sigfig

Release 1.2.0

Aug 17, 2021

Contents

1 Installation **3**
- 1.1 Automatic Install . . . 3
- 1.2 Manual Install . . . 3

2 Usage Guide **5**
- 2.1 Quickstart . . . 5

3 API Documentation **7**
- 3.1 Arguments . . . 7
- 3.2 Rounding Operations . . . 7
- 3.3 Uncertainty Rounding Rules . . . 8
- 3.4 Formatting Output . . . 9
- 3.5 Formatting Output with Uncertainty . . . 10
- 3.6 Other “Features” . . . 11

4 Project Development & Roadmap **13**
- 4.1 Initial Motivation . . . 13
- 4.2 Next Steps . . . 13
- 4.3 Possible Features . . . 14
- 4.4 Contributor Notes . . . 15

Index **17**

This is the `sigfig` Python package used for rounding numbers (with expected results).

```
>>> round(0.25, 1)
0.2
>>> from sigfig import round
>>> round(0.25, decimals=1)
0.3
>>> round(3.14159, 2)
3.1
>>> round(3.14159, 0.003839)
'3.142 ± 0.004'
>>> round(3.141592653589793, 0.00000002567, format='Drake')
'3.141 592 654(26)'
```

Key Features:

- round numbers by significant figures/digits
- round numbers by decimal places
- round numbers by uncertainty/error
- format numbers in a variety of common styles & notations
- read in numbers of any type

In-depth documentation can be found here:

CHAPTER 1

Installation

1.1 Automatic Install

To install the `sigfig` module, simply run this command in your terminal of choice:

```
$ pip install sigfig
```

If you don't have pip installed, the pip install page can guide you through the process. However if your version of Python doesn't come with pip consider updating your Python to the latest version.

1.2 Manual Install

Alternatively, you can `download sigfig.py` and extract it to either your Python Scripts directory or the same directory as the file from which you're using `sigfig`. You will need to do the same for the sortedcontainers module - a dependency for this project.

CHAPTER 2

Usage Guide

This page contains the most common use cases for the `sigfig` module.

First make sure that `sigfig` is *installed*.

2.1 Quickstart

Using the `sigfig` module's `round()` function is very simple.

Begin by importing the module and adding its features to the built-in round function:

```
>>> from sigfig import round
```

let's round a number to 4 significant figures

```
>>> round(123.456, 4)
123.5
```

but let's get in the habit of storing our numbers as `str`ings to circumvent `float`'s inherent lack of precision. `round()` will always return in the rounded number in the same type it was given unless told otherwise:

```
>>> round('123.456', 4)
'123.5'
>>> from decimal import Decimal
>>> round(Decimal(123456E-3), 3)
Decimal('123')
>>> round('123.456', 5, type=float)
123.46
```

We can round numbers by number of decimals, by significant figures/digits, or by the number's *uncertainty*:

```
>>> round('3.14159', sigfigs=2)
'3.1'
>>> round('3.14159', decimals=2)
```

(continues on next page)

(continued from previous page)

```
'3.14'
>>> round('3.14159', uncertainty=2)
'3 ± 2'
```

which the function will choose based on the type/context of the 2nd argument:

```
>>> round('3.14159', 2)
'3.1'
>>> round('3.14159', '0.0007524')
'3.1416 ± 0.0008'
```

When rounding by uncertainties we can isolate the rounded number and/or uncertainty by setting the `separation` keyword argument to `list` or `tuple`:

```
>>> round('123456E-5', '123E-6', sep=tuple)[0]
'1.2346'
>>> round(123456E-5, 123E-6, sep=list, type=Decimal)
[Decimal('1.2346'), Decimal('0.0001')]
```

2.1.1 Formatting

We can use `round()` to output numbers in a myriad of *different formats*:

```
>>> round('86375.25799', decimals=2, notation='sci') # scientific notation
'8.637526E4'
>>> round('86375.25799', sigfigs=3, notation='eng') # engineering notation
'86.4E3'
>>> round('86375.25799', '0.023759', format='PDG') # Particle Data Group preferred␣
↪formatting
'86375.258 ± 0.024'
>>> round('863192837.1176159248', '.00002742764', format='Drake') # Drake Group␣
↪preferred formatting
'863 192 837.117 616(27)'
```

or we can create our own custom format either from scratch

```
>>> round('17265098762.12345678', .000000289, spacing=5, spacer=',', decimal='_',␣
↪separation=' +/- ')
'1,72650,98762_12345,68 +/- 0_00000,03'
```

or by modifying an existing format:

```
>>> round('86375.25799', '0.023759', format='PDG', sep='brackets')
'86375.258(24)'
>>> round('863192837.1176159248', '.00002742764', format='Drake', cutoff=22)
'863 192 837.117 62(3)'
```

See the *API Documentation* for a more detailed explanation of all above-used features.

CHAPTER 3

API Documentation

This guide explains the interface to the `round()` function along with all the accepted parameters and how they effect the output:

3.1 Arguments

The first argument specifies the number to be rounded and/or reformatted. Can be of any numeric data type or type which can be interpreted as a number (ie. string '1.567'). The type of the returned value will be the same as this argument's type unless otherwise specified by the `type` keyword argument or by including uncertainty. `ValueError` will be raised in the event of uninterpretable input.

The second argument specifies either the number of significant figures (if `int` data type), or the uncertainty to which the first argument will be rounded (if any numeric-interpreted data type is given aside from `int`). `ValueError` will be raised in the event of uninterpretable input.

Additional arguments (aside from the keyword arguments specified below) are ignored

3.2 Rounding Operations

Only 1 of the 3 rounding operation may be used at a time. In the event multiple operations are requested, rounding by uncertainty will take precedence over rounding by significant figures which will take priority over rounding by number of decimals. Selecting a rounding operation is not mandatory and can be ignored when `round()` is being called strictly for formatting operations.

3.2.1 sigfigs

Default value: `None`

Controls how many significant figures the given number is to be rounded to in accordance with significant figures rounding rules. Can be specified with `sigfigs` keyword argument or by passing as 2nd argument of type `int`. When specified, it should be an `int` greater than 0.

```
>>> from sigfig import round
>>> round(12.7654, sigfigs=4)
12.77
>>> round(12.7654, 3)
12.8
```

3.2.2 decimals

Default value: `None`

Controls how many decimal places (or negative ten's power) the given number is to be rounded to in accordance with decimal place rounding rules. When specified, it should be an `int` of any value.

```
>>> from sigfig import round
>>> round(12.7654, decimals=3)
12.765
>>> round('12.7654', decimals=-1)
'10'
```

3.2.3 uncertainty

Default value: `None`

Takes the uncertainty which will determine how many decimal places the given number is rounded to in accordance with *Uncertainty Rounding Rules* and `cutoff` value (default value: 9). In the default `cutoff` case these rules dictate that the uncertainty is rounded to 1 significant figure and the given number is rounded to the same number of decimals as the uncertainty. By specifying an uncertainty, both the rounded number and rounded uncertainty will be returned (in a string separated by " ± " by default) Can be specified with `uncertainty`/`unc` keyword argument or by passing as 2nd argument in numeric-interpreted type (except `int`) to `round()`.

```
>>> from sigfig import round
>>> round('3.14159', uncertainty='0.6567')
'3.1 ± 0.7'
>>> round(3.14159, 0.001567)
'3.142 ± 0.002'
```

3.3 Uncertainty Rounding Rules

A number's uncertainty or error is a measure of how accurate that number is. Consequently, the uncertainty's order of magnitude (aka number of decimals) is of greater importance than it's value resulting in the uncertainty usually being displayed with only 1 significant figure so as to not distract from it's associated number. However, many of those in the scientific community will give 2 figures of uncertainty if the uncertainty begins with a 1 or 2. One prominent research group (The Particle Data Group) rounds their measured uncertainties to 2 decimal places if they begin with 35 (after being rounded) and will round to 1 decimal place if they begin with 36 or higher. This behavior is modified through

the `cutoff` keyword argument which will always round to 1 decimal place in the event of `cutoff=9`, round to 2 decimal places if the uncertainty begins with a 1 or 2 with `cutoff=29` (numbers beginning with 3-9 will be rounded to 1 decimal), and The Particle Data Group's preference sets `cutoff=35`.

Following the rounding of the uncertainty, the given number (not uncertainty) will be rounded to the smallest magnitude of the resulting rounded uncertainty. After all it would be confusing (or even misleading) to state a number with 6 decimals of accuracy when you're uncertain of any digit beyond the first decimal point.

3.3.1 cutoff (crop)

Default value: `9`

The uncertainty magnitude value (`int` 9) after which the uncertainty value is rounded with 1 less digit.

```
>>> from sigfig import round
>>> round('3.14159', '0.6567', cutoff=65)
'3.1 ± 0.7'
>>> round('3.14159', '0.6567', cutoff=66)
'3.14 ± 0.66'
>>> round('3.14159', '0.6567', crop=77)
'3.14 ± 0.66'
```

3.4 Formatting Output

3.4.1 notation (form)

Default value: `'standard'`

Output number format notation can be one of `standard`/`std` (default) for *standard notation* without exponentiation, `engineering`/`eng` for engineering notation, or `scientific`/`sci` for scientific notation.

```
>>> from sigfig import round
>>> round('3679.14159', decimals=2, notation='scientific')
'3.67914E3'
>>> round('16248055.209', notation='eng')
'16.248055209E6'
>>> round('16248055.209', '19923.456', notation='eng')
'16.25E6 ± 0.02E6'
```

Note: Should not be used in conjunction with kwarg `format`/`style` or `type`/`output_type` (since that would essentially be asking for conflicting outputs).

3.4.2 output_type (type)

Default value: `type(arg[0])`

Return type can be any numeric-interpreted type (i.e. `decimal.Decimal`, `float`, `str`, `int`) and should not be a string of that type (i.e. Use `float` instead of `'float'`).

```
>>> from sigfig import round
>>> from decimal import Decimal
>>> round('3679.14159', decimals=2, output_type=float)
3679.14
>>> round(16248055.209, type=Decimal)
Decimal('16248055.209')
```

Note: Should not be used in conjunction with kwarg `format`/`style` or `notation`/`form` (since these will require `str` output type).

3.4.3 spacing

Default value: `None`

Adds a `spacer` character every `spacing`'th digit. Should be `int` 1.

```
>>> from sigfig import round
>>> round('3679.14159', spacing=3, spacer=' ')
3 679.141 59
>>> round('94916248055.209', spacing=5, spacer=',')
'9,49162,48055.209'
```

3.4.4 spacer

Default value: `''`

Adds a `spacer` character (string) every `spacing`'th digit.

3.4.5 decimal

Default value: `'.'`

Changes the decimal point character (`str`).

```
>>> from sigfig import round
>>> round('3679.14159', decimals=2, decimal=',')
'3679,14'
```

3.5 Formatting Output with Uncertainty

3.5.1 separation (sep)

Default value: `' ± '`

Changes the string which separates a number from it's uncertainty. Recognizes the special strings `'brackets'` for in-line bracketed uncertainty, `'external_brackets'` for the special case of uncertainties greater than 10, and `tuple` or `list` which allows number and uncertainty to be stored independently.

```
>>> from sigfig import round
>>> round('3679.14159', '0.00123', separation='+/-')
'3679.142+/-0.001'
>>> round('3679.14159', 0.000123, sep='brackets')
'3679.1416(1)'
>>> round('97.74159', 0.393, sep=tuple)
('97.7', '0.4')
>>> round('3679990.14159', '123.00123', sep='brackets')
'36800(1)00'
>>> round('3679990.14159', '123.00123', sep='external_brackets')
'3680000(100)'
```

3.5.2 format (style)

Default value: `None`

Allows choice of predefined formats `'Drake'` and `'PDG'` for The Drake Group's preferred formatting of `cutoff=29, spacer=3, spacing=' ', separation='brackets'` and The Particle Data Group's preferred formatting of `cutoff=35` (see 5.3 Rounding).

```
>>> from sigfig import round
>>> round('3679990.14159', '0.00125', format='Drake')
'3 679 990.141 6(1 3)'
>>> round('3679990.14159', '0.00125', style='PDG')
'3679990.1416 ± 0.0013'
```

Note: Should not be used in conjunction with kwarg `output_type`/`type` or `notation`/`form`.

3.6 Other "Features"

3.6.1 order of keyword arguments

The interface for `round()` allows for conflicting keyword arguments (i.e. `cutoff=19, cutoff=20` or `format='Drake', sep='+/-'`) where subsequent kwargs overwrite what comes before them. However, this feature assumes insert-ordered `dict`ionaries which is not guaranteed until Python 3.7 (and beyond). If you are using `sigfig` with earlier versions of Python (before 3.7) without insert-ordered `dict`'s the recommended usage is to avoid conflicting keyword arguments.

3.6.2 prefix

Default value: `None`

This is an experimental feature which adds a metric SI unit prefix to the end of the outputted string (or multiple prefixes in the case of very big or very small numbers). This feature behaves similar to engineering notation except using prefixes instead of exponents. It has some unresolved edge cases that can be fully flushed out if found useful and requested.

```
>>> from sigfig import round
>>> round('3679990.14159', '97654', style='Drake', prefix=True)
'3.68(10)M'
>>> round('3.67999014159E-10', '0.00125E-10', prefix=True)
'368.0 ± 0.1p'
```

3.6.3 zero behaviour

Any number with a value of zero that is known to 1 or more decimal places will be represented with all trailing zeros (ie. 0.00 is known to 2 decimal places and all trailing zeros are displayed). Conversely any number with a value of zero that is known to -1 or fewer decimal places will be represented with only 1 digit (ie. 000 will only be displayed as 0). The only exception is in the case of (non-external) bracketed uncertainty when the number is zero and known to -1 or fewer decimal places. Below is an example of each scenario:

```
>>> from sigfig import round
>>> round('0.00004567', decimals=3)
'0.000'
>>> round('23', '4732')
'0 ± 5000'
>>> round('23', '4732', sep='brackets')
'0(5)000'
```

CHAPTER 4

Project Development & Roadmap

4.1 Initial Motivation

This project was created to add needed features to Python's built-in `round()` function. Namely:

- the ability to consistently round floating point numbers despite float's inherent lack of precision.
- the ability to round a number by number of significant figures/digits instead of by decimal places only.
- the ability to round a number by its associated uncertainty.

Additional features were also needed to format numeric values for scientific publications. Namely:

- displaying a number with its associated uncertainty - in bracketed form - with a space between every thousand(th)s

4.2 Next Steps

The Ultimate goal of the project is to add the included rounding features to the Python standard library. This will require some refactoring among other tasks before the first pull request should be made. This work is outlined in the following sections:

4.2.1 Separate out formatting code

The code for formatting the resultant rounded number string does not belong in the standard library's `round()` function but would make more sense as either it's own package, as part of the `numpy` package (ie. the `format_float_positional()` function), or as part of another package involving numeric or data visualization. This will have the added benefit of making `sigfig`'s code more readable which is never a bad thing.

4.2.2 Increase numeric storage efficiency and standardization

`sigfig` currently parses numbers by first converting to string and then storing in a {<ten's power>:<numeric value>} dict (see `_Number` in source for technical details). While this guarantees bug-free functionality for all numbers and is suitable for numbers already stored as strings, this lacks efficiency for `decimal.Decimal` and `float`. Numeric values could possibly be stored instead using the same storage technique employed by the `decimal.Decimal` package (after an investigation of that technique to ensure full code coverage). This should fully satisfy the `decimal.Decimal` case whereas the `float` case can be handled as-is by default but allowed to optionally (with `high_speed=True` instead of the default `high_accuracy=True`) use floating point arithmetic when speed trumps accuracy.

4.2.3 Interface Overhaul

The current interface is multi-dimensional and very forgiving which allows for a wide range of allowable but unexpected behaviours instead of warning/crashing if a user strays from typical use cases. While this suit's the project's current scrappy state, a redesign of the interface-handling `_arguments_parse()` function is recommended before merging with the standard library.

4.3 Possible Features

Other features looking for implementation by any potential contributers are welcomed, would be greatly appreciated, are detailed below, and (subjectively) ordered by priority:

4.3.1 Baking and User-Defined Formats/Styles

The ability to "bake" default behaviour into `round()` (essentially partial application, like you might do with `functools.partial()`) could allow for many desirable customizations. Some examples are:

- rounding by number of decimals instead of significant figures by default through something like `round.bake(round_by_decimals=True)`
- always spacing numbers by 3 (in the case where output is of type `str`) through `round.bake(spacing=3, spacer=' ')`

4.3.2 Warnings, Alerts, and Feedback

Certain actions and usages of `round()` warrant feedback given to the user. These include (but are not limited to) the following:

- warning for invalid keyword arguments
- warning for depreciated usages
- warning when out of range values are passed
- informing when conflicting inputs are provided
- informing when any data is passed implicitly instead of explicitly. For example: `round(3.2, 1)` versus `round(3.2, sigfigs=1)`

4.3.3 Units, Formatted Numbers, and Unit Prefixes

Modification of the `_num_parse()` function can be made without much effort to allow for formatted numbers (ie. `'1,237.0'`), currency (ie. `'$3,157.00'`), or numeric data with units (ie. `'3475.2753nm'`) to be accepted. This formatting data can be parsed and interpreted alongside the numeric data and the resulting output from the `round()` operation can be given (by default) in the same format as the input was given.

Also, common units with their prefixes can be parsed so that more suitable prefixes for units can be chosen or explicitly specified by a new keyword argument. For example:

```
>>> round('3475.2753nm', '45.9479nm')
'3.48 ± 0.05 μm'
>>> round('3475.2753nm', '45.9479nm', units='cm', sep='brackets')
'0.000348(5) cm'
```

4.3.4 Documentation: Figure(s) for Rounding Rules

The *Uncertainty Rounding Rules* section may be confusing to those unfamiliar with the concept and would benefit from visual aid. This can help to disambiguate like-sounding terms like "uncertainty", "magnitude of uncertainty", "number's uncertainty", and "error" as well as "number", "given number", and "number of decimals".

4.3.5 Input Precision

Input precision is not currently stored. In cases where a number is rounded to more decimals than it was given (ie. `round(1.23, 0.000073)`) a warning can be thrown stating *"implicit uncertainty (0.005) greater than provided uncertainty (0.000073). Provided uncertainty will be used."* since (in this case) the value 1.23 could be representing any value between 1.225 and 1.2349999....

4.3.6 Formatting of Exponentials

The exponentials resulting from scientific and engineering notation are separated from the number & uncertainty with an uppercase "E" present in both the number and resulting uncertainty. Some might find it useful to customize the character(s) and/or optionally only appended the character after the uncertainty and not after the number.

4.3.7 Parse Number Last (small efficiency increased)

A small gain to efficiency can be made by first parsing the uncertainty, number of decimals, or number of significant figures (aka the rounder) since these dictate how many digits are relevant in the given number. With the rounder known, the parsing of the given number can be quicker since digits beyond what the rounder dictates can be discarded. This will require a re-design of `_num_parse()` where the exponential information is parsed first and will only be of (limited) benefit when the number is given with exponential notation (unless it's known to not have a trailing exponent).

4.4 Contributor Notes

`sigfig` was developed with a few **PEP 20** idioms in mind:

- Beautiful is better than ugly.

- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Readability counts.

Refer to **PEP 8** and the Google Python Style Guide for best practices when in doubt and thank you for considering contribution :)

Useful links:

- Python Package Index entry: https://pypi.org/project/sigfig/
- Source Code: https://github.com/drakegroup/sigfig/

Please direct any comments/suggestions/feedback/bugs to mike.busuttil@gmail.com and valdezt@gmail.com

Thanks for downloading :)

Index

P

Python Enhancement Proposals
    PEP 20, 15
    PEP 8, 16