Contents

Chapter 1

Classes

1.1 real – real numbers and its functions

The module `real` provides arbitrary precision real numbers and their utilities. The functions provided are corresponding to the math standard module.

- **Classes**
  - **RealField**
  - **Real**
  - †**Constant**
  - †**ExponentialPowerSeries**
  - †**AbsoluteError**
  - †**RelativeError**
- **Functions**
  - **exp**
  - **sqrt**
  - **log**
  - **log1piter**
  - **piGaussLegendre**
  - **eContinuedFraction**
  - **floor**
  - **ceil**
  - **tranc**
  - **sin**
  - **cos**

- **tan**
- **sinh**
- **cosh**
- **tanh**
- **asin**
- **acos**
- **atan**
- **atan2**
- **hypot**
- **pow**
- **degrees**
- **radians**
- **fabs**
- **fmod**
- **frexp**
- **ldexp**
- **EulerTransform**

This module also provides following constants:

e :
`e` is the base of the natural logarithm function, also called Napier's constant.

pi :
`pi` is the circular constant, also denoted by π .

Log2 :
`Log2` is the natural logarithm of 2.

†**defaultError** :
`defaultError` is the instance of **RelativeError**.

theRealField :
`theRealField` is the instance of **RealField**.

1.1.1 RealField – field of real numbers

The class is for the field of real numbers. The class has the single instance **theRealField**.

This class is a subclass of **Field**.

Initialize (Constructor)

RealField() → *RealField*

Create an instance of RealField. You may not want to create an instance, since there is already **theRealField**.

Attribute

zero :
It expresses the additive unit 0. (read only)

one :
It expresses the multiplicative unit 1. (read only)

Operations

operator	explanation
`x in R`	membership test; return whether an element is in or not.
`repr(R)`	return representation string.
`str(R)`	return string.

Methods

1.1.1.1 getCharacteristic – get characteristic

getCharacteristic(`self`) → *integer*

Return the characteristic, zero.

1.1.1.2 issubring – subring test

issubring(`self`, `aRing`: **Ring**) → *bool*

Report whether another ring contains the real field as subring.

1.1.1.3 issuperring – superring test

issuperring(`self`, `aRing`: **Ring**) → *bool*

Report whether the real field contains another ring as subring.

1.1.2 Real – a Real number

Real is a class of real number. This class is only for consistency for other **Ring** object.

This class is a subclass of **CommutativeRingElement**.

All implemented operators in this class are delegated to Float type.

Initialize (Constructor)

Real(`value`: ***number***) → ***Real***

Construct a Real object.

`value` must be int, long, Float or **Rational**.

Methods

1.1.2.1 getRing – get ring object

getRing(`self`) → ***RealField***

Return the real field instance.

1.1.3 Constant – real number with error correction

Constant provides constant-like behavior for Float calculation context. It caches the constant value and re-computes for more precision by request.

Almost every operators are delegated to the cached value.

Initialize (Constructor)

Constant(`getValue`: ***numbers***, `err`: ***Error***=**defaultError**)
→ ***Constant***

Construct Constant from Float and error. `getValue` must be Float, and `err` must be **AbsoluteError** or **RelativeError**.

Operations

operator	explanation
`S(err)`	Return the value at least as accurate as the given error `err`. .

Examples

```
>>> pi = Constant(piGaussLegendre)
>>> print pi
3.14159265358979
>>> pi + 1
4.14159265358979
>>> pi(RelativeError(0,1,2**100)) # for 100 bit precision
3.14159265358979323846264338327795
```

Methods

1.1.3.1 inverse – inverse value

inverse(`self`) → ***Constant***

Return the inverse of the number.

1.1.3.2 toRational – convert to Rational

toRational(`self`) → ***Rational***

Return a rational number approximating the number.

1.1.4 ExponentialPowerSeries – exponential power series

ExponentialPowerSeries is a class for exponential power series, whose n-th term has form $\frac{a_n x^n}{n!}$.

Initialize (Constructor)

ExponentialPowerSeries(`iterator`: *iterator*) → ***ExponentialPowerSeries***

Construct an exponential power series with coefficient generated by the given `iterator`, which can be an infinite iterator.

Operations

operator	explanation
`S(x,maxerror)`	Return the value of the series with `x` assigned.The maximum error `maxerror` must be given as a **RelativeError** or **AbsoluteError** instance.

Examples

```
>>> expo = ExponentialPowerSeries(itertools.cycle([1]))
>>> expo(.5, defaultError)
Rational(5434422938503507, 3296144130048000)
```

Methods

1.1.4.1 terms – generator of terms of series

terms(`self`, `x`: *numbers*) → *ExponentialPowerSeries*

Generator of terms of series with assigned x value.
`x` must be int, long or Float.

1.1.5 AbsoluteError – absolute error

AbsoluteError is the class of absolute error of real numbers.
this class is deprecated.

1.1.6 RelativeError – relative error

AbsoluteError is the class of relative error of real numbers.
this class is deprecated.

1.1.7 exp(function) – exponential value

exp(x: *number*, err: *Error*=**defaultError**) → *number*

Return exponential of x.
err must be **AbsoluteError** or **RelativeError**.

1.1.8 sqrt(function) – square root

sqrt(x: *number*, err: *Error*=**defaultError**) → *number*

Return square root of x.
err must be **AbsoluteError** or **RelativeError**.

1.1.9 log(function) – logarithm

log(x: *number*, base: *number*=None, err: *Error*=**defaultError**)
→ *number*

Return logarithm of a positive number x.
If an additional argument base is given, it returns logarithm of x to the base.
err must be **AbsoluteError** or **RelativeError**.

1.1.10 log1piter(function) – iterator of log(1+x)

log1piter(xx: *number*) → *iterator*

Return iterator for $\log(1 + x)$.

1.1.11 piGaussLegendre(function) – pi by Gauss-Legendre

piGaussLegendre(err: *Error*=**defaultError**) → *number*

Return pi by Gauss-Legendre algorithm.
err must be **AbsoluteError** or **RelativeError**.

1.1.12 eContinuedFraction(function) – Napier's Constant by continued fraction expansion

eContinuedFraction(err: *Error*=**defaultError**) → *number*

Return the base of natural logarithm **e** by continued fraction expansion.
err must be **AbsoluteError** or **RelativeError**.

1.1.13 floor(function) – floor the number

floor(x: *number*) → *integer*

Return the biggest integer not more than x.

1.1.14 ceil(function) – ceil the number

ceil(x: *number*) → *integer*

Return the smallest integer not less than x.

1.1.15 tranc(function) – round-off the number

tranc(x: *number*) → *integer*

Return the number of rounded off x.

1.1.16 sin(function) – sine function

sin(x: *number*, err: *Error*=**defaultError**) → *number*

Return the sine of x.
err must be **AbsoluteError** or **RelativeError**.

1.1.17 cos(function) – cosine function

cos(x: *number*, err: *Error*=**defaultError**) → *number*

Return the cosine of x.
err must be **AbsoluteError** or **RelativeError**.

1.1.18 tan(function) – tangent function

tan(x: *number*, err: *Error*=**defaultError**) → *number*

Return the tangent of x.
err must be **AbsoluteError** or **RelativeError**.

1.1.19 sinh(function) – hyperbolic sine function

sinh(`x`: *number*, `err`: *Error*=**defaultError**) → *number*

Return the hyperbolic sine of `x`.
`err` must be **AbsoluteError** or **RelativeError**.

1.1.20 cosh(function) – hyperbolic cosine function

cosh(`x`: *number*, `err`: *Error*=**defaultError**) → *number*

Return the hyperbolic cosine of `x`.
`err` must be **AbsoluteError** or **RelativeError**.

1.1.21 tanh(function) – hyperbolic tangent function

tanh(`x`: *number*, `err`: *Error*=**defaultError**) → *number*

Return the hyperbolic tangent of `x`.
`err` must be **AbsoluteError** or **RelativeError**.

1.1.22 asin(function) – arc sine function

asin(`x`: *number*, `err`: *Error*=**defaultError**) → *number*

Return the arc sine of `x`.
`err` must be **AbsoluteError** or **RelativeError**.

1.1.23 acos(function) – arc cosine function

acos(`x`: *number*, `err`: *Error*=**defaultError**) → *number*

Return the arc cosine of `x`.
`err` must be **AbsoluteError** or **RelativeError**.

1.1.24 atan(function) – arc tangent function

atan(`x`: *number*, `err`: *Error*=**defaultError**) → *number*

Return the arc tangent of `x`.
`err` must be **AbsoluteError** or **RelativeError**.

1.1.25 atan2(function) – arc tangent function

atan2(y: *number*, x: *number*, err: *Error*=**defaultError**)
→ *number*

Return the arc tangent of y/x.
Unlike **atan**(y/x), the signs of both x and y are considered.
†It is unrecommended to obtain the value of **pi** with atan2(0,1).
err must be **AbsoluteError** or **RelativeError**.

1.1.26 hypot(function) – Euclidean distance function

hypot(x: *number*, y: *number*, err: *Error*=**defaultError**)
→ *number*

Return $\sqrt{x^2 + y^2}$.
err must be **AbsoluteError** or **RelativeError**.

1.1.27 pow(function) – power function

pow(x: *number*, y: *number*, err: *Error*=**defaultError**)
→ *number*

Return yth power of x.
err must be **AbsoluteError** or **RelativeError**.

1.1.28 degrees(function) – convert angle to degree

degrees(rad: *number*, err: *Error*=**defaultError**) → *number*

Converts angle rad from radians to degrees.
err must be **AbsoluteError** or **RelativeError**.

1.1.29 radians(function) – convert angle to radian

radians(deg: *number*, err: *Error*=**defaultError**) → *number*

Converts angle deg from degrees to radians.
err must be **AbsoluteError** or **RelativeError**.

1.1.30 fabs(function) – absolute value

fabs(x: *number*) → *number*

Return absolute value of x

1.1.31 fmod(function) – modulo function over real

fmod(`x`**:** *number*, `y`**:** *number*) → *number*

Return $x - ny$, where `n` is the quotient of `x / y`, rounded towards zero to an integer.

1.1.32 frexp(function) – expression with base and binary exponent

frexp(`x`**:** *number*) → (*m*,*e*)

Return a tuple `(m,e)`, where $x = m \times 2^e$, $1/2 \leq$ `abs(m)` < 1 and `e` is an integer.

†This function is provided as the counter-part of math.frexp, but it might not be useful.

1.1.33 ldexp(function) – construct number from base and binary exponent

ldexp(`x`**:** *number*, `i`**:** *number*) → *number*

Return $x \times 2^i$.

1.1.34 EulerTransform(function) – iterator yields terms of Euler transform

EulerTransform(`iterator`**:** *iterator*) → *iterator*

Return an iterator which yields terms of Euler transform of the given `iterator`.