public final class
Math
extends Object

Summary: Constants | Methods | Inherited Methods | [Expand All]
Added in API level 1

java.lang.Object
↳java.lang.Math

Class Overview

Class Math provides basic math constants and operations such as trigonometric functions, hyperbolic functions, exponential, logarithms, etc.

Summary

Constants	
double E	The double value closest to e, the base of the natural logarithm.
double PI	The double value closest to pi, the ratio of a circle's circumference to its diameter.
Public Methods	
static double	IEEEremainder (double x, double y) Returns the remainder of dividing x by y using the IEEE 754 rules.
static double	abs (double d) Returns the absolute value of the argument.
static long	abs (long l) Returns the absolute value of the argument.
static float	abs (float f) Returns the absolute value of the argument.
static int	abs (int i) Returns the absolute value of the argument.
static double	acos (double d) Returns the closest double approximation of the arc cosine of the argument within the range `[0..pi]`.

	asin (double d)
static double	Returns the closest double approximation of the arc sine of the argument within the range `[-pi/2..pi/2]`.
	atan (double d)
static double	Returns the closest double approximation of the arc tangent of the argument within the range `[-pi/2..pi/2]`.
	atan2 (double y, double x)
static double	Returns the closest double approximation of the arc tangent of `y/x` within the range `[-pi..pi]`.
	cbrt (double d)
static double	Returns the closest double approximation of the cube root of the argument.
	ceil (double d)
static double	Returns the double conversion of the most negative (closest to negative infinity) integer value greater than or equal to the argument.
	copySign (double magnitude, double sign)
static double	Returns a double with the given magnitude and the sign of `sign`.
	copySign (float magnitude, float sign)
static float	Returns a float with the given magnitude and the sign of `sign`.
	cos (double d)
static double	Returns the closest double approximation of the cosine of the argument.
	cosh (double d)
static double	Returns the closest double approximation of the hyperbolic cosine of the argument.
	exp (double d)
static double	Returns the closest double approximation of the raising "e" to the power of the argument.
	expm1 (double d)
static double	Returns the closest double

	approximation of $e^{d} - 1$.
	floor(double d)
static double	Returns the double conversion of the most positive (closest to positive infinity) integer value less than or equal to the argument.
	getExponent(float f)
static int	Returns the unbiased base-2 exponent of float f.
	getExponent(double d)
static int	Returns the unbiased base-2 exponent of double d.
	hypot(double x, double y)
static double	Returns $sqrt(x^2 + y^2)$.
	log(double d)
static double	Returns the closest double approximation of the natural logarithm of the argument.
	log10(double d)
static double	Returns the closest double approximation of the base 10 logarithm of the argument.
	log1p(double d)
static double	Returns the closest double approximation of the natural logarithm of the sum of the argument and 1.
	max(long l1, long l2)
static long	Returns the most positive (closest to positive infinity) of the two arguments.
	max(int i1, int i2)
static int	Returns the most positive (closest to positive infinity) of the two arguments.
	max(double d1, double d2)
static double	Returns the most positive (closest to positive infinity) of the two arguments.
	max(float f1, float f2)
static float	Returns the most positive (closest to positive infinity) of the two arguments.
	min(int i1, int i2)
static int	Returns the most negative (closest to negative infinity) of the two arguments.

	min (long l1, long l2)
static long	Returns the most negative (closest to negative infinity) of the two arguments.
	min (double d1, double d2)
static double	Returns the most negative (closest to negative infinity) of the two arguments.
	min (float f1, float f2)
static float	Returns the most negative (closest to negative infinity) of the two arguments.
	nextAfter (float start, double direction)
static float	Returns the next float after `start` in the given `direction`.
	nextAfter (double start, double direction)
static double	Returns the next double after `start` in the given `direction`.
static double	nextUp (double d) Returns the next double larger than `d`.
static float	nextUp (float f) Returns the next float larger than `f`.
	pow (double x, double y)
static double	Returns the closest double approximation of the result of raising `x` to the power of `y`.
	random ()
synchronized static double	Returns a pseudo-random double `n`, where `n >= 0.0 && n < 1.0`.
	rint (double d)
static double	Returns the double conversion of the result of rounding the argument to an integer.
	round (double d)
static long	Returns the result of rounding the argument to an integer.
	round (float f)
static int	Returns the result of rounding the argument to an integer.
static double	scalb (double d, int scaleFactor) Returns `d * 2^scaleFactor`.

static float	scalb (float d, int scaleFactor) Returns d * 2^scaleFactor.
static double	signum (double d) Returns the signum function of the argument.
static float	signum (float f) Returns the signum function of the argument.
static double	sin (double d) Returns the closest double approximation of the sine of the argument.
static double	sinh (double d) Returns the closest double approximation of the hyperbolic sine of the argument.
static double	sqrt (double d) Returns the closest double approximation of the square root of the argument.
static double	tan (double d) Returns the closest double approximation of the tangent of the argument.
static double	tanh (double d) Returns the closest double approximation of the hyperbolic tangent of the argument.
static double	toDegrees (double angrad) Returns the measure in degrees of the supplied radian angle.
static double	toRadians (double angdeg) Returns the measure in radians of the supplied degree angle.
static float	ulp (float f) Returns the argument's ulp (unit in the last place).
static double	ulp (double d) Returns the argument's ulp (unit in the last place).

Inherited Methods [Expand]

▸ From class java.lang.Object

Constants

public static final double **E** Added in API level 1

The double value closest to e, the base of the natural logarithm.

Constant Value: 2.718281828459045

public static final double **PI** Added in API level 1

The double value closest to pi, the ratio of a circle's circumference to its diameter.

Constant Value: 3.141592653589793

Public Methods

public static double **IEEEremainder** (double x, double y) Added in API level 1

Returns the remainder of dividing x by y using the IEEE 754 rules. The result is `x - round(x/p)*p` where `round(x/p)` is the nearest integer (rounded to even), but without numerical cancellation problems.

Special cases:

- `IEEEremainder((anything), 0) = NaN`
- `IEEEremainder(+infinity, (anything)) = NaN`
- `IEEEremainder(-infinity, (anything)) = NaN`
- `IEEEremainder(NaN, (anything)) = NaN`
- `IEEEremainder((anything), NaN) = NaN`
- `IEEEremainder(x, +infinity) = x` where x is anything but +/-infinity
- `IEEEremainder(x, -infinity) = x` where x is anything but +/-infinity

Parameters

x the numerator of the operation.

y the denominator of the operation.

Returns

the IEEE754 floating point reminder of of `x/y`.

public static double **abs** (double d)

Added in API level 1

Returns the absolute value of the argument.

Special cases:

- `abs(-0.0) = +0.0`
- `abs(+infinity) = +infinity`
- `abs(-infinity) = +infinity`
- `abs(NaN) = NaN`

public static long **abs** (long l)

Added in API level 1

Returns the absolute value of the argument. If the argument is `Long.MIN_VALUE`, `Long.MIN_VALUE` is returned.

public static float **abs** (float f)

Added in API level 1

Returns the absolute value of the argument.

Special cases:

- `abs(-0.0) = +0.0`
- `abs(+infinity) = +infinity`
- `abs(-infinity) = +infinity`
- `abs(NaN) = NaN`

public static int **abs** (int i)

Added in API level 1

Returns the absolute value of the argument.

If the argument is `Integer.MIN_VALUE`, `Integer.MIN_VALUE` is returned.

public static double **acos** (double d)

Added in API level 1

Returns the closest double approximation of the arc cosine of the argument within the range `[0..pi]`. The returned result is within 1 ulp (unit in the last place) of the real result.

Special cases:

- `acos((anything > 1) = NaN`
- `acos((anything < -1) = NaN`
- `acos(NaN) = NaN`

Parameters

d the value to compute arc cosine of.

Returns

the arc cosine of the argument.

public static double **asin** (double d) Added in API level 1

Returns the closest double approximation of the arc sine of the argument within the range `[-pi/2..pi/2]`. The returned result is within 1 ulp (unit in the last place) of the real result.

Special cases:

- `asin((anything > 1)) = NaN`
- `asin((anything < -1)) = NaN`
- `asin(NaN) = NaN`

Parameters

d the value whose arc sine has to be computed.

Returns

the arc sine of the argument.

public static double **atan** (double d) Added in API level 1

Returns the closest double approximation of the arc tangent of the argument within the range `[-pi/2..pi/2]`. The returned result is within 1 ulp (unit in the last place) of the real result.

Special cases:

- `atan(+0.0) = +0.0`
- `atan(-0.0) = -0.0`
- `atan(+infinity) = +pi/2`
- `atan(-infinity) = -pi/2`
- `atan(NaN) = NaN`

Parameters

d the value whose arc tangent has to be computed.

Returns

the arc tangent of the argument.

public static double **atan2** (double y, double x) Added in API level 1

Returns the closest double approximation of the arc tangent of `y/x` within the range `[-pi..pi]`. This is the angle of the polar

representation of the rectangular coordinates (x,y). The returned result is within 2 ulps (units in the last place) of the real result.

Special cases:

- `atan2((anything), NaN ) = NaN;`
- `atan2(NaN , (anything) ) = NaN;`
- `atan2(+0.0, +(anything but NaN)) = +0.0`
- `atan2(-0.0, +(anything but NaN)) = -0.0`
- `atan2(+0.0, -(anything but NaN)) = +pi`
- `atan2(-0.0, -(anything but NaN)) = -pi`
- `atan2(+(anything but 0 and NaN), 0) = +pi/2`
- `atan2(-(anything but 0 and NaN), 0) = -pi/2`
- `atan2(+(anything but infinity and NaN), +infinity) = +0.0`
- `atan2(-(anything but infinity and NaN), +infinity) = -0.0`
- `atan2(+(anything but infinity and NaN), -infinity) = +pi`
- `atan2(-(anything but infinity and NaN), -infinity) = -pi`
- `atan2(+infinity, +infinity ) = +pi/4`
- `atan2(-infinity, +infinity ) = -pi/4`
- `atan2(+infinity, -infinity ) = +3pi/4`
- `atan2(-infinity, -infinity ) = -3pi/4`
- `atan2(+infinity, (anything but,0, NaN, and infinity)) = +pi/2`
- `atan2(-infinity, (anything but,0, NaN, and infinity)) = -pi/2`

Parameters

y the numerator of the value whose atan has to be computed.

x the denominator of the value whose atan has to be computed.

Returns

the arc tangent of `y/x`.

public static double **cbrt** (double d)

Added in API level 1

Returns the closest double approximation of the cube root of the argument.

Special cases:

- `cbrt(+0.0) = +0.0`
- `cbrt(-0.0) = -0.0`
- `cbrt(+infinity) = +infinity`
- `cbrt(-infinity) = -infinity`
- `cbrt(NaN) = NaN`

Parameters

d the value whose cube root has to be computed.

Returns

the cube root of the argument.

public static double **ceil** (double d)

Added in API level 1

Returns the double conversion of the most negative (closest to negative infinity) integer value greater than or equal to the argument.

Special cases:

- `ceil(+0.0) = +0.0`
- `ceil(-0.0) = -0.0`
- `ceil((anything in range (-1,0)) = -0.0`
- `ceil(+infinity) = +infinity`
- `ceil(-infinity) = -infinity`
- `ceil(NaN) = NaN`

public static double **copySign** (double magnitude, double sign)

Added in API level 9

Returns a double with the given magnitude and the sign of `sign`. If `sign` is NaN, the sign of the result is arbitrary. If you need a determinate sign in such cases, use `StrictMath.copySign`.

public static float **copySign** (float magnitude, float sign)

Added in API level 9

Returns a float with the given magnitude and the sign of `sign`. If `sign` is NaN, the sign of the result is arbitrary. If you need a determinate sign in such cases, use `StrictMath.copySign`.

public static double **cos** (double d)

Added in API level 1

Returns the closest double approximation of the cosine of the argument. The returned result is within 1 ulp (unit in the last place) of the real result.

Special cases:

- `cos(+infinity) = NaN`
- `cos(-infinity) = NaN`
- `cos(NaN) = NaN`

Parameters

d the angle whose cosine has to be computed, in radians.

Returns

the cosine of the argument.

public static double **cosh** (double d)

Added in API level 1

Returns the closest double approximation of the hyperbolic cosine of the argument. The returned result is within 2.5 ulps (units in the last place) of the real result.

Special cases:

- `cosh(+infinity) = +infinity`
- `cosh(-infinity) = +infinity`
- `cosh(NaN) = NaN`

Parameters

d the value whose hyperbolic cosine has to be computed.

Returns

the hyperbolic cosine of the argument.

public static double **exp** (double d)

Added in API level 1

Returns the closest double approximation of the raising "e" to the power of the argument. The returned result is within 1 ulp (unit in the last place) of the real result.

Special cases:

- `exp(+infinity) = +infinity`
- `exp(-infinity) = +0.0`
- `exp(NaN) = NaN`

Parameters

d the value whose exponential has to be computed.

Returns
the exponential of the argument.

public static double **expm1** (double d)

Added in API level 1

Returns the closest double approximation of $e^{d} - 1$. If the argument is very close to 0, it is much more accurate to use `expm1(d)+1` than `exp(d)` (due to cancellation of significant digits). The returned result is within 1 ulp (unit in the last place) of the real result.

For any finite input, the result is not less than -1.0. If the real result is within 0.5 ulp of -1, -1.0 is returned.

Special cases:

- `expm1(+0.0) = +0.0`
- `expm1(-0.0) = -0.0`
- `expm1(+infinity) = +infinity`
- `expm1(-infinity) = -1.0`
- `expm1(NaN) = NaN`

Parameters

d the value to compute the $e^{d} - 1$ of.

Returns
the $e^{d} - 1$ value of the argument.

public static double **floor** (double d)

Added in API level 1

Returns the double conversion of the most positive (closest to positive infinity) integer value less than or equal to the argument.

Special cases:

- `floor(+0.0) = +0.0`
- `floor(-0.0) = -0.0`
- `floor(+infinity) = +infinity`
- `floor(-infinity) = -infinity`
- `floor(NaN) = NaN`

public static int **getExponent** (float f)

Added in API level 9

Returns the unbiased base-2 exponent of float `f`.

public static int **getExponent** (double d)

Added in API level 9

Returns the unbiased base-2 exponent of double d.

public static double **hypot** (double x, double y)

Added in API level 1

Returns `sqrt(`x^2`+` y^2`)`. The final result is without medium underflow or overflow. The returned result is within 1 ulp (unit in the last place) of the real result. If one parameter remains constant, the result should be semi-monotonic.

Special cases:

- `hypot(+infinity, (anything including NaN)) = +infinity`
- `hypot(-infinity, (anything including NaN)) = +infinity`
- `hypot((anything including NaN), +infinity) = +infinity`
- `hypot((anything including NaN), -infinity) = +infinity`
- `hypot(NaN, NaN) = NaN`

Parameters

x a double number.

y a double number.

Returns

the `sqrt(`x^2`+` y^2`)` value of the arguments.

public static double **log** (double d)

Added in API level 1

Returns the closest double approximation of the natural logarithm of the argument. The returned result is within 1 ulp (unit in the last place) of the real result.

Special cases:

- `log(+0.0) = -infinity`
- `log(-0.0) = -infinity`
- `log((anything < 0) = NaN`
- `log(+infinity) = +infinity`
- `log(-infinity) = NaN`
- `log(NaN) = NaN`

Parameters

d the value whose log has to be computed.

Returns

the natural logarithm of the argument.

public static double **log10** (double d)

Added in API level 1

Returns the closest double approximation of the base 10 logarithm of the argument. The returned result is within 1 ulp (unit in the last place) of the real result.

Special cases:

- `log10(+0.0) = -infinity`
- `log10(-0.0) = -infinity`
- `log10((anything < 0) = NaN`
- `log10(+infinity) = +infinity`
- `log10(-infinity) = NaN`
- `log10(NaN) = NaN`

Parameters

d the value whose base 10 log has to be computed.

Returns

the natural logarithm of the argument.

public static double **log1p** (double d)

Added in API level 1

Returns the closest double approximation of the natural logarithm of the sum of the argument and 1. If the argument is very close to 0, it is much more accurate to use `log1p(d)` than `log(1.0+d)` (due to numerical cancellation). The returned result is within 1 ulp (unit in the last place) of the real result and is semi-monotonic.

Special cases:

- `log1p(+0.0) = +0.0`
- `log1p(-0.0) = -0.0`
- `log1p((anything < 1)) = NaN`
- `log1p(-1.0) = -infinity`
- `log1p(+infinity) = +infinity`
- `log1p(-infinity) = NaN`
- `log1p(NaN) = NaN`

Parameters

d the value to compute the `ln(1+d)` of.

Returns

the natural logarithm of the sum of the argument and `1`.

public static long **max** (long l1, long l2)

Added in API level 1

Returns the most positive (closest to positive infinity) of the two arguments.

public static int **max** (int i1, int i2)

Added in API level 1

Returns the most positive (closest to positive infinity) of the two arguments.

public static double **max** (double d1, double d2)

Added in API level 1

Returns the most positive (closest to positive infinity) of the two arguments.

Special cases:

- `max(NaN, (anything)) = NaN`
- `max((anything), NaN) = NaN`
- `max(+0.0, -0.0) = +0.0`
- `max(-0.0, +0.0) = +0.0`

public static float **max** (float f1, float f2)

Added in API level 1

Returns the most positive (closest to positive infinity) of the two arguments.

Special cases:

- `max(NaN, (anything)) = NaN`
- `max((anything), NaN) = NaN`
- `max(+0.0, -0.0) = +0.0`
- `max(-0.0, +0.0) = +0.0`

public static int **min** (int i1, int i2)

Added in API level 1

Returns the most negative (closest to negative infinity) of the two arguments.

public static long **min** (long l1, long l2)

Added in API level 1

Returns the most negative (closest to negative infinity) of the two arguments.

public static double **min** (double d1, double d2)

Added in API level 1

Returns the most negative (closest to negative infinity) of the two arguments.

Special cases:

- `min(NaN, (anything)) = NaN`
- `min((anything), NaN) = NaN`
- `min(+0.0, -0.0) = -0.0`
- `min(-0.0, +0.0) = -0.0`

public static float **min** (float f1, float f2)

Added in API level 1

Returns the most negative (closest to negative infinity) of the two arguments.

Special cases:

- `min(NaN, (anything)) = NaN`
- `min((anything), NaN) = NaN`
- `min(+0.0, -0.0) = -0.0`
- `min(-0.0, +0.0) = -0.0`

public static float **nextAfter** (float start, double direction)

Added in API level 9

Returns the next float after `start` in the given `direction`.

public static double **nextAfter** (double start, double direction)

Added in API level 9

Returns the next double after `start` in the given `direction`.

public static double **nextUp** (double d)

Added in API level 9

Returns the next double larger than `d`.

public static float **nextUp** (float f)

Added in API level 9

Returns the next float larger than `f`.

public static double **pow** (double x, double y) Added in API level 1

Returns the closest double approximation of the result of raising x to the power of y.

Special cases:

- `pow((anything), +0.0) = 1.0`
- `pow((anything), -0.0) = 1.0`
- `pow(x, 1.0) = x`
- `pow((anything), NaN) = NaN`
- `pow(NaN, (anything except 0)) = NaN`
- `pow(+/-(|x| > 1), +infinity) = +infinity`
- `pow(+/-(|x| > 1), -infinity) = +0.0`
- `pow(+/-(|x| < 1), +infinity) = +0.0`
- `pow(+/-(|x| < 1), -infinity) = +infinity`
- `pow(+/-1.0 , +infinity) = NaN`
- `pow(+/-1.0 , -infinity) = NaN`
- `pow(+0.0, (+anything except 0, NaN)) = +0.0`
- `pow(-0.0, (+anything except 0, NaN, odd integer)) = +0.0`
- `pow(+0.0, (-anything except 0, NaN)) = +infinity`
- `pow(-0.0, (-anything except 0, NAN, odd integer)) = +infinity`
- `pow(-0.0, (odd integer)) = -pow( +0 , (odd integer) )`
- `pow(+infinity, (+anything except 0, NaN)) = +infinity`
- `pow(+infinity, (-anything except 0, NaN)) = +0.0`
- `pow(-infinity, (anything)) = -pow(0, (-anything))`
- `pow((-anything), (integer)) = pow(-1, (integer))*pow(+anything,integer)`
- `pow((-anything except 0 and inf), (non-integer)) = NAN`

Parameters

x the base of the operation.

y the exponent of the operation.

Returns

x to the power of y.

public static synchronized double **random** ()

Added in API level 1

Returns a pseudo-random double n, where `n >= 0.0 && n < 1.0`. This method reuses a single instance of `Random (/reference/java/util/Random.html)`. This method is thread-safe because access to the `Random` is synchronized, but this harms scalability. Applications may find a performance benefit from allocating a `Random` for each of their threads.

Returns

a pseudo-random number.

public static double **rint** (double d)

Added in API level 1

Returns the double conversion of the result of rounding the argument to an integer. Tie breaks are rounded towards even.

Special cases:

- `rint(+0.0) = +0.0`
- `rint(-0.0) = -0.0`
- `rint(+infinity) = +infinity`
- `rint(-infinity) = -infinity`
- `rint(NaN) = NaN`

Parameters

d the value to be rounded.

Returns

the closest integer to the argument (as a double).

public static long **round** (double d)

Added in API level 1

Returns the result of rounding the argument to an integer. The result is equivalent to `(long) Math.floor(d+0.5)`.

Special cases:

- `round(+0.0) = +0.0`
- `round(-0.0) = +0.0`
- `round((anything > Long.MAX_VALUE) = Long.MAX_VALUE`
- `round((anything < Long.MIN_VALUE) =`

Long.MIN_VALUE

- round(+infinity) = Long.MAX_VALUE
- round(-infinity) = Long.MIN_VALUE
- round(NaN) = +0.0

Parameters

d the value to be rounded.

Returns

the closest integer to the argument.

public static int **round** (float f)

Added in API level 1

Returns the result of rounding the argument to an integer. The result is equivalent to `(int) Math.floor(f+0.5)`.

Special cases:

- round(+0.0) = +0.0
- round(-0.0) = +0.0
- round((anything > Integer.MAX_VALUE) = Integer.MAX_VALUE
- round((anything < Integer.MIN_VALUE) = Integer.MIN_VALUE
- round(+infinity) = Integer.MAX_VALUE
- round(-infinity) = Integer.MIN_VALUE
- round(NaN) = +0.0

Parameters

f the value to be rounded.

Returns

the closest integer to the argument.

public static double **scalb** (double d, int scaleFactor)

Added in API level 9

Returns d * 2^`scaleFactor`. The result may be rounded.

public static float **scalb** (float d, int scaleFactor)

Added in API level 9

Returns d * 2^`scaleFactor`. The result may be rounded.

public static double **signum** (double d)

Added in API level 1

Returns the signum function of the argument. If the argument is less than zero, it returns -1.0. If the argument is greater than zero, 1.0 is returned. If the argument is either positive or negative zero, the argument is returned as result.

Special cases:

- `signum(+0.0) = +0.0`
- `signum(-0.0) = -0.0`
- `signum(+infinity) = +1.0`
- `signum(-infinity) = -1.0`
- `signum(NaN) = NaN`

Parameters

d the value whose signum has to be computed.

Returns

the value of the signum function.

public static float **signum** (float f)

Added in API level 1

Returns the signum function of the argument. If the argument is less than zero, it returns -1.0. If the argument is greater than zero, 1.0 is returned. If the argument is either positive or negative zero, the argument is returned as result.

Special cases:

- `signum(+0.0) = +0.0`
- `signum(-0.0) = -0.0`
- `signum(+infinity) = +1.0`
- `signum(-infinity) = -1.0`
- `signum(NaN) = NaN`

Parameters

f the value whose signum has to be computed.

Returns

the value of the signum function.

public static double **sin** (double d)

Added in API level 1

Returns the closest double approximation of the sine of the argument. The returned result is within 1 ulp (unit in the last place) of the real result.

Special cases:

- `sin(+0.0) = +0.0`
- `sin(-0.0) = -0.0`
- `sin(+infinity) = NaN`
- `sin(-infinity) = NaN`
- `sin(NaN) = NaN`

Parameters

d the angle whose sin has to be computed, in radians.

Returns

the sine of the argument.

public static double **sinh** (double d)

Added in API level 1

Returns the closest double approximation of the hyperbolic sine of the argument. The returned result is within 2.5 ulps (units in the last place) of the real result.

Special cases:

- `sinh(+0.0) = +0.0`
- `sinh(-0.0) = -0.0`
- `sinh(+infinity) = +infinity`
- `sinh(-infinity) = -infinity`
- `sinh(NaN) = NaN`

Parameters

d the value whose hyperbolic sine has to be computed.

Returns

the hyperbolic sine of the argument.

public static double **sqrt** (double d)

Added in API level 1

Returns the closest double approximation of the square root of the argument.

Special cases:

- `sqrt(+0.0) = +0.0`
- `sqrt(-0.0) = -0.0`
- `sqrt( (anything < 0) ) = NaN`
- `sqrt(+infinity) = +infinity`
- `sqrt(NaN) = NaN`

public static double **tan** (double d) Added in API level 1

Returns the closest double approximation of the tangent of the argument. The returned result is within 1 ulp (unit in the last place) of the real result.

Special cases:

- `tan(+0.0) = +0.0`
- `tan(-0.0) = -0.0`
- `tan(+infinity) = NaN`
- `tan(-infinity) = NaN`
- `tan(NaN) = NaN`

Parameters

d the angle whose tangent has to be computed, in radians.

Returns

the tangent of the argument.

public static double **tanh** (double d) Added in API level 1

Returns the closest double approximation of the hyperbolic tangent of the argument. The absolute value is always less than 1. The returned result is within 2.5 ulps (units in the last place) of the real result. If the real result is within 0.5ulp of 1 or -1, it should return exactly +1 or -1.

Special cases:

- `tanh(+0.0) = +0.0`
- `tanh(-0.0) = -0.0`
- `tanh(+infinity) = +1.0`
- `tanh(-infinity) = -1.0`
- `tanh(NaN) = NaN`

Parameters

d the value whose hyperbolic tangent has to be computed.

Returns

the hyperbolic tangent of the argument.

public static double **toDegrees** (double angrad) Added in API level 1

Returns the measure in degrees of the supplied radian angle. The result is `angrad * 180 / pi`.

Special cases:

- `toDegrees(+0.0) = +0.0`
- `toDegrees(-0.0) = -0.0`
- `toDegrees(+infinity) = +infinity`
- `toDegrees(-infinity) = -infinity`
- `toDegrees(NaN) = NaN`

Parameters

angrad an angle in radians.

Returns

the degree measure of the angle.

public static double **toRadians** (double angdeg) Added in API level 1

Returns the measure in radians of the supplied degree angle. The result is `angdeg / 180 * pi`.

Special cases:

- `toRadians(+0.0) = +0.0`
- `toRadians(-0.0) = -0.0`
- `toRadians(+infinity) = +infinity`
- `toRadians(-infinity) = -infinity`
- `toRadians(NaN) = NaN`

Parameters

angdeg an angle in degrees.

Returns

the radian measure of the angle.

public static float **ulp** (float f) Added in API level 1

Returns the argument's ulp (unit in the last place). The size of a ulp of a float value is the positive distance between this value and the float value next larger in magnitude. For non-NaN `x`, `ulp(-x) == ulp(x)`.

Special cases:

- `ulp(+0.0) = Float.MIN_VALUE`
- `ulp(-0.0) = Float.MIN_VALUE`
- `ulp(+infinity) = infinity`
- `ulp(-infinity) = infinity`

- `ulp(NaN) = NaN`

Parameters

f the floating-point value to compute ulp of.

Returns

the size of a ulp of the argument.

public static double **ulp** (double d)

Added in API level 1

Returns the argument's ulp (unit in the last place). The size of a ulp of a double value is the positive distance between this value and the double value next larger in magnitude. For non-NaN x, `ulp(-x) == ulp(x)`.

Special cases:

- `ulp(+0.0) = Double.MIN_VALUE`
- `ulp(-0.0) = Double.MIN_VALUE`
- `ulp(+infinity) = infinity`
- `ulp(-infinity) = infinity`
- `ulp(NaN) = NaN`

Parameters

d the floating-point value to compute ulp of.

Returns

the size of a ulp of the argument.