# Table of Contents

1 Getting started .................................................. 3  
1.1 Overview ...................................................... 3  
1.2 Contributing .................................................... 3  
1.3 Citing ........................................................... 3  
1.4 Related projects ............................................. 4  
1.5 Funding .......................................................... 4  
2 Installation ......................................................... 5  
3 Tutorials .......................................................... 7  
3.1 mendeleev tutorial ............................................ 7  
3.2 Getting tables from the database ............................ 15  
3.3 Jupyter notebooks ............................................ 19  
4 Data .............................................................. 21  
4.1 Elements ......................................................... 21  
4.2 Isotopes .......................................................... 25  
5 Electronegativities ............................................... 27  
5.1 Allen ............................................................. 28  
5.2 Allred and Rochow ............................................ 28  
5.3 Cottrell and Sutton ......................................... 28  
5.4 Ghosh ............................................................. 28  
5.5 Gordy ............................................................. 29  
5.6 Li and Xue ....................................................... 29  
5.7 Martynov and Batsanov ..................................... 29  
5.8 Mulliken .......................................................... 30  
5.9 Nagle ............................................................. 30  
5.10 Pauling ........................................................... 30  
5.11 Sanderson ....................................................... 31  
5.12 Bibliography .................................................. 31  
6 API Reference ..................................................... 33  
6.1 Accessing data ................................................ 33  
6.2 Classes ......................................................... 35  
7 License ............................................................ 45
8 mendeleev Changelog

8.1 Latest ................................................................. 47
8.2 v0.4.0 (22-11-2017) ................................................. 47
8.3 v0.3.6 (17-09-2017) ................................................ 47
8.4 v0.3.5 (07-09-2017) ................................................ 47
8.5 v0.3.4 (28-06-2017) ................................................ 48
8.6 v0.3.3 (16-05-2017) ................................................ 48
8.7 v0.3.2 (01-05-2017) ................................................ 48
8.8 v0.3.1 (25-01-2017) ................................................ 48
8.9 v0.3.0 (09-01-2017) ................................................ 48
8.10 v0.2.17 (08-01-2017) .............................................. 49
8.11 v0.2.16 (06-01-2017) .............................................. 49
8.12 v0.2.15 (02-01-2017) .............................................. 49
8.13 v0.2.14 (02-01-2017) .............................................. 49
8.14 v0.2.13 (01-01-2017) .............................................. 49
8.15 v0.2.12 (21-12-2016) .............................................. 49
8.16 v0.2.11 (10-11-2016) .............................................. 49
8.17 v0.2.10 (18-10-2016) .............................................. 49
8.18 v0.2.9 (16-10-2016) .............................................. 50
8.19 v0.2.8 (29-08-2016) .............................................. 50
8.20 v0.2.7 (02-04-2016) .............................................. 50
8.21 v0.2.6 (02-04-2016) .............................................. 50
8.22 v0.2.5 (02-04-2016) .............................................. 50
8.23 v0.2.4 (05-02-2016) .............................................. 50
8.24 v0.2.3 (27-01-2016) .............................................. 51
8.25 v0.2.2 (29-11-2015) .............................................. 51
8.26 v0.2.1 (26-10-2015) .............................................. 51
8.27 v0.2.0 (22-10-2015) .............................................. 52
8.28 v0.1.0 (11-07-2015) .............................................. 52

9 Indices and tables ...................................................... 53

Bibliography ............................................................... 55
This package provides an API for accessing various properties of elements from the periodic table of elements.
CHAPTER 1

Getting started

1.1 Overview

This package provides an API for accessing various properties of elements from the periodic table of elements.
The repository is hosted on bitbucket.

1.2 Contributing

All contributions are welcome!
If you would like to suggest an improvement or report a bug or data inconsistency please consider creating an issue on bitbucket. I would be especially grateful for references to possible data updates and sources and recommendations of new data.

1.3 Citing

If you use mendeleev in a scientific publication, please consider citing the software as

L. M. Mentel, mendeleev - A Python resource for properties of chemical elements, ions and isotopes. , 2014–.
Available at: https://bitbucket.org/lukaszmentel/mendeleev.

Here’s the reference in the BibLaTeX format

```latex
@software{mendeleev2014,
  author = {Mentel, Łukasz},
  title = {mendeleev} -- A Python resource for properties of chemical elements, ions and isotopes},
```
1.4 Related projects

periodictable This package provides a periodic table of the elements with support for mass, density and xray/neutron scattering information.

periodic Periodic is an open source simple python API/command line script for the periodic table.

1.5 Funding

This project is supported by the RCN (The Research Council of Norway) project number 239193.
CHAPTER 2

Installation

The package can be installed using pip

```
pip install mendeleev
```

You can also install the most recent version from the repository:

```
pip install https://bitbucket.org/lukaszmentel/mendeleev/get/tip.tar.gz
```

If you use conda you can install the package from my anaconda channel by

```
conda install -c lmmentel mendeleev=0.4.3
```
 CHAPTER 3

Tutorials

3.1 mendeleev tutorial

This simple tutorial will illustrate the basic capabilities of the package.

3.1.1 Basic interactive usage

Getting single elements

The simplest way of accessing the elements is importing them directly from mendeleev by symbols

In [6]: from mendeleev import Si, Fe, O
    print("Si's name: ", Si.name)
    print("Fe's atomic number: ", Fe.atomic_number)
    print("O's atomic weight: ", O.atomic_weight)

Si's name: Silicon
Fe's atomic number: 26
O's atomic weight: 15.999

An alternative interface to the data is through the element function that returns a single Element object or a list of Element object depending on the arguments.

The function can be imported directly from the mendeleev package

In [7]: from mendeleev import element

The element method accepts unique identifiers: atomic number, atomic symbol or element’s name in English. To retrieve the entries on Silicon by symbol type

In [8]: si = element('Si')

In [9]: si

Out[9]: Element(
    abundance_crust=282000.0,
    abundance_sea=2.2,
Metalloid element belonging to group 14 of the periodic table. It is the second most abundant element in the earth's crust. Chemically less reactive than carbon. First identified by Lavoisier in 1787 and first isolated in 1823 by Berzelius.
Similarly to access the data by atomic number or element names type

```python
In [10]: al = element(13)
print(al.name)
```

Aluminum

```python
In [11]: o = element('Oxygen')
print(o.atomic_number)
```

8

### Getting list of elements

The `element` method also accepts list or tuple of identifiers and then returns a list of `Element` objects

```python
In [12]: c, h, o = element(['C', 'Hydrogen', 8])
print(c.name, h.name, o.name)
```

Carbon Hydrogen Oxygen

#### 3.1.2 Extended attributes

Next to simple attributes returning `str`, `int` or `float`, there are extended attributes

- `oxistates`, returns a list of oxidation states
• **ionenergies**, returns a dictionary of ionization energies
• **isotopes**, returns a list of **Isotope** objects
• **ionic_radii** returns a list of **IonicRadius** objects
• **ec**, electronic configuration object

**Oxidation states**

**oxistates** returns a list of most common oxidation states for a given element

```python
In [13]: fe = element('Fe')
    print(fe.oxistates)
[6, 3, 2, 0, -2]
```

**Ionization energies**

The **ionenergies** returns a dictionary with ionization energies in eV as values and degrees of ionization as keys

```python
In [14]: o = element('O')
o.ionenergies
Out[14]: {1: 13.618054,
3: 35.12111,
4: 54.93554,
5: 77.4135,
6: 113.8989,
7: 138.1189,
8: 739.32679,
9: 871.40985}
```

**Isotopes**

The **isotopes** attribute returns a list of **Isotope** objects with the following attributes per isotope

• abundance
• atomic_number
• half_life
• half_life_unit
• is_radioactive
• mass
• mass_number
• mass_uncertainty

```python
In [15]: print("{0:<4s} {1:<4s} {2:<10s} {3:8s} {4:6s} {5:5s}
\n{6}").format("AN", "MN", "Mass", "Unc.", ", Abu.", ", Rad.

for iso in fe.isotopes:
    print("{0:<4d} {1:<4d} {2:<10.5f} {3:8.2e} {4:6.2f} {5:}").format(iso.atomic_number, iso.mass_number, iso.mass, iso.mass_uncertainty, iso.abundance * 

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>54</td>
<td>53.93961</td>
<td>3.00e-06</td>
<td>5.85</td>
<td>False</td>
</tr>
<tr>
<td>26</td>
<td>56</td>
<td>55.93494</td>
<td>3.00e-06</td>
<td>91.75</td>
<td>False</td>
</tr>
</tbody>
</table>
```
Ionic radii

Another composite attribute is ionic_radii which returns a list of IonicRadius object with the following attributes

- atomic_number, atomic number of the ion
- charge, charge of the ion
- econf, electronic configuration of the ion
- coordination, coordination type of the ion
- spin, spin state of the ion (HS or LS)
- crystal_radius, crystal radius in pm
- ionic_radius, ionic radius in pm
- origin, source of the data
- most_reliable, recommended value, (see the original paper for more information)

In [16]: for ir in fe.ionic_radii:
   print(ir)
charge= 2, coordination=IV , crystal_radius=77.000, ionic_radius=63.000
charge= 2, coordination=IVSQ , crystal_radius=78.000, ionic_radius=64.000
charge= 2, coordination=VI , crystal_radius=75.000, ionic_radius=61.000
charge= 2, coordination=VI , crystal_radius=92.000, ionic_radius=78.000
charge= 2, coordination=VIII , crystal_radius=106.000, ionic_radius=92.000
charge= 3, coordination=IV , crystal_radius=63.000, ionic_radius=49.000
charge= 3, coordination=V , crystal_radius=72.000, ionic_radius=58.000
charge= 3, coordination=VI , crystal_radius=69.000, ionic_radius=55.000
charge= 3, coordination=VI , crystal_radius=78.500, ionic_radius=64.500
charge= 3, coordination=VIII , crystal_radius=92.000, ionic_radius=78.000
charge= 4, coordination=VI , crystal_radius=72.500, ionic_radius=58.500
charge= 6, coordination=IV , crystal_radius=39.000, ionic_radius=25.000

Electronic configuration

ec attribute is an object from the ElectronicConfiguration class that has additional method for manipulating the configuration. Internally the configuration is represented as a OrderedDict from the collections module where tuples (n, s) (n is the principal quantum number and s is the subshell label) are used as keys and shell occupations are the values

In [17]: si.ec.conf
Out[17]: OrderedDict([(1, 's'), 2),
   ((2, 's'), 2),
   ((2, 'p'), 6),
   ((3, 's'), 2),
   ((3, 'p'), 2)])

the occupation of different subshells can be access supplying a proper key

In [18]: si.ec.conf[(1, 's')]
Out[18]: 2
to calculate the number of electrons per shell type

```
In [19]: si.ec.electrons_per_shell()
Out[19]: {'K': 2, 'L': 8, 'M': 4}
```

get the largest value of the principal quantum number

```
In [20]: si.ec.max_n()
Out[20]: 3
```

**Some useful functions**

Next to stored attributes there is a number of useful functions

```
In [21]: si = element('Si')
In [22]: # get the number of valence electrons
   ...: si.nvalence()
Out[22]: 4
In [23]: # calculate softness for an ion
   ...: si.softness(charge=2)
Out[23]: 0.058318712346158874
In [24]: # calculate hardness for an ion
   ...: si.hardness(charge=4)
Out[24]: 60.812605
In [25]: # calculate Mulliken electronegativity for a neutral atom or ion
   ...: si.en_mulliken(charge=1)
Out[25]: 8.1729225
In [26]: # calculate the effective nuclear charge for a subshell using Slater's rules
   ...: si.zeff(n=3, o='s')
Out[26]: 4.149999999999999
In [27]: # calculate the effective nuclear charge for a subshell using Clemneti's and Raimondi's exponents
   ...: si.zeff(n=3, o='s', method='clementi')
Out[27]: 4.9032
```

**Electronegativity**

Currently there are 9 electronegativity scales implemented that can be accessed though the common electronegativity method, the scales are:

- allen
- allred-rochow
- cottrell-sutton
- gordy
- li-xue
- mulliken
- nagle
More information can be found in the documentation.

In [28]: si.electronegativity(scale='pauling')
Out[28]: 1.9
In [29]: si.electronegativity(scale='allen')
Out[29]: 11.33

3.1.3 CLI utility

For those who work in the terminal there is a simple command line interface (CLI) for printing the information about a given element. The script name is element.py and it accepts either the symbol or name of the element as an argument and prints the data about it. For example, to print the properties of silicon type

In [31]: !element.py Si

Description
============
Metalloid element belonging to group 14 of the periodic table. It is the second most abundant element in the Earth's crust, making up 25.7% of it by weight. Chemically less reactive than carbon. First identified by Lavoisier in 1787 and first isolated in 1823 by Berzelius.

Properties
==========

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abundance crust</td>
<td>282000</td>
</tr>
<tr>
<td>Abundance sea</td>
<td>2.2</td>
</tr>
<tr>
<td>Annotation</td>
<td></td>
</tr>
<tr>
<td>Atomic number</td>
<td>14</td>
</tr>
<tr>
<td>Atomic radius</td>
<td>132</td>
</tr>
<tr>
<td>Atomic radius rahm</td>
<td>232</td>
</tr>
<tr>
<td>Atomic volume</td>
<td>12.1</td>
</tr>
<tr>
<td>Atomic weight</td>
<td>28.085</td>
</tr>
<tr>
<td>Atomic weight uncertainty</td>
<td>NaN</td>
</tr>
<tr>
<td>Block</td>
<td>p</td>
</tr>
<tr>
<td>Boiling point</td>
<td>2628</td>
</tr>
<tr>
<td>C6</td>
<td>305</td>
</tr>
<tr>
<td>C6 gb</td>
<td>308</td>
</tr>
<tr>
<td>Cas</td>
<td>7440-21-3</td>
</tr>
<tr>
<td>Covalent radius bragg</td>
<td>117</td>
</tr>
<tr>
<td>Covalent radius cordero</td>
<td>111</td>
</tr>
</tbody>
</table>
Covalent radius pyykko | 116
Covalent radius pyykko double | 107
Covalent radius pyykko triple | 102
Covalent radius slater | 110
Cpk color | #daa520
Density | 2.33
Dipole polarizability | 37.31
Discoverers | Jöns Berzelius
Discovery location | Sweden
Discovery year | 1824
Electron affinity | 1.38952
Electronic configuration | [Ne] 3s2 3p2
En allen | 11.33
En ghosh | 0.178503
En pauling | 1.9
Evaporation heat | 383
Fusion heat | 50.6
Gas basicity | 814.1
Geochemical class | major
Goldschmidt class | litophile
Group id | 14
Heat of formation | 450
Is monoisotopic | None
Is radioactive | False
Jmol color | #f0c8a0
Lattice constant | 5.43
Lattice structure | DIA
Melting point | 1683
Metallic radius | 117
Metallic radius cl2 | 138
Molcas gv color | #f0c8a0
Name | Silicon
Name origin | Latin: silex, silicus, (flint).
Period | 3
Proton affinity | 837
Series id | 5
Sources | Makes up major portion of clay, granite, quartz...
Specific heat | 0.703
Symbol | Si
Thermal conductivity | 149
Uses | Used in glass as silicon dioxide (SiO2). Silic...
Vdw radius | 210
Vdw radius alvarez | 219
Vdw radius batsanov | 210
Vdw radius bondi | 210
Vdw radius dreiding | 427
Vdw radius mm3 | 229
Vdw radius rt | NaN
Vdw radius truhlar | NaN
Vdw radius uff | 429.5

In [33]: %version_information mendeleev, sqlalchemy
3.2 Getting tables from the database

This short tutorial explains how to retrieve full tables from the database into pandas DataFrames.

3.2.1 The following tables are available from mendeleev

- elements
- ionicradii
- ionizationenergies
- oxidationstates
- groups
- series
- isotopes

mendeleev provides a convenient function `get_table` to perform the task at hand. The function can be directly imported from mendeleev

```python
In [4]: from mendeleev import get_table
```

To retrieve a table call the `get_table` with the table name as argument. Here we’ll get probably the most important table `elements` with basis data on each element

```python
In [5]: ptable = get_table('elements')
```

Now we can use pandas’ capabilities to work with the data.

```python
In [6]: ptable.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 118 entries, 0 to 117
Data columns (total 67 columns):
annotation 118 non-null object
atomic_number 118 non-null int64
atomic_radius 88 non-null float64
atomic_volume 91 non-null float64
block 118 non-null object
boiling_point 96 non-null float64
density 95 non-null float64
description 109 non-null object
dipole_polarizability 106 non-null float64
electron_affinity 77 non-null float64
electronic_configuration 118 non-null object
evaporation_heat 88 non-null float64
fusion_heat 75 non-null float64
group_id 90 non-null float64
```
For clarity let’s take only a subset of columns

```
In [7]: cols = ['atomic_number', 'symbol', 'atomic_radius', 'en_pauling', 'block', 'vdw_radius_mm3']
```
In [8]: ptable[cols].head()

Out[8]: atomic_number symbol atomic_radius en_pauling block vdw_radius_mm3
0 1 H 79.0 2.20 s 162.0
1 2 He NaN NaN s 153.0
2 3 Li 155.0 0.98 s 255.0
3 4 Be 112.0 1.57 s 223.0
4 5 B 98.0 2.04 p 215.0

It is quite easy now to get descriptive statistics on the data.

In [9]: ptable[cols].describe()

Out[9]:

<table>
<thead>
<tr>
<th></th>
<th>count</th>
<th>atomic_radius</th>
<th>en_pauling</th>
<th>vdw_radius_mm3</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean</td>
<td>59.500000</td>
<td>169.397727</td>
<td>1.748588</td>
<td>248.468085</td>
</tr>
<tr>
<td>std</td>
<td>34.207699</td>
<td>49.810108</td>
<td>0.634442</td>
<td>36.017828</td>
</tr>
<tr>
<td>min</td>
<td>1.000000</td>
<td>79.000000</td>
<td>0.700000</td>
<td>153.000000</td>
</tr>
<tr>
<td>25%</td>
<td>30.250000</td>
<td>137.000000</td>
<td>1.240000</td>
<td>229.000000</td>
</tr>
<tr>
<td>50%</td>
<td>59.500000</td>
<td>160.000000</td>
<td>1.700000</td>
<td>244.000000</td>
</tr>
<tr>
<td>75%</td>
<td>88.750000</td>
<td>181.000000</td>
<td>2.160000</td>
<td>269.250000</td>
</tr>
<tr>
<td>max</td>
<td>118.000000</td>
<td>299.000000</td>
<td>3.980000</td>
<td>364.000000</td>
</tr>
</tbody>
</table>

3.2.2 Isotopes table

Let try and retrieve another table, namely isotopes

In [10]: isotopes = get_table('isotopes', index_col='id')

In [11]: isotopes.info()

<class 'pandas.core.frame.DataFrame'>
 Int64Index: 406 entries, 1 to 406
Data columns (total 11 columns):
 atomic_number 406 non-null int64
 mass 377 non-null float64
 abundance 288 non-null float64
 mass_number 406 non-null int64
 mass_uncertainty 377 non-null float64
 is_radioactive 406 non-null bool
 half_life 121 non-null float64
 half_life_unit 85 non-null object
 spin 323 non-null float64
 g_factor 323 non-null float64
 quadrupole_moment 320 non-null float64
dtypes: bool(1), float64(7), int64(2), object(1)
memory usage: 35.3+ KB

Merge the elements table with the isotopes

We can now perform SQL-like merge operation on two DataFrames and produce an outer join

In [12]: import pandas as pd

In [13]: merged = pd.merge(ptable[cols], isotopes, how='outer', on='atomic_number')

now we have the following columns in the merged DataFrame

In [14]: merged.info()
To display all the isotopes of Silicon

```python
In [16]: merged[merged['symbol'] == 'Si']
```

```
Out[16]: atomic_number symbol atomic_radius en_pauling block vdw_radius_mm3 \\
28 14 Si 132.0 1.9 p 229.0
29 14 Si 132.0 1.9 p 229.0
30 14 Si 132.0 1.9 p 229.0

mass abundance mass_number mass_uncertainty is_radioactive \\
28 27.976927 0.92191 28 3.000000e-09 False
29 28.976495 0.04699 29 3.000000e-09 False
30 29.973770 0.03110 30 2.000000e-08 False

half_life half_life_unit spin g_factor quadrupole_moment \\
28 NaN None 0.0 0.00000 0.0
29 NaN None 0.0 0.00000 0.0
30 NaN None 0.0 0.00000 0.0
```
In [17]: %version_information mendeleev, sqlalchemy, pandas

<table>
<thead>
<tr>
<th>Software</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
<td>3.6.3 64bit [GCC 7.2.0]</td>
</tr>
<tr>
<td>IPython</td>
<td>6.2.1</td>
</tr>
<tr>
<td>OS</td>
<td>Linux 4.9.0 4 amd64 x86_64 with debian 9.1</td>
</tr>
<tr>
<td>mendeleev</td>
<td>0.3.6</td>
</tr>
<tr>
<td>sqlalchemy</td>
<td>1.1.13</td>
</tr>
<tr>
<td>pandas</td>
<td>0.20.3</td>
</tr>
<tr>
<td></td>
<td>Wed Nov 01 15:15:51 2017 CET</td>
</tr>
</tbody>
</table>

### 3.3 Jupyter notebooks

A series of short tutorials is available as Jupyter notebooks that present the main functionality and provide real-life examples

- Introduction
- Gettin tables
- Plotting periodic tables
### 4.1 Elements

The following data are currently available:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
<th>Unit</th>
<th>Data Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>abundance_crust</td>
<td>float</td>
<td>Abundance in the Earth’s crust</td>
<td>mg/kg</td>
<td>[21]</td>
</tr>
<tr>
<td>abundance_sea</td>
<td>float</td>
<td>Abundance in the seas</td>
<td>mg/L</td>
<td>[21]</td>
</tr>
<tr>
<td>annotation</td>
<td>str</td>
<td>Annotations regarding the data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>atomic_number</td>
<td>int</td>
<td>Atomic number</td>
<td></td>
<td></td>
</tr>
<tr>
<td>atomic_radius</td>
<td>float</td>
<td>Atomic radius</td>
<td>pm</td>
<td></td>
</tr>
<tr>
<td>atomic_radius_rahm</td>
<td>float</td>
<td>Atomic radius by Rahm et al.</td>
<td>pm</td>
<td>[38]</td>
</tr>
<tr>
<td>atomic_volume</td>
<td>float</td>
<td>Atomic volume</td>
<td>cm³/mol</td>
<td></td>
</tr>
<tr>
<td>atomic_weight</td>
<td>float</td>
<td>Atomic weight</td>
<td></td>
<td>[30][55]</td>
</tr>
<tr>
<td>atomic_weight_uncertainty</td>
<td>float</td>
<td>Atomic weight uncertainty¹</td>
<td></td>
<td>[30][55]</td>
</tr>
<tr>
<td>block</td>
<td>int</td>
<td>Block in periodic table</td>
<td></td>
<td></td>
</tr>
<tr>
<td>boiling_point</td>
<td>float</td>
<td>Boiling temperature</td>
<td>K</td>
<td></td>
</tr>
<tr>
<td>c6</td>
<td>float</td>
<td>C_6 dispersion coefficient in a.u.</td>
<td>a.u.</td>
<td>[13][47]</td>
</tr>
<tr>
<td>c6_gb</td>
<td>float</td>
<td>C_6 dispersion coefficient in a.u. (Gould &amp; Bučko)</td>
<td>a.u.</td>
<td>[20]</td>
</tr>
<tr>
<td>cas</td>
<td>str</td>
<td>Chemical Abstracts Service identifier</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Continued on next page
Table 1 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
<th>Unit</th>
<th>Data Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>covalent_radius_bragg</td>
<td>float</td>
<td>Covalent radius by Bragg</td>
<td>pm</td>
<td>[10]</td>
</tr>
<tr>
<td>covalent_radius_cordero</td>
<td>float</td>
<td>Covalent radius by Cerdero et al.²</td>
<td>pm</td>
<td>[16]</td>
</tr>
<tr>
<td>covalent_radius_pyykko</td>
<td>float</td>
<td>Single bond covalent radius by Pyykko et al.</td>
<td>pm</td>
<td>[36]</td>
</tr>
<tr>
<td>covalent_radius_pyykko_double</td>
<td>float</td>
<td>Double bond covalent radius by Pyykko et al.</td>
<td>pm</td>
<td>[35]</td>
</tr>
<tr>
<td>covalent_radius_pyykko_triple</td>
<td>float</td>
<td>Triple bond covalent radius by Pyykko et al.</td>
<td>pm</td>
<td>[37]</td>
</tr>
<tr>
<td>covalent_radius_slater</td>
<td>float</td>
<td>Covalent radius by Slater</td>
<td>pm</td>
<td>[44]</td>
</tr>
<tr>
<td>cpk_color</td>
<td>str</td>
<td>Element color in CPK convention</td>
<td>HEX</td>
<td>[52]</td>
</tr>
<tr>
<td>density</td>
<td>float</td>
<td>Density at 295K</td>
<td>g/cm³</td>
<td></td>
</tr>
<tr>
<td>description</td>
<td>str</td>
<td>Short description of the element</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dipole_polarizability</td>
<td>float</td>
<td>Dipole polarizability</td>
<td>a.u.</td>
<td>[58]</td>
</tr>
<tr>
<td>discoverers</td>
<td>str</td>
<td>The discoverers of the element</td>
<td></td>
<td></td>
</tr>
<tr>
<td>discovery_location</td>
<td>str</td>
<td>The location where the element was discovered</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dipole_year</td>
<td>int</td>
<td>The year the element was discovered</td>
<td></td>
<td></td>
</tr>
<tr>
<td>electron_affinity</td>
<td>float</td>
<td>Electron affinity²</td>
<td>eV</td>
<td>[21][6]</td>
</tr>
<tr>
<td>electrons</td>
<td>int</td>
<td>Number of electrons</td>
<td></td>
<td></td>
</tr>
<tr>
<td>en_allen</td>
<td>float</td>
<td>Allen’s scale of electronegativity⁴</td>
<td>eV</td>
<td>[26][27]</td>
</tr>
<tr>
<td>en_ghosh</td>
<td>float</td>
<td>Ghosh’s scale of electronegativity</td>
<td></td>
<td>[18]</td>
</tr>
<tr>
<td>en_mulliken</td>
<td>float</td>
<td>Mulliken’s scale of electronegativity</td>
<td>eV</td>
<td>[31]</td>
</tr>
<tr>
<td>en_pauling</td>
<td>float</td>
<td>Pauling’s scale of electronegativity</td>
<td></td>
<td>[21]</td>
</tr>
<tr>
<td>econf</td>
<td>str</td>
<td>Ground state electron configuration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>evaporation_heat</td>
<td>float</td>
<td>Evaporation heat</td>
<td>kJ/mol</td>
<td></td>
</tr>
<tr>
<td>fusion_heat</td>
<td>float</td>
<td>Fusion heat</td>
<td>kJ/mol</td>
<td></td>
</tr>
<tr>
<td>gas_basicity</td>
<td>float</td>
<td>Gas basicity</td>
<td>kJ/mol</td>
<td>[21]</td>
</tr>
<tr>
<td>geochemical_class</td>
<td>str</td>
<td>Geochemical classification</td>
<td></td>
<td>[50]</td>
</tr>
<tr>
<td>goldschmidt_class</td>
<td>str</td>
<td>Goldschmidt classification</td>
<td></td>
<td>[50][51]</td>
</tr>
</tbody>
</table>

Continued on next page
Table 1 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
<th>Unit</th>
<th>Data Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>group</td>
<td>int</td>
<td>Group in periodic table</td>
<td></td>
<td></td>
</tr>
<tr>
<td>heat_of Formation</td>
<td>float</td>
<td>Heat of formation</td>
<td>kJ/mol</td>
<td>[21]</td>
</tr>
<tr>
<td>ionenergy</td>
<td>tuple</td>
<td>Ionization energies</td>
<td>eV</td>
<td>[22]</td>
</tr>
<tr>
<td>ionic_radii</td>
<td>list</td>
<td>Ionic and crystal radii in pm</td>
<td>pm</td>
<td>[43]</td>
</tr>
<tr>
<td>is_monoisotopic</td>
<td>bool</td>
<td>Is the element monoisotopic</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_radioactive</td>
<td>bool</td>
<td>Is the element radioactive</td>
<td></td>
<td></td>
</tr>
<tr>
<td>isotopes</td>
<td>list</td>
<td>Isotopes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>jmol_color</td>
<td>str</td>
<td>Element color in Jmol convention</td>
<td>HEX</td>
<td>[56]</td>
</tr>
<tr>
<td>lattice_constant</td>
<td>float</td>
<td>Lattice constant</td>
<td>Angstrom</td>
<td></td>
</tr>
<tr>
<td>lattice_structure</td>
<td>str</td>
<td>Lattice structure code</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mass_number</td>
<td>int</td>
<td>Mass number (most abundant isotope)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>melting_point</td>
<td>float</td>
<td>Melting temperature</td>
<td>K</td>
<td></td>
</tr>
<tr>
<td>mendeleev_number</td>
<td>int</td>
<td>Mendeleev’s number</td>
<td></td>
<td>[34][48]</td>
</tr>
<tr>
<td>metallic_radius</td>
<td>float</td>
<td>Single-bond metallic radius</td>
<td>pm</td>
<td>[1]</td>
</tr>
<tr>
<td>metallic_radius_c12</td>
<td>float</td>
<td>Metallic radius with 12 nearest neigh-</td>
<td>pm</td>
<td>[1]</td>
</tr>
<tr>
<td>molcas_gv_color</td>
<td>str</td>
<td>Element color in MOCAS GV convention</td>
<td>HEX</td>
<td>[57]</td>
</tr>
<tr>
<td>name</td>
<td>str</td>
<td>Name in English</td>
<td></td>
<td></td>
</tr>
<tr>
<td>name_origin</td>
<td>str</td>
<td>Origin of the name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>neutrons</td>
<td>int</td>
<td>Number of neutrons (most abundant iso-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>oxistates</td>
<td>list</td>
<td>Oxidation states</td>
<td></td>
<td></td>
</tr>
<tr>
<td>period</td>
<td>int</td>
<td>Period in periodic table</td>
<td></td>
<td></td>
</tr>
<tr>
<td>proton_affinity</td>
<td>float</td>
<td>Proton affinity</td>
<td>kJ/mol</td>
<td>[21]</td>
</tr>
<tr>
<td>protons</td>
<td>int</td>
<td>Number of protons</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sconst</td>
<td>float</td>
<td>Nuclear charge screening constants⁶</td>
<td></td>
<td>[14][15]</td>
</tr>
<tr>
<td>series</td>
<td>int</td>
<td>Index to chemical series</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sources</td>
<td>str</td>
<td>Sources of the element</td>
<td></td>
<td></td>
</tr>
<tr>
<td>specific_heat</td>
<td>float</td>
<td>Specific heat @ 20°C</td>
<td>J/(g mol)</td>
<td></td>
</tr>
<tr>
<td>symbol</td>
<td>str</td>
<td>Chemical symbol</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
<th>Unit</th>
<th>Data Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>thermal_conductivity</td>
<td>float</td>
<td>Thermal conductivity @ 25°C</td>
<td>W/(m K)</td>
<td></td>
</tr>
<tr>
<td>uses</td>
<td>str</td>
<td>Applications of the element</td>
<td></td>
<td></td>
</tr>
<tr>
<td>vdw_radius</td>
<td>float</td>
<td>Van der Waals radius</td>
<td>pm</td>
<td>[21]</td>
</tr>
<tr>
<td>vdw_radius_alvarez</td>
<td>float</td>
<td>Van der Waals radius according to Alvarez</td>
<td>pm</td>
<td>[5][49]</td>
</tr>
<tr>
<td>vdw_radius_batsanov</td>
<td>float</td>
<td>Van der Waals radius according to Batsanov</td>
<td>pm</td>
<td>[8]</td>
</tr>
<tr>
<td>vdw_radius_bondi</td>
<td>float</td>
<td>Van der Waals radius according to Bondi</td>
<td>pm</td>
<td>[9]</td>
</tr>
<tr>
<td>vdw_radius_dreiding</td>
<td>float</td>
<td>Van der Waals radius from the DREIDING FF</td>
<td>pm</td>
<td>[29]</td>
</tr>
<tr>
<td>vdw_radius_mm3</td>
<td>float</td>
<td>Van der Waals radius from the MM3 FF</td>
<td>pm</td>
<td>[3]</td>
</tr>
<tr>
<td>vdw_radius_rt</td>
<td>float</td>
<td>Van der Waals radius according to Rowland and Taylor</td>
<td>pm</td>
<td>[40]</td>
</tr>
<tr>
<td>vdw_radius_truhlar</td>
<td>float</td>
<td>Van der Waals radius according to Truhlar</td>
<td>pm</td>
<td>[28]</td>
</tr>
<tr>
<td>vdw_radius_uff</td>
<td>float</td>
<td>Van der Waals radius from the UFF</td>
<td>pm</td>
<td>[39]</td>
</tr>
</tbody>
</table>

1 Atomic Weights
Atomic weights and their uncertainties were retrieved mainly from ref. [55]. For elements whose values were given as ranges the conventional atomic weights from Table 3 in ref. [30] were taken. For radioactive elements the standard approach was adopted where the weight is taken as the mass number of the most stable isotope. The data was obtained from CIAAW page on radioactive elements. In cases where two isotopes were specified the one with the smaller standard deviation was chosen. In case of Tc and Pm relative weights of their isotopes were used, for Tc isotope 98, and for Pm isotope 145 were taken from CIAAW.

2 Covalent Radius by Cordero et al.
In order to have a more homogeneous data for covalent radii taken from ref. [16] the values for 3 different valences for C, also the low and high spin values for Mn, Fe Co, were respectively averaged.

3 Electron affinity
Electron affinities were taken from [21] for the elements for which the data was available. For He, Be, N, Ar and Xe affinities were taken from [6] where they were specified for metastable ions and therefore the values are negative.

Updates
- Electron affinity of niobium was taken from [25].
- Electron affinity of cobalt was taken from [11].
- Electron affinity of lead was taken from [12].

4 Allen's configuration energies
The values of configurational energies from refs. [26] and [27] were taken as reported in eV without converting to Pauling units.

5 Mendeleev numbers
Mendeleev numbers were mostly taken from [48] but the range was extended to cover the whole periodic table following the prescription in the article of increasing the numbers going from top to bottom in each group and group by group from left to right in the periodic table.

6 Nuclear charge screening constants
The screening constants were calculated according to the following formula

\[ \sigma_{n,l,m} = Z - n \cdot \zeta_{n,l,m} \]
4.2 Isotopes

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
<th>Unit</th>
<th>Data Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>abundance</td>
<td>float</td>
<td>Relative Abundance</td>
<td></td>
<td>[54]</td>
</tr>
<tr>
<td>g_factor</td>
<td>float</td>
<td>Nuclear g-factor(^8)</td>
<td></td>
<td>[46]</td>
</tr>
<tr>
<td>half_life</td>
<td>float</td>
<td>Half life of the isotope</td>
<td></td>
<td>[30]</td>
</tr>
<tr>
<td>half_life_unit</td>
<td>str</td>
<td>Unit in which the half life is given</td>
<td></td>
<td>[30]</td>
</tr>
<tr>
<td>is_radioactive</td>
<td>bool</td>
<td>Is the isotope radioactive</td>
<td></td>
<td>[53]</td>
</tr>
<tr>
<td>mass</td>
<td>float</td>
<td>Atomic mass Da</td>
<td></td>
<td>[53]</td>
</tr>
<tr>
<td>mass_number</td>
<td>int</td>
<td>Mass number of the isotope</td>
<td></td>
<td>[53]</td>
</tr>
<tr>
<td>mass_uncertainty</td>
<td>float</td>
<td>Uncertainty of the atomic mass</td>
<td></td>
<td>[53]</td>
</tr>
<tr>
<td>spin</td>
<td>float</td>
<td>Nuclear spin quantum number</td>
<td></td>
<td></td>
</tr>
<tr>
<td>quadrupole_moment</td>
<td>float</td>
<td>Nuclear electric quadrupole moment(^8)</td>
<td>b [100 fm^2]</td>
<td>[45]</td>
</tr>
</tbody>
</table>

Data Footnotes

where \(n\) is the principal quantum number, \(Z\) is the atomic number, \(\sigma_{n,l,m}\) is the screening constant, \(\zeta_{n,l,m}\) is the optimized exponent from [14][15].

For elements Nb, Mo, Ru, Rh, Pd and Ag the exponent values corresponding to the ground state electronic configuration were taken (entries with superscript \(u\) in Table II in [15]).

For elements La, Pr, Nd and Pm two exponent were reported for 4f shell denoted 4f and 4f’ in [15]. The value corresponding to 4f were used since according to the authors these are the dominant ones.

\(^7\) van der Waals radii according to Alvarez.

The bulk of the radii data was taken from Ref. [5], but the radii for noble gasses were update according to the values in Ref. [49].

\(^8\) Isotope g-factors and quadrupole moments

The data regarding g-factors and electric quadrupole moments was parsed from easyspin webpage (accessed 25.01.2017) where additional notes are mentioned:

- Typo for Rh-103: Moment is factor of 10 too large
- 237Np, 239Pu, 243Am magnetic moment data from [21], section 11-2
- In quadrupole moment data - a typo for Ac-227: sign should be +
CHAPTER 5

Electronegativities

Since electronegativity is a useful concept rather than a physical observable, several scales of electronegativity exist and some of them are available in mendeleev. Depending on the definition of a particular scale the values are either stored directly or recomputed on demand with appropriate formulas. The following scales are stored:

- Allen
- Ghosh
- Pauling

Moreover there are electronegativity scales that can be computed from their respective definition and the atomic properties available in mendeleev:

- Allred-Rochow
- Cottrell-Sutton
- Gordy
- Li and Xue
- Martynov and Batsanov
- Mulliken
- Nagle
- Sanderson

For a short overview on electronegativity see this presentation.

All the examples shown below are for Silicon:

```python
>>> from mendeleev import element
>>> Si = element('Si')
```
5.1 Allen

The electronegativity scale proposed by Allen in ref [2] can be defined as:

\[ \chi_A = \frac{\sum_x n_x \varepsilon_x}{\sum_x n_x} \]

where: \( \varepsilon_x \) is the multiplet-averaged one-electron energy of the subshell \( x \) and \( n_x \) is the number of electrons in subshell \( x \) and the summation runs over the valence shell.

The values that are tabulated were obtained from refs. [26] and [27].

Example:

```python
>>> Si.en_allen
11.33
>>> Si.electronegativity('allen')
11.33
```

5.2 Allred and Rochow

The scale of Allred and Rochow [4] introduces the electronegativity as the electrostatic force exerted on the electron by the nuclear charge:

\[ \chi_{AR} = \frac{e^2 Z_{eff}}{r^2} \]

where: \( Z_{eff} \) is the effective nuclear charge and \( r \) is the covalent radius.

Example:

```python
>>> Si.electronegativity('allred-rochow')
0.00028240190249702736
```

5.3 Cottrell and Sutton

The scale proposed by Cottrell and Sutton [17] is derived from the equation:

\[ \chi_{CS} = \sqrt{\frac{Z_{eff}}{r}} \]

where: \( Z_{eff} \) is the effective nuclear charge and \( r \) is the covalent radius.

Example:

```python
>>> Si.electronegativity('cottrell-sutton')
0.18099342720014772
```

5.4 Ghosh

Ghosh [18] presented a scale of electronegativity based on the absolute radii of atoms computed as

\[ \chi_{GH} = a \cdot (1/R) + b \]
where: $R$ is the absolute atomic radius and $a$ and $b$ are empirical parameters.

Example:

```python
>>> Si.en_ghosh
0.178503
```

### 5.5 Gordy

Gordy’s scale [19] is based on the potential that measures the work necessary to achieve the charge separation, according to:

$$\chi_G = \frac{eZ_{\text{eff}}}{r}$$

where: $Z_{\text{eff}}$ is the effective nuclear charge and $r$ is the covalent radius.

Example:

```python
>>> Si.electronegativity('gordy')
0.03275862068965517
```

### 5.6 Li and Xue

Li and Xue [23][24] proposed a scale that takes into account different valence states and coordination environment of atoms and is calculated according to the following formula:

$$\chi_{LX} = n^* \sqrt{\sum_{j=1}^{n_v} I_j/R_y}$$

where: $n^*$ is the effective principal quantum number, $I_j$ is the $j$’th ionization energy in eV, $R_y$ is the Rydberg constant in eV and $r$ is either the crystal radius or ionic radius.

Example:

```python
>>> Si.en_li_xue(charge=4)
{u'IV': 13.16033405547733, u'VI': 9.748395596649873}
>>> Si.electronegativity('li-xue', charge=4)
{u'IV': 13.16033405547733, u'VI': 9.748395596649873}
```

### 5.7 Martynov and Batsanov

Martynov and Batsanov [7] used the square root of the averaged valence ionization energy as a measure of electronegativity:

$$\chi_{MB} = \sqrt{\frac{1}{n_v} \sum_{k=1}^{n_v} I_k}$$

where: $n_v$ is the number of valence electrons and $I_k$ is the $k$ th ionization potential.

Example:
5.8 Mulliken

Mulliken scale [31] is defined as the arithmetic average of the ionization potential \( IP \) and the electron affinity \( EA \):

\[
\chi_M = \frac{IP + EA}{2}
\]

Example:

```python
>>> Si.en_mulliken()
4.0758415
>>> Si.electronegativity('mulliken')
4.0758415
```

5.9 Nagle

Nagle [32] derived his scale from the atomic dipole polarizability:

\[
\chi_N = \frac{3}{\sqrt{n}} \sqrt[3]{\alpha}
\]

Example:

```python
>>> Si.electronegativity('nagle')
0.47505611644667534
```

5.10 Pauling

Pauling’s thermochemical scale was introduced in [33] as a relative scale based on electronegativity differences:

\[
\chi_A - \chi_B = \sqrt{E_d(AB)} - \frac{1}{2} \left[ E_d(AA) + E_d(BB) \right]
\]

where: \( E_d(XY) \) is the bond dissociation energy of a diatomic \( XY \). The values available in mendeleev are taken from ref. [21].

Example:

```python
>>> Si.en_pauling
1.9
>>> Si.electronegativity('pauling')
1.9
```
5.11 Sanderson

Sanderson [41][42] established his scale of electronegativity based on the stability ratio:

\[ \chi_S = \frac{\rho}{\rho_{ng}} \]

where: \( \rho \) is the average electron density \( \rho = \frac{Z}{4\pi r^3} \), and \( \rho_{ng} \) is the average electron density of a hypothetical noble gas atom with charge \( Z \).

Example:

```python
>>> Si.en_sanderson()
0.3468157872145231
>>> Si.electronegativity()
0.3468157872145231
```

5.12 Bibliography
6.1 Accessing data

6.1.1 Elements

The easiest way to access individual elements is simply by importing them from the `mendeleev` directly using their symbols:

```python
>>> from mendeleev import H, C, O, Og

>>> [x.name for x in [H, C, O, Og]]
['Hydrogen', 'Carbon', 'Oxygen', 'Oganesson']
```

An alternative method of access is through the `element()` function that returns either a single `Element` instance or a tuple of those instances depending on the input. It provides a more flexible interface since it accepts element names, atomic numbers and symbols as well as their combinations.

```python
def element(ids):
    pass
```

Based on the type of the `ids` identifier return either an `Element` object from the database, or a list of `Element` objects if the `ids` is a list or a tuple of identifiers. Valid identifiers for an element are: `name`, `symbol`, and `atomic number`.

**Args:** ids (str): element identifier

**Raises:** ValueError: when the identifier is not a list/tuple, int or str

**Example:** The element can be identified by symbol

```python
>>> from mendeleev import element
>>> si = element('Si')
>>> si.atomic_number
14
```

by the atomic number
>>> al = element(13)
>>> al.name
'Aluminum'

or by the name

>>> o = element('Oxygen')
>>> o.symbol
'O'

Multiple elements can be instantiated simultaneously through a combination of identifiers

>>> c, h, o = element(['C', 'Hydrogen', 8])
>>> print(c.name, h.name, o.name)
Carbon Hydrogen Oxygen

### 6.1.2 Tables

If you want a whole set of data you can retrieve one of the tables from the database as pandas DataFrame through the `get_table`. The following tables are available:

- elements
- groups
- ionicradii
- ionizationenergies
- isotopes
- oxidationstates
- screeningconstants
- series

**get_table**(tablename, **kwargs)
Return a table from the database as pandas DataFrame

**Args:**

- tablename: str  Name of the table from the database
- kwargs: A dictionary of keyword arguments to pass to the `pandas.read_qsl`

**Returns:**

- df: pandas.DataFrame  Pandas DataFrame with the contents of the table

**Example:**

```python
>>> from mendeleev import get_table
>>> df = get_table('elements')
>>> type(df)
pandas.core.frame.DataFrame
```
6.1.3 Database session and engine

For those who want to interact with the database through a layer of SQLAlchemy there are methods for getting the session or the engine:

```python
get_session(dbpath=None)
Return the database session connection.

get_engine(dbpath=None)
Return the db engine
```

6.2 Classes

6.2.1 Element

class Element(**kwargs)
Chemical element.

Attributes:

- `abundance_crust` [float] Abundance in the earth’s crust in mg/kg
- `abundance_sea` [float] Abundance in the seas in mg/L
- `annotation` [str] Annotations regarding the data
- `atomic_number` [int] Atomic number
- `atomic_radius` [float] Atomic radius in pm
- `atomic_radius_rahm` [float] Atomic radius by Rahm et al. in pm
- `atomic_volume` [float] Atomic volume in cm³/mol
- `atomic_weight` [float] Relative atomic weight as the ratio of the average mass of atoms of the element to 1/12 of the mass of an atom of 12C
- `block` [int] Block in periodic table, s, p, d, f
- `boiling_point` [float] Boiling temperature in K
- `cas` [str] Chemical Abstracts Service identifier
- `covalent_radius_bragg` [float] Covalent radius in pm from
- `covalent_radius_pyykko_double` [float] Double bond covalent radius in pm from P. Pyykkö et al.
covalent_radius_pyykko_triple [float] Triple bond covalent radius in pm from P. Pyykkö et al.
covalent_radius_slater [float] Covalent radius in pm from Slater

cpk_color [str] CPK color of the atom in HEX, see http://jmol.sourceforge.net/jscolors/#color_U
density [float] Density at 295K in g/cm3
description [str] Short description of the element
dipole_polarizability [float] Dipole polarizability in atomic units from P. Schwerdtfeger “Table of experimental and calculated static dipole polarizabilities for the electronic ground states of the neutral elements (in atomic units)”, February 11, 2014
discoverers: str The discoverers of the element
discovery_location: str The location where the element was discovered
discovery_year: int The year the element was discovered
electron_affinity [float] Electron affinity in eV
en_allen [float] Allen’s scale of electronegativity (Configurational energy)
en_ghosh [float] Ghosh’s scale of electronegativity
en_pauling [float] Pauling’s scale of electronegativity
econf [str] Ground state electron configuration
evaporation_heat [float] Evaporation heat in kJ/mol
fusion_heat [float] Fusion heat in kJ/mol
gas_basicity [float] Gas basicity
geochemical_class [String] Geochemical classification of the elements
goldschmidt_class [String] Goldschmidt classification of the elements
group [int] Group number
group_id [Group] Group details
heat_of_formation [float] Heat of formation in kJ/mol
is_monoisotopic [bool] A flag marking if the element is monoisotopic
jmol_color [str] Color of the atom as used in Jmol, in HEX, see http://jmol.sourceforge.net/jscolors/#color_U
lattice_constant [float] Lattice constant in ang
lattice_structure [str] Lattice structure code
mass [float] Relative atomic mass. Ratio of the average mass of atoms of the element to 1/12 of the mass of an atom of 12C
mendeleev_number [int] Mendeleev number
melting_point [float] Melting temperature in K
metallic_radius [Float] Single-bond metallic radius or metallic radius, have been calculated by Pauling using interatomic distances and an equation relating such distances with bond number
metallic_radius_c12 [Float] Metallic radius obtained by Pauling with an assumed number of nearest neighbors equal to 12
name [str] Name in English
name_origin: str Origin of the name
period [int] Period in periodic table
proton_affinity [Float] Proton affinity
series [int] Index to chemical series
sources: str Sources of the element
specific_heat [float] Specific heat in J/g mol @ 20 C
symbol [str of length 1 or 2] Chemical symbol
thermal_conductivity [float] Thermal conductivity in @/m K @25 C
uses: str Uses of the element
vdw_radius_bondi [float] Van der Waals radius according to Bondi in pm
vdw_radius_truhlar [float] Van der Waals radius according to Truhlar in pm
vdw_radius_rt [float] Van der Waals radius according to Rowland and Taylor in pm
vdw_radius_batsanov [float] Van der Waals radius according to Batsanov in pm
vdw_radius_dreiding [float] Van der Waals radius from the DREIDING force field in pm
vdw_radius_uff [float] Van der Waals radius from the UFF in pm
vdw_radius_mm3 [float] Van der Waals radius from MM3 in pm
oxistates [list] Oxidation states
calc_en_sanderson (radius='covalent_radius_pyykko')
  Sanderson electronegativity

  \[ \chi = \frac{AD}{AD_{eg}} \]

Args:
  radius [str] Radius to use in the calculation
covalent_radius
  Return the default covalent radius which is covalent_radius_pyykko

electronegativity (scale='pauling', charge=0)
  Calculate the electronegativity using one of the methods

Args:
  scale [str] Name of the electronegativity scale, one of
  • allen
  • allred-rochow
  • cottrell-sutton
  • gordy
electrons
   Return the number of electrons.

en_calc (radius='covalent_radius_pyykko', rpow=1, apow=1, **zeffkwargs)
   Calculate the electronegativity from a general formula
   \[ \chi = \left( \frac{Z_{\text{eff}}}{r^{\beta}} \right)^{\alpha} \]
   where
   • \(Z_{\text{eff}}\) is the effective nuclear charge
   • \(r\) is the covalent radius
   • \(\alpha, \beta\) parameters

en_li_xue (charge=0, radius='crystal_radius')
   Calculate the electronegativity of an atom according to the definition of Li and Xue
   Args:
       charge [int] Charge of the ion
       radius [str] Type of radius to be used in the calculation, either crystal_radius as recommended in the paper or ionic_radius
   Returns:
       out [dict] A dictionary with electronegativities as values and coordination string as keys or tuple of coordination and spin if the ion is LS or HS

en_martynov_batsanov ()
   Calculates the electronegativity value according to Martynov and Batsanov as the average of the ionization energies of the valence electrons
   \[ \chi_{MB} = \sqrt{\frac{1}{n_v} \sum_{k=1}^{n_v} I_k} \]
   where: \(n_v\) is the number of valence electrons and \(I_k\) is the \(k\) th ionization potential.

en_mulliken (charge=0, missingIsZero=False, useNegativeEA=False)
   Return the absolute electronegativity (Mulliken scale), calculated as
   \[ \chi = \frac{I + A}{2} \]
   where \(I\) is the ionization energy and \(A\) is the electron affinity

hardness (charge=0)
   Return the absolute hardness, calculated as
   \[ \eta = \frac{I - A}{2} \]
where I is the ionization energy and A is the electron affinity

Args:

charge: int Charge of the cation for which the hardness will be calculated

init_on_load()
Initialize the ElectronicConfiguration class as attribute of self

ionenergies
Return a dict with ionization degree as keys and ionization energies in eV as values.

mass
Return the atomic weight if defined or mass number otherwise.

mass_number
Return the mass number of the most abundant natural stable isotope

mass_str ()
String representation of atomic weight

neutrons
Return the number of neutrons of the most abundant natural stable isotope.

nvalence (method=None)
Return the number of valence electrons

oxistates
Return the oxidation states as a list of ints

protons
Return the number of protons.

sconst
Return a dict with screening constants with tuples (n, s) as keys and screening constants as values

softness (charge=0)
Return the absolute softness, calculated as

\[
S = \frac{1}{2\eta}
\]

where \(\eta\) is the absolute hardness

Args:

charge: int Charge of the cation for which the hardness will be calculated

zeff (n=None, o=None, method='slater', alle=False)
Return the effective nuclear charge for (n, s)

Args:

method [str]

Method to calculate the screening constant, the choices are

6.2.2 IonicRadius

**class IonicRadius(****kwargs**)**

Effective ionic radii and crystal radii in pm retrieved from\(^1\).

**Attributes:**

- **atomic_number** [int] Atomic number
- **charge** [int] Charge of the ion
- **econf** [str] Electronic configuration of the ion
- **coordination** [str] Type of coordination
- **spin** [str] Spin state: HS - high spin, LS - low spin
- **crystal_radius** [float] Crystal radius in pm
- **ionic_radius** [float] Ionic radius in pm
- **origin** [str] Source of the data
- **most_reliable** [bool] Most reliable value (see reference)

6.2.3 IonizationEnergy

**class IonizationEnergy(****kwargs**)**

Ionization energy of an element

**Attributes:**

- **atomic_number** [int] Atomic number
- **degree** [int] Degree of ionization with respect to neutral atom
- **energy** [float] Ionization energy in eV parsed from [http://physics.nist.gov/cgi-bin/ASD/ie.pl](http://physics.nist.gov/cgi-bin/ASD/ie.pl) on April 13, 2015

6.2.4 Isotope

**class Isotope(****kwargs**)**

Attributes:

- **abundance** [float] Abundance of the isotope
- **atomic_number** [int] Atomic number
- **half_life** [float] Half life time
- **half_life_unit** [str] Unit for the half life time

---
is_radioactive [bool] A flag marking whether the isotope is radioactive
mass [float] Mass of the isotope
mass_number [int] Mass number of the isotope
mass_uncertainty [float] Uncertainty of the mass

6.2.5 ScreeningConstant
class ScreeningConstant(**kwargs)
Attributes:
  atomic_number [int] Atomic number
  n [int] Principal quantum number
  s [str] Subshell label, (s, p, d, …)
  screening [float] Screening constant

6.2.6 Series
class Series(**kwargs)
Name of the series in the periodic table.
Attributes:
  name [str] Name of the series
  color [str] The HEX representation of a color of the series, the colors were obtained from ColorBrewer the qualitative 10-class paired colormap

6.2.7 Group
class Group(**kwargs)
Name of the group in the periodic table.

6.2.8 OxidationState
class OxidationState(**kwargs)
Oxidation states of an element
Attributes:
  atomic_number [int] Atomic number
  oxidation_state [int] Oxidation state
6.2.9 ElectronicConfiguration

class ElectronicConfiguration (conf=None, atomre=None, shellre=None)
    Electronic configuration handler

    atomre
        Regular expression for atomic symbols

    conf
        Return the configuration

    electrons_per_shell()
        Return number of electrons per shell as dict

    get_largest_core()
        Find the largest noble gas core possible for the current configuration and return the symbol of the corresponding noble gas element.

    get_valence()
        Find the valence configuration i.e. remove the largest noble gas core from the current configuration and return the result.

    ionize(n=1)
        Remove n electrons from and return a new ElectronicConfiguration object

    last_subshell(wrt='order')
        Return the valence shell

    max_l(n)
        Return the largest value of azimuthal quantum number for a given value of principal quantum number

        Args:
            n [int] Principal quantum number

    max_n()
        Return the largest value of principal quantum number for the atom

    ne()
        Return the number of electrons

    nvalence(block, method=None)
        Return the number of valence electrons

    parse(string)
        Parse a string with electronic configuration into an OrderedDict representation

    shell2int()
        configuration as list of tuples (n, l, e)

    shellre
        Regular expression for the shell

    slater_screening(n, o, alle=False)

        Args:
            n [int] Principal quantum number
            o [str] orbital label, (s, p, d, . . .)
            alle [bool] Use all the valence electrons, i.e. calculate screening for an extra electron
sort \texttt{(}\textit{inplace=True}\texttt{)}
Sort the occupations OD

spin\_occupations()
For each subshell calculate the number of \textit{alpha}, \textit{beta} electrons, electron pairs and unpaired electrons

spin\_only\_magnetic\_moment()
Return the magnetic moment including only spin of the electrons and not the angular momentum

to\_str()
Return a string with the configuration

unpaired\_electrons()
Return the number of unpaired electrons
The MIT License (MIT)

Copyright (c) 2015 Lukasz Mentel

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
8.1 Latest

- Added `mendelev_number` attribute to elements.
- Added footnotes to the data documentation.

8.2 v0.4.0 (22-11-2017)

- The elements can now be directly imported from `mendelev` by symbols.
- Added `sphinxcontrib.bibtex` extension to the docs to handle BibTeX style references to improve handling of the bibliographic entries.
- Added `nbsphinx` to include Jupyter Notebook tutorials in the docs.

8.3 v0.3.6 (17-09-2017)

- Added API documentation
- Corrected the sphinx configuration
- Updated the documentation

8.4 v0.3.5 (07-09-2017)

- Added a module with functions to scrape data from ciaaw.org
- Added new `Element` attributes, `name_origin`, `uses` and `sources`
• Added new `Element` attributes related to the discovery: `discoverers`, `discovery_location`, `discovery_year`

8.5 v0.3.4 (28-06-2017)

• Fixed python2.7 compatibility issue
• Added double and triple bond covalent radii from Pyykko
• Corrected minor error in the documentation
• Replaced lazy loading with eager in db queries

8.6 v0.3.3 (16-05-2017)

• Corrected the coordination of Br5+ ion in the ionic radii table

8.7 v0.3.2 (01-05-2017)

• Added `metallic_radius`
• Added Goldschmidt and geochemical classifications
• Corrected the docs configuration
• Added `cas` number attribute
• Added atomic radii by Rahm et al.
• Created a conda recipe
• Added a citation information to the readme
• Electronic configuration code was split into a separate module

8.8 v0.3.1 (25-01-2017)

• Added new properties of isotopes: `spin`, `g_factor`, `quadrupole_moment`

8.9 v0.3.0 (09-01-2017)

• Updates of the documentation and tutorials
• Added radioactive isotope half-lifes
8.10 v0.2.17 (08-01-2017)

- Extended the schema for isotopes with additional attributes and updated the values of abundancies, half lifes and mass uncertainties.
- Updates to the tutorials and docs.

8.11 v0.2.16 (06-01-2017)

- Corrected the radioactive attribute of Th, Pa and U elements.

8.12 v0.2.15 (02-01-2017)

- Patched the sphinx configuration.

8.13 v0.2.14 (02-01-2017)

- Patched typos in README.

8.14 v0.2.13 (01-01-2017)

- Updated atomic weight with the newest IUPAC and CIAAW recommendations.
- Added `is_radioactive` and `is_monoisotopic` attributes.
- Updated the docs.

8.15 v0.2.12 (21-12-2016)

- Got rid of the scipy dependency.

8.16 v0.2.11 (10-11-2016)

- Updated the names and symbols of elements 113, 115, 117, 118.
- Updated the docs.

8.17 v0.2.10 (18-10-2016)

- Added the C6 coefficients from Gould and Bucko.
- Added van der Waals radii from Alvarez.
8.18  v0.2.9 (16-10-2016)

• Added a scale of electronegativities by Ghosh.

8.19  v0.2.8 (29-08-2016)

• Updated the electron affinity of Pb and Co.
• Updates of the docs.

8.20  v0.2.7 (02-04-2016)

• Maintenance.

8.21  v0.2.6 (02-04-2016)

• Mainly maintenance updates to docs, sphinx conf.py, setup.py, requirements.

8.22  v0.2.5 (02-04-2016)

8.22.1 Features added

• Added calculation of Martynov and Batsanov scale of electronegativity in `en_martynov_batsanov` method in the `Element` class
• Added `abundance_crust` and `abundance_sea` with element abundancies in the crust and seas
• Added `molcas_gv_color` attribute with MOLCAS GV colors

8.22.2 Bugs fixed

• Restored Python 3.x compatibility

8.23  v0.2.4 (05-02-2016)

8.23.1 Features added

• Extended and corrected the documentation and Jupyter notebook tutorials on basic usage electronegativities, plotting and tables

8.23.2 Bugs fixed

• Corrected `raise` to `return` when calling `en_sanderson` from `electronegativity`
• Fixed and tested the formula for calculating the Li and Xue scale of electronegativity in `en_lie-xue`
8.24 v0.2.3 (27-01-2016)

8.24.1 Features added

- Added new vdW radii: vdw_radius_batsanov, vdw_radius_bondi, vdw_radius_dreiding, vdw_radius_mm3, vdw_radius_rt, vdw_radius_truhlar, vdw_radius_uff
- Added an option to plot the long (wide) version of the periodic table in periodic_plot

8.24.2 Bugs fixed

- Typos in the docstrings

8.25 v0.2.2 (29-11-2015)

8.25.1 Features added

- Added new covalent radii: covalent_radius_bragg, covalent_radius_slater
- Added the $c_6$ dispersion coefficients
- Added gas_basicity, proton_affinity and heat_of_formation
- Added periodic_plot function for producing Bokeh based plots of the periodic table
- Added jmol_color and cpk_color with different coloring schemes for atoms

8.25.2 Bug fixes

- Changed the series of elements 113, 114, 115, 116 to poor metals

8.26 v0.2.1 (26-10-2015)

8.26.1 Features added

- Extended the list of options for calculating Mulliken electronegativities in en_mulliken
- Added electrons_per_shell method
- Added a function to calculate linear interpolation of radii required for calculation of Sandersons electronegativity
- Added hybrid attributes electrons, protons, neutrons and mass_number

8.26.2 Bug fixes

- Changed the type of the melting_point from str to float
8.27 v0.2.0 (22-10-2015)

8.27.1 Features added

- Instead of covalent_radius added covalent_radius_2008 and covalent_radius_2009
- Instead of electronegativity added en_pauling and en_mulliken
- Added a method for getting ionic radii
- Improved the method for calculating the nuclear screening constants
- Added ElectronicConfiguration class initialized as Element attribute
- Added nuclear screening constants from Clementi and Raimondi
- Added a method to calculate the absolute softness, absolute hardness and absolute electronegativity
- Added get_table method to retrieve the tables as pandas DataFrames

8.27.2 Bug fixes

- Added missing electronic configurations
- Converted ionic radii from Angstrom to pico meters

8.28 v0.1.0 (11-07-2015)

First tagged version with the initial structure of the package and first version of the database and the python interface
CHAPTER 9

Indices and tables

- genindex
- modindex
- search
Bibliography


[28] Manjeera Mantina, Adam C Chamberlin, Rosendo Valero, Christopher J Cramer, and Donald G Truhlar. Consis-
tent van der Waals Radii for the Whole Main Group. *The Journal of Physical Chemistry A*, 113(19):5806–5812, 

10.1021/j100389a010, doi:10.1021/j100389a010.

Holden, Johanna Irgeher, Robert D. Loss, Thomas Walczyk, and Thomas Prohaska. Atomic weights of the ele-

[31] Robert S Mulliken. A New Electroaffinity Scale; Together with Data on Valence States and on Valence Ionization 


[33] Linus Pauling. THE NATURE OF THE CHEMICAL BOND. IV. THE ENERGY OF SINGLE BONDS 
AND THE RELATIVE ELECTRONEGATIVITY OF ATOMS. *Journal of the American Chemical Society*, 


odic table force field for molecular mechanics and molecular dynamics simulations. *Journal of the Ameri-
can Chemical Society*, 114(25):10024–10035, Dec 1992. URL: http://pubs.acs.org/doi/abs/10.1021/ja00051a040,
doi:10.1021/ja00051a040.

[40] R Scott Rowland and Robin Taylor. Intermolecular Nonbonded Contact Distances in Organic Crystal Struc-


doi:10.1021/ja01139a020.
[43] R. D. Shannon. Revised effective ionic radii and systematic studies of interatomic distances in halides and chalco-


[48] P. Villars, K. Cenzual, J. Daams, Y. Chen, and S. Iwata. Data-driven atomic environment prediction for bi-


Index

A
atomre (ElectronicConfiguration attribute), 42

C
calc_en_sanderson() (Element method), 37
conf (ElectronicConfiguration attribute), 42
covalent_radius (Element attribute), 37

electron negativity() (Element method), 37
ElectronicConfiguration (class in mendeleev.ecnf), 42
electrons (Element attribute), 38
electrons_per_shell() (ElectronicConfiguration method), 42
Element (class in mendeleev.tables), 35
element() (in module mendeleev.mendeleev), 33
en_calc() (Element method), 38
en_li_xue() (Element method), 38
en_martynov_batsanov() (Element method), 38
en_mulliken() (Element method), 38

G
get_engine() (in module mendeleev.mendeleev), 35
get_largest_core() (ElectronicConfiguration method), 42
get_session() (in module mendeleev.mendeleev), 35
get_table() (in module mendeleev.mendeleev), 34
get_valence() (ElectronicConfiguration method), 42
Group (class in mendeleev.tables), 41

H
hardness() (Element method), 38

I
init_on_load() (Element method), 39
ionenergies (Element attribute), 39
IonicRadius (class in mendeleev.tables), 40
IonizationEnergy (class in mendeleev.tables), 40
ionize() (ElectronicConfiguration method), 42
Isotope (class in mendeleev.tables), 40

L
last_subshell() (ElectronicConfiguration method), 42

M
mass (Element attribute), 39
mass_number (Element attribute), 39
mass_str() (Element method), 39
max_l() (ElectronicConfiguration method), 42
max_n() (ElectronicConfiguration method), 42

N
ne() (ElectronicConfiguration method), 42
neutrons (Element attribute), 39
nvalence() (ElectronicConfiguration method), 42
nvalence() (Element method), 39

O
OxidationState (class in mendeleev.tables), 41
oxistates (Element attribute), 39

P
parse() (ElectronicConfiguration method), 42
protons (Element attribute), 39

S
sconst (Element attribute), 39
ScreeningConstant (class in mendeleev.tables), 41
Series (class in mendeleev.tables), 41
shell2int() (ElectronicConfiguration method), 42
shellre (ElectronicConfiguration attribute), 42
slater_screening() (ElectronicConfiguration method), 42
softness() (Element method), 39
sort() (ElectronicConfiguration method), 42
spin_occupations() (ElectronicConfiguration method), 43
spin_only_magnetic_moment() (ElectronicConfiguration method), 43

to_str() (ElectronicConfiguration method), 43
U
unpaired_electrons() (ElectronicConfiguration method), 43

Z
zeff() (Element method), 39