nucleardata — provides nuclide information*

Bill Nettles†

Released 2016/04/01

The `nucleardata` package provides a method for quickly accessing information about atomic nuclides (isotopes of elements) by referring to the chemical symbol and mass number (A) or to the atomic number (Z) and mass number (A). This information can be inserted and typeset without the user having to search an outside source. The information available, via macro commands, in the current version includes chemical symbol or name given Z, Z given the chemical symbol or name, atomic mass, nuclear mass, Q-values for radioactive decay, half-life of ground states, binding energy, mass excess, and list of known isotopes of an element.

There is a macro command for generating a random set of nuclides, accessible as elements of lists of Z and corresponding (valid) A. The elements of the list can be used as arguments for Z and A arguments of other `nucleardata` macros. Another macro will randomly select an valid A for a user-specified Z.

Source of Data

The data is contained in two CSV format files: `massdata.csv` and `elementlist.csv`. These files must be installed in the same directory as the `nucleardata.sty` file. They are read during the LaTeX compile process and lookups are done by Python code embedded into the `.sty` file. Initially, the Python code loads all the data from the `CSV` files into Python arrays, `nucleardata` and `elementdata`.

The `massdata.csv` file was created by the author (Nettles) from ENSDF data files and the file `mass.mas03round` from "The Ame2003 atomic mass evaluation (II)" by G.Audi, A.H.Wapstra and C.Thibault, *Nuclear Physics* A729 p. 337-676, December 22, 2003. The `elementlist.csv` file was created by the author from public sources.

Using PythonTeX

Because Python is the basis of the lookup engine, the package `pythontex` is automatically loaded. The user must follow a three-step compiling sequence to get a semifinal/final document. For example, if the user's file is named `carbon.tex`, the sequence will be

*This file describes version v1.01c, last revised 2016/04/01.

†E-mail: bnettles@uu.edu

```
pdflatex carbon.tex
pythontex carbon.tex
pdflatex carbon.tex
```

If the user is familiar with PythonTEX this shouldn't seem unusual.

A Python Class–`Nucdata`

Python functions are called by customized LaTeX commands to extract the data from the `nucleardata` and `elementdata` arrays, so the user has the capability of using these functions to write custom Python routines within the default PythonTEX environment. The functions belong to a class defined in this package as `Nucdata`. The class is instantiated in the package Python code as `nuc`.

The functions will allow the user to use the data for more specific calculations such as Q-values of reactions or decay chain behaviors. PythonTEX was designed for tasks such as this. The user can utilize the functions as part of the `nuc` instantiation or can implement their own instance. The data arrays are loaded external to the class.

Neutron Notation

The neutron doesn't have a chemical symbol, but in this package the symbol **nn** can be used with a mass number, A, of 1. It can also be referenced with Z=0 and A=1 as arguments.

Rounding Option

Some of the macros below have an optional argument that lets the user specify rounding of decimal places. The rounding is accomplished using the Python `round(`⟨*float*⟩`,`⟨*places*⟩`)` function inside a PythonTEX `\py()` call. The ⟨*float*⟩ argument is the return value of the main function mentioned in each description. Rounding is not a part of the definition of the main functions. As an example, the LaTeX command definition for `nucamassu` is defined as

```
\newcommand{\nucamassu}[3][6]{\py{round(getMass_u('#2',#3),#1)}}
```

This definition gives a default rounding of six decimal places.

Using Python variable names as arguments

If a command calls for an integer value for either Z or A, you can supply the name of a Python variable which contains an integer value. This value could be set using a PythonTeX call or one of the `nucleardata` commands that selects random Z or A values (see `nucrandom[]{}` and `nucAran{}`).

```
\pyc{myz=6} Element Z=\py{myz} is called \nucname{myz}.
```

Macro Commands

`\nucsymbol` Command form: `\nucsymbol{`⟨*namez*⟩`}`.

The argument can be an unquoted string (the name of the element) or an integer (atomic number, Z). Returns a string with the element symbol. Calls a Python function `getSymbol('`⟨*namez*⟩`')`.

`\nucname` `\nucName` Command form: `\nucName{`⟨*namez*⟩`}` or `\nucname{`⟨*namez*⟩`}`.

The argument can be an unquoted string (the symbol of the element) or an integer (atomic number, Z). `\nucName` returns a string with the element name capitalized. `\nucname` returns the name in lower case. Calls a Python function `getName('`⟨*namez*⟩`')`.

`\nucz` Command form: `\nucz{`⟨*namez*⟩`}`.

The argument must be an unquoted string (the symbol or the name of the element). Returns the atomic number, Z. Calls a Python function `getZ('`⟨*namez*⟩`')`.

`\nuchalflife` `\nuchalfvalue` `\nuchalfunit` Command form: `\nuchalflife[`⟨*unit*⟩`]{`⟨*namez*⟩`}{`⟨*A*⟩`}`.

The optional argument is an unquoted string specifying the time unit to use for the return value. The chart below lists valid arguments. The first required argument can be an unquoted string (the symbol) or an integer (Z). The second required value must be an integer, the mass number, A. Returns a string with the value and units of the halflife of the specific nuclide. Calls a Python function `getHalfLife('`⟨*namez*⟩`', `⟨*A*⟩`, '`⟨*unit*⟩`')`.

There are two variations on this command:

`\nuchalfvalue` calls `getHalfLifeValue('`⟨*namez*⟩`', `⟨*A*⟩`, '`⟨*unit*⟩`')` and returns the unformatted numerical portion of the halflife and `\nuchalfunit` calls `getHalfLifeUnit('`⟨*namez*⟩`', `⟨*A*⟩`, '`⟨*unit*⟩`')` returns a string with the unit portion. They take the same arguments as `\nuchalflife`. If there is no half life listed, the call returns the None token.

argument	unit symbol	unit name
`ev`	eV	electron-volt
`mev`	MeV	mega-electron-volt
`kev`	keV	kilo-electron-volt
`ps`	ps	picosecond
`ns`	ns	nanosecond
`us`	µs	microsecond
`ms`	ms	millisecond
`s`	s	second
`m` or `min`	min	minute
`h` or `hr`	h	hour
`d` or `day`	d	day
`y` or `yr`	yr	year
`My`	My	megayear
`Gy`	Gy	gigayear

\nucspin Command form: \nucspin{⟨*namez*⟩}{⟨*A*⟩}, etc.

The first required argument can be an unquoted string (the symbol) or an integer (Z). The second must be an integer, the mass number, A. Returns the value of the **spin quantum number and parity** of the ground state of the nuclide. If no value has been assigned, it returns "None." Calls a Python function `getSpin('`⟨*namez*⟩`', `⟨*A*⟩`)`.

\nucamassu \nucamassmev \nucamasskev Command form: \nucamassu[⟨*rnd*⟩]{⟨*namez*⟩}{⟨*A*⟩}, \nucamassmev[⟨*rnd*⟩]{⟨*namez*⟩}{⟨*A*⟩}, \nucamasskev[⟨*rnd*⟩]{⟨*namez*⟩}{⟨*A*⟩}.

The optional argument is the number of decimal places for rounding; the default is 6 (or 3 for keV). The first required argument can be an unquoted string (the symbol) or an integer (Z). The second must be an integer, the mass number, A. Returns the value of the **atomic** mass of the nuclide in atomic mass units (u), MeV/c^2 or keV/c^2, respectively. Calls Python function `getMass_u('`⟨*namez*⟩`', `⟨*A*⟩`)` or `getMass_mev(...)` or `getMass_kev(...)`.

\nuclearmassu \nuclearmassmev \nuclearmasskev Command form: \nuclearmassu[⟨*rnd*⟩]{⟨*namez*⟩}{⟨*A*⟩}, etc.

The optional argument is the number of decimal places for rounding; the default is 6 (or 3 for keV). The first required argument can be a string (the symbol) or an integer (Z). The second must be an integer, the mass number, A. Returns the value of the **nuclear** mass of the nuclide in atomic mass units (u), MeV/c^2 or keV/c^2, respectively. Calls Python function `getNuclearMass_u('`⟨*namez*⟩`', `⟨*A*⟩`)`, etc.

\nucexcess Command form: \nucexcess[⟨*rnd*⟩]{⟨*namez*⟩}{⟨*A*⟩}.

The optional argument is the number of decimal places for rounding; the default is 3. The first required argument can be a string (the symbol) or an integer (Z). The second must be an integer, the mass number, A. Returns the mass excess (Δ) in keV/c^2. (Atomic mass = A×931502 + Δ, in keV). Calls Python function `getExcess('`⟨*namez*⟩`',`⟨*A*⟩`)`.

\nucbea Command form: \nucbea[⟨*rnd*⟩]{⟨*namez*⟩}{⟨*A*⟩}

The optional argument is the number of decimal places for rounding; the default is 3. The first required argument can be a string (the symbol) or an integer (Z). The second must be an integer, the mass number, A. Returns the binding energy per nucleon in MeV. (Z*atomic mass(^{1}H)+(A-Z)*mass neutron-atomic mass of nuclide)/A. Calls Python function `getBea('`⟨*namez*⟩`',`⟨*A*⟩`)`.

\nucisotopes Command form: \nucisotopes{⟨*namez*⟩}

The argument can be a string (the element symbol) or an integer (Z). The macro returns a string list of all the isotopes of that element which have mass information available in the database. Calls Python function `getIsotopes('`⟨*namez*⟩`')`.

The Python function `getIsotopes('`⟨*namez*⟩`')` produces a Python list of integers for the mass numbers, A, found in the data associated with the element. A

user can access the individual A values using a direct call in a PythonTex command like this:

```
The lightest isotope of \nucname{8} is \py{nuc.getIsotopes('8')[0]}
        and the heaviest is \py{nuc.getIsotopes('8')[-1]}.
```

Notice that in the direct call, the class instance, `nuc` is specified, and the argument is enclosed in single quotes. Quotes should NOT be used with the LaTeX macros in this package.

`\nucQalpha` `\nucQbeta` `\nucQposi` `\nucQec`

Command form: `\nucQ-----[`⟨*rnd*⟩`]{`⟨*namez*⟩`}{`⟨*A*⟩`}`

The optional argument is the number of decimal places for rounding; the default is 6. The first required argument can be an unquoted string (the element symbol) or an integer (Z). Returns the Q-value of the chosen decay in MeV. Decay type options are `alpha`, `beta`, `posi` (for positron), and `ec` for electron capture. Call Python functions `getQ-----('`⟨*namez*⟩`',`⟨*A*⟩`)`.

`\nucisalpha` `\nucisbeta` `\nucisposi` `\nucisec`

Command form: `\nucis-----{`⟨*namez*⟩`}{`⟨*A*⟩`}`.

The first argument can be an unquoted string (the element symbol) or an integer (Z). Returns the string **True** or **False** depending on whether the Q-value of a decay is positive or negative. Decay type options are the same as for the `\nucQalpha{-----` macros. Calls Python functions `getQ-----('`⟨*namez*⟩`',`⟨*A*⟩`)` and checks for positive value.

`\nucreactionqu` `\nucreactionqmev` `\nucreactionqkev`

Command form: `\nucreactionqu[`⟨*rnd*⟩`] {`⟨*namez1*⟩`} {`⟨*A1*⟩`} {`⟨*namez2*⟩`} {`⟨*A2*⟩`} {`⟨*namez3*⟩`} {`⟨*A3*⟩`} {`⟨*namez4*⟩`} {`⟨*A4*⟩`}`, etc.

The optional argument is the number of decimal places for rounding; the default is 6 (or 3 for keV). The first required argument can be a string (the symbol) or an integer (Z). The second must be an integer, the mass number, A. The numbers after ⟨*name*⟩ and ⟨*A*⟩ represent

1 – the target nucleus/particle

2 – the projectile nucleus/particle

3 – the ejected nucleus/particle

4 – the resultant nucleus/particle

Returns the Q-value of the reaction in atomic mass units (u), MeV/c^2 or keV/c^2, respectively. Calls Python function `getReaction_u('`⟨*namez1*⟩`',` ⟨*A1*⟩, `'`⟨*namez2*⟩`',` ⟨*A2*⟩, `'`⟨*namez3*⟩`',` ⟨*A3*⟩, `' `⟨*namez4*⟩`',` ⟨*A4*⟩`)`, etc.

`\nucAran`

Command form: `\nucAran{`⟨*namez*⟩`}`.

This macro generates a random A value for the given element specified by the required argument. The argument can be a string (the element symbol) or a number (Z). The generated A is stored in a Python variable, **singleAran**, accessible via `PythonTeX` macros such as `\py{singleAran}`. This list element name can also

be used as the A argument for other `nucleardata` macros. Calls Python function `nuc.getRandomA('`⟨*namez*⟩`')`.

`\nucrandom` Command form: `\nucrandom[`⟨*repeater*⟩`]{`⟨*listsize*⟩`}`.

This macro generates a list of random Z values and a list with a valid random A value for each chosen Z. The optional argument determines whether a Z value can selected more than once (0=no repeat [default], 1=repeats allowed). The required argument is an integer specifying the length of the lists. The generated list of Z is stored in a Python list, **zran**, and the associated A for each Z is stored in a Python list, **aran**. **zran[k]**→**aran[k]**. The values in each list are accessible via `PythonTeX` macros such as `\py{zran[0]}`. The list element members can also be used as Z or A arguments for other `nucleardata` macros.

Calls Python function `nuc.makeNucRandom(`⟨*listsize*⟩`,`⟨*repeater*⟩`)`.