

allclose (*a*, *b*, *rtol*=1e-08, *atol*=None)
Analogous to `numpy.allclose()`.

Parameters

- **a**
- **b**
- **rtol** – The relative tolerance. Default 1e-08.
- **atol** – The absolute tolerance. Default `None`.

Return type `bool`

as_latex (*quant*)
Returns the LaTeX representation of the unit of a quantity.

Example:

```
>>> print(as_latex(1/quantities.kelvin))  
\mathrm{\frac{1}{K}}
```

Return type `str`

cm3 = `array(1.) * cm**3`

Type: `Quantity`

Square centimetre

compare_equality (*a*, *b*)
Returns `True` if two arguments are equal.

Both arguments need to have the same dimensionality.

Examples:

```
>>> km, m = quantities.kilometre, quantities.metre  
>>> compare_equality(3*km, 3)  
False  
>>> compare_equality(3*km, 3000*m)  
True
```

Parameters

- **a** (`Union[Quantity, float]`)
- **b** (`Union[Quantity, float]`)

Return type `bool`

dimension_codes = {'amount': 'N', 'current': 'I', 'length': 'L', 'mass': 'M', 'temperat
Type: `dict`

Mapping of dimension names to symbols.

```
dm = UnitQuantity('decimetre', 0.1 * m)
```

Type: UnitQuantity

Decimetre

```
dm3 = array(1.) * decimetre**3
```

Type: Quantity

Square decimetre

```
format_si_units(value, *units)
```

Returns the given value, followed by the given units, and separated by a medium mathematical space.

Parameters

- **value** (float)
- ***units** (str)

New in version 0.4.0.

Return type str

```
format_string(value, precision='%.5g', tex=False)
```

Formats a scalar with unit as two strings.

Examples:

```
>>> print(' '.join(format_string(0.42*quantities.mol/decimetre**3)))
0.42 mol/decimetre**3
>>> print(' '.join(format_string(2/quantities.s, tex=True)))
2 \mathrm{\frac{1}{s}}
```

Parameters

- **value** (Quantity) – Value with unit
- **precision** (str) – Default '%.5g'.
- **tex** (bool) – Whether the string should be formatted for LaTeX. Default False.

Return type Tuple[str, str]

```
kilogray = UnitQuantity('kilogray', 1000.0 * Gy)
```

Type: UnitQuantity

Kilogray

```
kilojoule = UnitQuantity('kilojoule', 1000.0 * J)
```

Type: UnitQuantity

Kilojoule

```
m3 = array(1.) * m**3
```

Type: Quantity

Square metre

m_math_space = '\u205f'

Type: str

A medium mathematical space, `` ` / ``\u205f.

New in version 0.4.0.

micromole = UnitQuantity('micromole', 1e-06 * mol)

Type: UnitQuantity

Micromole

molal = UnitQuantity('molal', 1.0 * mol/kg)

Type: UnitQuantity

Molal (moles per kilogram)

nanomolar = UnitQuantity('nM', 1e-06 * mol/m**3)

Type: UnitQuantity

Nanomolar

nanomole = UnitQuantity('nanomole', 1e-09 * mol)

Type: UnitQuantity

Nanomole

per100eV = UnitQuantity('per_100_eV', 0.01 * 1/(N_A*eV))

Type: UnitQuantity

Per 100 electronVolts.

perMolar_perSecond = array(1.) * 1/(s*M)

Type: Quantity

Per Molar per second.

umol_per_J = array(1.) * umol/J

Type: Quantity

Micro mole per joule.

Part II

Contributing

Contributing

`chemistry_tools` uses `tox` to automate testing and packaging, and `pre-commit` to maintain code quality.

Install `pre-commit` with `pip` and install the git hook:

```
$ python -m pip install pre-commit
$ pre-commit install
```

11.1 Coding style

`formate` is used for code formatting.

It can be run manually via `pre-commit`:

```
$ pre-commit run formate -a
```

Or, to run the complete autoformatting suite:

```
$ pre-commit run -a
```

11.2 Automated tests

Tests are run with `tox` and `pytest`. To run tests for a specific Python version, such as Python 3.6:

```
$ tox -e py36
```

To run tests for all Python versions, simply run:

```
$ tox
```

11.3 Type Annotations

Type annotations are checked using `mypy`. Run `mypy` using `tox`:

```
$ tox -e mypy
```

11.4 Build documentation locally

The documentation is powered by Sphinx. A local copy of the documentation can be built with `tox`:

```
$ tox -e docs
```


Downloading source code

The `chemistry_tools` source code is available on GitHub, and can be accessed from the following URL: https://github.com/domdfcoding/chemistry_tools

If you have `git` installed, you can clone the repository with the following command:

```
$ git clone https://github.com/domdfcoding/chemistry_tools
```

```
Cloning into 'chemistry_tools'...
remote: Enumerating objects: 47, done.
remote: Counting objects: 100% (47/47), done.
remote: Compressing objects: 100% (41/41), done.
remote: Total 173 (delta 16), reused 17 (delta 6), pack-reused 126
Receiving objects: 100% (173/173), 126.56 KiB | 678.00 KiB/s, done.
Resolving deltas: 100% (66/66), done.
```

Alternatively, the code can be downloaded in a ‘zip’ file by clicking:

Clone or download → Download Zip

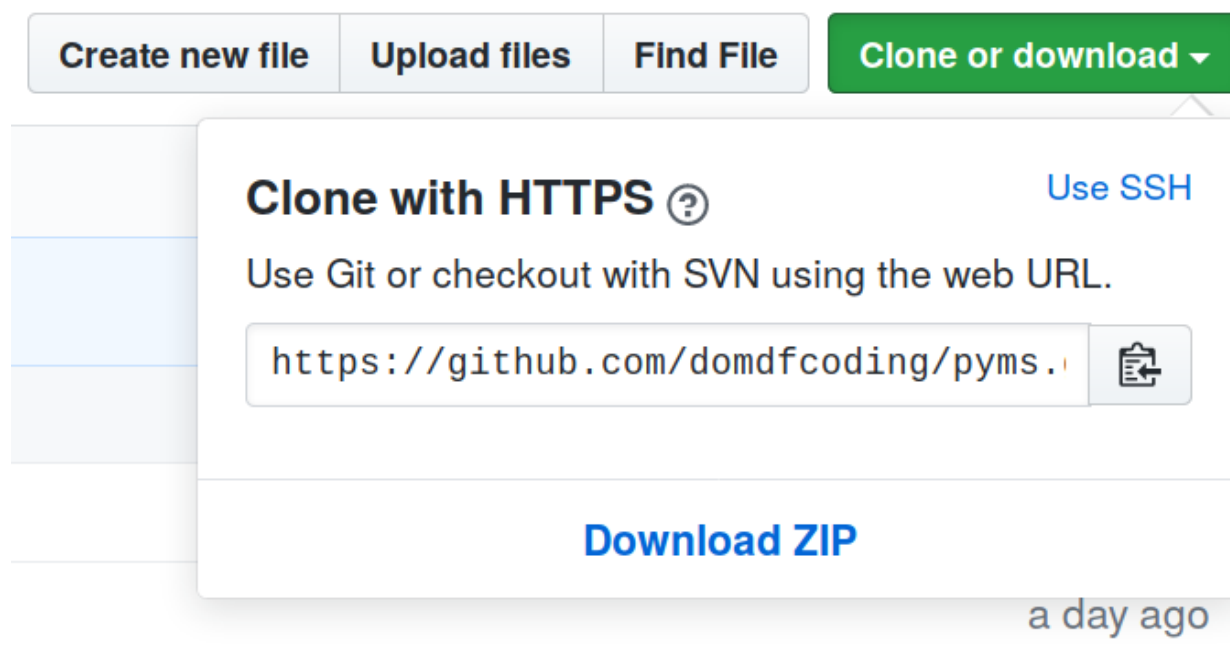


Fig. 1: Downloading a ‘zip’ file of the source code

12.1 Building from source

The recommended way to build `chemistry_tools` is to use `tox`:

```
$ tox -e build
```

The source and wheel distributions will be in the directory `dist`.

If you wish, you may also use `pep517.build` or another **PEP 517**-compatible build tool.

License

`chemistry_tools` is licensed under the [GNU Lesser General Public License v3.0](#)

Permissions of this copyleft license are conditioned on making available complete source code of licensed works and modifications under the same license or the GNU GPLv3. Copyright and license notices must be preserved. Contributors provide an express grant of patent rights. However, a larger work using the licensed work through interfaces provided by the licensed work may be distributed under different terms and without source code for the larger work.

Permissions

- Commercial use – The licensed material and derivatives may be used for commercial purposes.
- Modification – The licensed material may be modified.
- Distribution – The licensed material may be distributed.
- Patent use – This license provides an express grant of patent rights from contributors.
- Private use – The licensed material may be used and modified in private.

Conditions

- License and copyright notice – A copy of the license and copyright notice must be included with the licensed material.
- Disclose source – Source code must be made available when the licensed material is distributed.
- State changes – Changes made to the licensed material must be documented.
- Same license (library) – Modifications must be released under the same license when distributing the licensed material. In some cases a similar or related license may be used, or this condition may not apply to works that use the licensed material as a library.

Limitations

- Liability – This license includes a limitation of liability.
- Warranty – This license explicitly states that it does NOT provide any warranty.

[See more information on choosealicense.com](#) ⇒

GNU LESSER GENERAL PUBLIC LICENSE
Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <<https://fsf.org/>>
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

(continues on next page)

(continued from previous page)

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.

0. Additional Definitions.

As used herein, "this License" refers to version 3 of the GNU Lesser General Public License, and the "GNU GPL" refers to version 3 of the GNU General Public License.

"The Library" refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

An "Application" is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

A "Combined Work" is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the "Linked Version".

The "Minimal Corresponding Source" for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

The "Corresponding Application Code" for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work.

1. Exception to Section 3 of the GNU GPL.

You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL.

2. Conveying Modified Versions.

If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

- a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or
- b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.

(continues on next page)

(continued from previous page)

The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

- a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License.
- b) Accompany the object code with a copy of the GNU GPL and this license document.

4. Combined Works.

You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

- a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License.
- b) Accompany the Combined Work with a copy of the GNU GPL and this license document.
- c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document.
- d) Do one of the following:
 - 0) Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.
 - 1) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user's computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version.
- e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the

(continues on next page)

(continued from previous page)

Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

5. Combined Libraries.

You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

- a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License.
- b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.

The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy's public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.

Python Module Index

C

- `chemistry_tools.cache`, 75
- `chemistry_tools.cas`, 77
- `chemistry_tools.constants`, 79
- `chemistry_tools.elements`, 5
 - `chemistry_tools.elements.actinides`, 10
 - `chemistry_tools.elements.alkali_metals`, 6
 - `chemistry_tools.elements.alkaline_earth_metals`, 6
 - `chemistry_tools.elements.chalcogens`, 8
 - `chemistry_tools.elements.classes`, 11
 - `chemistry_tools.elements.halogens`, 9
 - `chemistry_tools.elements.lanthanides`, 9
 - `chemistry_tools.elements.noble_gases`, 9
 - `chemistry_tools.elements.pnictogens`, 8
 - `chemistry_tools.elements.tetrels`, 8
 - `chemistry_tools.elements.transition_metals`, 6
 - `chemistry_tools.elements.triels`, 7
- `chemistry_tools.formulae`, 21
 - `chemistry_tools.formulae.composition`, 21
 - `chemistry_tools.formulae.compound`, 23
 - `chemistry_tools.formulae.dataarray`, 25
 - `chemistry_tools.formulae.formula`, 28
 - `chemistry_tools.formulae.html`, 37
 - `chemistry_tools.formulae.iso_dist`, 38
 - `chemistry_tools.formulae.latex`, 42
 - `chemistry_tools.formulae.parser`, 43
 - `chemistry_tools.formulae.species`, 44
 - `chemistry_tools.formulae.unicode`, 47
 - `chemistry_tools.formulae.utils`, 48
- `chemistry_tools.names`, 83
- `chemistry_tools.pubchem`, 49
 - `chemistry_tools.pubchem.atom`, 51
 - `chemistry_tools.pubchem.bond`, 53
 - `chemistry_tools.pubchem.compound`, 55
 - `chemistry_tools.pubchem.description`, 59
 - `chemistry_tools.pubchem.enums`, 61
 - `chemistry_tools.pubchem.errors`, 63
 - `chemistry_tools.pubchem.full_record`, 64
 - `chemistry_tools.pubchem.images`, 65
 - `chemistry_tools.pubchem.lookup`, 66
 - `chemistry_tools.pubchem.properties`, 66
 - `chemistry_tools.pubchem.pug_rest`, 70
 - `chemistry_tools.pubchem.synonyms`, 71
 - `chemistry_tools.pubchem.utils`, 73
 - `chemistry_tools.spectrum_similarity`, 87
 - `chemistry_tools.units`, 91

Symbols

__add__() (*Formula method*), 29
 __class_getitem__ (*DataArray attribute*), 25
 __class_getitem__ (*IsotopeDistribution attribute*), 39
 __contains__() (*DataArray method*), 25
 __contains__() (*Elements method*), 16
 __contains__() (*IsotopeDistribution method*), 39
 __contains__() (*Synonyms method*), 71
 __eq__() (*Atom method*), 51
 __eq__() (*Bond method*), 53
 __eq__() (*Compound method*), 23
 __eq__() (*DataArray method*), 26
 __eq__() (*Formula method*), 29
 __eq__() (*IsotopeDistribution method*), 39
 __eq__() (*Species method*), 44
 __float__() (*Constant method*), 79
 __getitem__() (*DataArray method*), 26
 __getitem__() (*Elements method*), 16
 __getitem__() (*IsotopeDistribution method*), 39
 __iadd__() (*Formula method*), 29
 __imul__() (*Formula method*), 30
 __int__() (*Constant method*), 80
 __isub__() (*Formula method*), 30
 __iter__() (*DataArray method*), 26
 __iter__() (*Elements method*), 16
 __iter__() (*IsotopeDistribution method*), 39
 __len__() (*DataArray method*), 26
 __len__() (*Elements method*), 16
 __len__() (*IsotopeDistribution method*), 39
 __mul__() (*Formula method*), 30
 __new__() (*PubChemProperty static method*), 67
 __radd__() (*Formula method*), 30
 __repr__() (*Atom method*), 51
 __repr__() (*Bond method*), 53
 __repr__() (*Compound method*), 23, 56
 __repr__() (*Constant method*), 80
 __repr__() (*DataArray method*), 26
 __repr__() (*Element method*), 12
 __repr__() (*Elements method*), 16
 __repr__() (*Formula method*), 30
 __repr__() (*Isotope method*), 19
 __repr__() (*IsotopeDistribution method*), 39
 __repr__() (*PropData method*), 67

__rmul__() (*Formula method*), 30
 __rsub__() (*Formula method*), 30
 __setitem__() (*Formula method*), 30
 __str__() (*Composition method*), 21
 __str__() (*Compound method*), 24
 __str__() (*DataArray method*), 26
 __str__() (*Element method*), 12
 __str__() (*Elements method*), 16
 __str__() (*Formula method*), 30
 __str__() (*Isotope method*), 19
 __str__() (*IsotopeDistribution method*), 40
 __str__() (*PubChemHTTPError method*), 64
 __sub__() (*Formula method*), 30

A

Abundance (*IsoDistSort attribute*), 38
 abundance() (*Isotope property*), 19
 Ac (*in module chemistry_tools.elements.actinides*), 10
 add_alternate_spelling() (*Elements method*), 17
 Ag (*in module chemistry_tools.elements.transition_metals*), 7
 Al (*in module chemistry_tools.elements.triels*), 7
 ALIGNED (*CoordinateType attribute*), 61
 allclose() (*in module chemistry_tools.units*), 92
 Am (*in module chemistry_tools.elements.actinides*), 10
 append() (*Synonyms method*), 72
 Ar (*in module chemistry_tools.elements.noble_gases*), 9
 As (*in module chemistry_tools.elements.pnictogens*), 8
 as_array() (*Composition method*), 22
 as_array() (*DataArray method*), 26
 as_array() (*IsotopeDistribution method*), 40
 as_csv() (*DataArray method*), 26
 as_csv() (*IsotopeDistribution method*), 40
 as_dataframe() (*DataArray method*), 27
 as_dataframe() (*IsotopeDistribution method*), 40
 as_isotope() (*HeavyHydrogen property*), 18
 as_latex() (*in module chemistry_tools.units*), 92
 as_quantity() (*Constant method*), 80
 as_table() (*DataArray method*), 27
 as_table() (*IsotopeDistribution method*), 40
 async_get() (*in module chemistry_tools.pubchem.pug_rest*), 70
 At (*in module chemistry_tools.elements.halogens*), 9

atmrad() (*Element property*), 13
Atom (*class in chemistry_tools.pubchem.atom*), 51
atomic_mass_constant (*in module chemistry_tools.constants*), 80
atoms() (*Compound property*), 56
attr_name (*namedtuple field*)
 PropData (*namedtuple in chemistry_tools.pubchem.properties*), 67
Au (*in module chemistry_tools.elements.transition_metals*), 7
AUGMENTED (*CoordinateType attribute*), 61
average_mass() (*Formula property*), 31
average_mz() (*Formula property*), 31
avogadro_number (*in module chemistry_tools.constants*), 80

B

B (*in module chemistry_tools.elements.triels*), 7
Ba (*in module chemistry_tools.elements.alkaline_earth_metals*), 6
BadRequestError, 63
Be (*in module chemistry_tools.elements.alkaline_earth_metals*), 6
Bh (*in module chemistry_tools.elements.transition_metals*), 7
Bi (*in module chemistry_tools.elements.pnictogens*), 8
Bk (*in module chemistry_tools.elements.actinides*), 10
block() (*Element property*), 13
boltzmann_constant (*in module chemistry_tools.constants*), 80
Bond (*class in chemistry_tools.pubchem.bond*), 53
bonds() (*Compound property*), 56
Br (*in module chemistry_tools.elements.halogens*), 9

C

C (*in module chemistry_tools.elements.tetrels*), 8
C (*in module chemistry_tools.pubchem.compound*), 55
Ca (*in module chemistry_tools.elements.alkaline_earth_metals*), 6
cache (*in module chemistry_tools.cache*), 75
cache_dir (*in module chemistry_tools.cache*), 75
cached_requests (*in module chemistry_tools.cache*), 75
cactvs_fingerprint() (*Compound property*), 56
canonical_smiles() (*Compound property*), 56
canonicalized() (*Compound property*), 56
cas_from_iupac_name() (*in module chemistry_tools.names*), 83
cas_int_to_string() (*in module chemistry_tools.cas*), 77

cas_string_to_int() (*in module chemistry_tools.cas*), 77
Cd (*in module chemistry_tools.elements.transition_metals*), 7
Ce (*in module chemistry_tools.elements.lanthanides*), 9
Cf (*in module chemistry_tools.elements.actinides*), 10
charge() (*Compound property*), 24, 56
check_cas_number() (*in module chemistry_tools.cas*), 77
chemistry_tools.cache
 module, 75
chemistry_tools.cas
 module, 77
chemistry_tools.constants
 module, 79
chemistry_tools.elements
 module, 5
chemistry_tools.elements.actinides
 module, 10
chemistry_tools.elements.alkali_metals
 module, 6
chemistry_tools.elements.alkaline_earth_metals
 module, 6
chemistry_tools.elements.chalcogens
 module, 8
chemistry_tools.elements.classes
 module, 11
chemistry_tools.elements.halogens
 module, 9
chemistry_tools.elements.lanthanides
 module, 9
chemistry_tools.elements.noble_gases
 module, 9
chemistry_tools.elements.pnictogens
 module, 8
chemistry_tools.elements.tetrels
 module, 8
chemistry_tools.elements.transition_metals
 module, 6
chemistry_tools.elements.triels
 module, 7
chemistry_tools.formulae
 module, 21
chemistry_tools.formulae.composition
 module, 21
chemistry_tools.formulae.compound
 module, 23
chemistry_tools.formulae.dataarray
 module, 25
chemistry_tools.formulae.formula
 module, 28
chemistry_tools.formulae.html
 module, 37
chemistry_tools.formulae.iso_dist

module, 38
 chemistry_tools.formulae.latex
 module, 42
 chemistry_tools.formulae.parser
 module, 43
 chemistry_tools.formulae.species
 module, 44
 chemistry_tools.formulae.unicode
 module, 47
 chemistry_tools.formulae.utils
 module, 48
 chemistry_tools.names
 module, 83
 chemistry_tools.pubchem
 module, 49
 chemistry_tools.pubchem.atom
 module, 51
 chemistry_tools.pubchem.bond
 module, 53
 chemistry_tools.pubchem.compound
 module, 55
 chemistry_tools.pubchem.description
 module, 59
 chemistry_tools.pubchem.enums
 module, 61
 chemistry_tools.pubchem.errors
 module, 63
 chemistry_tools.pubchem.full_record
 module, 64
 chemistry_tools.pubchem.images
 module, 65
 chemistry_tools.pubchem.lookup
 module, 66
 chemistry_tools.pubchem.properties
 module, 66
 chemistry_tools.pubchem.pug_rest
 module, 70
 chemistry_tools.pubchem.synonyms
 module, 71
 chemistry_tools.pubchem.utils
 module, 73
 chemistry_tools.spectrum_similarity
 module, 87
 chemistry_tools.units
 module, 91
 CID (*PubChemNamespace* attribute), 62
 cid() (*Compound* property), 57
 Cl (*in module chemistry_tools.elements.halogens*), 9
 clear_cache() (*in module chemistry_tools.cache*),
 75
 Cm (*in module chemistry_tools.elements.actinides*), 10
 cm3 (*in module chemistry_tools.units*), 92
 Cn (*in module*
 chemistry_tools.elements.transition_metals), 7

Co (*in module*
 chemistry_tools.elements.transition_metals), 6
 COMPACT (*CoordinateType* attribute), 61
 compare_equality() (*in module*
 chemistry_tools.units), 92
 COMPLEX (*BondType* attribute), 54
 Composition (*class in*
 chemistry_tools.formulae.composition), 21
 composition() (*Formula* property), 31
 Compound (*class in*
 chemistry_tools.formulae.compound), 23
 Compound (*class in*
 chemistry_tools.pubchem.compound), 55
 compounds_to_frame() (*in module*
 chemistry_tools.pubchem.compound), 58
 COMPUTED (*CoordinateType* attribute), 61
 Constant (*class in chemistry_tools.constants*), 79
 coordinate_type() (*Atom* property), 51
 coordinate_type() (*Compound* property), 57
 copy() (*DataArray* method), 27
 copy() (*Formula* method), 31
 copy() (*IsotopeDistribution* method), 40
 copy() (*Species* method), 45
 count (*CompositionSort* attribute), 22
 covrad() (*Element* property), 13
 Cr (*in module*
 chemistry_tools.elements.transition_metals), 6
 create_array() (*in module*
 chemistry_tools.spectrum_similarity), 88
 Cs (*in module chemistry_tools.elements.alkali_metals*), 6
 CSV (*PubChemFormats* attribute), 62
 Cu (*in module*
 chemistry_tools.elements.transition_metals), 6

D

DataArray (*class in*
 chemistry_tools.formulae.dataarray), 25
 DATIVE (*BondType* attribute), 54
 Db (*in module*
 chemistry_tools.elements.transition_metals), 7
 density() (*Element* property), 13
 description (*namedtuple* field)
 PropData (*namedtuple in*
 chemistry_tools.pubchem.properties), 67
 description() (*Element* property), 13
 dimension_codes (*in module chemistry_tools.units*),
 92
 dm (*in module chemistry_tools.units*), 92
 dm3 (*in module chemistry_tools.units*), 93
 do_rest_get() (*in module*
 chemistry_tools.pubchem.pug_rest), 70
 DOUBLE (*BondType* attribute), 54
 Ds (*in module*
 chemistry_tools.elements.transition_metals), 7

`dtype` (*namedtuple field*)

`PubChemProperty` (*namedtuple in*
chemistry_tools.pubchem.properties), 67

`Dy` (*in module chemistry_tools.elements.lanthanides*), 9

E

`eleaffin` (*Element property*), 13

`eleconfig` (*Element property*), 13

`eleconfig_dict` (*Element property*), 13

`electron_radius` (*in module*
chemistry_tools.constants), 80

`electrons` (*Element property*), 13

`Element` (*class in chemistry_tools.elements.classes*), 11

`element` (*Atom property*), 52

`Elements` (*class in chemistry_tools.elements.classes*),
15

`elements` (*Compound property*), 57

`elements` (*Formula property*), 31

`eleneg` (*Element property*), 13

`elshells` (*Element property*), 14

`empirical_formula` (*Formula property*), 31

`empirical_formula` (*Species property*), 45

`Er` (*in module chemistry_tools.elements.lanthanides*), 10

`Es` (*in module chemistry_tools.elements.actinides*), 10

`Eu` (*in module chemistry_tools.elements.lanthanides*), 9

`exact_mass` (*Formula property*), 31

`exactmass` (*Element property*), 14

`EXPERIMENTAL` (*CoordinateType attribute*), 61

F

`F` (*in module chemistry_tools.elements.halogens*), 9

`F` (*in module chemistry_tools.formulae.formula*), 28

`faraday_constant` (*in module*
chemistry_tools.constants), 80

`Fe` (*in module*
chemistry_tools.elements.transition_metals), 6

`fingerprint` (*Compound property*), 57

`Fl` (*in module chemistry_tools.elements.tetrels*), 8

`Fm` (*in module chemistry_tools.elements.actinides*), 10

`force_valid_properties` (*in module*
chemistry_tools.pubchem.properties), 67

`format_si_units` (*in module*
chemistry_tools.units), 93

`format_string` (*in module*
chemistry_tools.pubchem.utils), 73

`format_string` (*in module chemistry_tools.units*),
93

`Formula` (*class in chemistry_tools.formulae.formula*),
28

`Formula` (*IsoDistSort attribute*), 38

`Fr` (*in module chemistry_tools.elements.alkali_metals*), 6

`from_cid` (*Compound class method*), 57

`from_kwargs` (*Formula class method*), 31

`from_kwargs` (*Species class method*), 45

`from_mass_fractions` (*Formula class method*),
33

`from_string` (*Formula class method*), 33

`from_string` (*Species class method*), 45

`fromkeys` (*DataArray class method*), 27

`fromkeys` (*IsotopeDistribution class method*), 40

G

`Ga` (*in module chemistry_tools.elements.triells*), 7

`Gd` (*in module chemistry_tools.elements.lanthanides*), 9

`Ge` (*in module chemistry_tools.elements.tetrels*), 8

`get` (*DataArray method*), 27

`get` (*IsotopeDistribution method*), 41

`get_common_name` (*in module*
chemistry_tools.pubchem.description), 59

`get_compound_id` (*in module*
chemistry_tools.pubchem.description), 59

`get_compounds` (*in module*
chemistry_tools.pubchem.lookup), 66

`get_description` (*in module*
chemistry_tools.pubchem.description), 59

`get_full_json` (*in module*
chemistry_tools.pubchem.pug_rest), 71

`get_iupac_name` (*Compound method*), 57

`get_iupac_name` (*in module*
chemistry_tools.pubchem.description), 59

`get_IUPAC_parts` (*in module*
chemistry_tools.names), 83

`get_IUPAC_sort_order` (*in module*
chemistry_tools.names), 83

`get_mz` (*Formula method*), 33

`get_properties` (*Compound method*), 57

`get_properties` (*in module*
chemistry_tools.pubchem.properties), 67

`get_property` (*Compound method*), 57

`get_property` (*in module*
chemistry_tools.pubchem.properties), 68

`get_sorted_parts` (*in module*
chemistry_tools.names), 84

`get_structure_image` (*in module*
chemistry_tools.pubchem.images), 65

`get_synonyms` (*in module*
chemistry_tools.pubchem.synonyms), 72

GNU Lesser General Public License
v3.0, 101

`group` (*Element property*), 14

`GROUPS` (*in module chemistry_tools.formulae.utils*), 48

H

`has_full_record` (*Compound property*), 57

`He` (*in module chemistry_tools.elements.noble_gases*), 9

`HeavyHydrogen` (*class in*
chemistry_tools.elements.classes), 17

Hf (in module *chemistry_tools.elements.transition_metals*), 7
 Hg (in module *chemistry_tools.elements.transition_metals*), 7
 hill_formula() (Formula property), 33
 hill_formula() (Species property), 46
 hill_order() (in module *chemistry_tools.formulae.utils*), 48
 Ho (in module *chemistry_tools.elements.lanthanides*), 9
 Hs (in module *chemistry_tools.elements.transition_metals*), 7
 html_subscript() (in module *chemistry_tools.formulae.html*), 37
 html_superscript() (in module *chemistry_tools.formulae.html*), 37
 HTTP_ERROR_CODES (in module *chemistry_tools.pubchem.errors*), 63
 HTTPTimeoutError, 63

I

I (in module *chemistry_tools.elements.halogens*), 9
 In (in module *chemistry_tools.elements.triels*), 7
 INCHIKEY (PubChemNamespace attribute), 62
 ionenergy() (Element property), 14
 IONIC (BondType attribute), 54
 Ir (in module *chemistry_tools.elements.transition_metals*), 7
 is_valid_value() (CoordinateType class method), 61
 is_valid_value() (PubChemFormats class method), 62
 is_valid_value() (PubChemNamespace class method), 63
 Isotope (class in *chemistry_tools.elements.classes*), 18
 isotope_distribution() (Formula method), 34
 IsotopeDict (in module *chemistry_tools.elements.classes*), 19
 IsotopeDistribution (class in *chemistry_tools.formulae.iso_dist*), 38
 isotopes() (Element property), 14
 isotopic_composition_abundance() (Formula property), 34
 items() (DataArray method), 27
 items() (IsotopeDistribution method), 41
 iter_isotopologues() (Formula method), 34
 iupac_name() (Compound property), 58
 iupac_name_from_cas() (in module *chemistry_tools.names*), 84

J

JSON (PubChemFormats attribute), 62

K

K (in module *chemistry_tools.elements.alkali_metals*), 6

keys() (DataArray method), 27
 keys() (IsotopeDistribution method), 41
 kilogray (in module *chemistry_tools.units*), 93
 kilojoule (in module *chemistry_tools.units*), 93
 Kr (in module *chemistry_tools.elements.noble_gases*), 9

L

La (in module *chemistry_tools.elements.lanthanides*), 9
 label (namedtuple field)
 PubChemProperty (namedtuple in *chemistry_tools.pubchem.properties*), 67
 latex_subscript() (in module *chemistry_tools.formulae.latex*), 42
 latex_superscript() (in module *chemistry_tools.formulae.latex*), 42
 Li (in module *chemistry_tools.elements.alkali_metals*), 6
 lower_names() (Elements property), 17
 Lr (in module *chemistry_tools.elements.actinides*), 10
 Lu (in module *chemistry_tools.elements.lanthanides*), 10
 Lv (in module *chemistry_tools.elements.chalcogens*), 8

M

m3 (in module *chemistry_tools.units*), 93
 m_math_space (in module *chemistry_tools.units*), 93
 Mass (IsoDistSort attribute), 38
 mass() (Compound property), 24
 mass() (Element property), 14
 mass() (Formula property), 35
 mass() (Isotope property), 19
 mass_fraction (CompositionSort attribute), 22
 mass_from_composition() (in module *chemistry_tools.formulae.parser*), 43
 massnumber() (Isotope property), 19
 Mc (in module *chemistry_tools.elements.pnictogens*), 8
 Md (in module *chemistry_tools.elements.actinides*), 10
 MethodNotAllowedError, 64
 Mg (in module *chemistry_tools.elements.alkaline_earth_metals*), 6
 micromole (in module *chemistry_tools.units*), 94
 Mn (in module *chemistry_tools.elements.transition_metals*), 6
 Mo (in module *chemistry_tools.elements.transition_metals*), 7
 module
 chemistry_tools.cache, 75
 chemistry_tools.cas, 77
 chemistry_tools.constants, 79
 chemistry_tools.elements, 5
 chemistry_tools.elements.actinides, 10
 chemistry_tools.elements.alkali_metals, 6

[chemistry_tools.elements.alkaline_earth_metals.constant](#) (in module [chemistry_tools.constants](#)), 81
[chemistry_tools.elements.alkaline_earth_metals.molar_mass](#) () (*Compound method*), 24
[chemistry_tools.elements.alkaline_earth_metals.molecular_formula](#) () (*Compound property*), 58
[chemistry_tools.elements.alkaline_earth_metals.molecular_mass](#) () (*Compound property*), 58
[chemistry_tools.elements.alkaline_earth_metals.molecular_weight](#) () (*Compound property*), 58
[chemistry_tools.elements.alkaline_earth_metals.molecular_weight](#) () (*Element property*), 14
[chemistry_tools.elements.alkaline_earth_metals.monoisotopic_mass](#) () (*Formula property*), 35
[chemistry_tools.elements.alkaline_earth_metals.most_probable_isotopic_composition](#) () (*Formula method*), 35
[chemistry_tools.elements.alkaline_earth_metals.Mt](#) (in module [chemistry_tools.elements.transition_metals](#)), 7
[chemistry_tools.elements.alkaline_earth_metals.multiplier_regex](#) (in module [chemistry_tools.names](#)), 84
[chemistry_tools.elements.alkaline_earth_metals.mz](#) () (*Formula property*), 35
[chemistry_tools.elements.alkaline_earth_metals.triels](#), 7
[chemistry_tools.formulae](#), 21
[chemistry_tools.formulae.composition](#), 21
[chemistry_tools.formulae.compound](#), 23
[chemistry_tools.formulae.dataarray](#), 25
[chemistry_tools.formulae.formula](#), 28
[chemistry_tools.formulae.html](#), 37
[chemistry_tools.formulae.iso_dist](#), 38
[chemistry_tools.formulae.latex](#), 42
[chemistry_tools.formulae.parser](#), 43
[chemistry_tools.formulae.species](#), 44
[chemistry_tools.formulae.unicode](#), 47
[chemistry_tools.formulae.utils](#), 48
[chemistry_tools.names](#), 83
[chemistry_tools.pubchem](#), 49
[chemistry_tools.pubchem.atom](#), 51
[chemistry_tools.pubchem.bond](#), 53
[chemistry_tools.pubchem.compound](#), 55
[chemistry_tools.pubchem.description](#), 59
[chemistry_tools.pubchem.enums](#), 61
[chemistry_tools.pubchem.errors](#), 63
[chemistry_tools.pubchem.full_record](#), 64
[chemistry_tools.pubchem.images](#), 65
[chemistry_tools.pubchem.lookup](#), 66
[chemistry_tools.pubchem.properties](#), 66
[chemistry_tools.pubchem.pug_rest](#), 70
[chemistry_tools.pubchem.synonyms](#), 71
[chemistry_tools.pubchem.utils](#), 73
[chemistry_tools.spectrum_similarity](#), 87
[chemistry_tools.units](#), 91
[molal](#) (in module [chemistry_tools.units](#)), 94

N
[N](#) (in module [chemistry_tools.elements.pnictogens](#)), 8
[n_atoms](#) () (*Formula property*), 35
[n_elements](#) () (*Composition property*), 22
[n_elements](#) () (*Formula property*), 35
[Na](#) (in module [chemistry_tools.elements.alkali_metals](#)), 6
[name](#) (*Constant attribute*), 80
[name](#) (*namedtuple field*)
 [PropData](#) (*namedtuple in chemistry_tools.pubchem.properties*), 67
 [PubChemProperty](#) (*namedtuple in chemistry_tools.pubchem.properties*), 67
[Name](#) (*PubChemNamespace attribute*), 62
[name](#) () (*Element property*), 14
[names](#) () (*Elements property*), 17
[nanomolar](#) (in module [chemistry_tools.units](#)), 94
[nanomole](#) (in module [chemistry_tools.units](#)), 94
[Nb](#) (in module [chemistry_tools.elements.transition_metals](#)), 6
[Nd](#) (in module [chemistry_tools.elements.lanthanides](#)), 9
[Ne](#) (in module [chemistry_tools.elements.noble_gases](#)), 9
[neutron_mass](#) (in module [chemistry_tools.constants](#)), 81
[neutrons](#) () (*Element property*), 14
[Nh](#) (in module [chemistry_tools.elements.triels](#)), 8
[Ni](#) (in module [chemistry_tools.elements.transition_metals](#)), 6
[No](#) (in module [chemistry_tools.elements.actinides](#)), 10
[no_isotope_hill_formula](#) () (*Formula property*), 36
[nominal_mass](#) () (*Formula property*), 36
[nominalmass](#) () (*Element property*), 14
[nominalmass](#) () (*HeavyHydrogen property*), 18
[normalize](#) () (in module [chemistry_tools.spectrum_similarity](#)), 88
[NotFoundError](#), 64
[Np](#) (in module [chemistry_tools.elements.actinides](#)), 10
[number](#) () (*Element property*), 15

O

O (in module *chemistry_tools.elements.chalcogens*), 8
 Og (in module *chemistry_tools.elements.noble_gases*), 9
 Os (in module
 chemistry_tools.elements.transition_metals), 7
 oxistates() (Element property), 15

P

P (in module *chemistry_tools.elements.pnictogens*), 8
 Pa (in module *chemistry_tools.elements.actinides*), 10
 parse_atoms() (in module
 chemistry_tools.pubchem.atom), 52
 parse_bonds() (in module
 chemistry_tools.pubchem.bond), 54
 parse_description() (in module
 chemistry_tools.pubchem.description), 59
 parse_full_record() (in module
 chemistry_tools.pubchem.full_record), 64
 parse_properties() (in module
 chemistry_tools.pubchem.properties), 69
 Pb (in module *chemistry_tools.elements.tetrels*), 8
 Pd (in module
 chemistry_tools.elements.transition_metals), 7
 per100eV (in module *chemistry_tools.units*), 94
 period() (Element property), 15
 perMolar_perSecond (in module
 chemistry_tools.units), 94
 phase (Species attribute), 46
 plancks_constant (in module
 chemistry_tools.constants), 81
 plot() (SpectrumSimilarity method), 87
 Pm (in module *chemistry_tools.elements.lanthanides*), 9
 PNG (PubChemFormats attribute), 62
 Po (in module *chemistry_tools.elements.chalcogens*), 8
 Pr (in module *chemistry_tools.elements.lanthanides*), 9
 precache() (Compound method), 58
 prefixes (in module *chemistry_tools.constants*), 81
 print_alignment() (SpectrumSimilarity method), 88
 PropData (namedtuple in
 chemistry_tools.pubchem.properties), 67
 attr_name (namedtuple field), 67
 description (namedtuple field), 67
 name (namedtuple field), 67
 type (namedtuple field), 67
 PROPERTY_MAP (in module
 chemistry_tools.pubchem.properties), 67
 protons() (Element property), 15
 Pt (in module
 chemistry_tools.elements.transition_metals), 7
 Pu (in module *chemistry_tools.elements.actinides*), 10
 PubChemHTTPError, 64
 PubChemProperty (namedtuple in
 chemistry_tools.pubchem.properties), 67

dtype (namedtuple field), 67
 label (namedtuple field), 67
 name (namedtuple field), 67
 source (namedtuple field), 67
 value (namedtuple field), 67

Python Enhancement Proposals
 PEP 517, 100

Q

QUADRUPLER (BondType attribute), 54

R

Ra (in module *chemistry_tools.elements.alkaline_earth_metals*), 6
 Rb (in module *chemistry_tools.elements.alkali_metals*), 6
 Re (in module
 chemistry_tools.elements.transition_metals), 7
 re_strings (in module *chemistry_tools.names*), 84
 rel_mass (CompositionSort attribute), 22
 Relative_Abandance (IsoDistSort attribute), 38
 request() (in module
 chemistry_tools.pubchem.pug_rest), 71
 ResponseParseError, 64
 rest_get_description() (in module
 chemistry_tools.pubchem.description), 59
 rest_get_full_record() (in module
 chemistry_tools.pubchem.full_record), 65
 rest_get_properties() (in module
 chemistry_tools.pubchem.properties), 69
 rest_get_properties_json() (in module
 chemistry_tools.pubchem.properties), 69
 rest_get_synonyms() (in module
 chemistry_tools.pubchem.synonyms), 72
 Rf (in module
 chemistry_tools.elements.transition_metals), 7
 Rg (in module
 chemistry_tools.elements.transition_metals), 7
 Rh (in module
 chemistry_tools.elements.transition_metals), 7
 Rn (in module *chemistry_tools.elements.noble_gases*), 9
 Ru (in module
 chemistry_tools.elements.transition_metals), 7

S

S (in module *chemistry_tools.elements.chalcogens*), 8
 S (in module *chemistry_tools.formulae.species*), 44
 Sb (in module *chemistry_tools.elements.pnictogens*), 8
 Sc (in module
 chemistry_tools.elements.transition_metals), 6
 score() (SpectrumSimilarity method), 88
 Se (in module *chemistry_tools.elements.chalcogens*), 8
 series() (Element property), 15
 ServerError, 64

set_coordinates() (*Atom method*), 52
 Sg (*in module chemistry_tools.elements.transition_metals*), 7
 Si (*in module chemistry_tools.elements.tetrels*), 8
 SI_base_registry (*in module chemistry_tools.units*), 91
 SINGLE (*BondType attribute*), 54
 Sm (*in module chemistry_tools.elements.lanthanides*), 9
 SMILES (*PubChemNamespace attribute*), 62
 smiles() (*Compound property*), 58
 Sn (*in module chemistry_tools.elements.tetrels*), 8
 sort_array_by_name() (*in module chemistry_tools.names*), 84
 sort_dataframe_by_name() (*in module chemistry_tools.names*), 85
 sort_IUPAC_names() (*in module chemistry_tools.names*), 84
 source (*namedtuple field*)
 PubChemProperty (*namedtuple in chemistry_tools.pubchem.properties*), 67
 Species (*class in chemistry_tools.formulae.species*), 44
 spectrum_similarity() (*in module chemistry_tools.spectrum_similarity*), 88
 SpectrumSimilarity (*class in chemistry_tools.spectrum_similarity*), 87
 speed_of_light (*in module chemistry_tools.constants*), 81
 split_isotope() (*Elements method*), 17
 split_isotope() (*in module chemistry_tools.formulae.utils*), 48
 Sr (*in module chemistry_tools.elements.alkaline_earth_metals*), 6
 STANDARDIZED (*CoordinateType attribute*), 61
 string_to_composition() (*in module chemistry_tools.formulae.parser*), 43
 string_to_html() (*in module chemistry_tools.formulae.html*), 37
 string_to_latex() (*in module chemistry_tools.formulae.latex*), 42
 string_to_unicode() (*in module chemistry_tools.formulae.unicode*), 47
 SUBMITTED (*CoordinateType attribute*), 61
 symbol (*CompositionSort attribute*), 22
 symbol (*Constant attribute*), 80
 symbol() (*Element property*), 15
 symbols() (*Elements property*), 17
 Synonyms (*class in chemistry_tools.pubchem.synonyms*), 71
 synonyms() (*Compound property*), 58
 systematic_name() (*Compound property*), 58

T

Ta (*in module chemistry_tools.elements.transition_metals*), 7
 Tb (*in module chemistry_tools.elements.lanthanides*), 9
 tboil() (*Element property*), 15
 Tc (*in module chemistry_tools.elements.transition_metals*), 7
 Te (*in module chemistry_tools.elements.chalcogens*), 8
 Th (*in module chemistry_tools.elements.actinides*), 10
 THREE_D (*CoordinateType attribute*), 61
 Ti (*in module chemistry_tools.elements.transition_metals*), 6
 TimeoutError (*in module chemistry_tools.pubchem.errors*), 64
 Tl (*in module chemistry_tools.elements.tries*), 7
 Tm (*in module chemistry_tools.elements.lanthanides*), 10
 tmelt() (*Element property*), 15
 to_dict() (*Atom method*), 52
 to_dict() (*Bond method*), 53
 to_series() (*Compound method*), 58
 total_mass() (*Composition property*), 22
 TRIPLE (*BondType attribute*), 54
 Ts (*in module chemistry_tools.elements.halogens*), 9
 TWO_D (*CoordinateType attribute*), 61
 type (*namedtuple field*)
 PropData (*namedtuple in chemistry_tools.pubchem.properties*), 67

U

U (*in module chemistry_tools.elements.actinides*), 10
 umol_per_J (*in module chemistry_tools.units*), 94
 unicode_subscript() (*in module chemistry_tools.formulae.unicode*), 47
 unicode_superscript() (*in module chemistry_tools.formulae.unicode*), 47
 NotImplementedError, 64
 unit (*Constant attribute*), 80
 UNITS_ANGSTROMS (*CoordinateType attribute*), 61
 UNITS_NANOMETERS (*CoordinateType attribute*), 61
 UNITS_PIXEL (*CoordinateType attribute*), 61
 UNITS_POINTS (*CoordinateType attribute*), 61
 UNITS_STDBONDS (*CoordinateType attribute*), 61
 UNITS_UNKNOWN (*CoordinateType attribute*), 61
 UNKNOWN (*BondType attribute*), 54

V

V (*in module chemistry_tools.elements.transition_metals*), 6
 vacuum_permittivity (*in module chemistry_tools.constants*), 81
 valid_properties (*in module chemistry_tools.pubchem.properties*), 69
 validate() (*Element method*), 15

`value` (*Constant attribute*), 80
`value` (*namedtuple field*)
 `PubChemProperty` (*namedtuple in*
 chemistry_tools.pubchem.properties), 67
`values()` (*DataArray method*), 27
`values()` (*IsotopeDistribution method*), 41
`vdwrad()` (*Element property*), 15

W

`W` (*in module*
 chemistry_tools.elements.transition_metals), 7

X

`Xe` (*in module chemistry_tools.elements.noble_gases*), 9
`XML` (*PubChemFormats attribute*), 62

Y

`Y` (*in module*
 chemistry_tools.elements.transition_metals), 6
`Yb` (*in module chemistry_tools.elements.lanthanides*), 10

Z

`Zn` (*in module*
 chemistry_tools.elements.transition_metals), 6
`Zr` (*in module*
 chemistry_tools.elements.transition_metals), 6