The mpFormula CPython Library and Toolbox Manual

Dietrich Hadler

July 2015
Version 0.1, Alpha 1 (Pre-Release)

Original authors of the mpmath documentation: Frederik Johanson and mpmath contributors
Original authors of the Python documentation: the Python development Team
Subsequent modifications: Dietrich Hadler

Copyright © 2015 Dietrich Hadler

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation.

Preface

The mpFormula CPython library provides a comprehensive set of real and complex functions in multiprecision arithmetic. For a subset of functions there is also support for decimal and interval arithmetic.

It is intended as a companion to the mpFormulaPy library, with a focus on speed and ease of development of additional numerical multiprecision Python code .

The current version is 0.1, alpha1 (pre-release), and much of the planned functionality is still missing.

The mpFormulaPy Library and Toolbox would not exist without the many authors and contributors of the underlying libraries. They are acknowledged in appendix C.

Dietrich Hadler

Contents

Preface i

Contents ii

List of Tables vi

List of Figures vii

I Getting Started 1

1 Introduction 2

1.1 Overview: Features and Setup . . . 2
1.1.1 Features . . . 2
1.1.2 The mpFormula CPython Library . . . 2
1.1.3 Installation . . . 3

1.2 License . . . 3

1.3 No Warranty . . . 3

1.4 Related Software . . . 3

2 Tutorials 4

2.1 Why multi-precision arithmetic? . . . 4
2.1.1 Example 1: Sums . . . 4
2.1.2 Example 2: Standard Deviation . . . 4
2.1.3 Example 3: Overflow and underflow . . . 4
2.1.4 Example 4: Polynomials . . . 5
2.1.5 Example 5: Trigonometric Functions . . . 5
2.1.6 Example 6: Logarithms and Exponential Functions . . . 5
2.1.7 Example 7: Linear Algebra . . . 5

3 Python: Built-in numerical types 6

3.1 Truth Value Testing . . . 6

3.2 Boolean Operations: and, or, not . . . 6
3.3 Comparisons . . . 7
3.4 Numeric Types - int, float, complex . . . 7
3.5 Long integers . . . 9
3.5.1 Bitwise Operations on Integer Types . . . 9
3.5.2 Additional Methods on Integer Types . . . 9
3.5.3 Additional Methods on Float . . . 11
3.6 Fractions . . . 13
3.6.1 Properties . . . 14
3.6.2 Methods . . . 14
3.7 Decimals . . . 16
3.7.1 Overview . . . 16
3.7.2 Quick-start Tutorial . . . 17
3.7.3 Decimal objects . . . 20
3.7.4 Methods . . . 22
3.7.5 Context objects . . . 29
3.7.6 Context Methods . . . 31
3.7.7 Constants . . . 37
3.7.8 Rounding modes . . . 38
3.7.9 Signals . . . 38
3.7.10 Floating Point Notes . . . 40
3.7.11 Working with threads . . . 42
3.7.12 Recipes . . . 42
3.7.13 Decimal FAQ . . . 45

4 gmpy2 49

4.1 Overview of gmpy2 . . . 49
4.1.1 Downloading and installing Gmpy2 . . . 49
4.1.2 Running Tests . . . 49
4.1.3 Tutorial . . . 50
4.1.4 Miscellaneous gmpy2 Functions . . . 51
4.2 Multiple-precision Integers . . . 53
4.2.1 mpz Methods . . . 53
4.2.2 mpz Functions . . . 54
4.3 Multiple-precision Integers (Advanced topics) . . . 61
4.3.1 The xmpz type . . . 61
4.3.2 Pseudoprimes . . . 63
4.3.3 Lucas Sequences . . . 65
4.4 Multiple-precision Rationals . . . 67
4.4.1 mpq Methods . . . 67
4.4.2 mpq Attributes . . . 67
4.4.3 mpq Functions . . . 67
4.5 Multiple-precision Reals . . . 69

4.5.1 Contexts . . . 69
4.5.2 Context Attributes . . . 70
4.5.3 Context Methods . . . 73
4.5.4 Contexts and the with statement . . . 73
4.5.5 mpfr Methods . . . 74
4.5.6 mpfr Attributes . . . 74
4.5.7 mpfr Functions . . . 75
4.5.8 mpfr Formatting . . . 84

4.6 Multiple-precision Complex . . . 86
4.6.1 mpc Methods . . . 87
4.6.2 mpc Attributes . . . 87
4.6.3 mpc Functions . . . 87
4.6.4 mpc Formatting . . . 91

II Appendices 93

A Python 94

A.1 Overview . . . 94

A.2 CPython . . . 95
A.2.1 Downloading and installing CPython 2.7 . . . 95
A.2.2 Using the C-API . . . 95
A.2.3 Interfaces to the C family of languages . . . 96
A.2.4 Cython: C extensions for the Python language . . . 101
A.2.5 A Windows-specific interface: using COM . . . 101

B Building the library and toolbox 103

C Acknowledgements 104

C.1 Contributors to libraries used in the numerical routines . . . 104
C.1.1 Contributors to mpMath . . . 104
C.1.2 Contributors to gmpy2 . . . 104

D Licenses 105

D.1 GNU Licenses . . . 105
D.1.1 GNU General Public License, Version 2 . . . 105
D.1.2 GNU Library General Public License, Version 2 . . . 111
D.1.3 GNU Lesser General Public License, Version 3 . . . 117
D.1.4 GNU General Public License, Version 3 . . . 119
D.1.5 GNU Free Documentation License, Version 1.3 . . . 128

D.2 Other Licenses . . . 134
D.2.1 Mozilla Public License, Version 2.0 . . . 134
D.2.2 New BSD License (for mpMath) . . . 139
D.2.3 MIT License . . . 140

D.2.4 Excel-DNA License 141

Appendices **94**

III Back Matter **142**

Bibliography **143**

Nomenclature **145**

Index **147**

List of Tables

List of Figures

Part I

Getting Started

Chapter 1

Introduction

1.1 Overview: Features and Setup

1.1.1 Features

The mpFormulaPy distribution consists of two parts: the mpFormulaPy Library and the mpFormulaPy Toolbox.

1.1.2 The mpFormula CPython Library

The mpFormulaPy Library is a collection of numerical functions and procedures in multiprecision arithmetic. It is intended to be usable on multiple platforms (i.e. platforms supported by a recent version of Python) and is provided in the form of source code in Python.
The following numerical types are supported:

- The conventional double (64 bit) precision binary floating point type (double in C).
- The mpf arbitray precision binary floating point type of the mpmath library.
- The mpi arbitray precision interval arithmetic binary floating point type of the mpmath library.
- The mpc arbitray precision complex binary floating point type of the mpmath library.
- The mpci arbitray precision complex interval arithmetic binary floating point type of the mpmath library.
- The long arbitray precision integer type of the Python library.
- The Fraction arbitray precision rational type of the Python library.
- The Decimal arbitray precision decimal floating point type of the Python library.

All of these types are available as real and complex scalars. vectors, and matrices.

The mpFormulaPy Library is based on mpmath Johansson *et al.* (2013), and the standard Python Library, including fractions.py and decimal.py.

1.1.3 Installation

The mpFormula CPython Library and Toolbox can be downloaded from

http://mpFormula.github.io/CPython/.

Unzip the downloaded file in a directory for which you have write-access.

1.2 License

The mpFormula CPython Library and Toolbox is free software. It is licensed under the GNU Lesser General Public License (LGPL), Version 3 (see appendix D.1.3). The manual for the mpFormula CPython Toolbox (this document) is licensed under the GNU Free Documentation License, Version 1.3 (see appendix D.1.5).

1.3 No Warranty

There is no warranty. See the GNU Lesser General Public License, Version 3 (see appendix D.1.3) for details.

1.4 Related Software

The mpFormulaC Library and Toolbox provides fast multiprecision routines written in C, with interfaces to CPython, R, .NET and COM. It can be downloaded from
http://mpFormula.github.io/C/.

Chapter 2

Tutorials

2.1 Why multi-precision arithmetic?

An introduction to the problems of rounding errors and catastrophic cancellation can be found in Goldberg (1991). Excellent reference texts are Higham (2002) and Higham (2009).
In the following paragraphs we will give a few examples of how widely used programs like MS Excel or Libreoffice Calc can give wrong results due to the fact that they are using double precision arithmetic and not multi-precision arithmetic

2.1.1 Example 1: Sums

Sums are often calculated exactly if all summands have an exact representation. If this is not the case, results can be unpredictable. In MS Excel, the formula

```
=SUM(10000000000,-16000000000,6000000000)
```

will give the correct result `0`, but the analogous formula

```
=SUM(1E+40,-1.6E+40,6E+39)
```

returns `1.20893E+24` instead of the correct result `0`.

2.1.2 Example 2: Standard Deviation

Like sums, variances and standard deviations are often calculated exactly if all arguments have an exact representation. If this is not the case, results can again be unpredictable. In MS Excel, the formula

```
=VAR(1E+30,1E+30,1E+30)
```

returns `2.97106E+28` instead of the correct result `0`, which should be the obvious results since all arguments are the same.

2.1.3 Example 3: Overflow and underflow

In many situations where the final result is representable in double precision, some of the interim results cause overflow or underflow. A popular example is the function $f(x, y) = \sqrt{x^2 + y^2}$. With $x = 3 \cdot 10^{300}$ and $y = 4 \cdot 10^{300}$ the result $f(x, y) = 5 \cdot 10^{300}$ is representable in double precision, but the (naive) calculation will overflow.

2.1.4 Example 4: Polynomials

Consider the following example Cuyt *et al.* (2001):

For $a = 77617$ and $b = 33096$, calculate

$$Y = 333.75b^6 + a^2(11a^2b^2 - b^6 - 121b^4 - 2) + 5.5b^8 + \frac{a}{2b} \tag{2.1.1}$$

The correct result is $Y = -54767/66192 = -0.827396...$

2.1.5 Example 5: Trigonometric Functions

Trigonometric functions are sensitive to small perturbations.

In double precision and binary floating point arithmetic, the tangent of $x = 1.57079632679489$ is calculated as $\tan(x) = 1.48752 \cdot 10^{14}$, whereas the correct result is $\tan(x) = 1.51075 \cdot 10^{14}$. This amounts to an absolute error of $2.32287 \cdot 10^{12}$ and a relative error of 1.54%.

There are also limits on the range of arguments, e.g. $\sin(10^8)$ returns the value 0.931639... (with an relative error of $-6.22776 \cdot 10^{-13}$), whereas $\sin(10^9)$ returns an invalid result (the exact result is 0.545843...)

2.1.6 Example 6: Logarithms and Exponential Functions

Consider the following example (Ghazi *et al.*, 2010) :
Determine 10 decimal digits of the constant

$$Y = 173746a + 94228b - 78487c, \quad \text{where} \tag{2.1.2}$$

$$a = \sin(10^{22}), b = \ln(17.1), c = \exp(0.42). \tag{2.1.3}$$

The expected result is $Y = -1.341818958 \cdot 10^{-12}$.

2.1.7 Example 7: Linear Algebra

2.1.7.1 Linear Solver

The following example is from Hofschuster & Krämer (2004):
We want to solve the (ill-conditioned) system of linear equations $Ax = b$ with

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} = \begin{pmatrix} 64919121 & -159018721 \\ 41869520.5 & -102558961 \end{pmatrix}, b = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \tag{2.1.4}$$

The correct solution is $x_1 = 205117922$, $x_2 = 83739041$.
To solve this 2×2 system numerically we first use the well known formulas

$$x_1 = \frac{a_{22}}{a_{11}a_{22} - a_{12}a_{21}}, \quad x_2 = \frac{-a_{21}}{a_{11}a_{22} - a_{12}a_{21}}, \tag{2.1.5}$$

Calculating this directly in double precision gives the following wrong result:
$x_1 = 102558961$, $x_2 = 41869520.5$

Chapter 3

Python: Built-in numerical types

The following sections describe the standard types that are built into the interpreter.
The principal built-in types are numerics, sequences, mappings, classes, instances and exceptions.
Some collection classes are mutable. The methods that add, subtract, or rearrange their members in place, and do not return a specific item, never return the collection instance itself but None.
Some operations are supported by several object types; in particular, practically all objects can be compared, tested for truth value, and converted to a string (with the repr() function or the slightly different str() function). The latter function is implicitly used when an object is written by the print() function.

3.1 Truth Value Testing

Any object can be tested for truth value, for use in an if or while condition or as operand of the Boolean operations below. The following values are considered false:

None
False
zero of any numeric type, for example, `0, 0.0, 0j`.
any empty sequence, for example, `'', (), []`.
any empty mapping, for example, `{}`.
instances of user-defined classes, if the class defines a __bool__() or __len__() method, when that method returns the integer zero or bool value False.

All other values are considered true — so objects of many types are always true.
Operations and built-in functions that have a Boolean result always return 0 or False for false and 1 or True for true, unless otherwise stated. (Important exception: the Boolean operations or and and always return one of their operands.)

3.2 Boolean Operations: and, or, not

These are the Boolean operations, ordered by ascending priority:

Operation	Result	Notes
`x or y`	if x is false, then y, else x	(1)
`x and y`	if x is false, then x, else y	(2)
`not x`	if x is false, then True, else False	(3)

Notes: 1.This is a short-circuit operator, so it only evaluates the second argument if the first one is False. 2.This is a short-circuit operator, so it only evaluates the second argument if the first one is True. 3.not has a lower priority than non-Boolean operators, so not a == b is interpreted as not (a == b), and a == not b is a syntax error.

3.3 Comparisons

There are eight comparison operations in Python. They all have the same priority (which is higher than that of the Boolean operations). Comparisons can be chained arbitrarily; for example, x ¡ y ¡= z is equivalent to x ¡ y and y ¡= z, except that y is evaluated only once (but in both cases z is not evaluated at all when x ¡ y is found to be false).
This table summarizes the comparison operations:

Operation	Meaning
`<`	strictly less than
`<=`	less than or equal
`>`	strictly greater than
`>=`	greater than or equal
`==`	equal
`!=`	not equal
`is`	object identity
`is not`	negated object identity

Objects of different types, except different numeric types, never compare equal. Furthermore, some types (for example, function objects) support only a degenerate notion of comparison where any two objects of that type are unequal. The ¡, ¡=, ¿ and ¿= operators will raise a TypeError exception when comparing a complex number with another built-in numeric type, when the objects are of different types that cannot be compared, or in other cases where there is no defined ordering.
Non-identical instances of a class normally compare as non-equal unless the class defines the __eq__() method.
Instances of a class cannot be ordered with respect to other instances of the same class, or other types of object, unless the class defines enough of the methods __lt__(), __le__(), __gt__(), and __ge__() (in general, __lt__() and __eq__() are sufficient, if you want the conventional meanings of the comparison operators).
The behavior of the is and is not operators cannot be customized; also they can be applied to any two objects and never raise an exception.
Two more operations with the same syntactic priority, in and not in, are supported only by sequence types (below).

3.4 Numeric Types - int, float, complex

There are three distinct numeric types: integers, floating point numbers, and complex numbers. In addition, Booleans are a subtype of integers. Integers have unlimited precision. Floating point numbers are usually implemented using double in C; information about the precision and internal representation of floating point numbers for the machine on which your program is running is available in sys.float_info. Complex numbers have a real and imaginary part, which are each a floating point number. To extract these parts from a complex number z, use z.real and z.imag.

(The standard library includes additional numeric types, fractions that hold rationals, and decimal that hold floating-point numbers with user-definable precision.)

Numbers are created by numeric literals or as the result of built-in functions and operators. Unadorned integer literals (including hex, octal and binary numbers) yield integers. Numeric literals containing a decimal point or an exponent sign yield floating point numbers. Appending 'j' or 'J' to a numeric literal yields an imaginary number (a complex number with a zero real part) which you can add to an integer or float to get a complex number with real and imaginary parts.

Python fully supports mixed arithmetic: when a binary arithmetic operator has operands of different numeric types, the operand with the "narrower" type is widened to that of the other, where integer is narrower than floating point, which is narrower than complex. Comparisons between numbers of mixed type use the same rule. [2] The constructors int(), float(), and complex() can be used to produce numbers of a specific type.

All numeric types (except complex) support the following operations, sorted by ascending priority (operations in the same box have the same priority; all numeric operations have a higher priority than comparison operations):

Operation	Result	Notes
x + y	sum of x and y	
x - y	difference of x and y	
x * y	product of x and y	
x / y	quotient of x and y	(1)
x // y	floored quotient of x and y	
x % y	remainder of x / y	(2)
-x	x negated	
+x	x unchanged	
abs(x)	absolute value or magnitude of x	
int(x)	x converted to integer	(3)(6)
float(x)	x converted to floating point	(4)(6)
complex(re, im)	a complex number with real part re, imaginary part im.	
	im defaults to zero.	(6)
c.conjugate()	conjugate of the complex number c	No
divmod(x, y)	the pair (x // y, x % y)	(2)
pow(x, y)	x to the power y	(5)
x ** y	x to the power y	(5)
math.trunc(x)	x truncated to Integral	(7)
round(x[, n])	x rounded to n digits, rounding half to even.	
	If n is omitted, it defaults to 0.	(7)
math.floor(x)	the greatest integral float $\leq$ x	(7)
math.ceil(x)	the least integral float $\geq$ x	(7)

Notes:
1.Also referred to as integer division. The resultant value is a whole integer, though the result's type is not necessarily int. The result is always rounded towards minus infinity: 1//2 is 0, (-1)//2 is -1, 1//(-2) is -1, and (-1)//(-2) is 0.

2.Not for complex numbers. Instead convert to floats using abs() if appropriate.

3.Conversion from floating point to integer may round or truncate as in C; see functions math.floor() and math.ceil() for well-defined conversions.

4.float also accepts the strings ”nan” and ”inf” with an optional prefix ”+” or ”-” for Not a Number (NaN) and positive or negative infinity.

5.Python defines pow(0, 0) and 0 ** 0 to be 1, as is common for programming languages.

6.The numeric literals accepted include the digits 0 to 9 or any Unicode equivalent (code points with the Nd property).
7.Only real types (int and float).
See http://www.unicode.org/Public/6.0.0/ucd/extracted/DerivedNumericType.txt for a complete list of code points with the Nd property.
For additional numeric operations see the math and cmath modules.

3.5 Long integers

3.5.1 Bitwise Operations on Integer Types

Bitwise operations only make sense for integers. Negative numbers are treated as their 2âĂŹs complement value (this assumes a sufficiently large number of bits that no overflow occurs during the operation).
The priorities of the binary bitwise operations are all lower than the numeric operations and higher than the comparisons; the unary operation has the same priority as the other unary numeric operations (+ and -).
This table lists the bitwise operations sorted in ascending priority (operations in the same box have the same priority):

Operation	Result	Notes
`x | y`	bitwise or of x and y	
`x ^ y`	bitwise exclusive or of x and y	
`x & y`	bitwise and of x and y	
`x << n`	x shifted left by n bits	(1)(2)
`x >> n`	x shifted right by n bits	(1)(3)
`~x`	the bits of x inverted	

Notes: 1.Negative shift counts are illegal and cause a ValueError to be raised. 2.A left shift by n bits is equivalent to multiplication by pow(2, n) without overflow check. 3.A right shift by n bits is equivalent to division by pow(2, n) without overflow check.

3.5.2 Additional Methods on Integer Types

The int type implements the numbers.Integral abstract base class. In addition, it provides one more method:

3.5.2.1 int.bit_length()

Return the number of bits necessary to represent an integer in binary, excluding the sign and leading zeros:

```
>>>>>> n = -37
>>> bin(n)
'-0b100101'
>>> n.bit_length()
6
```

More precisely, if x is nonzero, then x.bit_length() is the unique positive integer k such that 2**(k-1) ¡= abs(x) ¡ 2**k. Equivalently, when abs(x) is small enough to have a correctly rounded logarithm, then k = 1 + int(log(abs(x), 2)). If x is zero, then x.bit_length() returns 0.
Equivalent to:

```
def bit_length(self):
s = bin(self)     # binary representation: bin(-37) --> '-0b100101'
s = s.lstrip('-0b') # remove leading zeros and minus sign
return len(s)     # len('100101') --> 6
```

3.5.2.2 int.to_bytes

New in version 3.1.
int.to_bytes(length, byteorder, *, signed=False) Return an array of bytes representing an integer.

```
>>>>>> (1024).to_bytes(2, byteorder='big')
b'\x04\x00'
>>> (1024).to_bytes(10, byteorder='big')
b'\x00\x00\x00\x00\x00\x00\x00\x00\x04\x00'
>>> (-1024).to_bytes(10, byteorder='big', signed=True)
b'\xff\xff\xff\xff\xff\xff\xff\xff\xfc\x00'
>>> x = 1000
>>> x.to_bytes((x.bit_length() // 8) + 1, byteorder='little')
b'\xe8\x03'
```

The integer is represented using length bytes. An OverflowError is raised if the integer is not representable with the given number of bytes.
The byteorder argument determines the byte order used to represent the integer. If byteorder is "big", the most significant byte is at the beginning of the byte array. If byteorder is "little", the most significant byte is at the end of the byte array. To request the native byte order of the host system, use sys.byteorder as the byte order value.
The signed argument determines whether twoâĂŹs complement is used to represent the integer. If signed is False and a negative integer is given, an OverflowError is raised. The default value for signed is False.

3.5.2.3 int.from_bytes

New in version 3.2.
classmethod int.from_bytes(bytes, byteorder, *, signed=False) Return the integer represented by the given array of bytes.

```
>>>>>> int.from_bytes(b'\x00\x10', byteorder='big')
16
>>> int.from_bytes(b'\x00\x10', byteorder='little')
```

```
4096
>>> int.from_bytes(b'\xfc\x00', byteorder='big', signed=True)
-1024
>>> int.from_bytes(b'\xfc\x00', byteorder='big', signed=False)
64512
>>> int.from_bytes([255, 0, 0], byteorder='big')
16711680
```

The argument bytes must either be a bytes-like object or an iterable producing bytes.
The byteorder argument determines the byte order used to represent the integer. If byteorder is "big", the most significant byte is at the beginning of the byte array. If byteorder is "little", the most significant byte is at the end of the byte array. To request the native byte order of the host system, use sys.byteorder as the byte order value.
The signed argument indicates whether twoâĂŹs complement is used to represent the integer.

3.5.3 Additional Methods on Float

The float type implements the numbers.Real abstract base class. float also has the following additional methods.

3.5.3.1 float.as_integer_ratio()

Return a pair of integers whose ratio is exactly equal to the original float and with a positive denominator. Raises OverflowError on infinities and a ValueError on NaNs.

3.5.3.2 float.is_integer()

Return True if the float instance is finite with integral value, and False otherwise:

```
>>>>>> (-2.0).is_integer()
True
>>> (3.2).is_integer()
False
```

Two methods support conversion to and from hexadecimal strings. Since Python's floats are stored internally as binary numbers, converting a float to or from a decimal string usually involves a small rounding error. In contrast, hexadecimal strings allow exact representation and specification of floating-point numbers. This can be useful when debugging, and in numerical work.

3.5.3.3 float.hex()

Return a representation of a floating-point number as a hexadecimal string. For finite floating-point numbers, this representation will always include a leading 0x and a trailing p and exponent.

3.5.3.4 float.fromhex(s)

Class method to return the float represented by a hexadecimal string s. The string s may have leading and trailing whitespace.

Note that float.hex() is an instance method, while float.fromhex() is a class method.
A hexadecimal string takes the form:
[sign] ['0x'] integer ['.' fraction] ['p' exponent]

where the optional sign may by either + or -, integer and fraction are strings of hexadecimal digits, and exponent is a decimal integer with an optional leading sign. Case is not significant, and there must be at least one hexadecimal digit in either the integer or the fraction. This syntax is similar to the syntax specified in section 6.4.4.2 of the C99 standard, and also to the syntax used in Java 1.5 onwards. In particular, the output of float.hex() is usable as a hexadecimal floating-point literal in C or Java code, and hexadecimal strings produced by C's %a format character or Java's Double.toHexString are accepted by float.fromhex().

Note that the exponent is written in decimal rather than hexadecimal, and that it gives the power of 2 by which to multiply the coefficient. For example, the hexadecimal string 0x3.a7p10 represents the floating-point number (3 + 10./16 + 7./16**2) * 2.0**10, or 3740.0:

```
>>>>>> float.fromhex('0x3.a7p10')
3740.0
```

Applying the reverse conversion to 3740.0 gives a different hexadecimal string representing the same number:

```
>>>>>> float.hex(3740.0)
'0x1.d380000000000p+11'
```

3.6 Fractions

The fractions module provides support for rational number arithmetic.
A Fraction instance can be constructed from a pair of integers, from another rational number, or from a string.

class fractions.Fraction(numerator=0, denominator=1)
class fractions.Fraction(other_fraction)
class fractions.Fraction(float)
class fractions.Fraction(decimal)
class fractions.Fraction(string)

The first version requires that numerator and denominator are instances of numbers.Rational and returns a new Fraction instance with value numerator/denominator. If denominator is 0, it raises a ZeroDivisionError.

The second version requires that other_fraction is an instance of numbers.Rational and returns a Fraction instance with the same value.

The next two versions accept either a float or a decimal.Decimal instance, and return a Fraction instance with exactly the same value. Note that due to the usual issues with binary floating-point (see Floating Point Arithmetic: Issues and Limitations), the argument to Fraction(1.1) is not exactly equal to 11/10, and so Fraction(1.1) does not return Fraction(11, 10) as one might expect. (But see the documentation for the limit_denominator() method below.)

The last version of the constructor expects a string or unicode instance. The usual form for this instance is:
[sign] numerator ['/' denominator]
where the optional sign may be either '+' or '-' and numerator and denominator (if present) are strings of decimal digits. In addition, any string that represents a finite value and is accepted by the float constructor is also accepted by the Fraction constructor. In either form the input string may also have leading and/or trailing whitespace. Here are some examples:

The corresponding code is:

```
>>>>>> from fractions import Fraction
>>> Fraction(16, -10)
Fraction(-8, 5)
>>> Fraction(123)
Fraction(123, 1)
>>> Fraction()
Fraction(0, 1)
>>> Fraction('3/7')
Fraction(3, 7)
>>> Fraction(' -3/7 ')
Fraction(-3, 7)
>>> Fraction('1.414213 \t\n')
Fraction(1414213, 1000000)
>>> Fraction('-.125')
Fraction(-1, 8)
>>> Fraction('7e-6')
Fraction(7, 1000000)
>>> Fraction(2.25)
```

```
Fraction(9, 4)
>>> Fraction(1.1)
Fraction(2476979795053773, 2251799813685248)
>>> from decimal import Decimal
>>> Fraction(Decimal('1.1'))
Fraction(11, 10)
```

The Fraction class inherits from the abstract base class numbers.Rational, and implements all of the methods and operations from that class. Fraction instances are hashable, and should be treated as immutable. In addition, Fraction has the following properties and methods:
Changed in version 3.2: The Fraction constructor now accepts float and decimal.Decimal instances.

3.6.1 Properties

3.6.1.1 numerator

Numerator of the Fraction in lowest term.

3.6.1.2 denominator

Denominator of the Fraction in lowest term.

3.6.2 Methods

3.6.2.1 from_float(flt)

This class method constructs a Fraction representing the exact value of flt, which must be a float. Beware that Fraction.from_float(0.3) is not the same value as Fraction(3, 10)
Note: From Python 3.2 onwards, you can also construct a Fraction instance directly from a float.

3.6.2.2 from_decimal(dec)

This class method constructs a Fraction representing the exact value of dec, which must be a decimal.Decimal instance.
Note: From Python 3.2 onwards, you can also construct a Fraction instance directly from a decimal.Decimal instance.

3.6.2.3 limit_denominator()

limit_denominator(max_denominator=1000000) Finds and returns the closest Fraction to self that has denominator at most max_denominator. This method is useful for finding rational approximations to a given floating-point number:

```
>>>>>> from fractions import Fraction
>>> Fraction('3.1415926535897932').limit\_denominator(1000)
Fraction(355, 113)
```

or for recovering a rational number thatâĂŹs represented as a float:

```
>>>>>> from math import pi, cos
>>> Fraction(cos(pi/3))
```

```
Fraction(4503599627370497, 9007199254740992)
>>> Fraction(cos(pi/3)).limit\_denominator()
Fraction(1, 2)
>>> Fraction(1.1).limit\_denominator()
Fraction(11, 10)
```

3.6.2.4 __floor__()

Returns the greatest int ¡= self. This method can also be accessed through the math.floor() function:

```
>>>>>> from math import floor
>>> floor(Fraction(355, 113))
3
```

3.6.2.5 __ceil__()

Returns the least int ¿= self. This method can also be accessed through the math.ceil() function.

3.6.2.6 __round__()

__round__()
__round__(ndigits)
The first version returns the nearest int to self, rounding half to even. The second version rounds self to the nearest multiple of Fraction(1, 10**ndigits) (logically, if ndigits is negative), again rounding half toward even. This method can also be accessed through the round() function.

3.6.2.7 fractions.gcd(a, b)

Return the greatest common divisor of the integers a and b. If either a or b is nonzero, then the absolute value of gcd(a, b) is the largest integer that divides both a and b. gcd(a,b) has the same sign as b if b is nonzero; otherwise it takes the sign of a. gcd(0, 0) returns 0.

3.7 Decimals

3.7.1 Overview

The decimal module provides support for fast correctly-rounded decimal floating point arithmetic. It offers several advantages over the float datatype:

Decimal "is based on a floating-point model which was designed with people in mind, and necessarily has a paramount guiding principle – computers must provide an arithmetic that works in the same way as the arithmetic that people learn at school." – excerpt from the decimal arithmetic specification.

Decimal numbers can be represented exactly. In contrast, numbers like 1.1 and 2.2 do not have exact representations in binary floating point. End users typically would not expect 1.1 + 2.2 to display as 3.3000000000000003 as it does with binary floating point.

The exactness carries over into arithmetic. In decimal floating point, 0.1 + 0.1 + 0.1 - 0.3 is exactly equal to zero. In binary floating point, the result is 5.5511151231257827e-017. While near to zero, the differences prevent reliable equality testing and differences can accumulate. For this reason, decimal is preferred in accounting applications which have strict equality invariants.

The decimal module incorporates a notion of significant places so that 1.30 + 1.20 is 2.50. The trailing zero is kept to indicate significance. This is the customary presentation for monetary applications. For multiplication, the "schoolbook" approach uses all the figures in the multiplicands. For instance, 1.3 * 1.2 gives 1.56 while 1.30 * 1.20 gives 1.5600.

Unlike hardware based binary floating point, the decimal module has a user alterable precision (defaulting to 28 places) which can be as large as needed for a given problem:

```
>>> from decimal import *
>>> getcontext().prec = 6
>>> Decimal(1) / Decimal(7)
Decimal('0.142857')
>>> getcontext().prec = 28
>>> Decimal(1) / Decimal(7)
Decimal('0.1428571428571428571428571429')
```

Both binary and decimal floating point are implemented in terms of published standards. While the built-in float type exposes only a modest portion of its capabilities, the decimal module exposes all required parts of the standard. When needed, the programmer has full control over rounding and signal handling. This includes an option to enforce exact arithmetic by using exceptions to block any inexact operations.

The decimal module was designed to support "without prejudice, both exact unrounded decimal arithmetic (sometimes called fixed-point arithmetic) and rounded floating-point arithmetic." – excerpt from the decimal arithmetic specification.

The module design is centered around three concepts: the decimal number, the context for arithmetic, and signals.

A decimal number is immutable. It has a sign, coefficient digits, and an exponent. To preserve significance, the coefficient digits do not truncate trailing zeros. Decimals also include special values such as Infinity, -Infinity, and NaN. The standard also differentiates -0 from +0.

The context for arithmetic is an environment specifying precision, rounding rules, limits on exponents, flags indicating the results of operations, and trap enablers which determine whether signals are treated as exceptions. Rounding options include ROUND_CEILING, ROUND_DOWN, ROUND_FLOOR, ROUND_HALF_DOWN, ROUND_HALF_EVEN, ROUND_HALF_UP, ROUND_UP, and ROUND_05UP.

Signals are groups of exceptional conditions arising during the course of computation. Depending on the needs of the application, signals may be ignored, considered as informational, or treated as exceptions. The signals in the decimal module are: Clamped, InvalidOperation, DivisionByZero, Inexact, Rounded, Subnormal, Overflow, Underflow and FloatOperation.

For each signal there is a flag and a trap enabler. When a signal is encountered, its flag is set to one, then, if the trap enabler is set to one, an exception is raised. Flags are sticky, so the user needs to reset them before monitoring a calculation.

See also:
IBM's General Decimal Arithmetic Specification, The General Decimal Arithmetic Specification.
IEEE standard 854-1987, Unofficial IEEE 854 Text.

3.7.2 Quick-start Tutorial

The usual start to using decimals is importing the module, viewing the current context with getcontext() and, if necessary, setting new values for precision, rounding, or enabled traps:

```
>>> from decimal import *
>>> getcontext()
Context(prec=28, rounding=ROUND\_HALF\_EVEN, Emin=-999999, Emax=999999,
capitals=1, clamp=0, flags=[], traps=[Overflow, DivisionByZero,
InvalidOperation])

>>> getcontext().prec = 7    # Set a new precision
```

Decimal instances can be constructed from integers, strings, floats, or tuples. Construction from an integer or a float performs an exact conversion of the value of that integer or float. Decimal numbers include special values such as NaN which stands for âĂIJNot a numberâĂİ, positive and negative Infinity, and -0:

```
>>> getcontext().prec = 28
>>> Decimal(10)
Decimal('10')
>>> Decimal('3.14')
Decimal('3.14')
>>> Decimal(3.14)
Decimal('3.140000000000000124344978758017532527446746826171875')
>>> Decimal((0, (3, 1, 4), -2))
Decimal('3.14')
>>> Decimal(str(2.0 ** 0.5))
Decimal('1.4142135623730951')
>>> Decimal(2) ** Decimal('0.5')
Decimal('1.4142135623730950488016887242')
>>> Decimal('NaN')
Decimal('NaN')
```

```
>>> Decimal('-Infinity')
Decimal('-Infinity')
```

If the FloatOperation signal is trapped, accidental mixing of decimals and floats in constructors or ordering comparisons raises an exception:

```
>>> c = getcontext()
>>> c.traps[FloatOperation] = True
>>> Decimal(3.14)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
decimal.FloatOperation: [<class 'decimal.FloatOperation'>]
>>> Decimal('3.5') < 3.7
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
decimal.FloatOperation: [<class 'decimal.FloatOperation'>]
>>> Decimal('3.5') == 3.5
True
```

New in version 3.3.
The significance of a new Decimal is determined solely by the number of digits input. Context precision and rounding only come into play during arithmetic operations.

```
>>> getcontext().prec = 6
>>> Decimal('3.0')
Decimal('3.0')
>>> Decimal('3.1415926535')
Decimal('3.1415926535')
>>> Decimal('3.1415926535') + Decimal('2.7182818285')
Decimal('5.85987')
>>> getcontext().rounding = ROUND\_UP
>>> Decimal('3.1415926535') + Decimal('2.7182818285')
Decimal('5.85988')
```

If the internal limits of the C version are exceeded, constructing a decimal raises InvalidOperation:

```
>>> Decimal("1e9999999999999999999")
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
decimal.InvalidOperation: [<class 'decimal.InvalidOperation'>]
```

Changed in version 3.3.
Decimals interact well with much of the rest of Python. Here is a small decimal floating point flying circus:

```
>>> data = list(map(Decimal, '1.34 1.87 3.45 2.35 1.00 0.03 9.25'.split()))
>>> max(data)
Decimal('9.25')
>>> min(data)
Decimal('0.03')
>>> sorted(data)
[Decimal('0.03'), Decimal('1.00'), Decimal('1.34'), Decimal('1.87'),
Decimal('2.35'), Decimal('3.45'), Decimal('9.25')]
```

```
>>> sum(data)
Decimal('19.29')
>>> a,b,c = data[:3]
>>> str(a)
'1.34'
>>> float(a)
1.34
>>> round(a, 1)
Decimal('1.3')
>>> int(a)
1
>>> a * 5
Decimal('6.70')
>>> a * b
Decimal('2.5058')
>>> c % a
Decimal('0.77')
```

And some mathematical functions are also available to Decimal:

```
>>> getcontext().prec = 28
>>> Decimal(2).sqrt()
Decimal('1.4142135623730950488016887242')
>>> Decimal(1).exp()
Decimal('2.7182818284590452353602874713')
>>> Decimal('10').ln()
Decimal('2.3025850929940456840179914547')
>>> Decimal('10').log10()
Decimal('1')
```

The quantize() method rounds a number to a fixed exponent. This method is useful for monetary applications that often round results to a fixed number of places:

```
>>> Decimal('7.325').quantize(Decimal('.01'), rounding=ROUND\_DOWN)
Decimal('7.32')
>>> Decimal('7.325').quantize(Decimal('1.'), rounding=ROUND\_UP)
Decimal('8')
```

As shown above, the getcontext() function accesses the current context and allows the settings to be changed. This approach meets the needs of most applications.

For more advanced work, it may be useful to create alternate contexts using the Context() constructor. To make an alternate active, use the setcontext() function.

In accordance with the standard, the Decimal module provides two ready to use standard contexts, BasicContext and ExtendedContext. The former is especially useful for debugging because many of the traps are enabled:

```
>>> myothercontext = Context(prec=60, rounding=ROUND\_HALF\_DOWN)
>>> setcontext(myothercontext)
>>> Decimal(1) / Decimal(7)
Decimal('0.142857142857142857142857142857142857142857142857142857142857')

>>> ExtendedContext
```

```
Context(prec=9, rounding=ROUND\_HALF\_EVEN, Emin=-999999, Emax=999999,
capitals=1, clamp=0, flags=[], traps=[])
>>> setcontext(ExtendedContext)
>>> Decimal(1) / Decimal(7)
Decimal('0.142857143')
>>> Decimal(42) / Decimal(0)
Decimal('Infinity')

>>> setcontext(BasicContext)
>>> Decimal(42) / Decimal(0)
Traceback (most recent call last):
File "<pyshell#143>", line 1, in -toplevel-
Decimal(42) / Decimal(0)
DivisionByZero: x / 0
```

Contexts also have signal flags for monitoring exceptional conditions encountered during computations. The flags remain set until explicitly cleared, so it is best to clear the flags before each set of monitored computations by using the clear_flags() method.

```
>>> setcontext(ExtendedContext)
>>> getcontext().clear\_flags()
>>> Decimal(355) / Decimal(113)
Decimal('3.14159292')
>>> getcontext()
Context(prec=9, rounding=ROUND\_HALF\_EVEN, Emin=-999999, Emax=999999,
capitals=1, clamp=0, flags=[Inexact, Rounded], traps=[])
```

The flags entry shows that the rational approximation to Pi was rounded (digits beyond the context precision were thrown away) and that the result is inexact (some of the discarded digits were non-zero).
Individual traps are set using the dictionary in the traps field of a context:

```
>>> setcontext(ExtendedContext)
>>> Decimal(1) / Decimal(0)
Decimal('Infinity')
>>> getcontext().traps[DivisionByZero] = 1
>>> Decimal(1) / Decimal(0)
Traceback (most recent call last):
File "<pyshell#112>", line 1, in -toplevel-
Decimal(1) / Decimal(0)
DivisionByZero: x / 0
```

Most programs adjust the current context only once, at the beginning of the program. And, in many applications, data is converted to Decimal with a single cast inside a loop. With context set and decimals created, the bulk of the program manipulates the data no differently than with other Python numeric types.

3.7.3 Decimal objects

class decimal.Decimal(value=”0”, context=None)
Construct a new Decimal object based from value.
value can be an integer, string, tuple, float, or another Decimal object. If no value is given,

returns Decimal('0'). If value is a string, it should conform to the decimal numeric string syntax after leading and trailing whitespace characters are removed:

```
sign           ::= '+' | '-'
digit          ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
indicator      ::= 'e' | 'E'
digits         ::= digit [digit]...
decimal-part   ::= digits '.' [digits] | ['.'] digits
exponent-part  ::= indicator [sign] digits
infinity       ::= 'Infinity' | 'Inf'
nan            ::= 'NaN' [digits] | 'sNaN' [digits]
numeric-value  ::= decimal-part [exponent-part] | infinity
numeric-string ::= [sign] numeric-value | [sign] nan
```

Other Unicode decimal digits are also permitted where digit appears above. These include decimal digits from various other alphabets (for example, Arabic-Indic and Devanagari digits) along with the fullwidth digits '
uff10' through '
uff19'.

If value is a tuple, it should have three components, a sign (0 for positive or 1 for negative), a tuple of digits, and an integer exponent. For example, Decimal((0, (1, 4, 1, 4), -3)) returns Decimal('1.414').

If value is a float, the binary floating point value is losslessly converted to its exact decimal equivalent. This conversion can often require 53 or more digits of precision. For example, Decimal(float('1.1')) converts to
Decimal('1.100000000000000088817841970012523233890533447265625').

The context precision does not affect how many digits are stored. That is determined exclusively by the number of digits in value. For example, Decimal('3.00000') records all five zeros even if the context precision is only three.

The purpose of the context argument is determining what to do if value is a malformed string. If the context traps InvalidOperation, an exception is raised; otherwise, the constructor returns a new Decimal with the value of NaN.

Once constructed, Decimal objects are immutable.
Changed in version 3.2: The argument to the constructor is now permitted to be a float instance.
Changed in version 3.3: float arguments raise an exception if the FloatOperation trap is set. By default the trap is off.

Decimal floating point objects share many properties with the other built-in numeric types such as float and int. All of the usual math operations and special methods apply. Likewise, decimal objects can be copied, pickled, printed, used as dictionary keys, used as set elements, compared, sorted, and coerced to another type (such as float or int).

There are some small differences between arithmetic on Decimal objects and arithmetic on integers and floats. When the remainder operator % is applied to Decimal objects, the sign of the result is the sign of the dividend rather than the sign of the divisor:

```
>>> (-7) % 4
1
```

```
>>> Decimal(-7) % Decimal(4)
Decimal('-3')
```

The integer division operator // behaves analogously, returning the integer part of the true quotient (truncating towards zero) rather than its floor, so as to preserve the usual identity
x == (x // y) * y + x

```
>>> -7 // 4
-2
>>> Decimal(-7) // Decimal(4)
Decimal('-1')
```

The % and // operators implement the remainder and divide-integer operations (respectively) as described in the specification.

Decimal objects cannot generally be combined with floats or instances of fractions.Fraction in arithmetic operations: an attempt to add a Decimal to a float, for example, will raise a TypeError. However, it is possible to use Python's comparison operators to compare a Decimal instance x with another number y. This avoids confusing results when doing equality comparisons between numbers of different types.
Changed in version 3.2: Mixed-type comparisons between Decimal instances and other numeric types are now fully supported.

3.7.4 Methods

In addition to the standard numeric properties, decimal floating point objects also have a number of specialized methods:

3.7.4.1 adjusted()

Return the adjusted exponent after shifting out the coefficientâĂŹs rightmost digits until only the lead digit remains: Decimal('321e+5').adjusted() returns seven. Used for determining the position of the most significant digit with respect to the decimal point.

3.7.4.2 as_tuple()

Return a named tuple representation of the number: DecimalTuple(sign, digits, exponent).

3.7.4.3 canonical()

Return the canonical encoding of the argument. Currently, the encoding of a Decimal instance is always canonical, so this operation returns its argument unchanged.

3.7.4.4 compare(other, context=None)

Compare the values of two Decimal instances. compare() returns a Decimal instance, and if either operand is a NaN then the result is a NaN:

```
a or b is a NaN ==> Decimal('NaN')
a < b           ==> Decimal('-1')
a == b          ==> Decimal('0')
a > b           ==> Decimal('1')
```

3.7.4.5 compare_signal(other, context=None)

This operation is identical to the compare() method, except that all NaNs signal. That is, if neither operand is a signaling NaN then any quiet NaN operand is treated as though it were a signaling NaN.

3.7.4.6 compare_total(other, context=None)

Compare two operands using their abstract representation rather than their numerical value. Similar to the compare() method, but the result gives a total ordering on Decimal instances. Two Decimal instances with the same numeric value but different representations compare unequal in this ordering:

```
>>> Decimal('12.0').compare\_total(Decimal('12'))
Decimal('-1')
```

Quiet and signaling NaNs are also included in the total ordering. The result of this function is Decimal('0') if both operands have the same representation, Decimal('-1') if the first operand is lower in the total order than the second, and Decimal('1') if the first operand is higher in the total order than the second operand. See the specification for details of the total order.
This operation is unaffected by context and is quiet: no flags are changed and no rounding is performed. As an exception, the C version may raise InvalidOperation if the second operand cannot be converted exactly.

3.7.4.7 compare_total_mag(other, context=None)

Compare two operands using their abstract representation rather than their value as in compare_total(), but ignoring the sign of each operand. x.compare_total_mag(y) is equivalent to x.copy_abs().compare_total(y.copy_abs()).
This operation is unaffected by context and is quiet: no flags are changed and no rounding is performed. As an exception, the C version may raise InvalidOperation if the second operand cannot be converted exactly.

3.7.4.8 conjugate()

Just returns self, this method is only to comply with the Decimal Specification.

3.7.4.9 copy_abs()

Return the absolute value of the argument. This operation is unaffected by the context and is quiet: no flags are changed and no rounding is performed.

3.7.4.10 copy_negate()

Return the negation of the argument. This operation is unaffected by the context and is quiet: no flags are changed and no rounding is performed.

3.7.4.11 copy_sign(other, context=None)

Return a copy of the first operand with the sign set to be the same as the sign of the second operand. For example:

```
>>> Decimal('2.3').copy\_sign(Decimal('-1.5'))
Decimal('-2.3')
```

This operation is unaffected by context and is quiet: no flags are changed and no rounding is performed. As an exception, the C version may raise InvalidOperation if the second operand cannot be converted exactly.

3.7.4.12 exp(context=None)

Return the value of the (natural) exponential function e**x at the given number. The result is correctly rounded using the ROUND_HALF_EVEN rounding mode.

```
>>> Decimal(1).exp()
Decimal('2.718281828459045235360287471')
>>> Decimal(321).exp()
Decimal('2.561702493119680037517373933E+139')
```

3.7.4.13 from_float(f)

Classmethod that converts a float to a decimal number, exactly.
Note Decimal.from_float(0.1) is not the same as Decimal('0.1'). Since 0.1 is not exactly representable in binary floating point, the value is stored as the nearest representable value which is 0x1.999999999999ap-4. That equivalent value in decimal is 0.1000000000000000055511151231257827021181583404541015625.

Note: From Python 3.2 onwards, a Decimal instance can also be constructed directly from a float.

```
>>> Decimal.from_float(0.1)
Decimal('0.1000000000000000055511151231257827021181583404541015625')
>>> Decimal.from_float(float('nan'))
Decimal('NaN')
>>> Decimal.from_float(float('inf'))
Decimal('Infinity')
>>> Decimal.from_float(float('-inf'))
Decimal('-Infinity')
```

3.7.4.14 fma(other, third, context=None)

New in version 3.1.
Fused multiply-add. Return self*other+third with no rounding of the intermediate product self*other.

```
>>> Decimal(2).fma(3, 5)
Decimal('11')
```

3.7.4.15 is_canonical()

Return True if the argument is canonical and False otherwise. Currently, a Decimal instance is always canonical, so this operation always returns True.

3.7.4.16 is_finite()

Return True if the argument is a finite number, and False if the argument is an infinity or a NaN.

3.7.4.17 is_infinite()

Return True if the argument is either positive or negative infinity and False otherwise.

3.7.4.18 is_nan()

Return True if the argument is a (quiet or signaling) NaN and False otherwise.

3.7.4.19 is_normal(context=None)

Return True if the argument is a normal finite number. Return False if the argument is zero, subnormal, infinite or a NaN.

3.7.4.20 is_qnan()

Return True if the argument is a quiet NaN, and False otherwise.

3.7.4.21 is_signed()

Return True if the argument has a negative sign and False otherwise. Note that zeros and NaNs can both carry signs.

3.7.4.22 is_snan()

Return True if the argument is a signaling NaN and False otherwise.

3.7.4.23 is_subnormal(context=None)

Return True if the argument is subnormal, and False otherwise.

3.7.4.24 is_zero()

Return True if the argument is a (positive or negative) zero and False otherwise.

3.7.4.25 ln(context=None)

Return the natural (base e) logarithm of the operand. The result is correctly rounded using the ROUND_HALF_EVEN rounding mode.

3.7.4.26 log10(context=None)

Return the base ten logarithm of the operand. The result is correctly rounded using the ROUND_HALF_EVE rounding mode.

3.7.4.27 logb(context=None)

For a nonzero number, return the adjusted exponent of its operand as a Decimal instance. If the operand is a zero then Decimal('-Infinity') is returned and the DivisionByZero flag is raised. If the operand is an infinity then Decimal('Infinity') is returned.

3.7.4.28 logical_and(other, context=None)

logical_and() is a logical operation which takes two logical operands (see Logical operands). The result is the digit-wise and of the two operands.

3.7.4.29 logical_invert(context=None)

logical_invert() is a logical operation. The result is the digit-wise inversion of the operand.

3.7.4.30 logical_or(other, context=None)

logical_or() is a logical operation which takes two logical operands (see Logical operands). The result is the digit-wise or of the two operands.

3.7.4.31 logical_xor(other, context=None)

logical_xor() is a logical operation which takes two logical operands (see Logical operands). The result is the digit-wise exclusive or of the two operands.

3.7.4.32 max(other, context=None)

Like max(self, other) except that the context rounding rule is applied before returning and that NaN values are either signaled or ignored (depending on the context and whether they are signaling or quiet).

3.7.4.33 max_mag(other, context=None)

Similar to the max() method, but the comparison is done using the absolute values of the operands.

3.7.4.34 min(other, context=None)

Like min(self, other) except that the context rounding rule is applied before returning and that NaN values are either signaled or ignored (depending on the context and whether they are signaling or quiet).

3.7.4.35 min_mag(other, context=None)

Similar to the min() method, but the comparison is done using the absolute values of the operands. next_minus(context=None) Return the largest number representable in the given context (or in the current threadâĂŹs context if no context is given) that is smaller than the given operand.

3.7.4.36 next_plus(context=None)

Return the smallest number representable in the given context (or in the current threadâĂŹs context if no context is given) that is larger than the given operand.

3.7.4.37 next_toward(other, context=None)

If the two operands are unequal, return the number closest to the first operand in the direction of the second operand. If both operands are numerically equal, return a copy of the first operand with the sign set to be the same as the sign of the second operand.

3.7.4.38 normalize(context=None)

Normalize the number by stripping the rightmost trailing zeros and converting any result equal to Decimal('0') to Decimal('0e0'). Used for producing canonical values for attributes of an equivalence class. For example, Decimal('32.100') and Decimal('0.321000e+2') both normalize to the equivalent value Decimal('32.1').

3.7.4.39 number_class(context=None)

Return a string describing the class of the operand. The returned value is one of the following ten strings.

```
"-Infinity", indicating that the operand is negative infinity.
"-Normal", indicating that the operand is a negative normal number.
"-Subnormal", indicating that the operand is negative and subnormal.
"-Zero", indicating that the operand is a negative zero.
"+Zero", indicating that the operand is a positive zero.
"+Subnormal", indicating that the operand is positive and subnormal.
"+Normal", indicating that the operand is a positive normal number.
"+Infinity", indicating that the operand is positive infinity.
"NaN", indicating that the operand is a quiet NaN (Not a Number).
"sNaN", indicating that the operand is a signaling NaN.
```

3.7.4.40 quantize(exp, rounding=None, context=None, watchexp=True)

Return a value equal to the first operand after rounding and having the exponent of the second operand.

```
>>> Decimal('1.41421356').quantize(Decimal('1.000'))
Decimal('1.414')
```

Unlike other operations, if the length of the coefficient after the quantize operation would be greater than precision, then an InvalidOperation is signaled. This guarantees that, unless there is an error condition, the quantized exponent is always equal to that of the right-hand operand.
Also unlike other operations, quantize never signals Underflow, even if the result is subnormal and inexact.
If the exponent of the second operand is larger than that of the first then rounding may be necessary. In this case, the rounding mode is determined by the rounding argument if given, else by the given context argument; if neither argument is given the rounding mode of the current threadâĂŹs context is used.
If watchexp is set (default), then an error is returned whenever the resulting exponent is greater than Emax or less than Etiny.
Deprecated since version 3.3: watchexp is an implementation detail from the pure Python version and is not present in the C version. It will be removed in version 3.4, where it defaults to True.

3.7.4.41 radix()

Return Decimal(10), the radix (base) in which the Decimal class does all its arithmetic. Included for compatibility with the specification.

3.7.4.42 remainder_near(other, context=None)

Return the remainder from dividing self by other. This differs from self % other in that the sign of the remainder is chosen so as to minimize its absolute value. More precisely, the return value is self - n * other where n is the integer nearest to the exact value of self / other, and if two integers are equally near then the even one is chosen.
If the result is zero then its sign will be the sign of self.

```
>>> Decimal(18).remainder\_near(Decimal(10))
Decimal('-2')
>>> Decimal(25).remainder\_near(Decimal(10))
Decimal('5')
>>> Decimal(35).remainder\_near(Decimal(10))
Decimal('-5')
```

3.7.4.43 rotate(other, context=None)

Return the result of rotating the digits of the first operand by an amount specified by the second operand. The second operand must be an integer in the range -precision through precision. The absolute value of the second operand gives the number of places to rotate. If the second operand is positive then rotation is to the left; otherwise rotation is to the right. The coefficient of the first operand is padded on the left with zeros to length precision if necessary. The sign and exponent of the first operand are unchanged.

3.7.4.44 same_quantum(other, context=None)

Test whether self and other have the same exponent or whether both are NaN.
This operation is unaffected by context and is quiet: no flags are changed and no rounding is performed. As an exception, the C version may raise InvalidOperation if the second operand cannot be converted exactly.

3.7.4.45 scaleb(other, context=None)

Return the first operand with exponent adjusted by the second. Equivalently, return the first operand multiplied by 10**other. The second operand must be an integer.

3.7.4.46 shift(other, context=None)

Return the result of shifting the digits of the first operand by an amount specified by the second operand. The second operand must be an integer in the range -precision through precision. The absolute value of the second operand gives the number of places to shift. If the second operand is positive then the shift is to the left; otherwise the shift is to the right. Digits shifted into the coefficient are zeros. The sign and exponent of the first operand are unchanged.

3.7.4.47 sqrt(context=None)

Return the square root of the argument to full precision.

3.7.4.48 to_eng_string(context=None)

Convert to an engineering-type string.
Engineering notation has an exponent which is a multiple of 3, so there are up to 3 digits left of the decimal place. For example, converts Decimal('123E+1') to Decimal('1.23E+3')

3.7.4.49 to_integral(rounding=None, context=None)

Identical to the to_integral_value() method. The to_integral name has been kept for compatibility with older versions.

3.7.4.50 to_integral_exact(rounding=None, context=None)

Round to the nearest integer, signaling Inexact or Rounded as appropriate if rounding occurs. The rounding mode is determined by the rounding parameter if given, else by the given context. If neither parameter is given then the rounding mode of the current context is used.

3.7.4.51 to_integral_value(rounding=None, context=None)

Round to the nearest integer without signaling Inexact or Rounded. If given, applies rounding; otherwise, uses the rounding method in either the supplied context or the current context.

3.7.4.52 Logical operands

The logical_and(), logical_invert(), logical_or(), and logical_xor() methods expect their arguments to be logical operands. A logical operand is a Decimal instance whose exponent and sign are both zero, and whose digits are all either 0 or 1.

3.7.5 Context objects

Contexts are environments for arithmetic operations. They govern precision, set rules for rounding, determine which signals are treated as exceptions, and limit the range for exponents.
Each thread has its own current context which is accessed or changed using the getcontext() and setcontext() functions:

3.7.5.1 decimal.getcontext()

Return the current context for the active thread.

3.7.5.2 decimal.setcontext(c)

Set the current context for the active thread to c.
You can also use the with statement and the localcontext() function to temporarily change the active context.

3.7.5.3 decimal.localcontext(ctx=None)

Return a context manager that will set the current context for the active thread to a copy of ctx on entry to the with-statement and restore the previous context when exiting the with-statement. If no context is specified, a copy of the current context is used.

For example, the following code sets the current decimal precision to 42 places, performs a calculation, and then automatically restores the previous context:

```
from decimal import localcontext

with localcontext() as ctx:
ctx.prec = 42  # Perform a high precision calculation
s = calculate\_something()
s = +s # Round the final result back to the default precision
```

New contexts can also be created using the Context constructor described below. In addition, the module provides three pre-made contexts:

3.7.5.4 class decimal.BasicContext

This is a standard context defined by the General Decimal Arithmetic Specification. Precision is set to nine. Rounding is set to ROUND_HALF_UP. All flags are cleared. All traps are enabled (treated as exceptions) except Inexact, Rounded, and Subnormal.
Because many of the traps are enabled, this context is useful for debugging.

3.7.5.5 class decimal.ExtendedContext

This is a standard context defined by the General Decimal Arithmetic Specification. Precision is set to nine. Rounding is set to ROUND_HALF_EVEN. All flags are cleared. No traps are enabled (so that exceptions are not raised during computations).
Because the traps are disabled, this context is useful for applications that prefer to have result value of NaN or Infinity instead of raising exceptions. This allows an application to complete a run in the presence of conditions that would otherwise halt the program.

3.7.5.6 class decimal.DefaultContext

This context is used by the Context constructor as a prototype for new contexts. Changing a field (such a precision) has the effect of changing the default for new contexts created by the Context constructor.
This context is most useful in multi-threaded environments. Changing one of the fields before threads are started has the effect of setting system-wide defaults. Changing the fields after threads have started is not recommended as it would require thread synchronization to prevent race conditions.
In single threaded environments, it is preferable to not use this context at all. Instead, simply create contexts explicitly as described below.
The default values are prec=28, rounding=ROUND_HALF_EVEN, and enabled traps for Overflow, InvalidOperation, and DivisionByZero.

In addition to the three supplied contexts, new contexts can be created with the Context constructor.

3.7.5.7 class decimal.Context(prec=None, rounding=None, Emin=None, Emax=None, capitals=None, clamp=None, flags=None, traps=None)

Creates a new context. If a field is not specified or is None, the default values are copied from the DefaultContext. If the flags field is not specified or is None, all flags are cleared.

prec is an integer in the range [1, MAX_PREC] that sets the precision for arithmetic operations in the context.

The rounding option is one of the constants listed in the section Rounding Modes.

The traps and flags fields list any signals to be set. Generally, new contexts should only set traps and leave the flags clear.

The Emin and Emax fields are integers specifying the outer limits allowable for exponents. Emin must be in the range [MIN_EMIN, 0], Emax in the range [0, MAX_EMAX].

The capitals field is either 0 or 1 (the default). If set to 1, exponents are printed with a capital E; otherwise, a lowercase e is used: Decimal('6.02e+23').

The clamp field is either 0 (the default) or 1. If set to 1, the exponent e of a Decimal instance representable in this context is strictly limited to the range Emin - prec + 1 ¡= e ¡= Emax - prec + 1. If clamp is 0 then a weaker condition holds: the adjusted exponent of the Decimal instance is at most Emax. When clamp is 1, a large normal number will, where possible, have its exponent reduced and a corresponding number of zeros added to its coefficient, in order to fit the exponent constraints; this preserves the value of the number but loses information about significant trailing zeros.
For example:

```
>>> Context(prec=6, Emax=999, clamp=1).create\_decimal('1.23e999')
Decimal('1.23000E+999')
```

A clamp value of 1 allows compatibility with the fixed-width decimal interchange formats specified in IEEE 754.

3.7.6 Context Methods

The Context class defines several general purpose methods as well as a large number of methods for doing arithmetic directly in a given context. In addition, for each of the Decimal methods described above (with the exception of the adjusted() and as_tuple() methods) there is a corresponding Context method. For example, for a Context instance C and Decimal instance x, C.exp(x) is equivalent to x.exp(context=C). Each Context method accepts a Python integer (an instance of int) anywhere that a Decimal instance is accepted.

3.7.6.1 clear_flags()

Resets all of the flags to 0.

3.7.6.2 clear_traps()

Resets all of the traps to 0.
New in version 3.3.

3.7.6.3 copy()

Return a duplicate of the context.

3.7.6.4 copy_decimal(num)

Return a copy of the Decimal instance num.

3.7.6.5 create_decimal(num)

Creates a new Decimal instance from num but using self as context. Unlike the Decimal constructor, the context precision, rounding method, flags, and traps are applied to the conversion. This is useful because constants are often given to a greater precision than is needed by the application. Another benefit is that rounding immediately eliminates unintended effects from digits beyond the current precision. In the following example, using unrounded inputs means that adding zero to a sum can change the result:

```
>>> getcontext().prec = 3
>>> Decimal('3.4445') + Decimal('1.0023')
Decimal('4.45')
>>> Decimal('3.4445') + Decimal(0) + Decimal('1.0023')
Decimal('4.44')
```

This method implements the to-number operation of the IBM specification. If the argument is a string, no leading or trailing whitespace is permitted.

3.7.6.6 create_decimal_from_float(f)

Creates a new Decimal instance from a float f but rounding using self as the context. Unlike the Decimal.from_float() class method, the context precision, rounding method, flags, and traps are applied to the conversion.

```
>>> context = Context(prec=5, rounding=ROUND\_DOWN)
>>> context.create\_decimal\_from\_float(math.pi)
Decimal('3.1415')
>>> context = Context(prec=5, traps=[Inexact])
>>> context.create\_decimal\_from\_float(math.pi)
Traceback (most recent call last):
...
decimal.Inexact: None
```

New in version 3.1.

3.7.6.7 Etiny()

Returns a value equal to Emin - prec + 1 which is the minimum exponent value for subnormal results. When underflow occurs, the exponent is set to Etiny.

3.7.6.8 Etop()

Returns a value equal to Emax - prec + 1.

The usual approach to working with decimals is to create Decimal instances and then apply arithmetic operations which take place within the current context for the active thread. An alternative approach is to use context methods for calculating within a specific context. The methods are similar to those for the Decimal class and are only briefly recounted here.

3.7.6.9 abs(x)

Returns the absolute value of x.

3.7.6.10 add(x, y)

Return the sum of x and y.

3.7.6.11 canonical(x)

Returns the same Decimal object x.

3.7.6.12 compare(x, y)

Compares x and y numerically.

3.7.6.13 compare_signal(x, y)

Compares the values of the two operands numerically.

3.7.6.14 compare_total(x, y)

Compares two operands using their abstract representation.

3.7.6.15 compare_total_mag(x, y)

Compares two operands using their abstract representation, ignoring sign.

3.7.6.16 copy_abs(x)

Returns a copy of x with the sign set to 0.

3.7.6.17 copy_negate(x)

Returns a copy of x with the sign inverted.

3.7.6.18 copy_sign(x, y)

Copies the sign from y to x.

3.7.6.19 divide(x, y)

Return x divided by y.

3.7.6.20 divide_int(x, y)

Return x divided by y, truncated to an integer.

3.7.6.21 divmod(x, y)

Divides two numbers and returns the integer part of the result.

3.7.6.22 exp(x)

Returns e ** x.

3.7.6.23 fma(x, y, z)

Returns x multiplied by y, plus z.

3.7.6.24 is_canonical(x)

Returns True if x is canonical; otherwise returns False.

3.7.6.25 is_finite(x)

Returns True if x is finite; otherwise returns False.

3.7.6.26 is_infinite(x)

Returns True if x is infinite; otherwise returns False.

3.7.6.27 is_nan(x)

Returns True if x is a qNaN or sNaN; otherwise returns False.

3.7.6.28 is_normal(x)

Returns True if x is a normal number; otherwise returns False.

3.7.6.29 is_qnan(x)

Returns True if x is a quiet NaN; otherwise returns False.

3.7.6.30 is_signed(x)

Returns True if x is negative; otherwise returns False.

3.7.6.31 is_snan(x)

Returns True if x is a signaling NaN; otherwise returns False.

3.7.6.32 is_subnormal(x)

Returns True if x is subnormal; otherwise returns False.

3.7.6.33 is_zero(x)

Returns True if x is a zero; otherwise returns False.

3.7.6.34 ln(x)

Returns the natural (base e) logarithm of x.

3.7.6.35 log10(x)

Returns the base 10 logarithm of x.

3.7.6.36 logb(x)

Returns the exponent of the magnitude of the operand's MSD.

3.7.6.37 logical_and(x, y)

Applies the logical operation and between each operand's digits.

3.7.6.38 logical_invert(x)

Invert all the digits in x.

3.7.6.39 logical_or(x, y)

Applies the logical operation or between each operand's digits.

3.7.6.40 logical_xor(x, y)

Applies the logical operation xor between each operand's digits.

3.7.6.41 max(x, y)

Compares two values numerically and returns the maximum.

3.7.6.42 max_mag(x, y)

Compares the values numerically with their sign ignored.

3.7.6.43 min(x, y)

Compares two values numerically and returns the minimum.

3.7.6.44 min_mag(x, y)

Compares the values numerically with their sign ignored.

3.7.6.45 minus(x)

Minus corresponds to the unary prefix minus operator in Python.

3.7.6.46 multiply(x, y)

Return the product of x and y.

3.7.6.47 next_minus(x)

Returns the largest representable number smaller than x.

3.7.6.48 next_plus(x)

Returns the smallest representable number larger than x.

3.7.6.49 next_toward(x, y)

Returns the number closest to x, in direction towards y.

3.7.6.50 normalize(x)

Reduces x to its simplest form.

3.7.6.51 number_class(x)

Returns an indication of the class of x.

3.7.6.52 plus(x)

Plus corresponds to the unary prefix plus operator in Python. This operation applies the context precision and rounding, so it is not an identity operation.

3.7.6.53 power(x, y, modulo=None)

Return x to the power of y, reduced modulo modulo if given.
With two arguments, compute x**y. If x is negative then y must be integral. The result will be inexact unless y is integral and the result is finite and can be expressed exactly in âĂŸprecisionâĂŹ digits. The rounding mode of the context is used. Results are always correctly-rounded in the Python version.
Changed in version 3.3: The C module computes power() in terms of the correctly-rounded exp() and ln() functions. The result is well-defined but only âĂIJalmost always correctly-roundedâĂİ.
With three arguments, compute (x**y) % modulo. For the three argument form, the following restrictions on the arguments hold:

```
all three arguments must be integral
y must be nonnegative
at least one of x or y must be nonzero
modulo must be nonzero and have at most âĂŸprecisionâĂŹ digits
```

The value resulting from Context.power(x, y, modulo) is equal to the value that would be obtained by computing (x**y) % modulo with unbounded precision, but is computed more efficiently. The exponent of the result is zero, regardless of the exponents of x, y and modulo. The result is always exact.

3.7.6.54 quantize(x, y)

Returns a value equal to x (rounded), having the exponent of y.

3.7.6.55 radix()

radix() Just returns 10, as this is Decimal.

3.7.6.56 remainder(x, y)

Returns the remainder from integer division.
The sign of the result, if non-zero, is the same as that of the original dividend.

3.7.6.57 remainder_near(x, y)

Returns x - y * n, where n is the integer nearest the exact value of x / y (if the result is 0 then its sign will be the sign of x).

3.7.6.58 rotate(x, y)

Returns a rotated copy of x, y times.

3.7.6.59 same_quantum(x, y)

Returns True if the two operands have the same exponent.

3.7.6.60 scaleb(x, y)

Returns the first operand after adding the second value its exp.

3.7.6.61 shift(x, y)

Returns a shifted copy of x, y times.

3.7.6.62 sqrt(x)

Square root of a non-negative number to context precision.

3.7.6.63 subtract(x, y)

Return the difference between x and y.

3.7.6.64 to_eng_string(x)

to_eng_string(x) Converts a number to a string, using scientific notation.

3.7.6.65 to_integral_exact(x)

to_integral_exact(x) Rounds to an integer.

3.7.6.66 to_sci_string(x)

Converts a number to a string using scientific notation.

3.7.7 Constants

The constants in this section are only relevant for the C module. They are also included in the pure Python version for compatibility.
decimal.HAVE_THREADS The default value is True. If Python is compiled without threads, the C version automatically disables the expensive thread local context machinery. In this case, the value is False.

	32-bit	64-bit
decimal.MAX_PREC	425000000	999999999999999999
decimal.MAX_EMAX	425000000	999999999999999999
decimal.MIN_EMIN	-425000000	-999999999999999999
decimal.MIN_ETINY	-849999999	-1999999999999999997

3.7.8 Rounding modes

```
decimal.ROUND\_CEILING
Round towards Infinity.
decimal.ROUND\_DOWN
Round towards zero.
decimal.ROUND\_FLOOR
Round towards -Infinity.
decimal.ROUND\_HALF\_DOWN
Round to nearest with ties going towards zero.
decimal.ROUND\_HALF\_EVEN
Round to nearest with ties going to nearest even integer.
decimal.ROUND\_HALF\_UP
Round to nearest with ties going away from zero.
decimal.ROUND\_UP
Round away from zero.
decimal.ROUND\_05UP
Round away from zero if last digit after rounding towards zero would have been 0 or
    5; otherwise round towards zero.
```

3.7.9 Signals

Signals represent conditions that arise during computation. Each corresponds to one context flag and one context trap enabler.

The context flag is set whenever the condition is encountered. After the computation, flags may be checked for informational purposes (for instance, to determine whether a computation was exact). After checking the flags, be sure to clear all flags before starting the next computation.

If the context's trap enabler is set for the signal, then the condition causes a Python exception to be raised. For example, if the DivisionByZero trap is set, then a DivisionByZero exception is raised upon encountering the condition.

3.7.9.1 class decimal.Clamped

Altered an exponent to fit representation constraints.
Typically, clamping occurs when an exponent falls outside the contextâĂŹs Emin and Emax limits. If possible, the exponent is reduced to fit by adding zeros to the coefficient.

3.7.9.2 class decimal.DecimalException

Base class for other signals and a subclass of ArithmeticError.

3.7.9.3 class decimal.DivisionByZero

Signals the division of a non-infinite number by zero.
Can occur with division, modulo division, or when raising a number to a negative power. If this signal is not trapped, returns Infinity or -Infinity with the sign determined by the inputs to the calculation.

3.7.9.4 class decimal.Inexact

Indicates that rounding occurred and the result is not exact.
Signals when non-zero digits were discarded during rounding. The rounded result is returned. The signal flag or trap is used to detect when results are inexact.

3.7.9.5 class decimal.InvalidOperation

An invalid operation was performed.
Indicates that an operation was requested that does not make sense. If not trapped, returns NaN. Possible causes include:

```
Infinity - Infinity
0 * Infinity
Infinity / Infinity
x % 0
Infinity % x
sqrt(-x) and x > 0
0 ** 0
x ** (non-integer)
x ** Infinity
```

3.7.9.6 class decimal.Overflow

Numerical overflow.
Indicates the exponent is larger than Emax after rounding has occurred. If not trapped, the result depends on the rounding mode, either pulling inward to the largest representable finite number or rounding outward to Infinity. In either case, Inexact and Rounded are also signaled.

3.7.9.7 class decimal.Rounded

Rounding occurred though possibly no information was lost.
Signaled whenever rounding discards digits; even if those digits are zero (such as rounding 5.00 to 5.0). If not trapped, returns the result unchanged. This signal is used to detect loss of significant digits.

3.7.9.8 class decimal.Subnormal

Exponent was lower than Emin prior to rounding.
Occurs when an operation result is subnormal (the exponent is too small). If not trapped, returns the result unchanged.

3.7.9.9 class decimal.Underflow

Numerical underflow with result rounded to zero.
Occurs when a subnormal result is pushed to zero by rounding. Inexact and Subnormal are also signaled.

3.7.9.10 class decimal.FloatOperation

Enable stricter semantics for mixing floats and Decimals.

If the signal is not trapped (default), mixing floats and Decimals is permitted in the Decimal constructor, create_decimal() and all comparison operators. Both conversion and comparisons are exact. Any occurrence of a mixed operation is silently recorded by setting FloatOperation in the context flags. Explicit conversions with from_float() or create_decimal_from_float() do not set the flag.

Otherwise (the signal is trapped), only equality comparisons and explicit conversions are silent. All other mixed operations raise FloatOperation.

The following table summarizes the hierarchy of signals:

```
exceptions.ArithmeticError(exceptions.Exception)
DecimalException
Clamped
DivisionByZero(DecimalException, exceptions.ZeroDivisionError)
Inexact
Overflow(Inexact, Rounded)
Underflow(Inexact, Rounded, Subnormal)
InvalidOperation
Rounded
Subnormal
FloatOperation(DecimalException, exceptions.TypeError)
```

3.7.10 Floating Point Notes

3.7.10.1 Mitigating round-off error with increased precision

The use of decimal floating point eliminates decimal representation error (making it possible to represent 0.1 exactly); however, some operations can still incur round-off error when non-zero digits exceed the fixed precision.

The effects of round-off error can be amplified by the addition or subtraction of nearly offsetting quantities resulting in loss of significance. Knuth provides two instructive examples where rounded floating point arithmetic with insufficient precision causes the breakdown of the associative and distributive properties of addition:

Examples from Seminumerical Algorithms, Section 4.2.2.

```
>>> from decimal import Decimal, getcontext
>>> getcontext().prec = 8

>>> u, v, w = Decimal(11111113), Decimal(-11111111), Decimal('7.51111111')
>>> (u + v) + w
```

```
Decimal('9.5111111')
>>> u + (v + w)
Decimal('10')

>>> u, v, w = Decimal(20000), Decimal(-6), Decimal('6.0000003')
>>> (u*v) + (u*w)
Decimal('0.01')
>>> u * (v+w)
Decimal('0.0060000')
```

The decimal module makes it possible to restore the identities by expanding the precision sufficiently to avoid loss of significance:

```
>>> getcontext().prec = 20
>>> u, v, w = Decimal(11111113), Decimal(-11111111), Decimal('7.51111111')
>>> (u + v) + w
Decimal('9.51111111')
>>> u + (v + w)
Decimal('9.51111111')
>>>
>>> u, v, w = Decimal(20000), Decimal(-6), Decimal('6.0000003')
>>> (u*v) + (u*w)
Decimal('0.0060000')
>>> u * (v+w)
Decimal('0.0060000')
```

3.7.10.2 Special values

The number system for the decimal module provides special values including NaN, sNaN, -Infinity, Infinity, and two zeros, +0 and -0.

Infinities can be constructed directly with: Decimal('Infinity'). Also, they can arise from dividing by zero when the DivisionByZero signal is not trapped. Likewise, when the Overflow signal is not trapped, infinity can result from rounding beyond the limits of the largest representable number.

The infinities are signed (affine) and can be used in arithmetic operations where they get treated as very large, indeterminate numbers. For instance, adding a constant to infinity gives another infinite result.

Some operations are indeterminate and return NaN, or if the InvalidOperation signal is trapped, raise an exception. For example, 0/0 returns NaN which means "not a number". This variety of NaN is quiet and, once created, will flow through other computations always resulting in another NaN. This behavior can be useful for a series of computations that occasionally have missing inputs — it allows the calculation to proceed while flagging specific results as invalid.

A variant is sNaN which signals rather than remaining quiet after every operation. This is a useful return value when an invalid result needs to interrupt a calculation for special handling.

The behavior of Python's comparison operators can be a little surprising where a NaN is involved. A test for equality where one of the operands is a quiet or signaling NaN always returns False (even when doing Decimal('NaN')==Decimal('NaN')), while a test for inequality always returns True. An attempt to compare two Decimals using any of the $<, <=, >$ or $>=$ operators will

raise the InvalidOperation signal if either operand is a NaN, and return False if this signal is not trapped. Note that the General Decimal Arithmetic specification does not specify the behavior of direct comparisons; these rules for comparisons involving a NaN were taken from the IEEE 854 standard (see Table 3 in section 5.7). To ensure strict standards-compliance, use the compare() and compare-signal() methods instead.

The signed zeros can result from calculations that underflow. They keep the sign that would have resulted if the calculation had been carried out to greater precision. Since their magnitude is zero, both positive and negative zeros are treated as equal and their sign is informational.

In addition to the two signed zeros which are distinct yet equal, there are various representations of zero with differing precisions yet equivalent in value. This takes a bit of getting used to. For an eye accustomed to normalized floating point representations, it is not immediately obvious that the following calculation returns a value equal to zero:

```
>>> 1 / Decimal('Infinity')
Decimal('0E-1000026')
```

3.7.11 Working with threads

The getcontext() function accesses a different Context object for each thread. Having separate thread contexts means that threads may make changes (such as getcontext().prec=10) without interfering with other threads.

Likewise, the setcontext() function automatically assigns its target to the current thread.

If setcontext() has not been called before getcontext(), then getcontext() will automatically create a new context for use in the current thread.

The new context is copied from a prototype context called DefaultContext. To control the defaults so that each thread will use the same values throughout the application, directly modify the DefaultContext object. This should be done before any threads are started so that there wonâĂŹt be a race condition between threads calling getcontext(). For example:

```
# Set applicationwide defaults for all threads about to be launched
DefaultContext.prec = 12
DefaultContext.rounding = ROUND\_DOWN
DefaultContext.traps = ExtendedContext.traps.copy()
DefaultContext.traps[InvalidOperation] = 1
setcontext(DefaultContext)

# Afterwards, the threads can be started
t1.start()
t2.start()
t3.start()
. . .
```

3.7.12 Recipes

Here are a few recipes that serve as utility functions and that demonstrate ways to work with the Decimal class:

```
def moneyfmt(value, places=2, curr='', sep=',', dp='.',
pos='', neg='-', trailneg=''):
"""Convert Decimal to a money formatted string.

places: required number of places after the decimal point
curr:   optional currency symbol before the sign (may be blank)
sep:    optional grouping separator (comma, period, space, or blank)
dp:     decimal point indicator (comma or period)
only specify as blank when places is zero
pos:    optional sign for positive numbers: '+', space or blank
neg:    optional sign for negative numbers: '-', '(', space or blank
trailneg:optional trailing minus indicator: '-', ')', space or blank

>>> d = Decimal('-1234567.8901')
>>> moneyfmt(d, curr='$')
'-$1,234,567.89'
>>> moneyfmt(d, places=0, sep='.', dp='', neg='', trailneg='-')
'1.234.568-'
>>> moneyfmt(d, curr='$', neg='(', trailneg=')')
'($1,234,567.89)'
>>> moneyfmt(Decimal(123456789), sep=' ')
'123 456 789.00'
>>> moneyfmt(Decimal('-0.02'), neg='<', trailneg='>')
'<0.02>'

"""
q = Decimal(10) ** -places   # 2 places --> '0.01'
sign, digits, exp = value.quantize(q).as\_tuple()
result = []
digits = list(map(str, digits))
build, next = result.append, digits.pop
if sign:
build(trailneg)
for i in range(places):
build(next() if digits else '0')
if places:
build(dp)
if not digits:
build('0')
i = 0
while digits:
build(next())
i += 1
if i == 3 and digits:
i = 0
build(sep)
build(curr)
build(neg if sign else pos)
return ''.join(reversed(result))

def pi():
```

```
"""Compute Pi to the current precision.

>>> print(pi())
3.141592653589793238462643383

"""
getcontext().prec += 2 # extra digits for intermediate steps
three = Decimal(3)    # substitute "three=3.0" for regular floats
lasts, t, s, n, na, d, da = 0, three, 3, 1, 0, 0, 24
while s != lasts:
lasts = s
n, na = n+na, na+8
d, da = d+da, da+32
t = (t * n) / d
s += t
getcontext().prec -= 2
return +s             # unary plus applies the new precision

def exp(x):
"""Return e raised to the power of x. Result type matches input type.

>>> print(exp(Decimal(1)))
2.718281828459045235360287471
>>> print(exp(Decimal(2)))
7.389056098930650227230427461
>>> print(exp(2.0))
7.38905609893
>>> print(exp(2+0j))
(7.38905609893+0j)

"""
getcontext().prec += 2
i, lasts, s, fact, num = 0, 0, 1, 1, 1
while s != lasts:
lasts = s
i += 1
fact *= i
num *= x
s += num / fact
getcontext().prec -= 2
return +s

def cos(x):
"""Return the cosine of x as measured in radians.

The Taylor series approximation works best for a small value of x.
For larger values, first compute x = x % (2 * pi).

>>> print(cos(Decimal('0.5')))
0.8775825618903727161162815826
>>> print(cos(0.5))
0.87758256189
```

```
>>> print(cos(0.5+0j))
(0.87758256189+0j)

"""
getcontext().prec += 2
i, lasts, s, fact, num, sign = 0, 0, 1, 1, 1, 1
while s != lasts:
lasts = s
i += 2
fact *= i * (i-1)
num *= x * x
sign *= -1
s += num / fact * sign
getcontext().prec -= 2
return +s

def sin(x):
"""Return the sine of x as measured in radians.

The Taylor series approximation works best for a small value of x.
For larger values, first compute x = x % (2 * pi).

>>> print(sin(Decimal('0.5')))
0.47942553860420300027328793521
>>> print(sin(0.5))
0.479425538604
>>> print(sin(0.5+0j))
(0.479425538604+0j)

"""
getcontext().prec += 2
i, lasts, s, fact, num, sign = 1, 0, x, 1, x, 1
while s != lasts:
lasts = s
i += 2
fact *= i * (i-1)
num *= x * x
sign *= -1
s += num / fact * sign
getcontext().prec -= 2
return +s
```

3.7.13 Decimal FAQ

Q. It is cumbersome to type decimal.Decimal('1234.5'). Is there a way to minimize typing when using the interactive interpreter?
A. Some users abbreviate the constructor to just a single letter:

```
>>> D = decimal.Decimal
>>> D('1.23') + D('3.45')
Decimal('4.68')
```

Q. In a fixed-point application with two decimal places, some inputs have many places and need to be rounded. Others are not supposed to have excess digits and need to be validated. What methods should be used?

A. The quantize() method rounds to a fixed number of decimal places. If the Inexact trap is set, it is also useful for validation:

```
>>> TWOPLACES = Decimal(10) ** -2   # same as Decimal('0.01')



>>> # Round to two places
>>> Decimal('3.214').quantize(TWOPLACES)
Decimal('3.21')



>>> # Validate that a number does not exceed two places
>>> Decimal('3.21').quantize(TWOPLACES, context=Context(traps=[Inexact]))
Decimal('3.21')



>>> Decimal('3.214').quantize(TWOPLACES, context=Context(traps=[Inexact]))
Traceback (most recent call last):
...
Inexact: None
```

Q. Once I have valid two place inputs, how do I maintain that invariant throughout an application?

A. Some operations like addition, subtraction, and multiplication by an integer will automatically preserve fixed point. Others operations, like division and non-integer multiplication, will change the number of decimal places and need to be followed-up with a quantize() step:

```
>>> a = Decimal('102.72')           # Initial fixed-point values
>>> b = Decimal('3.17')
>>> a + b                           # Addition preserves fixed-point
Decimal('105.89')
>>> a - b
Decimal('99.55')
>>> a * 42                          # So does integer multiplication
Decimal('4314.24')
>>> (a * b).quantize(TWOPLACES)  # Must quantize non-integer multiplication
Decimal('325.62')
>>> (b / a).quantize(TWOPLACES)  # And quantize division
Decimal('0.03')
```

In developing fixed-point applications, it is convenient to define functions to handle the quantize() step:

```
>>> def mul(x, y, fp=TWOPLACES):
...     return (x * y).quantize(fp)
>>> def div(x, y, fp=TWOPLACES):
...     return (x / y).quantize(fp)
```

```
>>> mul(a, b)                   # Automatically preserve fixed-point
Decimal('325.62')
>>> div(b, a)
Decimal('0.03')
```

Q. There are many ways to express the same value. The numbers 200, 200.000, 2E2, and 02E+4 all have the same value at various precisions. Is there a way to transform them to a single recognizable canonical value?
A. The normalize() method maps all equivalent values to a single representative:

```
>>> values = map(Decimal, '200 200.000 2E2 .02E+4'.split())
>>> [v.normalize() for v in values]
[Decimal('2E+2'), Decimal('2E+2'), Decimal('2E+2'), Decimal('2E+2')]
```

Q. Some decimal values always print with exponential notation. Is there a way to get a non-exponential representation?
A. For some values, exponential notation is the only way to express the number of significant places in the coefficient. For example, expressing 5.0E+3 as 5000 keeps the value constant but cannot show the original's two-place significance.
If an application does not care about tracking significance, it is easy to remove the exponent and trailing zeroes, losing significance, but keeping the value unchanged:

```
>>> def remove\_exponent(d):
...     return d.quantize(Decimal(1)) if d == d.to\_integral() else d.normalize()

>>> remove\_exponent(Decimal('5E+3'))
Decimal('5000')
```

Q. Is there a way to convert a regular float to a Decimal?
A. Yes, any binary floating point number can be exactly expressed as a Decimal though an exact conversion may take more precision than intuition would suggest:

```
>>> Decimal(math.pi)
Decimal('3.141592653589793115997963468544185161590576171875')
```

Q. Within a complex calculation, how can I make sure that I have not gotten a spurious result because of insufficient precision or rounding anomalies.
A. The decimal module makes it easy to test results. A best practice is to re-run calculations using greater precision and with various rounding modes. Widely differing results indicate insufficient precision, rounding mode issues, ill-conditioned inputs, or a numerically unstable algorithm.
Q. I noticed that context precision is applied to the results of operations but not to the inputs. Is there anything to watch out for when mixing values of different precisions?
A. Yes. The principle is that all values are considered to be exact and so is the arithmetic on those values. Only the results are rounded. The advantage for inputs is that "what you type is what you get". A disadvantage is that the results can look odd if you forget that the inputs have not been rounded:

```
>>> getcontext().prec = 3
>>> Decimal('3.104') + Decimal('2.104')
```

```
Decimal('5.21')
>>> Decimal('3.104') + Decimal('0.000') + Decimal('2.104')
Decimal('5.20')
```

The solution is either to increase precision or to force rounding of inputs using the unary plus operation:

```
>>> getcontext().prec = 3
>>> +Decimal('1.23456789')   # unary plus triggers rounding
Decimal('1.23')
```

Alternatively, inputs can be rounded upon creation using the Context.create_decimal() method:

```
>>> Context(prec=5, rounding=ROUND\_DOWN).create\_decimal('1.2345678')
Decimal('1.2345')
```

Chapter 4

gmpy2

4.1 Overview of gmpy2

gmpy2 is a C-coded Python extension module that supports multiple-precision arithmetic. gmpy2 is the successor to the original gmpy module. The gmpy module only supported the GMP multiple-precision library. gmpy2 adds support for the MPFR (correctly rounded real floating-point arithmetic) and MPC (correctly rounded complex floating-point arithmetic) libraries. gmpy2 also updates the API and naming conventions to be more consistent and support the additional functionality.
The following libraries are supported:

GMP for integer and rational arithmetic (Granlund & the GMP development team, 2013)

MPIR is based on the GMP library but adds support for Microsoft's Visual Studio compiler . It is used to create the Windows binaries (Hart & Gladman, 2014).

MPFR for correctly rounded real floating-point arithmetic (Fousse *et al.*, 2007)

MPC for correctly rounded complex floating-point arithmetic MPC (Enge *et al.*, 2012)

Generalized Lucas sequences and primality tests are based on the following code:

mpz_lucas: http://sourceforge.net/projects/mpzlucas/.
mpz_prp: http://sourceforge.net/projects/mpzprp/.

4.1.1 Downloading and installing Gmpy2

GMPY2 can be downloaded from

https://code.google.com/p/gmpy/.
The file which needs to downloaded is specific for the Python version. For Python 2.7x 32 bit, the file gmpy2-2.0.3.win32-py2.7.exe needs to be downloaded. After download, start the executable file and follow the instructions.

The license can be found in appendix D.1.3)
The contributors are listed in section C.1.2

4.1.2 Running Tests

gmpy2 comes with a test suite, which is pre-installed in the directory

..\mpFormulaPy\Win32_Python33\Lib\site-packages\gmpy2_tests.

The tests can be run from the CPython console or from within PyScripter. All tests should pass.

4.1.3 Tutorial

The mpz type is compatible with Python's built-in int/long type but is significanly faster for large values. The cutover point for performance varies, but can be as low as 20 to 40 digits. A variety of additional integer functions are provided.

```
>>> import gmpy2
>>> from gmpy2 import mpz,mpq,mpfr,mpc
>>> mpz(99) * 43
mpz(4257)
>>> pow(mpz(99), 37, 59)
mpz(18)
>>> gmpy2.isqrt(99)
mpz(9)
>>> gmpy2.isqrt\_rem(99)
(mpz(9), mpz(18))
>>> gmpy2.gcd(123,27)
mpz(3)
>>> gmpy2.lcm(123,27)
mpz(1107)
```

The mpq type is compatible with the fractions.Fraction type included with Python.

```
>>> mpq(3,7)/7
mpq(3,49)
>>> mpq(45,3) * mpq(11,8)
mpq(165,8)
```

The most significant new features in gmpy2 are support for correctly rounded arbitrary precision real and complex arithmetic based on the MPFR and MPC libraries. Floating point contexts are used to control exceptional conditions. For example, division by zero can either return an Infinity or raise an exception.

```
>>> mpfr(1)/7
mpfr('0.14285714285714285')
>>> gmpy2.get\_context().precision=200
>>> mpfr(1)/7
mpfr('0.1428571428571428571428571428571428571428571428571428571428571',200)
>>> gmpy2.get\_context()
context(precision=200, real\_prec=Default, imag\_prec=Default,
round=RoundToNearest, real\_round=Default, imag\_round=Default,
emax=1073741823, emin=-1073741823,
subnormalize=False,
trap\_underflow=False, underflow=False,
trap\_overflow=False, overflow=False,
trap\_inexact=False, inexact=True,
trap_invalid=False, invalid=False,
trap_erange=False, erange=False,
trap_divzero=False, divzero=False,
```

```
trap_expbound=False,
allow_complex=False)
>>> mpfr(1)/0
mpfr('inf')
>>> gmpy2.get_context().trap_divzero=True
>>> mpfr(1)/0
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
gmpy2.DivisionByZeroError: 'mpfr' division by zero in division
>>> gmpy2.get_context()
context(precision=200, real_prec=Default, imag_prec=Default,
round=RoundToNearest, real_round=Default, imag_round=Default,
emax=1073741823, emin=-1073741823,
subnormalize=False,
trap_underflow=False, underflow=False,
trap_overflow=False, overflow=False,
trap_inexact=False, inexact=True,
trap_invalid=False, invalid=False,
trap_erange=False, erange=False,
trap_divzero=True, divzero=True,
trap_expbound=False,
allow_complex=False)
>>> gmpy2.sqrt(mpfr(-2))
mpfr('nan')
>>> gmpy2.get_context().allow_complex=True
>>> gmpy2.get_context().precision=53
>>> gmpy2.sqrt(mpfr(-2))
mpc('0.0+1.4142135623730951j')
>>>
>>> gmpy2.set_context(gmpy2.context())
>>> with gmpy2.local_context() as ctx:
... print(gmpy2.const_pi())
... ctx.precision+=20
... print(gmpy2.const_pi())
... ctx.precision+=20
... print(gmpy2.const_pi())
...
3.1415926535897931
3.141592653589793238462643383279
3.14159265358979323846264338327950288419716939937510
>>> print(gmpy2.const_pi())
3.1415926535897931
>>>
```

4.1.4 Miscellaneous gmpy2 Functions

4.1.4.1 from_binary(...)

from_binary(bytes) returns a gmpy2 object from a byte sequence created by to_binary().

4.1.4.2 get_cache(...)

get_cache() returns the current cache size (number of objects) and the maximum size per object (number of limbs).
gmpy2 maintains an internal list of freed mpz, xmpz, mpq, mpfr, and mpc objects for reuse. The cache significantly improves performance but also increases the memory footprint.

4.1.4.3 license(...)

license() returns the gmpy2 license information.

4.1.4.4 mp_limbsize(...)

mp_limbsize() returns the number of bits per limb used by the GMP or MPIR libarary.

4.1.4.5 mp_version(...)

mp_version() returns the version of the GMP or MPIR library.

4.1.4.6 mpc_version(...)

mpc_version() returns the version of the MPC library.

4.1.4.7 mpfr_version(...)

mpfr_version() returns the version of the MPFR library.

4.1.4.8 random_state(...)

random_state([seed]) returns a new object containing state information for the random number generator. An optional integer argument can be specified as the seed value. Only the Mersenne Twister random number generator is supported.

4.1.4.9 set_cache(...)

set_cache(number, size) updates the maximum number of freed objects of each type that are cached and the maximum size (in limbs) of each object. The maximum number of objects of each type that can be cached is 1000. The maximum size of an object is 16384. The maximum size of an object is approximately 64K on 32-bit systems and 128K on 64-bit systems. Note The caching options are global to gmpy2. Changes are not thread-safe. A change in one thread will impact all threads.

4.1.4.10 to_binary(...)

to_binary(x) returns a byte sequence from a gmpy2 object. All object types are supported.

4.1.4.11 version(...)

version() returns the version of gmpy2.

4.2 Multiple-precision Integers

The gmpy2 mpz type supports arbitrary precision integers. It should be a drop-in replacement for Python's long type. Depending on the platform and the specific operation, an mpz will be faster than Python's long once the precision exceeds 20 to 50 digits. All the special integer functions in GMP are supported.
Examples

```
>>> import gmpy2
>>> from gmpy2 import mpz
>>> mpz('123') + 1
mpz(124)
>>> 10 - mpz(1)
mpz(9)
>>> gmpy2.is_prime(17)
True
```

Note: The use of from gmpy2 import * is not recommended. The names in gmpy2 have been chosen to avoid conflict with Python's builtin names but gmpy2 does use names that may conflict with other modules or variable names.

4.2.1 mpz Methods

4.2.1.1 bit_clear(...)

x.bit_clear(n) returns a copy of x with bit n set to 0.

4.2.1.2 bit_flip(...)

x.bit_flip(n) returns a copy of x with bit n inverted.

4.2.1.3 bit_length(...)

x.bit_length() returns the number of significant bits in the radix-2 representation of x. For compatibility with Python, mpz(0).bit_length() returns 0.

4.2.1.4 bit_scan0(...)

x.bit_scan0(n) returns the index of the first 0-bit of x with index $\geq$ n. If there are no more 0-bits in x at or above index n (which can only happen for x $<$ 0, assuming an infinitely long 2's complement format), then None is returned. n must be $\geq$ 0.

4.2.1.5 bit_scan1(...)

x.bit_scan1(n) returns the index of the first 1-bit of x with index $\geq$ n. If there are no more 1-bits in x at or above index n (which can only happen for x $\geq$ 0, assuming an infinitely long 2's complement format), then None is returned. n must be $\geq$ 0.

4.2.1.6 bit_set(...)

x.bit_set(n) returns a copy of x with bit n set to 0.

4.2.1.7 bit_test(...)

x.bit_test(n) returns True if bit n of x is set, and False if it is not set.

4.2.1.8 denominator(...)

x.denominator() returns mpz(1).

4.2.1.9 digits(...)

x.digits([base=10]) returns a string representing x in radix base.

4.2.1.10 numerator(...)

x.numerator() returns a copy of x.

4.2.1.11 num_digits(...)

x.num_digits([base=10]) returns the length of the string representing the absolute value of x in radix base. The result is correct if base is a power of 2. For other other bases, the result is usually correct but may be 1 too large. base can range between 2 and 62, inclusive.

4.2.2 mpz Functions

4.2.2.1 add(...)

add(x, y) returns x + y. The result type depends on the input types.

4.2.2.2 bincoef(...)

bincoef(x, n) returns the binomial coefficient. n must be ≥ 0.

4.2.2.3 bit_clear(...)

bit_clear(x, n) returns a copy of x with bit n set to 0.

4.2.2.4 bit_flip(...)

bit_flip(x, n) returns a copy of x with bit n inverted.

4.2.2.5 bit_length(...)

bit_length(x) returns the number of significant bits in the radix-2 representation of x. For compatibility with Python, mpz(0).bit_length() returns 0 while mpz(0).num_digits(2) returns 1.

4.2.2.6 bit_mask(...)

bit_mask(n) returns an mpz object exactly n bits in length with all bits set.

4.2.2.7 bit_scan0(...)

bit_scan0(x, n) returns the index of the first 0-bit of x with index $\geq$ n. If there are no more 0-bits in x at or above index n (which can only happen for x $<$ 0, assuming an infinitely long 2's complement format), then None is returned. n must be $\geq$ 0.

4.2.2.8 bit_scan1(...)

bit_scan1(x, n) returns the index of the first 1-bit of x with index $\geq$ n. If there are no more 1-bits in x at or above index n (which can only happen for x $\geq$ 0, assuming an infinitely long 2's complement format), then None is returned. n must be $\geq$ 0.

4.2.2.9 bit_set(...)

bit_set(x, n) returns a copy of x with bit n set to 0.

4.2.2.10 bit_test(...)

bit_test(x, n) returns True if bit n of x is set, and False if it is not set.

4.2.2.11 c_div(...)

c_div(x, y) returns the quotient of x divided by y. The quotient is rounded towards +Inf (ceiling rounding). x and y must be integers.

4.2.2.12 c_div_2exp(...)

c_div_2exp(x, n) returns the quotient of x divided by 2**n. The quotient is rounded towards +Inf (ceiling rounding). x must be an integer and n must be $>$ 0.

4.2.2.13 c_divmod(...)

c_divmod(x, y) returns the quotient and remainder of x divided by y. The quotient is rounded towards +Inf (ceiling rounding) and the remainder will have the opposite sign of y. x and y must be integers.

4.2.2.14 c_divmod_2exp(...)

c_divmod_2exp(x ,n) returns the quotient and remainder of x divided by 2**n. The quotient is rounded towards +Inf (ceiling rounding) and the remainder will be negative or zero. x must be an integer and n must be $>$ 0.

4.2.2.15 c_mod(...)

c_mod(x, y) returns the remainder of x divided by y. The remainder will have the opposite sign of y. x and y must be integers.

4.2.2.16 c_mod_2exp(...)

c_mod_2exp(x, n) returns the remainder of x divided by 2**n. The remainder will be negative. x must be an integer and n must be $>$ 0.

4.2.2.17 comb(...)

comb(x, n) returns the number of combinations of x things, taking n at a time. n must be ≥ 0.

4.2.2.18 digits(...)

digits(x[, base=10]) returns a string representing x in radix base.

4.2.2.19 div(...)

div(x, y) returns x / y. The result type depends on the input types.

4.2.2.20 divexact(...)

divexact(x, y) returns the quotient of x divided by y. Faster than standard division but requires the remainder is zero!

4.2.2.21 divm(...)

divm(a, b, m) returns x such that b * x == a modulo m. Raises a ZeroDivisionError exception if no such value x exists.

4.2.2.22 f_div(...)

f_div(x, y) returns the quotient of x divided by y. The quotient is rounded towards -Inf (floor rounding). x and y must be integers.

4.2.2.23 f_div_2exp(...)

f_div_2exp(x, n) returns the quotient of x divided by 2**n. The quotient is rounded towards -Inf (floor rounding). x must be an integer and n must be > 0.

4.2.2.24 f_divmod(...)

f_divmod(x, y) returns the quotient and remainder of x divided by y. The quotient is rounded towards -Inf (floor rounding) and the remainder will have the same sign as y. x and y must be integers.

4.2.2.25 f_divmod_2exp(...)

f_divmod_2exp(x, n) returns quotient and remainder after dividing x by 2**n. The quotient is rounded towards -Inf (floor rounding) and the remainder will be positive. x must be an integer and n must be > 0.

4.2.2.26 f_mod(...)

f_mod(x, y) returns the remainder of x divided by y. The remainder will have the same sign as y. x and y must be integers.

4.2.2.27 f_mod_2exp(...)

f_mod_2exp(x, n) returns remainder of x divided by 2**n. The remainder will be positive. x must be an integer and n must be > 0.

4.2.2.28 fac(...)

fac(n) returns the exact factorial of n. Use factorial() to get the floating-point approximation.

4.2.2.29 fib(...)

fib(n) returns the n-th Fibonacci number.

4.2.2.30 fib2(...)

fib2(n) returns a 2-tuple with the (n-1)-th and n-th Fibonacci numbers.

4.2.2.31 gcd(...)

gcd(a, b) returns the greatest common denominator of integers a and b.

4.2.2.32 gcdext(...)

gcdext(a, b) returns a 3-element tuple (g, s, t) such that g == gcd(a, b) and g == a * s + b * t

4.2.2.33 hamdist(...)

hamdist(x, y) returns the Hamming distance (number of bit-positions where the bits differ) between integers x and y.

4.2.2.34 invert(...)

invert(x, m) returns y such that x * y == 1 modulo m, or 0 if no such y exists.

4.2.2.35 iroot(...)

iroot(x,n) returns a 2-element tuple (y, b) such that y is the integer n-th root of x and b is True if the root is exact. x must be ≥ 0 and n must be > 0.

4.2.2.36 iroot_rem(...)

iroot_rem(x,n) returns a 2-element tuple (y, r) such that y is the integer n-th root of x and x = y**n + r. x must be ≥ 0 and n must be > 0.

4.2.2.37 is_even(...)

is_even(x) returns True if x is even, False otherwise.

4.2.2.38 is_odd(...)

is_odd(x) returns True if x is odd, False otherwise.

4.2.2.39 is_power(...)

is_power(x) returns True if x is a perfect power, False otherwise.

4.2.2.40 is_prime(...)

is_prime(x[, n=25]) returns True if x is probably prime. False is returned if x is definately composite. x is checked for small divisors and up to n Miller-Rabin tests are performed. The actual tests performed may vary based on version of GMP or MPIR used.

4.2.2.41 is_square(...)

is_square(x) returns True if x is a perfect square, False otherwise.

4.2.2.42 isqrt(...)

isqrt(x) returns the integer square root of an integer x. x must be ≥ 0.

4.2.2.43 isqrt_rem(...)

isqrt_rem(x) returns a 2-tuple (s, t) such that s = isqrt(x) and t = x - s * s. x must be ≥ 0.

4.2.2.44 jacobi(...)

jacobi(x, y) returns the Jacobi symbol (x — y). y must be odd and > 0.

4.2.2.45 kronecker(...)

kronecker(x, y) returns the Kronecker-Jacobi symbol (x — y).

4.2.2.46 lcm(...)

lcm(a, b) returns the lowest common multiple of integers a and b.

4.2.2.47 legendre(...)

legendre(x, y) returns the Legendre symbol (x — y). y is assumed to be an odd prime.

4.2.2.48 lucas(...)

lucas(n) returns the n-th Lucas number.

4.2.2.49 lucas2(...)

lucas2(n) returns a 2-tuple with the (n-1)-th and n-th Lucas numbers.

4.2.2.50 mpz(...)

mpz() returns a new mpz object set to 0.

4.2.2.51 mpz(n)

mpz(n) returns a new mpz object from a numeric value n. If n is not an integer, it will be truncated to an integer.

4.2.2.52 mpz(s, base=0)

mpz(s[, base=0]) returns a new mpz object from a string s made of digits in the given base. If base = 0, then binary, octal, or hex Python strings are recognized by leading 0b, 0o, or 0x characters. Otherwise the string is assumed to be decimal. Values for base can range between 2 and 62.

4.2.2.53 mpz_random(...)

mpz_random(random_state, n) returns a uniformly distributed random integer between 0 and n-1. The parameter random_state must be created by random_state() first.

4.2.2.54 mpz_rrandomb(...)

mpz_rrandomb(random_state, b) returns a random integer between 0 and 2**b - 1 with long sequences of zeros and one in its binary representation. The parameter random_state must be created by random_state() first.

4.2.2.55 mpz_urandomb(...)

mpz_urandomb(random_state, b) returns a uniformly distributed random integer between 0 and 2**b - 1. The parameter random_state must be created by random_state() first.

4.2.2.56 mul(...)

mul(x, y) returns x * y. The result type depends on the input types.

4.2.2.57 next_prime(...)

next_prime(x) returns the next probable prime number > x.

4.2.2.58 num_digits(...)

num_digits(x[, base=10]) returns the length of the string representing the absolute value of x in radix base. The result is correct if base is a power of 2. For other other bases, the result is usually correct but may be 1 too large. base can range between 2 and 62, inclusive.

4.2.2.59 popcount(...)

popcount(x) returns the number of bits with value 1 in x. If x < 0, the number of bits with value 1 is infinite so -1 is returned in that case.

4.2.2.60 powmod(...)

powmod(x, y, m) returns (x ** y) mod m. The exponenent y can be negative, and the correct result will be returned if the inverse of x mod m exists. Otherwise, a ValueError is raised.

4.2.2.61 remove(...)

remove(x, f) will remove the factor f from x as many times as possible and return a 2-tuple (y, m) where y = x // (f ** m). f does not divide y. m is the multiplicity of the factor f in x. f must be > 1.

4.2.2.62 sub(...)

sub(x, y) returns x - y. The result type depends on the input types.

4.2.2.63 t_div(...)

t_div(x, y) returns the quotient of x divided by y. The quotient is rounded towards zero (truncation). x and y must be integers.

4.2.2.64 t_div_2exp(...)

t_div_2exp(x, n) returns the quotient of x divided by 2**n. The quotient is rounded towards zero (truncation). n must be > 0.

4.2.2.65 t_divmod(...)

t_divmod(x, y) returns the quotient and remainder of x divided by y. The quotient is rounded towards zero (truncation) and the remainder will have the same sign as x. x and y must be integers.

4.2.2.66 t_divmod_2exp(...)

t_divmod_2exp(x, n) returns the quotient and remainder of x divided by 2**n. The quotient is rounded towards zero (truncation) and the remainder will have the same sign as x. x must be an integer and n must be > 0.

4.2.2.67 t_mod(...)

t_mod(x, y) returns the remainder of x divided by y. The remainder will have the same sign as x. x and y must be integers.

4.2.2.68 t_mod_2exp(...)

t_mod_2exp(x, n) returns the remainder of x divided by 2**n. The remainder will have the same sign as x. x must be an integer and n must be > 0.

4.3 Multiple-precision Integers (Advanced topics)

4.3.1 The xmpz type

gmpy2 provides access to an experimental integer type called xmpz. The xmpz type is a mutable integer type. In-place operations (+=, //=, etc.) modify the orignal object and do not create a new object. Instances of xmpz cannot be used as dictionary keys.

```
>>> import gmpy2
>>> from gmpy2 import xmpz
>>> a = xmpz(123)
>>> b = a
>>> a += 1
>>> a
xmpz(124)
>>> b
xmpz(124)
```

The ability to change an xmpz object in-place allows for efficient and rapid bit manipulation. Individual bits can be set or cleared:

```
>>> a[10]=1
>>> a
xmpz(1148)
```

Slice notation is supported. The bits referenced by a slice can be either "read from" or "written to". To clear a slice of bits, use a source value of 0. In 2s-complement format, 0 is represented by an arbitrary number of 0-bits. To set a slice of bits, use a source value of 0.
The tilde operator inverts, or complements the bits in an integer. (0 is -1 so you can also use -1.) In 2s-complement format, -1 is represented by an arbitrary number of 1-bits. If a value for stop is specified in a slice assignment and the actual bit-length of the xmpz is less than stop, then the destination xmpz is logically padded with 0-bits to length stop.

```
>>> a=xmpz(0)
>>> a[8:16] = ~0
>>> bin(a)
'0b1111111100000000'
>>> a[4:12] = ~a[4:12]
>>> bin(a)
'0b1111000011110000'
```

Bits can be reversed:

```
>>> bin(a)
'0b10001111100'
>>> a[::] = a[::-1]
>>> bin(a)
'0b111110001'
```

The iter_bits() method returns a generator that returns True or False for each bit position. The methods iter_clear(), and iter_set() return generators that return the bit positions that are 1 or 0. The methods support arguments start and stop that define the beginning and ending bit positions

that are used. To mimic the behavior of slices. the bit positions checked include start but the last position checked is stop - 1.

```
>>> a=xmpz(117)
>>> bin(a)
'0b1110101'
>>> list(a.iter_bits())
[True, False, True, False, True, True, True]
>>> list(a.iter_clear())
[1, 3]
>>> list(a.iter_set())
[0, 2, 4, 5, 6]
>>> list(a.iter_bits(stop=12))
[True, False, True, False, True, True, True, False, False, False, False, False]
```

The following program uses the Sieve of Eratosthenes to generate a list of prime numbers.

```
from __future__ import print_function
import time
import gmpy2
def sieve(limit=1000000):
'''Returns a generator that yields the prime numbers up to limit.'''
# Increment by 1 to account for the fact that slices do not include
# the last index value but we do want to include the last value for
# calculating a list of primes.
sieve_limit = gmpy2.isqrt(limit) + 1
limit += 1
# Mark bit positions 0 and 1 as not prime.
bitmap = gmpy2.xmpz(3)
# Process 2 separately. This allows us to use p+p for the step size
# when sieving the remaining primes.
bitmap[4 : limit : 2] = -1
# Sieve the remaining primes.
for p in bitmap.iter_clear(3, sieve_limit):
bitmap[p*p : limit : p+p] = -1
return bitmap.iter_clear(2, limit)
if __name__ == "__main__":
start = time.time()
result = list(sieve())
print(time.time() - start)
print(len(result))
```

4.3.2 Pseudoprimes

An overview is provided by Grantham (2001).

Function **gmpy2.is_bpsw_prp**(***n*** *As Integer*) As Boolean

---

The function gmpy2.is_bpsw_prp returns True if n is a Baillie-Pomerance-Selfridge-Wagstaff (BPSW) probable prime.

Parameter:
n: An Integer.

A BPSW probable prime passes the is_strong_prp() test with base 2 and the is_selfridge_prp() test.

Function **gmpy2.is_euler_prp**(***n*** *As Integer*, ***a*** *As Integer*) As Boolean

---

The function gmpy2.is_euler_prp returns True if n is an Euler (also known as Solovay-Strassen) probable prime to the base a.

Parameters:
n: An odd integer.
a: An Integer.

Assuming: gcd(n, a) == 1, and n is odd.
Then an Euler probable prime requires: a**((n-1)/2) == 1 (mod n).

Function **gmpy2.is_extra_strong_lucas_prp**(***n*** *As Integer*, ***p*** *As Integer*) As Boolean

---

The function gmpy2.is_extra_strong_lucas_prp returns True if n is an extra strong Lucas probable prime with parameters (p, 1).

Parameters:
n: An odd Integer.
p: An Integer.

Assuming: n is odd; D = p*p - 4, D != 0.
gcd(n, 2*D) == 1; n = s*(2**r) + Jacobi(D,n), s odd.

Then an extra strong Lucas probable prime requires:
lucasu(p,1,s) == 0 (mod n), or
lucasv(p,1,s) == +/-2 (mod n), or
lucasv(p,1,s*(2**t)) == 0 (mod n)
for some t, $0 \leq t < r$.

Function **gmpy2.is_fermat_prp**(***n*** *As Integer*, ***a*** *As Integer*) As Boolean

---

The function gmpy2.is_fermat_prp returns True if n is a Fermat probable prime to the base a.

Parameters:
n: An Integer.
a: An Integer.

Assuming: gcd(n,a) == 1.
Then a Fermat probable prime requires: a**(n-1) == 1 (mod n).

Function **gmpy2.is_fibonacci_prp(n,p,q)**(***n*** *As Integer,* ***p*** *As Integer,* ***q*** *As Integer*) As Boolean

---

The function gmpy2.is_fibonacci_prp(n,p,q) returns True if n is an Fibonacci probable prime with parameters (p, q).

Parameters:
n: An Integer.
p: An Integer.
q: An Integer.

Assuming: n is odd; p > 0, q = +/-1; p*p - 4*q != 0.
Then a Fibonacci probable prime requires: lucasv(p,q,n) == p (mod n).

Function **gmpy2.is_lucas_prp**(***n*** *As Integer,* ***p*** *As Integer,* ***q*** *As Integer*) As Boolean

---

The function gmpy2.is_lucas_prp returns True if n is a Lucas probable prime with parameters (p, q).

Parameters:
n: An Integer.
p: An Integer.
q: An Integer.

Assuming: n is odd; D = p*p - 4*q, D != 0; gcd(n, 2*q*D) == 1.
Then a Lucas probable prime requires: lucasu(p,q,n - Jacobi(D,n)) == 0 (mod n).

Function **gmpy2.is_selfridge_prp**(***n*** *As Integer*) As Boolean

---

The function gmpy2.is_selfridge_prp returns True if n is a Lucas probable prime with Selfridge parameters (p,q).

Parameter:
n: An Integer.

The Selfridge parameters are chosen by finding the first element D in the sequence 5, -7, 9, -11, 13, ... such that Jacobi(D,n) == -1. Let p=1 and q = (1-D)/4 and then perform a Lucas probable prime test.

Function **gmpy2.is_strong_bpsw_prp**(***n*** *As Integer*) As Boolean

---

The function gmpy2.is_strong_bpsw_prp returns True if n is a strong Baillie-Pomerance-Selfridge-Wagstaff probable prime

Parameter:
n: An Integer.

A strong BPSW probable prime passes the is_strong_prp() test with base 2 and the is_strongselfridge_prp() test.

Function **gmpy2.is_strong_lucas_prp**(***n*** *As Integer*, ***p*** *As Integer*, ***q*** *As Integer*) As Boolean

The function gmpy2.is_strong_lucas_prp returns True if n is a strong Lucas probable prime with parameters (p, q).

Parameters:
n: An Integer.
p: An Integer.
q: An Integer.

Assuming: n is odd; D = p*p - 4*q, D != 0.
gcd(n, 2*q*D) == 1; n = s*(2**r) + Jacobi(D,n), s odd.

Then a strong Lucas probable prime requires: lucasu(p,q,s) == 0 (mod n), or
lucasv(p,q,s*(2**t)) == 0 (mod n) for some t, 0 ≤ t < r.

Function **gmpy2.is_strong_prp**(***n*** *As Integer*, ***a*** *As Integer*) As Boolean

The function gmpy2.is_strong_prp returns True if n is an strong (also known as Miller-Rabin) probable prime to the base a.

Parameters:
n: An odd Integer.
a: An Integer.

Assuming: gcd(n,a) == 1; n is odd; n = s*(2**r) + 1, with s odd.
Then a strong probable prime requires one of the following is true:
a**s == 1 (mod n), or
a**(s*(2**t)) == -1 (mod n) for some t, 0 ≤ t < r.

Function **gmpy2.is_strong_selfridge_prp**(***n*** *As mpNum*) As mpNum

The function gmpy2.is_strong_selfridge_prp returns True if n is a strong Lucas probable prime with Selfridge parameters

Parameter:
n: An Integer.

The Selfridge parameters are chosen by finding the first element D in the sequence 5, -7, 9, -11, 13, ... such that Jacobi(D,n) == -1. Let p=1 and q = (1-D)/4 and then perform a strong Lucas probable prime test.

4.3.3 Lucas Sequences

An overview is provided by Joye & Quisquater (1996).

Function **gmpy2.lucasu**(***p*** *As Integer*, ***q*** *As Integer*, ***k*** *As Integer*) As Integer

The function gmpy2.lucasu returns the k-th element of the Lucas U sequence defined by p, q

Parameters:

p: An Integer.
q: An Integer.
k: An Integer.

p*p - 4*q must not equal 0; k must be greater than or equal to 0.

Function **gmpy2.lucasu_mod**(***p*** *As Integer*, ***q*** *As Integer*, ***k*** *As Integer*, ***n*** *As Integer*) As Integer

---

The function gmpy2.lucasu_mod returns the k-th element of the Lucas U sequence defined by p, q (mod n)

Parameters:
p: An Integer.
q: An Integer.
k: An Integer.
n: An Integer.

p*p - 4*q must not equal 0; k must be greater than or equal to 0; n must be greater than 0.

Function **gmpy2.lucasv**(***p*** *As Integer*, ***q*** *As Integer*, ***k*** *As Integer*) As Integer

---

The function gmpy2.lucasv returns the k-th element of the Lucas V sequence defined by parameters (p, q)

Parameters:
p: An Integer.
q: An Integer.
k: An Integer.

p*p - 4*q must not equal 0; k must be greater than or equal to 0.

Function **gmpy2.lucasv_mod**(***p*** *As Integer*, ***q*** *As Integer*, ***k*** *As Integer*, ***n*** *As Integer*) As Integer

---

The function gmpy2.lucasv_mod returns the k-th element of the Lucas V sequence defined by p, q (mod n)

Parameters:
p: An Integer.
q: An Integer.
k: An Integer.
n: An Integer.

p*p - 4*q must not equal 0; k must be greater than or equal to 0; n must be greater than 0.

4.4 Multiple-precision Rationals

gmpy2 provides a rational type call mpq. It should be a replacement for Python's fractions.Fraction module.

```
>>> import gmpy2
>>> from gmpy2 import mpq
>>> mpq(1,7)
mpq(1,7)
>>> mpq(1,7) * 11
mpq(11,7)
>>> mpq(11,7)/13
mpq(11,91)
```

4.4.1 mpq Methods

4.4.1.1 digits(...)

x.digits([base=10]) returns a Python string representing x in the given base (2 to 62, default is 10). A leading "-" is present if x < 0, but no leading "+" is present if x ≥ 0.

4.4.2 mpq Attributes

4.4.2.1 denominator

x.denominator returns the denominator of x.

4.4.2.2 numerator

x.numerator returns the numerator of x.

4.4.3 mpq Functions

4.4.3.1 add(...)

add(x, y) returns x + y. The result type depends on the input types.

4.4.3.2 div(...)

div(x, y) returns x / y. The result type depends on the input types.

4.4.3.3 f2q(...)

f2q(x[, err]) returns the best mpq approximating x to within relative error err. Default is the precision of x. If x is not an mpfr, it is converted to an mpfr. Uses Stern-Brocot tree to find the best approximation. An mpz is returned if the the denominator is 1. If err < 0, then the relative error sought is 2.0 ** err.

4.4.3.4 mpq(...)

mpq() returns an mpq object set to 0/1.

4.4.3.5 mpq(n)

mpq(n) returns an mpq object with a numeric value n. Decimal and Fraction values are converted exactly.

4.4.3.6 mpq(n, m)

mpq(n, m) returns an mpq object with a numeric value n / m.

4.4.3.7 mpq(s, base=10)

mpq(s[, base=10]) returns an mpq object from a string s made up of digits in the given base. s may be made up of two numbers in the same base separated by a "/" character. If base == 10, then an embedded "." indicates a number with a decimal fractional part.

4.4.3.8 mul(...)

mul(x, y) returns x * y. The result type depends on the input types.

4.4.3.9 qdiv(...)

qdiv(x[, y=1]) returns x/y as mpz if possible, or as mpq if x is not exactly divisible by y.

4.4.3.10 sub(...)

sub(x, y) returns x - y. The result type depends on the input types.

4.5 Multiple-precision Reals

gmpy2 replaces the mpf type from gmpy 1.x with a new mpfr type based on the MPFR library. The new mpfr type supports correct rounding, selectable rounding modes, and many trigonometric, exponential, and special functions. A context manager is used to control precision, rounding modes, and the behavior of exceptions.

The default precision of an mpfr is 53 bits - the same precision as Python's float type. If the precision is changed, then mpfr(float('1.2')) differs from mpfr('1.2'). To take advantage of the higher precision provided by the mpfr type, always pass constants as strings.

```
>>> import gmpy2
>>> from gmpy2 import mpfr
>>> mpfr('1.2')
mpfr('1.2')
>>> mpfr(float('1.2'))
mpfr('1.2')
>>> gmpy2.get_context().precision=100
>>> mpfr('1.2')
mpfr('1.2000000000000000000000000000006',100)
>>> mpfr(float('1.2'))
mpfr('1.1999999999999999555910790149937',100)
>>>
```

4.5.1 Contexts

Contexts and context managers are not thread-safe! Modifying the context in one thread will impact all other threads.

A context is used to control the behavior of mpfr and mpc arithmetic. In addition to controlling the precision, the rounding mode can be specified, minimum and maximum exponent values can be changed, various exceptions can be raised or ignored, gradual underflow can be enabled, and returning complex results can be enabled. gmpy2.context() creates a new context with all options set to default.

gmpy2.set_context(ctx) will set the active context to ctx. gmpy2.get_context() will return a reference to the active context. Note that contexts are mutable: modifying the reference returned by get_context() will modify the active context until a new context is enabled with set_context(). The copy() method of a context will return a copy of the context.

The following example just modifies the precision. The remaining options will be discussed later.

```
>>> gmpy2.set_context(gmpy2.context())
>>> gmpy2.get_context()
context(precision=53, real_prec=Default, imag_prec=Default,
round=RoundToNearest, real_round=Default, imag_round=Default,
emax=1073741823, emin=-1073741823,
subnormalize=False,
trap_underflow=False, underflow=False,
trap_overflow=False, overflow=False,
trap_inexact=False, inexact=False,
trap_invalid=False, invalid=False,
```

```
trap_erange=False, erange=False,
trap_divzero=False, divzero=False,
trap_expbound=False,
allow_complex=False)
>>> gmpy2.sqrt(5)
mpfr('2.2360679774997898')
>>> gmpy2.get_context().precision=100
>>> gmpy2.sqrt(5)
mpfr('2.2360679774997896964091736687316',100)
>>> gmpy2.get_context().precision+=20
>>> gmpy2.sqrt(5)
mpfr('2.2360679774997896964091736687312762351',120)
>>> ctx=gmpy2.get_context()
>>> ctx.precision+=20
>>> gmpy2.sqrt(5)
mpfr('2.2360679774997896964091736687312762354406182',140)
>>> gmpy2.set_context(gmpy2.context())
>>> gmpy2.sqrt(5)
mpfr('2.2360679774997898')
>>> ctx.precision+=20
>>> gmpy2.sqrt(5)
mpfr('2.2360679774997898')
>>> gmpy2.set_context(ctx)
>>> gmpy2.sqrt(5)
mpfr('2.2360679774997896964091736687312762354406183596116',160)
>>>
```

4.5.2 Context Attributes

4.5.2.1 precision

This attribute controls the precision of an mpfr result. The precision is specified in bits, not decimal digits. The maximum precision that can be specified is platform dependent and can be retrieved with get_max_precision().
Note: Specifying a value for precision that is too close to the maximum precision will cause the MPFR library to fail.

4.5.2.2 real_prec

This attribute controls the precision of the real part of an mpc result. If the value is Default, then the value of the precision attribute is used.

4.5.2.3 imag_prec

This attribute controls the precision of the imaginary part of an mpc result. If the value is Default, then the value of real_prec is used.

4.5.2.4 round

There are five rounding modes availble to mpfr types:
RoundAwayZero: The result is rounded away from 0.0.

RoundDown: The result is rounded towards -Infinity.
RoundToNearest: Round to the nearest value; ties are rounded to an even value.
RoundToZero: The result is rounded towards 0.0.
RoundUp: The result is rounded towards +Infinity.

4.5.2.5 real_round

This attribute controls the rounding mode for the real part of an mpc result. If the value is Default, then the value of the round attribute is used. Note: RoundAwayZero is not a valid rounding mode for mpc.

4.5.2.6 imag_round

This attribute controls the rounding mode for the imaginary part of an mpc result. If the value is Default, then the value of the real_round attribute is used. Note: RoundAwayZero is not a valid rounding mode for mpc.

4.5.2.7 emax

This attribute controls the maximum allowed exponent of an mpfr result. The maximum exponent is platform dependent and can be retrieved with get_emax_max().

4.5.2.8 emin

This attribute controls the minimum allowed exponent of an mpfr result. The minimum exponent is platform dependent and can be retrieved with get_emin_min().

Note: It is possible to change the values of emin/emax such that previous mpfr values are no longer valid numbers but should either underflow to +/-0.0 or overflow to +/-Infinity. To raise an exception if this occurs, see trap_expbound.

4.5.2.9 subnormalize

The usual IEEE-754 floating point representation supports gradual underflow when the minimum exponent is reached. The MFPR library does not enable gradual underflow by default but it can be enabled to precisely mimic the results of IEEE-754 floating point operations.

4.5.2.10 trap_underflow

If set to False, a result that is smaller than the smallest possible mpfr given the current exponent range will be replaced by +/-0.0. If set to True, an UnderflowResultError exception is raised.

4.5.2.11 underflow

This flag is not user controllable. It is automatically set if a result underflowed to +/-0.0 and trap_underflow is False.

4.5.2.12 trap_overflow

If set to False, a result that is larger than the largest possible mpfr given the current exponent range will be replaced by +/-Infinity. If set to True, an OverflowResultError exception is raised.

4.5.2.13 overflow

This flag is not user controllable. It is automatically set if a result overflowed to +/-Infinity and trap_overflow is False.

4.5.2.14 trap_inexact

This attribute controls whether or not an InexactResultError exception is raised if an inexact result is returned. To check if the result is greater or less than the exact result, check the rc attribute of the mpfr result.

4.5.2.15 inexact

This flag is not user controllable. It is automatically set if an inexact result is returned.

4.5.2.16 trap_invalid

This attribute controls whether or not an InvalidOperationError exception is raised if a numerical result is not defined. A special NaN (Not-A-Number) value will be returned if an exception is not raised. The InvalidOperationError is a sub-class of Python's ValueError. For example, gmpy2.sqrt(-2) will normally return mpfr('nan'). However, if allow_complex is set to True, then an mpc result will be returned.

4.5.2.17 invalid

This flag is not user controllable. It is automatically set if an invalid (Not-A-Number) result is returned.

4.5.2.18 trap_erange

This attribute controls whether or not a RangeError exception is raised when certain operations are performed on NaN and/or Infinity values. Setting trap_erange to True can be used to raise an exception if comparisons are attempted with a NaN.

```
>>> gmpy2.set_context(gmpy2.context())
>>> mpfr('nan') == mpfr('nan')
False
>>> gmpy2.get_context().trap_erange=True
>>> mpfr('nan') == mpfr('nan')
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
gmpy2.RangeError: comparison with NaN
>>>
```

4.5.2.19 erange

This flag is not user controllable. It is automatically set if an erange error occurred.

4.5.2.20 trap_divzero

This attribute controls whether or not a DivisionByZeroError exception is raised if division by 0 occurs. The DivisionByZeroError is a sub-class of Python's ZeroDivisionError.

4.5.2.21 divzero

This flag is not user controllable. It is automatically set if a division by zero occurred and NaN result was returned.

4.5.2.22 trap_expbound

This attribute controls whether or not an ExponentOutOfBoundsError exception is raised if exponents in an operand are outside the current emin/emax limits.

4.5.2.23 allow_complex

This attribute controls whether or not an mpc result can be returned if an mpfr result would normally not be possible.

4.5.3 Context Methods

4.5.3.1 clear_flags()

Clear the underflow, overflow, inexact, invalid, erange, and divzero flags.

4.5.3.2 copy()

Return a copy of the context.

4.5.4 Contexts and the with statement

Contexts can also be used in conjunction with Python's with ... statement to temporarily change the context settings for a block of code and then restore the original settings when the block of code exits.

gmpy2.local_context() first save the current context and then creates a new context based on a context passed as the first argument, or the current context if no context is passed. The new context is modified if any optional keyword arguments are given. The orginal active context is restored when the block completes.

In the following example, the current context is saved by gmpy2.local_context() and then the block begins with a copy of the default context and the precision set to 100. When the block is finished, the original context is restored.

```
>>> with gmpy2.local_context(gmpy2.context(), precision=100) as ctx
... print(gmpy2.sqrt(2))
... ctx.precision += 100
... print(gmpy2.sqrt(2))
...
1.4142135623730950488016887242092
1.41421356237309504880168872420969807856967187537694807317667960
>>>
```

A context object can also be used directly to create a context manager block. However, instead of restoring the context to the active context when the with ... statement is executed, the restored context is the context used before any keyword argument modifications. The code:

```
::
with gmpy2.ieee(64) as ctx:
```

is equivalent to:

```
::
gmpy2.set\_context(gmpy2.ieee(64)) with gmpy2.local\_context() as ctx:
```

Contexts that implement the standard single, double, and quadruple precision floating point types can be created using ieee().

4.5.5 mpfr Methods

4.5.5.1 as_integer_ratio()

Returns a 2-tuple containing the numerator and denominator after converting the mpfr object into the exact rational equivalent. The return 2-tuple is equivalent to Python's as_integer_ratio() method of built-in float objects.

4.5.5.2 as_mantissa_exp()

Returns a 2-tuple containing the mantissa and exponent.

4.5.5.3 as_simple_fraction()

Returns an mpq containing the simplest rational value that approximates the mpfr value with an error less than 1/(2**precision).

4.5.5.4 conjugate()

Returns the complex conjugate. For mpfr objects, returns a copy of the original object.

4.5.5.5 digits()

Returns a 3-tuple containing the mantissa, the exponent, and the number of bits of precision. The mantissa is represented as a string in the specified base with up to "prec" digits. If "prec" is 0, as many digits that are available are returned. No more digits than available given x's precision are returned. "base" must be between 2 and 62, inclusive.

4.5.5.6 is_integer()

Returns True if the mpfr object is an integer.

4.5.6 mpfr Attributes

4.5.6.1 imag

Returns the imaginary component. For mpfr objects, returns 0.

4.5.6.2 precision

Returns the precision of the mpfr object.

4.5.6.3 rc

The result code (also known as ternary value in the MPFR documentation) is 0 if the value of the mpfr object is exactly equal to the exact, infinite precision value. If the result code is 1, then the value of the mpfr object is greater than the exact value. If the result code is -1, then the value of the mpfr object is less than the exact, infinite precision value.

4.5.6.4 real

Returns the real component. For mpfr objects, returns a copy of the original object.

4.5.7 mpfr Functions

4.5.7.1 acos(...)

acos(x) returns the arc-cosine of x. x is measured in radians. If context.allow_complex is True, then an mpc result will be returned for abs(x) ¿ 1.

4.5.7.2 acosh(...)

acosh(x) returns the inverse hyperbolic cosine of x.

4.5.7.3 add(...)

add(x, y) returns x + y. The type of the result is based on the types of the arguments.

4.5.7.4 agm(...)

agm(x, y) returns the arithmetic-geometric mean of x and y.

4.5.7.5 ai(...)

ai(x) returns the Airy function of x.

4.5.7.6 asin(...)

asin(x) returns the arc-sine of x. x is measured in radians. If context.allow_complex is True, then an mpc result will be returned for abs(x) $>$ 1.

4.5.7.7 asinh(...)

asinh(x) return the inverse hyperbolic sine of x.

4.5.7.8 atan(...)

atan(x) returns the arc-tangent of x. x is measured in radians.

4.5.7.9 atan2(...)

atan2(y, x) returns the arc-tangent of (y/x).

4.5.7.10 atanh(...)

atanh(x) returns the inverse hyperbolic tangent of x. If context.allow_complex is True, then an mpc result will be returned for abs(x) > 1.

4.5.7.11 cbrt(...)

cbrt(x) returns the cube root of x.

4.5.7.12 ceil(...)

ceil(x) returns the "mpfr" that is the smallest integer $\geq$ x.

4.5.7.13 check_range(...)

check_range(x) return a new "mpfr" with exponent that lies within the current range of emin and emax.

4.5.7.14 const_catalan(...)

const_catalan([precision=0]) returns the catalan constant using the specified precision. If no precision is specified, the default precision is used.

4.5.7.15 const_euler(...)

const_euler([precision=0]) returns the euler constant using the specified precision. If no precision is specified, the default precision is used.

4.5.7.16 const_log2(...)

const_log2([precision=0]) returns the log2 constant using the specified precision. If no precision is specified, the default precision is used.

4.5.7.17 const_pi(...)

const_pi([precision=0]) returns the constant pi using the specified precision. If no precision is specified, the default precision is used.

4.5.7.18 context(...)

context() returns a new context manager controlling MPFR and MPC arithmetic.

4.5.7.19 cos(...)

cos(x) seturns the cosine of x. x is measured in radians.

4.5.7.20 cosh(...)

cosh(x) returns the hyperbolic cosine of x.

4.5.7.21 cot(...)

cot(x) returns the cotangent of x. x is measured in radians.

4.5.7.22 coth(...)

coth(x) returns the hyperbolic cotangent of x.

4.5.7.23 csc(...)

csc(x) returns the cosecant of x. x is measured in radians.

4.5.7.24 csch(...)

csch(x) returns the hyperbolic cosecant of x.

4.5.7.25 degrees(...)

degrees(x) converts an angle measurement x from radians to degrees.

4.5.7.26 digamma(...)

digamma(x) returns the digamma of x.

4.5.7.27 div(...)

div(x, y) returns x / y. The type of the result is based on the types of the arguments.

4.5.7.28 div_2exp(...)

div_2exp(x, n) returns an "mpfr" or "mpc" divided by 2**n.

4.5.7.29 eint(...)

eint(x) returns the exponential integral of x.

4.5.7.30 erf(...)

erf(x) returns the error function of x.

4.5.7.31 erfc(...)

erfc(x) returns the complementary error function of x.

4.5.7.32 exp(...)

exp(x) returns e**x.

4.5.7.33 exp10(...)

exp10(x) returns 10**x.

4.5.7.34 exp2(...)

exp2(x) returns 2**x.

4.5.7.35 expm1(...)

expm1(x) returns e**x - 1. expm1() is more accurate than exp(x) - 1 when x is small.

4.5.7.36 f2q(...)

f2q(x[,err]) returns the simplest mpq approximating x to within relative error err. Default is the precision of x. Uses Stern-Brocot tree to find the simplist approximation. An mpz is returned if the the denominator is 1. If err < 0, error sought is 2.0 ** err.

4.5.7.37 factorial(...)

factorial(n) returns the floating-point approximation to the factorial of n. See fac(n) to get the exact integer result.

4.5.7.38 floor(...)

floor(x) returns the "mpfr" that is the smallest integer ≤ x.

4.5.7.39 fma(...)

fma(x, y, z) returns correctly rounded result of (x * y) + z.

4.5.7.40 fmod(...)

fmod(x, y) returns x - n*y where n is the integer quotient of x/y, rounded to 0.

4.5.7.41 fms(...)

fms(x, y, z) returns correctly rounded result of (x * y) - z.

4.5.7.42 frac(...)

frac(x) returns the fractional part of x.

4.5.7.43 frexp(...)

frexp(x) returns a tuple containing the exponent and mantissa of x.

4.5.7.44 fsum(...)

fsum(iterable) returns the accurate sum of the values in the iterable.

4.5.7.45 gamma(...)

gamma(x) returns the gamma of x.

4.5.7.46 get_exp(...)

get_exp(mpfr) returns the exponent of an mpfr. Returns 0 for NaN or Infinity and sets the erange flag and will raise an exception if trap_erange is set.

4.5.7.47 hypot(...)

hypot(y, x) returns square root of (x**2 + y**2).

4.5.7.48 ieee(...)

ieee(bitwidth) returns a context with settings for 32-bit (aka single), 64-bit (aka double), or 128-bit (aka quadruple) precision floating point types.

4.5.7.49 inf(...)

inf(n) returns an mpfr initialized to Infinity with the same sign as n. If n is not given, +Infinity is returned.

4.5.7.50 is_finite(...)

is_finite(x) returns True if x is an actual number (i.e. not NaN or Infinity).

4.5.7.51 is_inf(...)

is_inf(x) returns True if x is Infinity or -Infinity.
Note: is_inf() is deprecated; please use if_infinite().

4.5.7.52 is_infinite(...)

is_infinite(x) returns True if x Infinity or -Infinity.

4.5.7.53 is_nan(...)

is_nan(x) returns True if x is NaN (Not-A-Number).

4.5.7.54 is_number(...)

is_number(x) returns True if x is an actual number (i.e. not NaN or Infinity).
Note: is_number() is deprecated; please use is_finite().

4.5.7.55 is_regular(...)

is_regular(x) returns True if x is not zero, NaN, or Infinity.

4.5.7.56 is_signed(...)

is_signed(x) returns True if the sign bit of x is set.

4.5.7.57 is_unordered(...)

is_unordered(x,y) returns True if either x and/or y is NaN.

4.5.7.58 is_zero(...)

is_zero(x) returns True if x is zero.

4.5.7.59 j0(...)

j0(x) returns the Bessel function of the first kind of order 0 of x.

4.5.7.60 j1(...)

j1(x) returns the Bessel function of the first kind of order 1 of x.

4.5.7.61 jn(...)

jn(x,n) returns the Bessel function of the first kind of order n of x.

4.5.7.62 lgamma(...)

lgamma(x) returns a tuple containing the logarithm of the absolute value of gamma(x) and the sign of gamma(x)

4.5.7.63 li2(...)

li2(x) returns the real part of dilogarithm of x.

4.5.7.64 lngamma(...)

lngamma(x) returns the logarithm of gamma(x).

4.5.7.65 log(...)

log(x) returns the natural logarithm of x.

4.5.7.66 log10(...)

log10(x) returns the base-10 logarithm of x.

4.5.7.67 log1p(...)

log1p(x) returns the natural logarithm of (1+x).

4.5.7.68 log2(...)

log2(x) returns the base-2 logarithm of x.

4.5.7.69 max2(...)

max2(x, y) returns the maximum of x and y. The result may be rounded to match the current context. Use the builtin max() to get an exact copy of the largest object without any rounding.

4.5.7.70 min2(...)

min2(x, y) returns the minimum of x and y. The result may be rounded to match the current context. Use the builtin min() to get an exact copy of the smallest object without any rounding.

4.5.7.71 modf(...)

modf(x) returns a tuple containing the integer and fractional portions of x.

4.5.7.72 mpfr(...)

mpfr() returns and mpfr object set to 0.0.

4.5.7.73 mpfr(n, precison=0)

mpfr(n[, precison=0]) returns an mpfr object after converting a numeric value n. If no precision, or a precision of 0, is specified; the precision is taken from the current context. mpfr(s[, precision=0[, [base=0]]) returns an mpfr object after converting a string "s" made up of digits in the given base, possibly with fractional part (with period as a separator) and/or exponent (with exponent marker "e" for base ≤ 10, else "@"). If no precision, or a precision of 0, is specified; the precison is taken from the current context. The base of the string representation must be 0 or in the interval 2 ... 62. If the base is 0, the leading digits of the string are used to identify the base: 0b implies base=2, 0x implies base=16, otherwise base=10 is assumed.

4.5.7.74 mpfr_from_old_binary(...)

mpfr_from_old_binary(string) returns an mpfr from a GMPY 1.x binary mpf format. Please use to_binary()/from_binary() to convert GMPY2 objects to or from a binary format.

4.5.7.75 mpfr_grandom(...)

mpfr_grandom(random_state) returns two random numbers with gaussian distribution. The parameter random_state must be created by random_state() first.

4.5.7.76 mpfr_random(...)

mpfr_random(random_state) returns a uniformly distributed number between [0,1]. The parameter random_state must be created by random_state() first.

4.5.7.77 mul(...)

mul(x, y) returns x * y. The type of the result is based on the types of the arguments.

4.5.7.78 mul_2exp(...)

mul_2exp(x, n) returns "mpfr" or "mpc" multiplied by 2**n.

4.5.7.79 nan(...)

nan() returns an "mpfr" initialized to NaN (Not-A-Number).

4.5.7.80 next_above(...)

next_above(x) returns the next "mpfr" from x toward +Infinity.

4.5.7.81 next_below(...)

next_below(x) returns the next "mpfr" from x toward -Infinity.

4.5.7.82 radians(...)

radians(x) converts an angle measurement x from degrees to radians.

4.5.7.83 rec_sqrt(...)

rec_sqrt(x) returns the reciprocal of the square root of x.

4.5.7.84 reldiff(...)

reldiff(x, y) returns the relative difference between x and y. Result is equal to abs(x-y)/x.

4.5.7.85 remainder(...)

remainder(x, y) returns x - n*y where n is the integer quotient of x/y, rounded to the nearest integer and ties rounded to even.

4.5.7.86 remquo(...)

remquo(x, y) returns a tuple containing the remainder(x,y) and the low bits of the quotient.

4.5.7.87 rint(...)

rint(x) returns x rounded to the nearest integer using the current rounding mode.

4.5.7.88 rint_ceil(...)

rint_ceil(x) returns x rounded to the nearest integer by first rounding to the next higher or equal integer and then, if needed, using the current rounding mode.

4.5.7.89 rint_floor(...)

rint_floor(x) returns x rounded to the nearest integer by first rounding to the next lower or equal integer and then, if needed, using the current rounding mode.

4.5.7.90 rint_round(...)

rint_round(x) returns x rounded to the nearest integer by first rounding to the nearest integer (ties away from 0) and then, if needed, using the current rounding mode.

4.5.7.91 rint_trunc(...)

rint_trunc(x) returns x rounded to the nearest integer by first rounding towards zero and then, if needed, using the current rounding mode.

4.5.7.92 root(...)

root(x, n) returns n-th root of x. The result always an mpfr.

4.5.7.93 round2(...)

round2(x[, n]) returns x rounded to n bits. Uses default precision if n is not specified. See round_away() to access the mpfr_round() function. Use the builtin round() to round x to n decimal digits.

4.5.7.94 round_away(...)

round_away(x) returns an mpfr by rounding x the nearest integer, with ties rounded away from 0.

4.5.7.95 sec(...)

sec(x) returns the secant of x. x is measured in radians.

4.5.7.96 sech(...)

sech(x) returns the hyperbolic secant of x.

4.5.7.97 set_exp(...)

set_exp(x, n) sets the exponent of a given mpfr to n. If n is outside the range of valid exponents, set_exp() will set the erange flag and either return the original value or raise an exception if trap_erange is set.

4.5.7.98 set_sign(...)

set_sign(x, bool) returns a copy of x with its sign bit set if bool evaluates to True.

4.5.7.99 sign(...)

sign(x) returns -1 if x $<$ 0, 0 if x == 0, or +1 if x $>$ 0.

4.5.7.100 sin(...)

sin(x) returns the sine of x. x is measured in radians.

4.5.7.101 sin_cos(...)

sin_cos(x) returns a tuple containing the sine and cosine of x. x is measured in radians.

4.5.7.102 sinh(...)

sinh(x) returns the hyberbolic sine of x.

4.5.7.103 sinh_cosh(...)

sinh_cosh(x) returns a tuple containing the hyperbolic sine and cosine of x.

4.5.7.104 sqrt(...)

sqrt(x) returns the square root of x. If x is integer, rational, or real, then an mpfr will be returned. If x is complex, then an mpc will be returned. If context.allow_complex is True, negative values of x will return an mpc.

4.5.7.105 square(...)

square(x) returns x * x. The type of the result is based on the types of the arguments.

4.5.7.106 sub(...)

sub(x, y) returns x - y. The type of the result is based on the types of the arguments.

4.5.7.107 tan(...)

tan(x) returns the tangent of x. x is measured in radians.

4.5.7.108 tanh(...)

tanh(x) returns the hyperbolic tangent of x.

4.5.7.109 trunc(...)

trunc(x) returns an "mpfr" that is x truncated towards 0. Same as x.floor() if $x \geq 0$ or x.ceil() if $x < 0$.

4.5.7.110 y0(...)

y0(x) returns the Bessel function of the second kind of order 0 of x.

4.5.7.111 y1(...)

y1(x) returns the Bessel function of the second kind of order 1 of x.

4.5.7.112 yn(...)

yn(x,n) returns the Bessel function of the second kind of order n of x.

4.5.7.113 zero(...)

zero(n) returns an mpfr inialized to 0.0 with the same sign as n. If n is not given, +0.0 is returned.

4.5.7.114 zeta(...)

zeta(x) returns the Riemann zeta of x.

4.5.8 mpfr Formatting

The mpfr type supports the __format__() special method to allow custom output formatting.

4.5.8.1 __format__(...)

x.__format__(fmt) returns a Python string by formatting "x" using the format string "fmt".
A valid format string consists of:
optional alignment code:

```
"<" : left shifted in field
">" : right shifted in field
"^" : centered in field
```

optional leading sign code

```
"+" : always display leading sign
"-" : only display minus for negative values
" " : minus for negative values, space for positive values
```

optional width.precision
optional rounding mode:

```
"U" : round toward plus infinity
"D" : round toward minus infinity
"Y" : round away from zero
"Z" : round toward zero
"N" : round to nearest
```

optional conversion code:

```
"a","A" : hex format
"b" : binary format
"e","E" : scientific format
"f","F" : fixed point format
"g","G" : fixed or scientific format
```

Note: The formatting codes must be specified in the order shown above.

```
>>> import gmpy2
>>> from gmpy2 import mpfr
>>> a=mpfr("1.23456")
>>> "{0:15.3f}".format(a)
' 1.235'
>>> "{0:15.3Uf}".format(a)
' 1.235'
>>> "{0:15.3Df}".format(a)
' 1.234'
>>> "{0:.3Df}".format(a)
'1.234'
>>> "{0:+.3Df}".format(a)
'+1.234'
```

4.6 Multiple-precision Complex

gmpy2 adds a multiple-precision complex type called mpc that is based on the MPC library. The context manager settings for mpfr arithmetic are applied to mpc arithmetic by default. It is possible to specifiy different precision and rounding modes for both the real and imaginary components of an mpc.

```
>>> import gmpy2
>>> from gmpy2 import mpc
>>> gmpy2.sqrt(mpc("1+2j"))
mpc('1.272019649514069+0.78615137775742328j')
>>> gmpy2.get_context(real_prec=100,imag_prec=200)
context(precision=53, real_prec=100, imag_prec=200,
round=RoundToNearest, real_round=Default, imag_round=Default,
emax=1073741823, emin=-1073741823,
subnormalize=False,
trap_underflow=False, underflow=False,
trap_overflow=False, overflow=False,
trap_inexact=False, inexact=True,
trap_invalid=False, invalid=False,
trap_erange=False, erange=False,
trap_divzero=False, divzero=False,
trap_expbound=False,
allow_complex=False)
>>> gmpy2.sqrt(mpc("1+2j"))
mpc('1.27201964951406896425242246173760+0.78615137775742328606955858584295892952312200
```

Exceptions are normally raised in Python when the result of a real operation is not defined over the reals; for example, sqrt(-4) will raise an exception. The default context in gmpy2 implements the same behavior but by setting allow_complex to True, complex results will be returned.

```
>>> import gmpy2
>>> from gmpy2 import mpc
>>> gmpy2.sqrt(-4)
mpfr('nan')
>>> gmpy2.get_context(allow_complex=True)
context(precision=53, real_prec=Default, imag_prec=Default,
round=RoundToNearest, real_round=Default, imag_round=Default,
emax=1073741823, emin=-1073741823,
subnormalize=False,
trap_underflow=False, underflow=False,
trap_overflow=False, overflow=False,
trap_inexact=False, inexact=False,
trap_invalid=False, invalid=True,
trap_erange=False, erange=False,
trap_divzero=False, divzero=False,
trap_expbound=False,
allow_complex=True)
>>> gmpy2.sqrt(-4)
mpc('0.0+2.0j')
```

4.6.1 mpc Methods

4.6.1.1 conjugate()

Returns the complex conjugate.

4.6.1.2 digits()

Returns a two element tuple where each element represents the real and imaginary components as a 3-tuple containing the mantissa, the exponent, and the number of bits of precision. The mantissa is represented as a string in the specified base with up to "prec" digits. If "prec" is 0, as many digits that are available are returned. No more digits than available given x's precision are returned. "base" must be between 2 and 62, inclusive.

4.6.2 mpc Attributes

4.6.2.1 imag

Returns the imaginary component.

4.6.2.2 precision

Returns a 2-tuple containing the the precision of the real and imaginary components.

4.6.2.3 rc

Returns a 2-tuple containing the ternary value of the real and imaginary components. The ternary value is 0 if the value of the component is exactly equal to the exact, infinite precision value. If the result code is 1, then the value of the component is greater than the exact value. If the result code is -1, then the value of the component is less than the exact, infinite precision value.

4.6.2.4 real

Returns the real component.

4.6.3 mpc Functions

4.6.3.1 acos(...)

acos(x) returns the arc-cosine of x.

4.6.3.2 acosh(...)

acosh(x) returns the inverse hyperbolic cosine of x.

4.6.3.3 add(...)

add(x, y) returns x + y. The type of the result is based on the types of the arguments.

4.6.3.4 asin(...)

asin(x) returns the arc-sine of x.

4.6.3.5 asinh(...)

asinh(x) return the inverse hyperbolic sine of x.

4.6.3.6 atan(...)

atan(x) returns the arc-tangent of x.

4.6.3.7 atanh(...)

atanh(x) returns the inverse hyperbolic tangent of x.

4.6.3.8 cos(...)

cos(x) seturns the cosine of x.

4.6.3.9 cosh(...)

cosh(x) returns the hyperbolic cosine of x.

4.6.3.10 div(...)

div(x, y) returns x / y. The type of the result is based on the types of the arguments.

4.6.3.11 div_2exp(...)

div_2exp(x, n) returns an "mpfr" or "mpc" divided by 2**n.

4.6.3.12 exp(...)

exp(x) returns e**x.

4.6.3.13 fma(...)

fma(x, y, z) returns correctly rounded result of (x * y) + z.

4.6.3.14 fms(...)

fms(x, y, z) returns correctly rounded result of (x * y) - z.

4.6.3.15 is_inf(...)

is_inf(x) returns True if either the real or imaginary component of x is Infinity or -Infinity.

4.6.3.16 is_nan(...)

is_nan(x) returns True if either the real or imaginary component of x is NaN (Not-A-Number).

4.6.3.17 is_zero(...)

is_zero(x) returns True if x is zero.

4.6.3.18 log(...)

log(x) returns the natural logarithm of x.

4.6.3.19 log10(...)

log10(x) returns the base-10 logarithm of x.

4.6.3.20 mpc(...)

mpc() returns an mpc object set to 0.0+0.0j.

4.6.3.21 mpc(c, precision=0)

returns a new "mpc" object from an existing complex number (either a Python complex object or another "mpc" object). If the precision is not specified, then the precision is taken from the current context. The rounding mode is always taken from the current context.

4.6.3.22 mpc(r, i=0, precision=0)

mpc(r[, i=0[, precision=0]]) returns a new "mpc" object by converting two non-complex numbers into the real and imaginary components of an "mpc" object. If the precision is not specified, then the precision is taken from the current context. The rounding mode is always taken from the current context.

4.6.3.23 mpc(s, precision=0, base=10)

mpc(s[, [precision=0[, base=10]]) returns a new "mpc" object by converting a string s into a complex number. If base is omitted, then a base-10 representation is assumed otherwise a base between 2 and 36 can be specified. If the precision is not specified, then the precision is taken from the current context. The rounding mode is always taken from the current context. In addition to the standard Python string representation of a complex number: "1+2j", the string representation used by the MPC library: "(1 2)" is also supported.

Note: The precision can be specified either a single number that is used for both the real and imaginary components, or as a 2-tuple that can specify different precisions for the real and imaginary components.

4.6.3.24 mpc_random(...)

mpfc_random(random_state) returns a uniformly distributed number in the unit square [0,1]x[0,1]. The parameter random_state must be created by random_state() first.

4.6.3.25 mul(...)

mul(x, y) returns x * y. The type of the result is based on the types of the arguments.

4.6.3.26 mul_2exp(...)

mul_2exp(x, n) returns "mpfr" or "mpc" multiplied by 2**n.

4.6.3.27 norm(...)

norm(x) returns the norm of a complex x. The norm(x) is defined as x.real**2 + x.imag**2.

4.6.3.28 abs(...)

abs(x) is the square root of norm(x).

4.6.3.29 phase(...)

phase(x) returns the phase angle, also known as argument, of a complex x.

4.6.3.30 polar(...)

polar(x) returns the polar coordinate form of a complex x that is in rectangular form.

4.6.3.31 proj(...)

proj(x) returns the projection of a complex x on to the Riemann sphere.

4.6.3.32 rect(...)

rect(x) returns the polar coordinate form of a complex x that is in rectangular form.

4.6.3.33 sin(...)

sin(x) returns the sine of x.

4.6.3.34 sinh(...)

sinh(x) returns the hyberbolic sine of x.

4.6.3.35 sqrt(...)

sqrt(x) returns the square root of x. If x is integer, rational, or real, then an mpfr will be returned. If x is complex, then an mpc will be returned. If context.allow_complex is True, negative values of x will return an mpc.

4.6.3.36 square(...)

square(x) returns x * x. The type of the result is based on the types of the arguments.

4.6.3.37 sub(...)

sub(x, y) returns x - y. The type of the result is based on the types of the arguments.

4.6.3.38 tan(...)

tan(x) returns the tangent of x. x is measured in radians.

4.6.3.39 tanh(...)

tanh(x) returns the hyperbolic tangent of x.

4.6.4 mpc Formatting

The mpc type supports the __format__() special method to allow custom output formatting.

4.6.4.1 __format__(...)

x.__format__(fmt) returns a Python string by formatting "x" using the format string "fmt". A valid format string consists of:
optional alignment code:

```
"<" : left shifted in field
">" : right shifted in field
"^" : centered in field
```

optional leading sign code

```
"+" : always display leading sign
"-" : only display minus for negative values
" " : minus for negative values, space for positive values
```

optional width.real.precision, width.imag.precision
optional rounding mode:

```
"U" : round toward plus infinity
"D" : round toward minus infinity
"Z" : round toward zero
"N" : round to nearest
```

optional output style:

```
"P" : Python style, 1+2j, (default)
"M" : MPC style, (1 2)
```

optional conversion code:

```
"a","A" : hex format
"b" : binary format
"e","E" : scientific format
"f","F" : fixed point format
"g","G" : fixed or scientific format
```

Note: The formatting codes must be specified in the order shown above.

```
>>> import gmpy2
>>> from gmpy2 import mpc
>>> a=gmpy2.sqrt(mpc("1+2j"))
>>> a
mpc('1.272019649514069+0.78615137775742328j')
>>> "{0:.4.4Mf}".format(a)
'(1.2720 0.7862)'
>>> "{0:.4.4f}".format(a)
'1.2720+0.7862j'
>>> "{0:^20.4.4U}".format(a)
'  1.2721+0.7862j  '
```

```
>>> "{0:^20.4.4D}".format(a)
' 1.2720+0.7861j '
```

Part II

Appendices

Appendix A

Python

A.1 Overview

Python is a widely used general-purpose, high-level programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C. The language provides constructs intended to enable clear programs on both a small and large scale.

Python supports multiple programming paradigms, including object-oriented, imperative and functional programming or procedural styles. It features a dynamic type system and automatic memory management and has a large and comprehensive standard library.

Like other dynamic languages, Python is often used as a scripting language, but is also used in a wide range of non-scripting contexts. Using third-party tools, Python code can be packaged into standalone executable programs. Python interpreters are available for many operating systems.

A.2 CPython

CPython, the reference implementation of Python, is free and open source software and has a community-based development model, as do nearly all of its alternative implementations. CPython is managed by the non-profit Python Software Foundation.

For further information on Python, see Wikipedia: Python (the text above has been copied from this reference), or the Python Homepage. Support for COM is included in the distribution of the ActivePython Community Edition.

Python can use GMP und MPFR thanks to GMPY2, with documentation here.

IPython is an integrations platform for various scientific libraries (NumPy, SciPy, matlibplot, pandas etc.) http://ipython.org/. Popular distributions are the
Community Edition of Anaconda: http://docs.continuum.io/anaconda/index.html,

Book recommendation: McKinney (2012).
To compile the mpmath library libraries, Python 2.7 is required.

A.2.1 Downloading and installing CPython 2.7

ActivePython is ActiveState's complete and ready-to-install distribution of Python. It provides a one-step installation of all essential Python modules, as well as extensive documentation. The Windows distribution ships with PyWin32 – a suite of Windows tools developed by Mark Hammond, including bindings to the Win32 API and Windows COM. ActivePython can be downloaded from

http://www.activestate.com/activepython/downloads.

The latest release version of the 2.7x series is 2.7.6.9. You need to download 2 separate files to support compilation of both 32 bit and 64 bit dlls.

A.2.2 Using the C-API

This section describes how to write modules in C or C++ to extend the Python interpreter with new modules. Those modules can not only define new functions but also new object types and their methods. The document also describes how to embed the Python interpreter in another application, for use as an extension language. Finally, it shows how to compile and link extension modules so that they can be loaded dynamically (at run time) into the interpreter, if the underlying operating system supports this feature.
This document assumes basic knowledge about Python. For an informal introduction to the language, see The Python Tutorial. The Python Language Reference gives a more formal definition of the language. The Python Standard Library documents the existing object types, functions and modules (both built-in and written in Python) that give the language its wide application range.
For a detailed description of the whole Python/C API, see the separate Python/C API Reference Manual.
. Extending Python with C or C++
It is quite easy to add new built-in modules to Python, if you know how to program in C. Such extension modules can do two things that canâĂŹt be done directly in Python: they can implement new built-in object types, and they can call C library functions and system calls.

To support extensions, the Python API (Application Programmers Interface) defines a set of functions, macros and variables that provide access to most aspects of the Python run-time system. The Python API is incorporated in a C source file by including the header "Python.h". The compilation of an extension module depends on its intended use as well as on your system setup; details are given in later chapters.
Note: The C extension interface is specific to CPython, and extension modules do not work on other Python implementations. In many cases, it is possible to avoid writing C extensions and preserve portability to other implementations. For example, if your use case is calling C library functions or system calls, you should consider using the ctypes module or the cffi library rather than writing custom C code. These modules let you write Python code to interface with C code and are more portable between implementations of Python than writing and compiling a C extension module.
As an example for an extension module which provides additional functionality in multi-precision computing, see the documentation on gmpy2 (section 4.1).

A.2.3 Interfaces to the C family of languages

A.2.3.1 Windows, GNU/Linux, Mac OSX: GNU Compiler Collection

The GNU Compiler Collection (GCC) is a compiler system produced by the GNU Project supporting various programming languages. GCC is a key component of the GNU toolchain. The Free Software Foundation (FSF) distributes GCC under the GNU General Public License (GNU GPL). GCC has played an important role in the growth of free software, as both a tool and an example.

Originally named the GNU C Compiler, because it only handled the C programming language, GCC 1.0 was released in 1987 and the compiler was extended to compile C++ in December of that year.[1] Front ends were later developed for Objective-C, Objective-C++, Fortran, Java, Ada, and Go among others.[3]

As well as being the official compiler of the unfinished GNU operating system, GCC has been adopted as the standard compiler by most other modern Unix-like computer operating systems, including Linux and the BSD family. A port to RISC OS has also been developed extensively in recent years. There is also an old (3.0) port of GCC to Plan9, running under its ANSI/POSIX Environment (APE).[4] GCC is also available for Microsoft Windows operating systems and for the ARM processor used by many portable devices.

For further information on the GNU Compiler Collection, see Wikipedia: GCC (the text above has been copied from this reference), or the GCC Homepage.

A.2.3.2 Windows: MSVC

Microsoft Visual C++ (often abbreviated as MSVC or VC++) is a commercial (free version available), integrated development environment (IDE) product from Microsoft for the C, C++, and C++/CLI programming languages. It features tools for developing and debugging C++ code, especially code written for the Microsoft Windows API, the DirectX API, and the Microsoft .NET Framework.

Although the product originated as an IDE for the C programming language, the compiler's support for that language conforms only to the original edition of the C standard, dating from 1989. The later revisions of the standard, C99 and C11, are not supported.[41] According to Herb

Sutter, the C compiler is only included for "historical reasons" and is not planned to be further developed. Users are advised to either use only the subset of the C language that is also valid C++, and then use the C++ compiler to compile their code, or to just use a different compiler such as Intel C++ Compiler or the GNU Compiler Collection instead.[42]

For further information on Microsoft Visual C++, see Wikipedia: MSVC (the text above has been copied from this reference), or the MSVC Homepage.

The following C file makes a number of direct calls into Python, passing arguments as strings and receiving a result as string:

```
#include "stdafx.h"
#include "CallPython.h"

int main(int argc, const char *argv[])
{
long ResultLong = SetSpecialValue_Long(2,3,4);
printf( "This is a long: %d\n", ResultLong);

double ResultDouble = SetSpecialValue_Double(2.0,3.0,4.0);
printf( "This is a double: %f\n", ResultDouble);

const char *sLong2[] = {"TestLong", "lll", "3", "1432"};
MyPythonFunction(4, sLong2);

const char *sDouble2[] = {"TestDouble", "fff", "13.5", "265.34"};
MyPythonFunction(4, sDouble2);

const char *sString2[] = {"TestStringFunc", "sss", "3", "2"};
MyPythonFunction(4, sString2);

const char *sString3[] = {"TestStringMpf2", "3", "2.456"};
MyPythonFunctionString2(3, sString3);

char buffer[1600]; // 1600 bytes allocated here on the stack.
int size0a = MyPythonFunctionStringReturn(3, sString3, buffer, sizeof(buffer));
printf("New New0a %s\n", buffer); // prints "Mar"
//printf("Length of string: %ld\n", size0a);

char buffer0[1600]; // 1600 bytes allocated here on the stack.
int size0 = MyPythonFunctionStringReturn00("TestStringMpf0", buffer0,
    sizeof(buffer0));
printf("New New0 %s\n", buffer0); // prints "Mar"
//printf("Length of string0: %ld\n", size0);

char buffer1[1600]; // 1600 bytes allocated here on the stack.
int size1 = MyPythonFunctionStringReturn01("TestStringMpf1", "3", buffer1,
    sizeof(buffer1));
printf("New New1 %s\n", buffer1); // prints "Mar"
//printf("Length of string: %ld\n", size1);
```

```
char buffer2[1600]; // 1600 bytes allocated here on the stack.
int size2 = MyPythonFunctionStringReturn02("TestStringMpf2", "3", "2.456", buffer2,
    sizeof(buffer2));
printf("New New2 %s\n", buffer2); // prints "Mar"
//printf("Length of string: %ld\n", size2);

ClosePy();
return 0;
}
```

The header file CallPython.h looks like this:

```
#pragma warning(disable: 4244)

#ifdef CALLPYTHON_EXPORTS
#define MPNUMC_DLL_IMPORTEXPORT __declspec(dllexport)
#else
#define MPNUMC_DLL_IMPORTEXPORT __declspec(dllimport)
#endif

#ifdef __cplusplus
extern "C" {
#endif

MPNUMC_DLL_IMPORTEXPORT long SetSpecialValue_Long(long m, long n, long what);
MPNUMC_DLL_IMPORTEXPORT double SetSpecialValue_Double(double m, double n, double
    what);
MPNUMC_DLL_IMPORTEXPORT int CallPythonFunction(int argc, const char *argv[]);
MPNUMC_DLL_IMPORTEXPORT int MyPythonFunction(int argc, const char *argv[]);
MPNUMC_DLL_IMPORTEXPORT int MyPythonFunctionString(int argc, const char *argv[]);
MPNUMC_DLL_IMPORTEXPORT int MyPythonFunctionString2(int argc, const char *argv[]);
MPNUMC_DLL_IMPORTEXPORT int MyPythonFunctionStringReturn(int argc, const char
    *argv[], char* buffer, int buffersize);

MPNUMC_DLL_IMPORTEXPORT int MyPythonFunctionStringReturn00(const char* FuncName,
    char* buffer, int buffersize);
MPNUMC_DLL_IMPORTEXPORT int MyPythonFunctionStringReturn01(const char* FuncName,
    const char* Arg01, char* buffer, int buffersize);
MPNUMC_DLL_IMPORTEXPORT int MyPythonFunctionStringReturn02(const char* FuncName,
    const char* Arg01, const char* Arg02, char* buffer, int buffersize);

MPNUMC_DLL_IMPORTEXPORT void ClosePy();
#ifdef __cplusplus
}
#endif
```

The C file CallPython.cpp (which produces the dynamic link library) looks like this:

```
#define _CRT_SECURE_NO_WARNINGS
#include "CallPython.h"
#include "stdafx.h"
```

```
#include<stdio.h>
#include <Python.h>

long SetSpecialValue_Long(long m, long n, long what)
{
return (m + n + 1) * what;
}

double SetSpecialValue_Double(double m, double n, double what)
{
return (m + n) * what;
}


int MyPythonFunctionStringReturn00(const char* FuncName, char* buffer, int buffersize)
{
PyObject *pFunc;
PyObject *pValue;
Py_ssize_t size;
PyObject *pModule= GetPythonModule2();
pFunc = PyObject_GetAttrString(pModule, FuncName);
if (pFunc && PyCallable_Check(pFunc)) {
pValue = PyObject_CallObject(pFunc, NULL);
if (pValue != NULL) {
strncpy(buffer, PyUnicode_AsUTF8AndSize(pValue, &size), buffersize-1);
Py_DECREF(pValue);
}
}
Py_XDECREF(pFunc);
return size;
}


int MyPythonFunctionStringReturn01(const char* FuncName, const char* Arg01, char*
    buffer, int buffersize)
{
PyObject *pFunc, *pArgs, *pValue;
PyObject *pModule= GetPythonModule2();
Py_ssize_t size;
pFunc = PyObject_GetAttrString(pModule, FuncName);
if (pFunc && PyCallable_Check(pFunc)) {
pArgs = PyTuple_New(1);
pValue = PyUnicode_FromString(Arg01);
PyTuple_SetItem(pArgs, 0, pValue);

pValue = PyObject_CallObject(pFunc, pArgs);
Py_DECREF(pArgs);
if (pValue != NULL) {
strncpy(buffer, PyUnicode_AsUTF8AndSize(pValue, &size), buffersize-1);
Py_DECREF(pValue);
}
}
```

```
Py_XDECREF(pFunc);
return size;
}


int MyPythonFunctionStringReturn02(const char* FuncName, const char* Arg01, const
    char* Arg02, char* buffer, int buffersize)
{
PyObject *pFunc, *pArgs, *pValue;
PyObject *pModule= GetPythonModule2();
Py_ssize_t size;

pFunc = PyObject_GetAttrString(pModule, FuncName);

if (pFunc && PyCallable_Check(pFunc)) {
pArgs = PyTuple_New(2);
pValue = PyUnicode_FromString(Arg01);
PyTuple_SetItem(pArgs, 0, pValue);

pValue = PyUnicode_FromString(Arg02);
PyTuple_SetItem(pArgs, 1, pValue);

pValue = PyObject_CallObject(pFunc, pArgs);
Py_DECREF(pArgs);
if (pValue != NULL) {
strncpy(buffer, PyUnicode_AsUTF8AndSize(pValue, &size), buffersize-1);
Py_DECREF(pValue);
}
}
Py_XDECREF(pFunc);
return size;
}

void ClosePy()
{
PyObject *pModule;
pModule = GetPythonModule2();
Py_DECREF(pModule);
Py_Finalize();
}
```

A.2.4 Cython: C extensions for the Python language

[Cython] is a programming language that makes writing C extensions for the Python language as easy as Python itself. It aims to become a superset of the [Python] language which gives it high-level, object-oriented, functional, and dynamic programming. Its main feature on top of these is support for optional static type declarations as part of the language. The source code gets translated into optimized C/C++ code and compiled as Python extension modules. This allows for both very fast program execution and tight integration with external C libraries, while keeping up the high programmer productivity for which the Python language is well known.
The primary Python execution environment is commonly referred to as CPython, as it is written in C. Other major implementations use Java (Jython [Jython]), C# (IronPython [IronPython]) and Python itself (PyPy [PyPy]). Written in C, CPython has been conducive to wrapping many external libraries that interface through the C language. It has, however, remained non trivial to write the necessary glue code in C, especially for programmers who are more fluent in a high-level language like Python than in a close-to-the-metal language like C.
Originally based on the well-known Pyrex [Pyrex], the Cython project has approached this problem by means of a source code compiler that translates Python code to equivalent C code. This code is executed within the CPython runtime environment, but at the speed of compiled C and with the ability to call directly into C libraries. At the same time, it keeps the original interface of the Python source code, which makes it directly usable from Python code. These two-fold characteristics enable CythonâĂŹs two major use cases: extending the CPython interpreter with fast binary modules, and interfacing Python code with external C libraries.
While Cython can compile (most) regular Python code, the generated C code usually gains major (and sometime impressive) speed improvements from optional static type declarations for both Python and C types. These allow Cython to assign C semantics to parts of the code, and to translate them into very efficient C code. Type declarations can therefore be used for two purposes: for moving code sections from dynamic Python semantics into static-and-fast C semantics, but also for directly manipulating types defined in external libraries. Cython thus merges the two worlds into a very broadly applicable programming language.

A.2.5 A Windows-specific interface: using COM

Example for using the library

```
#Enable COM support
from win32com.client import Dispatch

#Load the mpNumerics library
mp = Dispatch("mpNumerics.mp_Lib")

#Set Floating point type to MPFR with 60 decimal digits precision
mp.FloatingPointType = 3
mp.Prec10 = 60

#Assign values to x1 and x2
x1 = mp.Real(4.5)
x2 = mp.Real(1.21)

#Calculate x3 = x1 / x2
x3 = x1.Div(x2)
```

```
#Print the value of x3
print (x3.Str())
```

Example for using Excel

```
#Enable COM support
from win32com.client import Dispatch

#Load the Excel library
xl = Dispatch("Excel.Application")
xl.Visible = 1
xl.Workbooks.Add()
xl.Cells(1,1).Value = "Hello442"
print("From Python")
```

Appendix B

Building the library and toolbox

Building the toolbox and the library from scratch is a much more involved process than just using them.
Conceptually, it could be described as a top-down process which starts at the level of the modification of the source files for the documentation, the following (automated) generation of various *.xml, *.cs, *.h files and their processing with appropriate tools, which create the .NET, COM, native DLL and spreadsheet interfaces, ultimately leading to the connecting point of the mpNumC.h header file.
It could also be described as a bottom-up process, starting with the compilation of the *.c, *.h amd *.asm of the GMP, MPFR and FLINT library. followed by the compilation of the Eigen and Boost template libraries with the various supported data types, again leading to the connecting point of the mpNumC.h header file.
In practice, it is easiest to start any rebuilding of the toolbox or the library with an already working installation, with the following steps in mind:

- When changing a function, or introducing a new one, always start at the documentation, and provide all information which is required for automated generation of dependent files.
- Compile the documentation in latex, and process the output with makemenu etc.
- Run the routines which are necessary to update the .NET, COM, native DLL and spreadsheet interfaces.
- Decide whether you need to update the mpNumC.h header file.

Alternatively, you could start with a breaking change in one of the underlying libraries (e.g. GMP), recompile them first, then recompile all of the dependent libraries.

Appendix C

Acknowledgements

C.1 Contributors to libraries used in the numerical routines

C.1.1 Contributors to mpMath

The following text has been copied from the mpMath manual (0.19):

xxxx

C.1.2 Contributors to gmpy2

The following text has been copied from the gmpy2 manual (2.05):

xxxx

Appendix D

Licenses

D.1 GNU Licenses

D.1.1 GNU General Public License, Version 2

GNU LIBRARY GENERAL PUBLIC LICENSE
Version 2, June 1991

Copyright (C) 1991 Free Software Foundation, Inc.
51 Franklin St, Fifth Floor, Boston, MA 02110-1301, USA
Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.
[This is the first released version of the library GPL. It is numbered 2 because it goes with version 2 of the ordinary GPL.] Preamble The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users.
This license, the Library General Public License, applies to some specially designated Free Software Foundation software, and to any other libraries whose authors decide to use it. You can use it for your libraries, too.
When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.
To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library, or if you modify it.
For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link a program with the library, you must provide complete object files to the recipients so that they can relink them with the library, after making changes to the library and recompiling it. And you must show them these terms so they know their rights.
Our method of protecting your rights has two steps: (1) copyright the library, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the library.
Also, for each distributor's protection, we want to make certain that everyone understands that there is no warranty for this free library. If the library is modified by someone else and passed on, we want its recipients to know that what they have is not the original version, so that any problems introduced by others will not reflect on the original authors' reputations.
Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that companies distributing free software will individually obtain patent licenses, thus in effect transforming

the program into proprietary software. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License, which was designed for utility programs. This license, the GNU Library General Public License, applies to certain designated libraries. This license is quite different from the ordinary one; be sure to read it in full, and don't assume that anything in it is the same as in the ordinary license.

The reason we have a separate public license for some libraries is that they blur the distinction we usually make between modifying or adding to a program and simply using it. Linking a program with a library, without changing the library, is in some sense simply using the library, and is analogous to running a utility program or application program. However, in a textual and legal sense, the linked executable is a combined work, a derivative of the original library, and the ordinary General Public License treats it as such.

Because of this blurred distinction, using the ordinary General Public License for libraries did not effectively promote software sharing, because most developers did not use the libraries. We concluded that weaker conditions might promote sharing better.

However, unrestricted linking of non-free programs would deprive the users of those programs of all benefit from the free status of the libraries themselves. This Library General Public License is intended to permit developers of non-free programs to use free libraries, while preserving your freedom as a user of such programs to change the free libraries that are incorporated in them. (We have not seen how to achieve this as regards changes in header files, but we have achieved it as regards changes in the actual functions of the Library.) The hope is that this will lead to faster development of free libraries.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, while the latter only works together with the library.

Note that it is possible for a library to be covered by the ordinary General Public License rather than by this special one.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Library General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:
âĂća) The modified work must itself be a software library. âĂćb) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change. âĂćc) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License. âĂćd) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful. (For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)
These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.
Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.
In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.
Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.
This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.
If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a ”work that uses the Library”. Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.
However, linking a ”work that uses the Library” with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a ”work that uses the library”.

The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also compile or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

âĂća) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.) âĂćb) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution. âĂćc) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place. âĂćd) Verify that the user has already received a copy of these materials or that you have already sent this user a copy. For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

âĂća) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above. âĂćb) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.
If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.
It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.
This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Library General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.
Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions

are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
END OF TERMS AND CONDITIONS How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).
To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.
one line to give the library's name and an idea of what it does. Copyright (C) year name of author
This library is free software; you can redistribute it and/or modify it under the terms of the GNU Library General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Library General Public License for more details.
You should have received a copy of the GNU Library General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301, USA. Also add information on how to contact you by electronic and paper mail.
You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary. Here is a sample; alter the names:
Yoyodyne, Inc., hereby disclaims all copyright interest in the library 'Frob' (a library for tweaking knobs) written by James Random Hacker.
signature of Ty Coon, 1 April 1990 Ty Coon, President of Vice That's all there is to it!

D.1.2 GNU Library General Public License, Version 2

GNU LIBRARY GENERAL PUBLIC LICENSE
Version 2, June 1991

Copyright (C) 1991 Free Software Foundation, Inc.
51 Franklin St, Fifth Floor, Boston, MA 02110-1301, USA
Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

[This is the first released version of the library GPL. It is numbered 2 because it goes with version 2 of the ordinary GPL.] Preamble The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users.

This license, the Library General Public License, applies to some specially designated Free Software Foundation software, and to any other libraries whose authors decide to use it. You can use it for your libraries, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library, or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link a program with the library, you must provide complete object files to the recipients so that they can relink them with the library, after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

Our method of protecting your rights has two steps: (1) copyright the library, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the library.

Also, for each distributor's protection, we want to make certain that everyone understands that there is no warranty for this free library. If the library is modified by someone else and passed on, we want its recipients to know that what they have is not the original version, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that companies distributing free software will individually obtain patent licenses, thus in effect transforming the program into proprietary software. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License, which was designed for utility programs. This license, the GNU Library General Public License, applies to certain designated libraries. This license is quite different from the ordinary one; be sure to read it in full, and don't assume that anything in it is the same as in the ordinary license.

The reason we have a separate public license for some libraries is that they blur the distinction we usually make between modifying or adding to a program and simply using it. Linking a program with a library, without changing the library, is in some sense simply using the library, and is analogous to running a utility program or application program. However, in a textual and legal sense, the linked executable is a combined work, a derivative of the original library, and the ordinary General Public License treats it as such.

Because of this blurred distinction, using the ordinary General Public License for libraries did not effectively promote software sharing, because most developers did not use the libraries. We concluded that weaker conditions might promote sharing better.

However, unrestricted linking of non-free programs would deprive the users of those programs of all benefit from the free status of the libraries themselves. This Library General Public License is intended

to permit developers of non-free programs to use free libraries, while preserving your freedom as a user of such programs to change the free libraries that are incorporated in them. (We have not seen how to achieve this as regards changes in header files, but we have achieved it as regards changes in the actual functions of the Library.) The hope is that this will lead to faster development of free libraries.
The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, while the latter only works together with the library.
Note that it is possible for a library to be covered by the ordinary General Public License rather than by this special one.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Library General Public License (also called "this License"). Each licensee is addressed as "you".
A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.
The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)
"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.
Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.
You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:
âĂća) The modified work must itself be a software library. âĂćb) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change. âĂćc) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License. âĂćd) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful. (For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)
These requirements apply to the modified work as a whole. If identifiable sections of that work are

not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.
Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.
In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.
Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.
This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.
If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a ”work that uses the Library”. Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.
However, linking a ”work that uses the Library” with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a ”work that uses the library”. The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.
When a ”work that uses the Library” uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.
If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)
Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also compile or link a ”work that uses the Library” with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer’s own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:
âĂća) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.) âĂćb) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution. âĂćc) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place. âĂćd) Verify that the user has already received a copy of these materials or that you have already sent this user a copy. For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.
It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:
âĂća) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above. âĂćb) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason

(not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.
If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.
It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.
This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Library General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.
Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and ”any later version”, you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY ”AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR RE-

DISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
END OF TERMS AND CONDITIONS How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).
To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the ”copyright” line and a pointer to where the full notice is found.
one line to give the library’s name and an idea of what it does. Copyright (C) year name of author
This library is free software; you can redistribute it and/or modify it under the terms of the GNU Library General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Library General Public License for more details.
You should have received a copy of the GNU Library General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301, USA. Also add information on how to contact you by electronic and paper mail.
You should also get your employer (if you work as a programmer) or your school, if any, to sign a ”copyright disclaimer” for the library, if necessary. Here is a sample; alter the names:
Yoyodyne, Inc., hereby disclaims all copyright interest in the library ‘Frob’ (a library for tweaking knobs) written by James Random Hacker.
signature of Ty Coon, 1 April 1990 Ty Coon, President of Vice That’s all there is to it!

D.1.3 GNU Lesser General Public License, Version 3

GNU LESSER GENERAL PUBLIC LICENSE
Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

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

0. Additional Definitions.
As used herein, "this License" refers to version 3 of the GNU Lesser General Public License, and the "GNU GPL" refers to version 3 of the GNU General Public License.
"The Library" refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.
An "Application" is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.
A "Combined Work" is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the "Linked Version".
The "Minimal Corresponding Source" for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.
The "Corresponding Application Code" for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work.

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

2. Conveying Modified Versions.
If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:
a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or
b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.
The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:
a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License.
b) Accompany the object code with a copy of the GNU GPL and this license document.

4. Combined Works.
You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:
a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License.
b) Accompany the Combined Work with a copy of the GNU GPL and this license document.
c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document.
d) Do one of the following:
0) Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.
1) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user's computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version.
e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

5. Combined Libraries.
You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:
a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License.
b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.
The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.
Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.
If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy's public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.

D.1.4 GNU General Public License, Version 3

GNU GENERAL PUBLIC LICENSE
Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble
The GNU General Public License is a free, copyleft license for software and other kinds of works.
The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.
When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.
To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.
For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.
Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.
For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.
Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.
Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.
The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.
"This License" refers to version 3 of the GNU General Public License.
"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.
To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.
A "covered work" means either the unmodified Program or a work based on the Program.
To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.
To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.
An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.
The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.
A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.
The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.
The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.
The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.
The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.
All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.
You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose

of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.
Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.
No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.
When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.
You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.
You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.
You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:
a) The work must carry prominent notices stating that you modified it, and giving a relevant date.
b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.
A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.
You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:
a) Convey the object code in, or embodied in, a physical product (including a physical distribution

medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.
c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.
A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.
A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.
"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.
If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).
The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.
Corresponding Source conveyed, and Installation Information provided, in accord with this section must

be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.
"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.
When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.
Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:
a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
d) Limiting the use for publicity purposes of names of licensors or authors of the material; or
e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.
All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.
If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.
Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.
You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).
However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.
Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.
You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.
Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.
An ”entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.
You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.
A ”contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's ”contributor version”.
A contributor's ”essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, ”control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.
Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.
In the following three paragraphs, a ”patent license” is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To ”grant” such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.
If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. ”Knowingly relying” means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.
A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.
Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.
If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.
Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.
The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.
Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.
If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.
Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.
THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY

KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.
IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.
If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.
To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the ”copyright” line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.> Copyright (C) <year> <name of author>
This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>

Also add information on how to contact you by electronic and paper mail.
If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:
<program> Copyright (C) <year> <name of author> This program comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’. This is free software, and you are welcome to redistribute it under certain conditions; type ‘show c’ for details.
The hypothetical commands ‘show w’ and ‘show c’ should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would

use an ”about box”.
You should also get your employer (if you work as a programmer) or school, if any, to sign a ”copyright disclaimer” for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <http://www.gnu.org/licenses/>
The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read
<http://www.gnu.org/philosophy/why-not-lgpl.html>

D.1.5 GNU Free Documentation License, Version 1.3

GNU Free Documentation License
1.3, 3 November 2008

Copyright (C) 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE
The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.
This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.
We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS
This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.
A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.
A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.
The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.
The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.
A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise

Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".
Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.
The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.
The "publisher" means any person or entity that distributes copies of the Document to the public.
A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.
The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING
You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.
You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY
If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.
If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.
If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy

(directly or through your agents or retailers) of that edition to the public.
It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS
You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:
A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
H. Include an unaltered copy of this License.
I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
O. Preserve any Warranty Disclaimers.
If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.
You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties-for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one. The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS
You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.
The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.
In the combination, you must combine any sections Entitled ”History” in the various original documents, forming one section Entitled ”History”; likewise combine any sections Entitled ”Acknowledgements”, and any sections Entitled ”Dedications”. You must delete all sections Entitled ”Endorsements”.

6. COLLECTIONS OF DOCUMENTS
You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.
You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS
A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an ”aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.
If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION
Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or

disclaimer, the original version will prevail.
If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION
You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.
However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.
Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.
Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE
The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.
Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING
"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.
"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.
"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.
An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.
The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:
Copyright (C) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled ”GNU Free Documentation License”.
If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the ”with ? Texts.” line with this:
with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.
If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.
If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

D.2 Other Licenses

D.2.1 Mozilla Public License, Version 2.0

Mozilla Public License Version 2.0

1. Definitions
1.1. ”Contributor”
means each individual or legal entity that creates, contributes to the creation of, or owns Covered Software.
1.2. ”Contributor Version”
means the combination of the Contributions of others (if any) used by a Contributor and that particular ContributorâĂŹs Contribution.
1.3. ”Contribution”
means Covered Software of a particular Contributor.
1.4. ”Covered Software”
means Source Code Form to which the initial Contributor has attached the notice in Exhibit A, the Executable Form of such Source Code Form, and Modifications of such Source Code Form, in each case including portions thereof.
1.5. ”Incompatible With Secondary Licenses”
means
a.that the initial Contributor has attached the notice described in Exhibit B to the Covered Software; or
b.that the Covered Software was made available under the terms of version 1.1 or earlier of the License, but not also under the terms of a Secondary License.
1.6. ”Executable Form”
means any form of the work other than Source Code Form.
1.7. ”Larger Work”
means a work that combines Covered Software with other material, in a separate file or files, that is not Covered Software.
1.8. ”License”
means this document.
1.9. ”Licensable”
means having the right to grant, to the maximum extent possible, whether at the time of the initial grant or subsequently, any and all of the rights conveyed by this License.
1.10. ”Modifications”
means any of the following:
a.any file in Source Code Form that results from an addition to, deletion from, or modification of the contents of Covered Software; or
b.any new file in Source Code Form that contains any Covered Software.
1.11. ”Patent Claims” of a Contributor
means any patent claim(s), including without limitation, method, process, and apparatus claims, in any patent Licensable by such Contributor that would be infringed, but for the grant of the License, by the making, using, selling, offering for sale, having made, import, or transfer of either its Contributions or its Contributor Version.
1.12. ”Secondary License”
means either the GNU General Public License, Version 2.0, the GNU Lesser General Public License, Version 2.1, the GNU Affero General Public License, Version 3.0, or any later versions of those licenses.
1.13. ”Source Code Form”
means the form of the work preferred for making modifications.
1.14. ”You” (or ”Your”)

means an individual or a legal entity exercising rights under this License. For legal entities, "You" includes any entity that controls, is controlled by, or is under common control with You. For purposes of this definition, "control" means (a) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (b) ownership of more than fifty percent (50%) of the outstanding shares or beneficial ownership of such entity.

2. License Grants and Conditions
2.1. Grants
Each Contributor hereby grants You a world-wide, royalty-free, non-exclusive license:
a.under intellectual property rights (other than patent or trademark) Licensable by such Contributor to use, reproduce, make available, modify, display, perform, distribute, and otherwise exploit its Contributions, either on an unmodified basis, with Modifications, or as part of a Larger Work; and
b.under Patent Claims of such Contributor to make, use, sell, offer for sale, have made, import, and otherwise transfer either its Contributions or its Contributor Version.
2.2. Effective Date
The licenses granted in Section 2.1 with respect to any Contribution become effective for each Contribution on the date the Contributor first distributes such Contribution.
2.3. Limitations on Grant Scope
The licenses granted in this Section 2 are the only rights granted under this License. No additional rights or licenses will be implied from the distribution or licensing of Covered Software under this License. Notwithstanding Section 2.1(b) above, no patent license is granted by a Contributor:
a.for any code that a Contributor has removed from Covered Software; or
b.for infringements caused by: (i) Your and any other third partyâĂŹs modifications of Covered Software, or (ii) the combination of its Contributions with other software (except as part of its Contributor Version); or
c.under Patent Claims infringed by Covered Software in the absence of its Contributions.
This License does not grant any rights in the trademarks, service marks, or logos of any Contributor (except as may be necessary to comply with the notice requirements in Section 3.4).
2.4. Subsequent Licenses
No Contributor makes additional grants as a result of Your choice to distribute the Covered Software under a subsequent version of this License (see Section 10.2) or under the terms of a Secondary License (if permitted under the terms of Section 3.3).
2.5. Representation
Each Contributor represents that the Contributor believes its Contributions are its original creation(s) or it has sufficient rights to grant the rights to its Contributions conveyed by this License.
2.6. Fair Use
This License is not intended to limit any rights You have under applicable copyright doctrines of fair use, fair dealing, or other equivalents.
2.7. Conditions
Sections 3.1, 3.2, 3.3, and 3.4 are conditions of the licenses granted in Section 2.1.

3. Responsibilities
3.1. Distribution of Source Form
All distribution of Covered Software in Source Code Form, including any Modifications that You create or to which You contribute, must be under the terms of this License. You must inform recipients that the Source Code Form of the Covered Software is governed by the terms of this License, and how they can obtain a copy of this License. You may not attempt to alter or restrict the recipientsâĂŹ rights in the Source Code Form.
3.2. Distribution of Executable Form
If You distribute Covered Software in Executable Form then:
a.such Covered Software must also be made available in Source Code Form, as described in Section 3.1, and You must inform recipients of the Executable Form how they can obtain a copy of such Source Code

Form by reasonable means in a timely manner, at a charge no more than the cost of distribution to the recipient; and

b.You may distribute such Executable Form under the terms of this License, or sublicense it under different terms, provided that the license for the Executable Form does not attempt to limit or alter the recipientsâĂŹ rights in the Source Code Form under this License.

3.3. Distribution of a Larger Work

You may create and distribute a Larger Work under terms of Your choice, provided that You also comply with the requirements of this License for the Covered Software. If the Larger Work is a combination of Covered Software with a work governed by one or more Secondary Licenses, and the Covered Software is not Incompatible With Secondary Licenses, this License permits You to additionally distribute such Covered Software under the terms of such Secondary License(s), so that the recipient of the Larger Work may, at their option, further distribute the Covered Software under the terms of either this License or such Secondary License(s).

3.4. Notices

You may not remove or alter the substance of any license notices (including copyright notices, patent notices, disclaimers of warranty, or limitations of liability) contained within the Source Code Form of the Covered Software, except that You may alter any license notices to the extent required to remedy known factual inaccuracies.

3.5. Application of Additional Terms

You may choose to offer, and to charge a fee for, warranty, support, indemnity or liability obligations to one or more recipients of Covered Software. However, You may do so only on Your own behalf, and not on behalf of any Contributor. You must make it absolutely clear that any such warranty, support, indemnity, or liability obligation is offered by You alone, and You hereby agree to indemnify every Contributor for any liability incurred by such Contributor as a result of warranty, support, indemnity or liability terms You offer. You may include additional disclaimers of warranty and limitations of liability specific to any jurisdiction.

4. Inability to Comply Due to Statute or Regulation

If it is impossible for You to comply with any of the terms of this License with respect to some or all of the Covered Software due to statute, judicial order, or regulation then You must: (a) comply with the terms of this License to the maximum extent possible; and (b) describe the limitations and the code they affect. Such description must be placed in a text file included with all distributions of the Covered Software under this License. Except to the extent prohibited by statute or regulation, such description must be sufficiently detailed for a recipient of ordinary skill to be able to understand it.

5. Termination

5.1. The rights granted under this License will terminate automatically if You fail to comply with any of its terms. However, if You become compliant, then the rights granted under this License from a particular Contributor are reinstated (a) provisionally, unless and until such Contributor explicitly and finally terminates Your grants, and (b) on an ongoing basis, if such Contributor fails to notify You of the non-compliance by some reasonable means prior to 60 days after You have come back into compliance. Moreover, Your grants from a particular Contributor are reinstated on an ongoing basis if such Contributor notifies You of the non-compliance by some reasonable means, this is the first time You have received notice of non-compliance with this License from such Contributor, and You become compliant prior to 30 days after Your receipt of the notice.

5.2. If You initiate litigation against any entity by asserting a patent infringement claim (excluding declaratory judgment actions, counter-claims, and cross-claims) alleging that a Contributor Version directly or indirectly infringes any patent, then the rights granted to You by any and all Contributors for the Covered Software under Section 2.1 of this License shall terminate.

5.3. In the event of termination under Sections 5.1 or 5.2 above, all end user license agreements (excluding distributors and resellers) which have been validly granted by You or Your distributors under this License prior to termination shall survive termination.

6. Disclaimer of Warranty
Covered Software is provided under this License on an ”as is” basis, without warranty of any kind, either expressed, implied, or statutory, including, without limitation, warranties that the Covered Software is free of defects, merchantable, fit for a particular purpose or non-infringing. The entire risk as to the quality and performance of the Covered Software is with You. Should any Covered Software prove defective in any respect, You (not any Contributor) assume the cost of any necessary servicing, repair, or correction. This disclaimer of warranty constitutes an essential part of this License. No use of any Covered Software is authorized under this License except under this disclaimer.

7. Limitation of Liability
Under no circumstances and under no legal theory, whether tort (including negligence), contract, or otherwise, shall any Contributor, or anyone who distributes Covered Software as permitted above, be liable to You for any direct, indirect, special, incidental, or consequential damages of any character including, without limitation, damages for lost profits, loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses, even if such party shall have been informed of the possibility of such damages. This limitation of liability shall not apply to liability for death or personal injury resulting from such partyâĂŹs negligence to the extent applicable law prohibits such limitation. Some jurisdictions do not allow the exclusion or limitation of incidental or consequential damages, so this exclusion and limitation may not apply to You.

8. Litigation
Any litigation relating to this License may be brought only in the courts of a jurisdiction where the defendant maintains its principal place of business and such litigation shall be governed by laws of that jurisdiction, without reference to its conflict-of-law provisions. Nothing in this Section shall prevent a partyâĂŹs ability to bring cross-claims or counter-claims.

9. Miscellaneous
This License represents the complete agreement concerning the subject matter hereof. If any provision of this License is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable. Any law or regulation which provides that the language of a contract shall be construed against the drafter shall not be used to construe this License against a Contributor.

10. Versions of the License
10.1. New Versions
Mozilla Foundation is the license steward. Except as provided in Section 10.3, no one other than the license steward has the right to modify or publish new versions of this License. Each version will be given a distinguishing version number.
10.2. Effect of New Versions
You may distribute the Covered Software under the terms of the version of the License under which You originally received the Covered Software, or under the terms of any subsequent version published by the license steward.
10.3. Modified Versions
If you create software not governed by this License, and you want to create a new license for such software, you may create and use a modified version of this License if you rename the license and remove any references to the name of the license steward (except to note that such modified license differs from this License).
10.4. Distributing Source Code Form that is Incompatible With Secondary Licenses
If You choose to distribute Source Code Form that is Incompatible With Secondary Licenses under the terms of this version of the License, the notice described in Exhibit B of this License must be attached.

Exhibit A - Source Code Form License Notice
This Source Code Form is subject to the terms of the Mozilla Public License, v. 2.0. If a copy of the MPL was not distributed with this file, You can obtain one at http://mozilla.org/MPL/2.0/.

If it is not possible or desirable to put the notice in a particular file, then You may include the notice in a location (such as a LICENSE file in a relevant directory) where a recipient would be likely to look for such a notice.
You may add additional accurate notices of copyright ownership.

Exhibit B - ”Incompatible With Secondary Licenses” Notice
This Source Code Form is ”Incompatible With Secondary Licenses”, as defined by the Mozilla Public License, v. 2.0.

D.2.2 New BSD License (for mpMath)

New BSD License (for mpMath)

Copyright (c) 2005-2013 Fredrik Johansson and mpmath contributors
All rights reserved.
Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
a. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. b. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. c. Neither the name of mpmath nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.
THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS ”AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

D.2.3 MIT License

MIT License

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.

D.2.4 Excel-DNA License

Excel-DNA License

Excel-DNA License Copyright (C) 2005-2009 Govert van Drimmelen
This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.

Govert van Drimmelen govert@icon.co.za

Part III

Back Matter

Bibliography

Cuyt, A.A.M., Verdonk, B., Becuve, S., & Kuterna, P. 2001. A remarkable example of catastrophic cancellation unraveled. *Computing*, **66**(3), 309–320. 5

Enge, Andreas, Gastineau, Mickaël, Théveny, Philippe, & Zimmermann, Paul. 2012 (July). *mpc — A library for multiprecision complex arithmetic with exact rounding*. 1.0 edn. INRIA. http://mpc.multiprecision.org/. The MPC manual is available from http://www.multiprecision.org/mpc/download/mpc-1.0.1.pdf. 49

Fousse, Laurent, Hanrot, Guillaume, Lefèvre, Vincent, Pélissier, Patrick, & Zimmermann, Paul. 2007. MPFR: A Multiple-Precision Binary Floating-Point Library with Correct Rounding. *ACM Transactions on Mathematical Software*, **33**(2), 13:1–13:15. Online resource: http://www.mpfr.org/. The MPFR manual is available from http://www.mpfr.org/mpfr-current/mpfr.pdf, and a description of the algorithms from http://www.mpfr.org/algorithms.pdf. 49

Ghazi, Kaveh R., Lefèvre, Vincent, Théveny, Philippe, & Zimmermann, Paul. 2010. Why and How to Use Arbitrary Precision. *Computing in Science and Engineering*, **12**(3), 62–65. Preprint available at http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.212.6321&rep=rep1&type=pdf. 5

Goldberg, David. 1991. What Every Computer Scientist Should Know About Floating Point Arithmetic. *ACM Computing Surveys*, **23**(1), 5–48. Extended and edited reprint from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.22.6768. 4

Granlund, Torbjörn, & the GMP development team. 2013. *GNU MP: The GNU Multiple Precision Arithmetic Library*. 5.1.2 edn. Online resource: http://gmplib.org/. The GMP manual is available from http://gmplib.org/gmp-man-5.1.2.pdf. 49

Grantham, Jon. 2001. Frobenius pseudoprimes. *Mathematics of Computation*, **70**, 873–891. Available at http://www.ams.org/journals/mcom/2001-70-234/S0025-5718-00-01197-2/home.html. 63

Hart, W., & Gladman, B. 2014. *MPIR: Multiple Precision Integers and Rationals*. Version 2.6.0, http://mpir.org/. 49

Higham, Nicholas J. 2002. *Accuracy and Stability of Numerical Algorithms*. 1st edn. SIAM. Online resource: http://www.maths.manchester.ac.uk/~higham/asna/. 4

Higham, Nicholas J. 2009. *Accuracy and Stability of Numerical Algorithms - Presentation given at 3rd Many-core and ReconïňĄgurable Supercomputing Network Workshop QueenâĂŹs University, Belfast, January 15-16, 2009*. Available as http://cpc.cs.qub.ac.uk/MRSN/higham.pdf. 4

Hofschuster, W., & Krämer, W. 2004. C-XSC 2.0: A C++ Library for Extended Scientific Computing. *Pages 15–35 of:* Alt, R., Frommer, A., Kearfott, R.B., & Luther, W. (eds), *Numerical Software with Result Verification, Lecture Notes in Computer Science*. Springer-Verlag, Heidelberg. Online resource: http://www2.math.uni-wuppertal.de/~xsc/xsc-sprachen.html. A preprint is available from http://www2.math.uni-wuppertal.de/~xsc/preprints/prep_03_5.pdf. 5

Johansson, Fredrik, *et al.* 2013 (December). *mpmath: a Python library for arbitrary-precision floating-point arithmetic (version 0.18)*. http://mpmath.org/. 2

Joye, M., & Quisquater, J.-J. 1996. Efficient computation of full Lucas sequences. *Electronics Letters*, **32(6)**, 537–538. Available at http://joye.site88.net/papers/JQ96lucas.pdf. 65

McKinney, Wes. 2012. *Python for Data Analysis*. O'Reilly Media. Online resource: http://shop.oreilly.com/product/0636920023784.do. 95

Nomenclature

$\chi^2_{\nu,\alpha}$	α quantile of the central χ^2-distribution with ν degrees of freedom (page 188)
$\Gamma(x)$	Gamma Function (page 154)
$\Gamma_p(x)$	Multivariate Gamma Function (page 592)
$\mathbb{C}$	Set of complex numbers (page 887)
$\mathbb{N}$	Set of natural numbers (page 887)
$\mathbb{Q}$	Set of rational numbers (page 887)
$\mathbb{R}$	Set of real numbers (page 887)
$\mathbb{Z}$	Set of integer numbers (page 887)
$\Phi(x)$	CDF of the standardized normal distribution (page 217)
$\phi(x)$	pdf of the standardized normal distribution (page 216)
$\Phi^{-1}(\alpha)$	Inverse CDF of the standardized normal distribution (page 217)
$F_F(m,n,x)$	CDF of the central F-distribution (page 196)
$f_F(m,n,x)$	pdf of the central F-distribution (page 196)
$F_N(x;\mu,\sigma^2)$	CDF of the normal distribution with mean μ and variance σ^2 (page 217)
$F_N(x;\mu,\sigma^2)$	pdf of the normal distribution with mean μ and variance σ^2 (page 216)
$F_N^{-1}(\alpha;\mu,\sigma^2)$	Inverse CDF of the normal distribution with mean μ and variance σ^2 (page 217)
$f_R(r,N;\rho)$	pdf of the Distribution of the Sample Correlation Coefficient (page 616)
$F_t(n,x)$	CDF of the central t-distribution (page 224)
$f_t(n,x)$	pdf of the central t-distribution (page 224)
$F_{\chi^2}(n,x)$	CDF of the central chi-square distribution (page 187)
$f_{\chi^2}(n,x)$	pdf of the central chi-square distribution (page 187)
$F_{\chi^2}(n,x;\lambda)$	CDF of the noncentral chi-square distribution (page 642)
$f_{\chi^2}(n,x;\lambda)$	pdf of the noncentral chi-square distribution (page 641)
$F_{\nu_1,\nu_2,\alpha}$	α quantile of the central F-distribution with ν_1 and ν_2 degrees of freedom (page 196)
$F_{\text{Beta'}}(x;a,b,\lambda)$	CDF of the (singly) noncentral Beta-distribution (page 649)
$f_{\text{Beta'}}(x;a,b,\lambda)$	pdf of the (singly) noncentral Beta-distribution (page 649)
$F_{\text{Beta}}(a,b,x)$	CDF of the central Beta-distribution (page 178)
$f_{\text{Beta}}(a,b,x)$	pdf of the central Beta-distribution (page 178)
$F_{\text{Bin}}(n,k;p)$	CDF of the binomial distribution (page 183)
$f_{\text{Bin}}(n,k;p)$	pmf of the binomial distribution (page 183)
$F_{\text{NegBin}}(n,k;p)$	CDF of the negative binomial distribution (page 213)
$f_{\text{NegBin}}(n,k;p)$	pmf of the negative binomial distribution (page 213)
$f_{F''}(x;m,n)$	pdf of the doubly noncentral F-distribution (page 671)
$f_{F'}(x;m,n)$	CDF of the (singly) noncentral F-distribution (page 665)
$f_{F'}(x;m,n)$	pdf of the (singly) noncentral F-distribution (page 665)
$F_{R^2}(x;p,n,\rho^2)$	CDF of the Square of the Multiple Sample Correlation Coefficient (page 625)
$f_{R^2}(x;p,n,\rho^2)$	pdf of the Square of the Multiple Sample Correlation Coefficient (page 625)

$F_{t''}(t; n; \delta, \theta)$	CDF of the doubly noncentral t-square distribution (page 660)
$f_{t''}(t; n; \delta, \theta)$	pdf of the doubly noncentral t-square distribution (page 659)
$F_{t'}(n, x, \delta)$	CDF of the (singly) noncentral t-distribution (page 653)
$f_{t'}(n, x, \delta)$	pdf of the (singly) noncentral t-distribution (page 652)
$I_\nu(z)$	Modified Bessel function of the first kind of real order ν (page 152)
$J_\nu(z)$	Bessel function of the first kind of real order ν (page 152)
$K_\nu(z)$	Modified Bessel function of the second kind of real order ν (page 153)
$N_{Rho}(\alpha, \beta, \widetilde{\rho})$	Sample size function of the noncentral t-distribution for a given confidence level α, power β and modified noncentrality parameter $\widetilde{\rho}$ (page 623)
$N_{t''}(\alpha, \beta, \widetilde{\rho})$	Sample size function of the doubly noncentral t-distribution for a given confidence level α, power β and modified noncentrality parameter $\widetilde{\rho}$ (page 658)
$N_{t''}(\alpha, \beta, \widetilde{\rho})$	Sample size function of the doubly noncentral t-distribution for a given confidence level α, power β and modified noncentrality parameter $\widetilde{\rho}$ (page 663)
$t_{\nu,\alpha}$	α quantile of the central t-distribution with ν degrees of freedom (page 225)
$T_{\text{Owen}}(a, b)$	Owen's T-Function (page 633)
$t_{n,\delta;\alpha}$	α quantile of the noncentral t-distribution with ν degrees of freedom and noncentrality parameter δ (page 655)
$Y_\nu(z)$	Bessel function of the second kind of real order ν (page 152)
z_α	α quantile of the standardized normal distribution (page 217)
${}_0\widetilde{F}_1(b; x)$	Regularized Confluent Hypergeometric Limit Function (page 422)
${}_0F_1(a; \Omega)$	Confluent Hypergeometric Limit Function for Matrix Argument (page 596)
${}_1\widetilde{F}_1(a, b; z)$	Kummer's Regularized Confluent Hypergeometric Function (page 424)
${}_1F_1(a, b; \Omega)$	Kummer's Confluent Hypergeometric Function for Matrix Argument (page 595)
${}_1F_1(a, b; z)$	Kummer's Confluent Hypergeometric Function (page 423)
${}_2\widetilde{F}_1(a, b; c; z)$	Gauss Regularized Hypergeometric Function (page 431)
${}_2F_1(a, b; c; \mathbf{T})$	Gauss Hypergeometric Function of Matrix Argument (page 594)
${}_2F_1(a, b; c; x)$	Gauss Hypergeometric Function (page 429)
CDF	cumulative distribution function (page 170)
pdf	probability density function (page 170)
pmf	probability mass function (page 170)

Index

Multiprecision Functions
- gmpy2.is_bpsw_prp, 63
- gmpy2.is_euler_prp, 63
- gmpy2.is_extra_strong_lucas_prp, 63
- gmpy2.is_fermat_prp, 63
- gmpy2.is_fibonacci_prp(n,p,q), 64
- gmpy2.is_lucas_prp, 64
- gmpy2.is_selfridge_prp, 64
- gmpy2.is_strong_bpsw_prp, 64
- gmpy2.is_strong_lucas_prp, 65
- gmpy2.is_strong_prp, 65
- gmpy2.is_strong_selfridge_prp, 65
- gmpy2.lucasu, 65
- gmpy2.lucasu_mod, 66
- gmpy2.lucasv, 66
- gmpy2.lucasv_mod, 66