eGenix.com

mxNumber

Fast High-Precision Number Types for Python

Version 3.0

Contents

1. Introduction

mxNumber was created to experiment with rational numbers and to explore means of dealing with decimal number types in database interfacing.

Since the *GNU Multi-Precision Library (GMP)* already has all these number types and also provides what people want most when it comes to numbers: precision and speed, wrapping these as Python types seemed a natural approach.

Unfortunately, GMP 3.1 - the current version of GMP at the time the extension was developed - was only available for Unix platforms and MacOS, but there was no reliable port for Windows.

As this was a show-stopper, we decided to port GMP 3.1 to Windows, which proved to be quite challenging. The gmp31.dll is included in the Windows version of mxNumber. The source archive with the changes and the DLL is available on our web-site as *gmp-3.1.1.win32.zip*. Since the GMP library is LGPLed, we distribute the changes under the LGPL as well.

GMP 4 and later are available for Windows as well, so this is no longer an issue.

Note:
We are currently not continuing the development of this extension, but still support it.

This documentation is still incomplete.

Please still consider the package experimental.

2. Available Number Types

The mxNumber package defines the following number types and implements most interoperability features needed to use these as if they were native Python number types:

Integer

This is an arbitrary precision integer object (like `longs` in Python) based on the GMP `mpz` type.

Rational

This is an arbitrary precision rational object based on the GMP `mpq` type. It uses two `Integer` objects for numerator and denominator which are always normalized (they don't have common factors except 1).

Float

This is a variable precision float object based on the GMP `mpf` type. The precision can be defined at creation time. Floats created in numeric operations use the packages current default precision as basis.

2.1 Conversion from and to other formats

TBD

2.2 Rounding errors

TBD

2.3 Immutability

One other thing to keep in mind when working with mx.Number objects is that they are immutable (like tuples). Once an object is created you can not change its value. Instead, you will have to create a new object with modified values.

The advantage of having immutable objects is that they can be used as dictionary keys and cached in various ways.

2.4 Interaction with other types

mx.Number objects can be compared and hashed, making them compatible to the dictionary implementation Python uses (they can be used as keys).

The copy protocol, standard arithmetic and `pickle()` are also supported.

2.5 String formats

TBD

2.6 Speed and Memory

TBD

2.7 Arithmetic & Coercion

TBD

The different types of this package are coerced in the following ways (whereever possible):

```
                 mx.Number.Float
                       ^
                       |
       --------> Python float
      |                ^
      |                |
      |         mx.Number.Rational
      |                ^
      |                |
Python long --> mx.Number.Integer
      ^                ^
      |                |
       --------  Python integer
```

3. mx.Number Objects

The package provides the following data structures for working with numeric values. These are:

- *Integer* for storing arbitrary precision whole numbers,
- *Rational* for storing exact rational numbers with infinite precision,
- *Float* for storing configurable precision floating point values

3.1 mx.Number.Integer Object

The Integer object is an interface to the GMP `mpz` number type.

Integers can store arbitrary precision whole numbers.

3.1.1 Integer Object Constructors

`Integer(value)`

Constructs an Integer instance from the given value.

value can be a Python integer, string, float, long or another Integer object.

3.1.2 Integer Object Methods

An `Integer` object has the following methods.

`.copy([memo])`

Return a new reference for the instance. This function is used for the copy-protocol. Real copying doesn't take place, since the instances are immutable.

`.even()`

True iff number is even.

`.factorial()`

Return the factorial of the number.

`.gcd(other)`

Return the (positive) GCD of number and other.

`.hamdist(other)`

Return the Hamming Distance between number and other. Both values must be positive.

`.has_root(n)`

Return 1/0 iff number has an (exact) n-th root.

`.is_perfect_power()`

True iff number is a perfect power.

`.is_perfect_square()`

True iff number is a perfect square.

`.jacobi()`

Return the Jacobi symbol for number.

`.lcm(other)`

Return the (positive) LCM of number and other.

`.legendre()`

Return the Legendre symbol for number.

`.odd()`

True iff number is odd.

`.over(k)`

Return the binomial coefficient number over k.

`.popcount()`

Return the population count for number. Number must be positive.

`.prime(reps)`

Return 1 if number is a prime, 2 if number is probably prime and 3 if number surely prime according to the Miller-Rabin test. 0 is returned for non-primes. Higher values for reps increase the probability.

`.root(n)`

Return the (truncated) n-th root of the number.

`.sign()`

Return the sign of the number.

`.sqrt()`

Return the square root of the number.

3.1.3 Integer Object Attributes

`Integer` objects currently don't have attributes.

3.2 mx.Number.Rational Object

The Rational object is an interface to the GMP `mpq` number type.

Rationals can store exact rational numbers with infinite precision.

3.2.1 Rational Object Constructors

`Rational(value[,denominator])`

Constructs a Rational instance from the given value. If denominator is given, value is interpreted as numerator.

value and denominator can be Python integers, strings, floats, longs or Integer objects.

`FareyRational(value, maxden)`

Returns a Rational-object reflecting the given value and using maxden as maximum denominator.

3.2.2 Rational Object Methods

An `Rational` object has the following methods.

`.copy([memo])`

Return a new reference for the instance. This function is used for the copy-protocol. Real copying doesn't take place, since the instances are immutable.

`.format(base, precision=0)`

Return a string representing the Rational in the given base.

For base10, a precision value >= 0 will return an approximate decimal point representation of the Rational, while setting precision to 0 causes the 'nominator/denominator' format to be used. precision has no meaning for non-base10 values.

Note that representation using the decimal point are only accurate to approximately 17 digits (C doubles are used for the formatting).

`.sign()`

Return the sign of the number.

3.2.3 Rational Object Attributes

An `Rational` object has the following attributes.

`.denominator`

Denominator Integer-object of the Rational.

`.numerator`

Numerator Integer-object of the Rational.

3.3 mx.Number.Float Object

The Float object is an interface to the GMP `mpf` number type.

Floats can store configurable precision floating point values.

3.3.1 Float Object Constructors

`Float(value[,precision=64])`

Constructs a Float instance from the given value.

precision gives the number of bits which should at least be used to store the floating point value.

3.3.2 Float Object Methods

A `Float` object has the following methods.

`.sign()`

Return the sign of the Float.

`.format(base, precision=0)`

Return a string representing the Float.

precision defines the maximum number of significant digits to use, 0 means to let the implementation choose the value depending on the Float's storage precision.

3.3.3 Float Attributes

A `Float` object has the following attributes.

`.precision`

The number of bits used by the object to store the floating point value.

4. mx.Number Constants

The package defines these constants:

`Error`

This exception will be raised for problems related to the types. Exceptions will normally only be raised by functions, methods or arithmetic operations.

`IntegerType, RationalType, FloatType`

The type objects for the objects.

`mxNumberAPI`

The C API wrapped by a C object. See mxNumber.h for details.

5. mx.Number Functions

The package currently does not define any additional functions.

6. Examples of Use

TBD

This snippet demonstrates some of the possible interactions of mxNumber types and Python number types:

```
>>> from mx.Number import *

>>> # To be written...
```

More examples will appear in the `Examples` subdirectory of the package.

7. mx.Number Python C-API

mxNumber exposes a Python C-API which can easily be used by other Python extensions. Please have look at the file `mxNumber.h` for details.

To access the module, do the following (note the similarities with Python's way of accessing functions from a module):

```
#include "mxNumber.h"

...
    PyObject *v;

    /* Import the mxNumber module */
    if (mxNumber_ImportModuleAndAPI())
   goto onError;

    /* Access functions from the exported C API through mxNumber */
    v = mxNumber.Integer_FromString("123");
    if (!v)
   goto onError;

    /* Type checking */
    if (mxNumber_Check(v))
        printf("Works.\n");

    Py_DECREF(v);
...
```

8. Package Structure

```
[Number]
       Doc/
       [Examples]
       [mxNumber]
              win32/
              test.py
       LazyModule.py
       Number.py
```

Names with trailing / are plain directories, ones with []-brackets are Python packages, ones with ".py" extension are Python submodules.

The package imports all symbols from the extension module and also registers the types so that they become compatible to the pickle and copy mechanisms in Python.

9. Support

eGenix.com is providing commercial support for this package. If you are interested in receiving information about this service please see the *eGenix.com Support Conditions*.

10. Copyright & License

EGENIX.COM PUBLIC LICENSE AGREEMENT

Version 1.1.0

This license agreement is based on the Python CNRI License Agreement, a widely accepted open-source license.

1. Introduction

This "License Agreement" is between eGenix.com Software, Skills and Services GmbH ("eGenix.com"), having an office at Pastor-Loeh-Str. 48, D-40764 Langenfeld, Germany, and the Individual or Organization ("Licensee") accessing and otherwise using this software in source or binary form and its associated documentation ("the Software").

2. License

Subject to the terms and conditions of this eGenix.com Public License Agreement, eGenix.com hereby grants Licensee a non-exclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use the Software alone or in any derivative version, provided, however, that the eGenix.com Public License Agreement is retained in the Software, or in any derivative version of the Software prepared by Licensee.

3. NO WARRANTY

eGenix.com is making the Software available to Licensee on an "AS IS" basis. SUBJECT TO ANY STATUTORY WARRANTIES WHICH CAN NOT BE EXCLUDED, EGENIX.COM MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, EGENIX.COM MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

4. LIMITATION OF LIABILITY

EGENIX.COM SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF THE SOFTWARE FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR OTHER PECUNIARY LOSS) AS A RESULT OF USING, MODIFYING OR

DISTRIBUTING THE SOFTWARE, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE EXCLUSION OR LIMITATION MAY NOT APPLY TO LICENSEE.

5. Termination

This License Agreement will automatically terminate upon a material breach of its terms and conditions.

6. Third Party Rights

Any software or documentation in source or binary form provided along with the Software that is associated with a separate license agreement is licensed to Licensee under the terms of that license agreement. This License Agreement does not apply to those portions of the Software. Copies of the third party licenses are included in the Software Distribution.

7. General

Nothing in this License Agreement affects any statutory rights of consumers that cannot be waived or limited by contract.

Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between eGenix.com and Licensee.

If any provision of this License Agreement shall be unlawful, void, or for any reason unenforceable, such provision shall be modified to the extent necessary to render it enforceable without losing its intent, or, if no such modification is possible, be severed from this License Agreement and shall not affect the validity and enforceability of the remaining provisions of this License Agreement.

This License Agreement shall be governed by and interpreted in all respects by the law of Germany, excluding conflict of law provisions. It shall not be governed by the United Nations Convention on Contracts for International Sale of Goods.

This License Agreement does not grant permission to use eGenix.com trademarks or trade names in a trademark sense to endorse or promote products or services of Licensee, or any third party.

The controlling language of this License Agreement is English. If Licensee has received a translation into another language, it has been provided for Licensee's convenience only.

8. Agreement

By downloading, copying, installing or otherwise using the Software, Licensee agrees to be bound by the terms and conditions of this License Agreement.

For question regarding this License Agreement, please write to:

eGenix.com Software, Skills and Services GmbH

Pastor-Loeh-Str. 48

D-40764 Langenfeld

Germany