math

Package math go1.15.2 Latest

Published: **Sep 9, 2020** | License: BSD-3-Clause | Standard library

Doc Overview Subdirectories Versions Imports Imported By Licenses

Overview

Package math provides basic constants and mathematical functions.

This package does not guarantee bit-identical results across architectures.

Constants

```
const (
    E   = 2.71828182845904523536028747135266249775724709369995957496696763 // https:/
    Pi  = 3.14159265358979323846264338327950288419716939937510582097494459 // https:/
    Phi = 1.61803398874989484820458683436563811772030917980576286213544862 // https:/

    Sqrt2   = 1.41421356237309504880168872420969807856967187537694807317667974 // htt
    SqrtE   = 1.64872127070012814684865078781416357165377610071014801157507931 // htt
    SqrtPi  = 1.77245385090551602729816748334114518279754945612238712821380779 // htt
    SqrtPhi = 1.27201964951406896425242246173749149171560804184009624861664038 // htt

    Ln2    = 0.693147180559945309417232121458176568075500134360255254120680009 // htt
    Log2E  = 1 / Ln2
    Ln10   = 2.30258509299404568401799145468436420760110148862877297603332790 // http
    Log10E = 1 / Ln10
)
```

Mathematical constants.

```
const (
    MaxFloat32             = 3.40282346638528859811704183484516925440e+38  // 2**127
    SmallestNonzeroFloat32 = 1.401298464324817070923729583289916131280e-45 // 1 / 2**

    MaxFloat64             = 1.797693134862315708145274237317043567981e+308 // 2**102
    SmallestNonzeroFloat64 = 4.940656458412465441765687928682213723651e-324 // 1 / 2*
)
```

Floating-point limit values. Max is the largest finite value representable by the type. SmallestNonzero is the smallest positive, non-zero value representable by the type.

```
const (
    MaxInt8   = 1<<7 - 1
    MinInt8   = -1 << 7
    MaxInt16  = 1<<15 - 1
    MinInt16  = -1 << 15
    MaxInt32  = 1<<31 - 1
    MinInt32  = -1 << 31
    MaxInt64  = 1<<63 - 1
    MinInt64  = -1 << 63
    MaxUint8  = 1<<8 - 1
    MaxUint16 = 1<<16 - 1
    MaxUint32 = 1<<32 - 1
    MaxUint64 = 1<<64 - 1
)
```

Integer limit values.

func Abs

```
func Abs(x float64) float64
```

Abs returns the absolute value of x.

Special cases are:

```
Abs(±Inf) = +Inf
Abs(NaN) = NaN
```

func Acos

```
func Acos(x float64) float64
```

Acos returns the arccosine, in radians, of x.

Special case is:

```
Acos(x) = NaN if x < -1 or x > 1
```

func Acosh

```
func Acosh(x float64) float64
```

Acosh returns the inverse hyperbolic cosine of x.

Special cases are:

```
Acosh(+Inf) = +Inf
Acosh(x) = NaN if x < 1
Acosh(NaN) = NaN
```

func Asin

```
func Asin(x float64) float64
```

Asin returns the arcsine, in radians, of x.

Special cases are:

```
Asin(±0) = ±0
Asin(x) = NaN if x < -1 or x > 1
```

func Asinh

```
func Asinh(x float64) float64
```

Asinh returns the inverse hyperbolic sine of x.

Special cases are:

```
Asinh(±0) = ±0
Asinh(±Inf) = ±Inf
Asinh(NaN) = NaN
```

func Atan

```
func Atan(x float64) float64
```

Atan returns the arctangent, in radians, of x.

Special cases are:

```
Atan(±0) = ±0
Atan(±Inf) = ±Pi/2
```

func Atan2

```
func Atan2(y, x float64) float64
```

Atan2 returns the arc tangent of y/x, using the signs of the two to determine the quadrant of the return value.

Special cases are (in order):

```
Atan2(y, NaN) = NaN
Atan2(NaN, x) = NaN
Atan2(+0, x>=0) = +0
Atan2(-0, x>=0) = -0
Atan2(+0, x<=-0) = +Pi
```

```
Atan2(-0, x<=-0) = -Pi
Atan2(y>0, 0) = +Pi/2
Atan2(y<0, 0) = -Pi/2
Atan2(+Inf, +Inf) = +Pi/4
Atan2(-Inf, +Inf) = -Pi/4
Atan2(+Inf, -Inf) = 3Pi/4
Atan2(-Inf, -Inf) = -3Pi/4
Atan2(y, +Inf) = 0
Atan2(y>0, -Inf) = +Pi
Atan2(y<0, -Inf) = -Pi
Atan2(+Inf, x) = +Pi/2
Atan2(-Inf, x) = -Pi/2
```

func Atanh

```
func Atanh(x float64) float64
```

Atanh returns the inverse hyperbolic tangent of x.

Special cases are:

```
Atanh(1) = +Inf
Atanh(±0) = ±0
Atanh(-1) = -Inf
Atanh(x) = NaN if x < -1 or x > 1
Atanh(NaN) = NaN
```

func Cbrt

```
func Cbrt(x float64) float64
```

Cbrt returns the cube root of x.

Special cases are:

```
Cbrt(±0) = ±0
Cbrt(±Inf) = ±Inf
Cbrt(NaN) = NaN
```

func Ceil

```
func Ceil(x float64) float64
```

Ceil returns the least integer value greater than or equal to x.

Special cases are:

```
Ceil(±0) = ±0
Ceil(±Inf) = ±Inf
Ceil(NaN) = NaN
```

func Copysign

```
func Copysign(x, y float64) float64
```

Copysign returns a value with the magnitude of x and the sign of y.

func Cos

```
func Cos(x float64) float64
```

Cos returns the cosine of the radian argument x.

Special cases are:

```
Cos(±Inf) = NaN
Cos(NaN) = NaN
```

func Cosh

```
func Cosh(x float64) float64
```

Cosh returns the hyperbolic cosine of x.

Special cases are:

```
Cosh(±0) = 1
Cosh(±Inf) = +Inf
Cosh(NaN) = NaN
```

func Dim

```
func Dim(x, y float64) float64
```

Dim returns the maximum of x-y or 0.

Special cases are:

```
Dim(+Inf, +Inf) = NaN
Dim(-Inf, -Inf) = NaN
Dim(x, NaN) = Dim(NaN, x) = NaN
```

func Erf

```
func Erf(x float64) float64
```

Erf returns the error function of x.

Special cases are:

```
Erf(+Inf) = 1
Erf(-Inf) = -1
Erf(NaN) = NaN
```

func Erfc

```
func Erfc(x float64) float64
```

Erfc returns the complementary error function of x.

Special cases are:

```
Erfc(+Inf) = 0
Erfc(-Inf) = 2
Erfc(NaN) = NaN
```

func Erfcinv

```
func Erfcinv(x float64) float64
```

Erfcinv returns the inverse of Erfc(x).

Special cases are:

```
Erfcinv(0) = +Inf
Erfcinv(2) = -Inf
Erfcinv(x) = NaN if x < 0 or x > 2
Erfcinv(NaN) = NaN
```

func Erfinv

```
func Erfinv(x float64) float64
```

Erfinv returns the inverse error function of x.

Special cases are:

```
Erfinv(1) = +Inf
Erfinv(-1) = -Inf
Erfinv(x) = NaN if x < -1 or x > 1
Erfinv(NaN) = NaN
```

func Exp

```
func Exp(x float64) float64
```

Exp returns e**x, the base-e exponential of x.

Special cases are:

```
Exp(+Inf) = +Inf
Exp(NaN) = NaN
```

Very large values overflow to 0 or +Inf. Very small values underflow to 1.

func Exp2

```
func Exp2(x float64) float64
```

Exp2 returns 2**x, the base-2 exponential of x.

Special cases are the same as Exp.

func Expm1

```
func Expm1(x float64) float64
```

Expm1 returns e**x - 1, the base-e exponential of x minus 1. It is more accurate than Exp(x) - 1 when x is near zero.

Special cases are:

```
Expm1(+Inf) = +Inf
Expm1(-Inf) = -1
Expm1(NaN) = NaN
```

Very large values overflow to -1 or +Inf.

func FMA

```
func FMA(x, y, z float64) float64
```

FMA returns x * y + z, computed with only one rounding. (That is, FMA returns the fused multiply-add of x, y, and z.)

func Float32bits

```
func Float32bits(f float32) uint32
```

Float32bits returns the IEEE 754 binary representation of f, with the sign bit of f and the result in the same bit position. Float32bits(Float32frombits(x)) == x.

func Float32frombits

```
func Float32frombits(b uint32) float32
```

Float32frombits returns the floating-point number corresponding to the IEEE 754 binary representation b, with the sign bit of b and the result in the same bit position. Float32frombits(Float32bits(x)) == x.

func Float64bits

```
func Float64bits(f float64) uint64
```

Float64bits returns the IEEE 754 binary representation of f, with the sign bit of f and the result in the same bit position, and Float64bits(Float64frombits(x)) == x.

func Float64frombits

```
func Float64frombits(b uint64) float64
```

Float64frombits returns the floating-point number corresponding to the IEEE 754 binary representation b, with the sign bit of b and the result in the same bit position. Float64frombits(Float64bits(x)) == x.

func Floor

```
func Floor(x float64) float64
```

Floor returns the greatest integer value less than or equal to x.

Special cases are:

```
Floor(±0) = ±0
Floor(±Inf) = ±Inf
Floor(NaN) = NaN
```

func Frexp

```
func Frexp(f float64) (frac float64, exp int)
```

Frexp breaks f into a normalized fraction and an integral power of two. It returns frac and exp satisfying f == frac × 2**exp, with the absolute value of frac in the interval [½, 1).

Special cases are:

```
Frexp(±0) = ±0, 0
Frexp(±Inf) = ±Inf, 0
Frexp(NaN) = NaN, 0
```

func Gamma

```
func Gamma(x float64) float64
```

Gamma returns the Gamma function of x.

Special cases are:

```
Gamma(+Inf) = +Inf
Gamma(+0) = +Inf
Gamma(-0) = -Inf
Gamma(x) = NaN for integer x < 0
Gamma(-Inf) = NaN
Gamma(NaN) = NaN
```

func Hypot

```
func Hypot(p, q float64) float64
```

Hypot returns Sqrt(p*p + q*q), taking care to avoid unnecessary overflow and underflow.

Special cases are:

```
Hypot(±Inf, q) = +Inf
Hypot(p, ±Inf) = +Inf
Hypot(NaN, q) = NaN
Hypot(p, NaN) = NaN
```

func Ilogb

```
func Ilogb(x float64) int
```

Ilogb returns the binary exponent of x as an integer.

Special cases are:

```
Ilogb(±Inf) = MaxInt32
Ilogb(0) = MinInt32
Ilogb(NaN) = MaxInt32
```

func Inf

```
func Inf(sign int) float64
```

Inf returns positive infinity if sign >= 0, negative infinity if sign < 0.

func IsInf

```
func IsInf(f float64, sign int) bool
```

IsInf reports whether f is an infinity, according to sign. If sign > 0, IsInf reports whether f is positive infinity. If sign < 0, IsInf reports whether f is negative infinity. If sign == 0, IsInf reports whether f is either infinity.

func IsNaN

```
func IsNaN(f float64) (is bool)
```

IsNaN reports whether f is an IEEE 754 ``not-a-number'' value.

func J0

```
func J0(x float64) float64
```

J0 returns the order-zero Bessel function of the first kind.

Special cases are:

```
J0(±Inf) = 0
J0(0) = 1
J0(NaN) = NaN
```

func J1

```
func J1(x float64) float64
```

J1 returns the order-one Bessel function of the first kind.

Special cases are:

```
J1(±Inf) = 0
J1(NaN) = NaN
```

func Jn

```
func Jn(n int, x float64) float64
```

Jn returns the order-n Bessel function of the first kind.

Special cases are:

```
Jn(n, ±Inf) = 0
Jn(n, NaN) = NaN
```

func Ldexp

```
func Ldexp(frac float64, exp int) float64
```

Ldexp is the inverse of Frexp. It returns frac × 2**exp.

Special cases are:

```
Ldexp(±0, exp) = ±0
Ldexp(±Inf, exp) = ±Inf
Ldexp(NaN, exp) = NaN
```

func Lgamma

```
func Lgamma(x float64) (lgamma float64, sign int)
```

Lgamma returns the natural logarithm and sign (-1 or +1) of Gamma(x).

Special cases are:

```
Lgamma(+Inf) = +Inf
Lgamma(0) = +Inf
Lgamma(-integer) = +Inf
Lgamma(-Inf) = -Inf
Lgamma(NaN) = NaN
```

func Log

```
func Log(x float64) float64
```

Log returns the natural logarithm of x.

Special cases are:

```
Log(+Inf) = +Inf
Log(0) = -Inf
Log(x < 0) = NaN
Log(NaN) = NaN
```

func Log10

```
func Log10(x float64) float64
```

Log10 returns the decimal logarithm of x. The special cases are the same as for Log.

func Log1p

```
func Log1p(x float64) float64
```

Log1p returns the natural logarithm of 1 plus its argument x. It is more accurate than Log(1 + x) when x is near zero.

Special cases are:

```
Log1p(+Inf) = +Inf
Log1p(±0) = ±0
Log1p(-1) = -Inf
Log1p(x < -1) = NaN
Log1p(NaN) = NaN
```

func Log2

```
func Log2(x float64) float64
```

Log2 returns the binary logarithm of x. The special cases are the same as for Log.

func Logb

```
func Logb(x float64) float64
```

Logb returns the binary exponent of x.

Special cases are:

```
Logb(±Inf) = +Inf
Logb(0) = -Inf
Logb(NaN) = NaN
```

func Max

```
func Max(x, y float64) float64
```

Max returns the larger of x or y.

Special cases are:

```
Max(x, +Inf) = Max(+Inf, x) = +Inf
Max(x, NaN) = Max(NaN, x) = NaN
Max(+0, ±0) = Max(±0, +0) = +0
Max(-0, -0) = -0
```

func Min

```
func Min(x, y float64) float64
```

Min returns the smaller of x or y.

Special cases are:

```
Min(x, -Inf) = Min(-Inf, x) = -Inf
Min(x, NaN) = Min(NaN, x) = NaN
Min(-0, ±0) = Min(±0, -0) = -0
```

func Mod

```
func Mod(x, y float64) float64
```

Mod returns the floating-point remainder of x/y. The magnitude of the result is less than y and its sign agrees with that of x.

Special cases are:

```
Mod(±Inf, y) = NaN
Mod(NaN, y) = NaN
Mod(x, 0) = NaN
Mod(x, ±Inf) = x
Mod(x, NaN) = NaN
```

func Modf

```
func Modf(f float64) (int float64, frac float64)
```

Modf returns integer and fractional floating-point numbers that sum to f. Both values have the same sign as f.

Special cases are:

```
Modf(±Inf) = ±Inf, NaN
Modf(NaN) = NaN, NaN
```

func NaN

```
func NaN() float64
```

NaN returns an IEEE 754 ``not-a-number'' value.

func Nextafter

```
func Nextafter(x, y float64) (r float64)
```

Nextafter returns the next representable float64 value after x towards y.

Special cases are:

```
Nextafter(x, x)   = x
Nextafter(NaN, y) = NaN
Nextafter(x, NaN) = NaN
```

func Nextafter32

```
func Nextafter32(x, y float32) (r float32)
```

Nextafter32 returns the next representable float32 value after x towards y.

Special cases are:

```
Nextafter32(x, x)   = x
Nextafter32(NaN, y) = NaN
Nextafter32(x, NaN) = NaN
```

func Pow

```
func Pow(x, y float64) float64
```

Pow returns x**y, the base-x exponential of y.

Special cases are (in order):

```
Pow(x, ±0) = 1 for any x
Pow(1, y) = 1 for any y
Pow(x, 1) = x for any x
Pow(NaN, y) = NaN
Pow(x, NaN) = NaN
Pow(±0, y) = ±Inf for y an odd integer < 0
Pow(±0, -Inf) = +Inf
Pow(±0, +Inf) = +0
Pow(±0, y) = +Inf for finite y < 0 and not an odd integer
Pow(±0, y) = ±0 for y an odd integer > 0
Pow(±0, y) = +0 for finite y > 0 and not an odd integer
Pow(-1, ±Inf) = 1
Pow(x, +Inf) = +Inf for |x| > 1
Pow(x, -Inf) = +0 for |x| > 1
Pow(x, +Inf) = +0 for |x| < 1
Pow(x, -Inf) = +Inf for |x| < 1
Pow(+Inf, y) = +Inf for y > 0
Pow(+Inf, y) = +0 for y < 0
Pow(-Inf, y) = Pow(-0, -y)
Pow(x, y) = NaN for finite x < 0 and finite non-integer y
```

func Pow10

```
func Pow10(n int) float64
```

Pow10 returns 10**n, the base-10 exponential of n.

Special cases are:

```
Pow10(n) =    0 for n < -323
Pow10(n) = +Inf for n > 308
```

func Remainder

```
func Remainder(x, y float64) float64
```

Remainder returns the IEEE 754 floating-point remainder of x/y.

Special cases are:

```
Remainder(±Inf, y) = NaN
Remainder(NaN, y) = NaN
Remainder(x, 0) = NaN
Remainder(x, ±Inf) = x
Remainder(x, NaN) = NaN
```

func Round

```
func Round(x float64) float64
```

Round returns the nearest integer, rounding half away from zero.

Special cases are:

```
Round(±0) = ±0
Round(±Inf) = ±Inf
Round(NaN) = NaN
```

func RoundToEven

```
func RoundToEven(x float64) float64
```

RoundToEven returns the nearest integer, rounding ties to even.

Special cases are:

```
RoundToEven(±0) = ±0
RoundToEven(±Inf) = ±Inf
RoundToEven(NaN) = NaN
```

func Signbit

```
func Signbit(x float64) bool
```

Signbit reports whether x is negative or negative zero.

func Sin

```
func Sin(x float64) float64
```

Sin returns the sine of the radian argument x.

Special cases are:

```
Sin(±0) = ±0
Sin(±Inf) = NaN
Sin(NaN) = NaN
```

func Sincos

```
func Sincos(x float64) (sin, cos float64)
```

Sincos returns Sin(x), Cos(x).

Special cases are:

```
Sincos(±0) = ±0, 1
Sincos(±Inf) = NaN, NaN
Sincos(NaN) = NaN, NaN
```

func Sinh

```
func Sinh(x float64) float64
```

Sinh returns the hyperbolic sine of x.

Special cases are:

```
Sinh(±0) = ±0
Sinh(±Inf) = ±Inf
Sinh(NaN) = NaN
```

func Sqrt

```
func Sqrt(x float64) float64
```

Sqrt returns the square root of x.

Special cases are:

```
Sqrt(+Inf) = +Inf
Sqrt(±0) = ±0
Sqrt(x < 0) = NaN
Sqrt(NaN) = NaN
```

func Tan

```
func Tan(x float64) float64
```

Tan returns the tangent of the radian argument x.

Special cases are:

```
Tan(±0) = ±0
Tan(±Inf) = NaN
Tan(NaN) = NaN
```

func Tanh

```
func Tanh(x float64) float64
```

Tanh returns the hyperbolic tangent of x.

Special cases are:

```
Tanh(±0) = ±0
Tanh(±Inf) = ±1
Tanh(NaN) = NaN
```

func Trunc

```
func Trunc(x float64) float64
```

Trunc returns the integer value of x.

Special cases are:

```
Trunc(±0) = ±0
Trunc(±Inf) = ±Inf
Trunc(NaN) = NaN
```

func Y0

```
func Y0(x float64) float64
```

Y0 returns the order-zero Bessel function of the second kind.

Special cases are:

```
Y0(+Inf) = 0
Y0(0) = -Inf
Y0(x < 0) = NaN
Y0(NaN) = NaN
```

func Y1

```
func Y1(x float64) float64
```

Y1 returns the order-one Bessel function of the second kind.

Special cases are:

```
Y1(+Inf) = 0
Y1(0) = -Inf
Y1(x < 0) = NaN
Y1(NaN) = NaN
```

func Yn

```
func Yn(n int, x float64) float64
```

Yn returns the order-n Bessel function of the second kind.

Special cases are:

```
Yn(n, +Inf) = 0
Yn(n ≥ 0, 0) = -Inf
Yn(n < 0, 0) = +Inf if n is odd, -Inf if n is even
Yn(n, x < 0) = NaN
Yn(n, NaN) = NaN
```