Math Class

Contains methods for mathematical operations.

Namespace

System

Math Fields

The following are fields for `Math`.

- **E**
  Returns the mathematical constant *e*, which is the base of natural logarithms.
- **PI**
  Returns the mathematical constant *pi*, which is the ratio of the circumference of a circle to its diameter.

E

Returns the mathematical constant *e*, which is the base of natural logarithms.

Signature

```
public static final Double E
```

Property Value

Type: Double

PI

Returns the mathematical constant *pi*, which is the ratio of the circumference of a circle to its diameter.

Signature

```
public static final Double PI
```

Property Value

Type: Double

Math Methods

The following are methods for `Math`. All methods are static.

- **abs(decimalValue)**
  Returns the absolute value of the specified Decimal.
- **abs(doubleValue)**
  Returns the absolute value of the specified Double.

- **acos(decimalAngle)**
  Returns the arc cosine of an angle, in the range of 0.0 through *pi*.
- **acos(doubleAngle)**
  Returns the arc cosine of an angle, in the range of 0.0 through *pi*.
- **asin(decimalAngle)**
  Returns the arc sine of an angle, in the range of *-pi*/2 through *pi*/2.
- **asin(doubleAngle)**
  Returns the arc sine of an angle, in the range of *-pi*/2 through *pi*/2.
- **atan(decimalAngle)**
  Returns the arc tangent of an angle, in the range of *-pi*/2 through *pi*/2.
- **atan(doubleAngle)**
  Returns the arc tangent of an angle, in the range of *-pi*/2 through *pi*/2.
- **atan2(xCoordinate, yCoordinate)**
  Converts rectangular coordinates (*xCoordinate* and *yCoordinate*) to polar (*r* and *theta*). This method computes the phase *theta* by computing an arc tangent of *xCoordinate*/*yCoordinate* in the range of *-pi* to *pi*.
- **atan2(xCoordinate, yCoordinate)**
  Converts rectangular coordinates (*xCoordinate* and *yCoordinate*) to polar (*r* and *theta*). This method computes the phase *theta* by computing an arc tangent of *xCoordinate*/*yCoordinate* in the range of *-pi* to *pi*.
- **cbrt(decimalValue)**
  Returns the cube root of the specified Decimal. The cube root of a negative value is the negative of the cube root of that value's magnitude.
- **cbrt(doubleValue)**
  Returns the cube root of the specified Double. The cube root of a negative value is the negative of the cube root of that value's magnitude.
- **ceil(decimalValue)**
  Returns the smallest (closest to negative infinity) Decimal that is not less than the argument and is equal to a mathematical integer.
- **ceil(doubleValue)**
  Returns the smallest (closest to negative infinity) Double that is not less than the argument and is equal to a mathematical integer.
- **cos(decimalAngle)**
  Returns the trigonometric cosine of the angle specified by *decimalAngle*.
- **cos(doubleAngle)**
  Returns the trigonometric cosine of the angle specified by *doubleAngle*.
- **cosh(decimalAngle)**
  Returns the hyperbolic cosine of *decimalAngle*. The hyperbolic cosine of *d* is defined to be $(e^x + e^{-x})/2$ where *e* is Euler's number.
- **cosh(doubleAngle)**
  Returns the hyperbolic cosine of *doubleAngle*. The hyperbolic cosine of *d* is defined to be $(e^x + e^{-x})/2$ where *e* is Euler's number.
- **exp(exponentDecimal)**
  Returns Euler's number *e* raised to the power of the specified Decimal.
- **exp(exponentDouble)**
  Returns Euler's number *e* raised to the power of the specified Double.
- **floor(decimalValue)**
  Returns the largest (closest to positive infinity) Decimal that is not greater than the argument and is equal to a mathematical integer.
- **floor(doubleValue)**
  Returns the largest (closest to positive infinity) Double that is not greater than the argument and is equal to a mathematical integer.
- **log(decimalValue)**
  Returns the natural logarithm (base *e*) of the specified Decimal.

- **log10(doubleValue)**
  Returns the logarithm (base *10*) of the specified Double.
- **max(decimalValue1, decimalValue2)**
  Returns the larger of the two specified Decimals.
- **max(doubleValue1, doubleValue2)**
  Returns the larger of the two specified Doubles.
- **max(integerValue1, integerValue2)**
  Returns the larger of the two specified Integers.
- **max(longValue1, longValue2)**
  Returns the larger of the two specified Longs.
- **min(decimalValue1, decimalValue2)**
  Returns the smaller of the two specified Decimals.
- **min(doubleValue1, doubleValue2)**
  Returns the smaller of the two specified Doubles.
- **min(integerValue1, integerValue2)**
  Returns the smaller of the two specified Integers.
- **min(longValue1, longValue2)**
  Returns the smaller of the two specified Longs.
- **mod(integerValue1, integerValue2)**
  Returns the remainder of *integerValue1* divided by *integerValue2*.
- **mod(longValue1, longValue2)**
  Returns the remainder of *longValue1* divided by *longValue2*.
- **pow(doubleValue, exponent)**
  Returns the value of the first Double raised to the power of *exponent*.
- **random()**
  Returns a positive Double that is greater than or equal to 0.0 and less than 1.0.
- **rint(decimalValue)**
  Returns the value that is closest in value to *decimalValue* and is equal to a mathematical integer.
- **rint(doubleValue)**
  Returns the value that is closest in value to *doubleValue* and is equal to a mathematical integer.
- **round(doubleValue)**
  Do not use. This method is deprecated as of the Winter '08 release. Instead, use `Math.roundToLong`. Returns the closest Integer to the specified Double. If the result is less than -2,147,483,648 or greater than 2,147,483,647, Apex generates an error.
- **round(decimalValue)**
  Returns the rounded approximation of this Decimal. The number is rounded to zero decimal places using half-even rounding mode, that is, it rounds towards the “nearest neighbor” unless both neighbors are equidistant, in which case, this mode rounds towards the even neighbor. If the result is less than -2,147,483,648 or greater than 2,147,483,647, Apex generates an error.
- **roundToLong(decimalValue)**
  Returns the rounded approximation of this Decimal. The number is rounded to zero decimal places using half-even rounding mode, that is, it rounds towards the “nearest neighbor” unless both neighbors are equidistant, in which case, this mode rounds towards the even neighbor.
- **roundToLong(doubleValue)**
  Returns the closest Long to the specified Double.
- **signum(decimalValue)**
  Returns the signum function of the specified Decimal, which is 0 if *decimalValue* is 0, 1.0 if *decimalValue* is greater than 0, -1.0 if *decimalValue* is less than 0.
- **signum(doubleValue)**
  Returns the signum function of the specified Double, which is 0 if *doubleValue* is 0, 1.0 if

Returns the trigonometric sine of the angle specified by *doubleAngle*.

- **sinh(decimalAngle)**
  Returns the hyperbolic sine of *decimalAngle*. The hyperbolic sine of *decimalAngle* is defined to be $(e^x - e^{-x})/2$ where *e* is Euler's number.
- **sinh(doubleAngle)**
  Returns the hyperbolic sine of *doubleAngle*. The hyperbolic sine of *doubleAngle* is defined to be $(e^x - e^{-x})/2$ where *e* is Euler's number.
- **sqrt(decimalValue)**
  Returns the correctly rounded positive square root of *decimalValue*.
- **sqrt(doubleValue)**
  Returns the correctly rounded positive square root of *doubleValue*.
- **tan(decimalAngle)**
  Returns the trigonometric tangent of the angle specified by *decimalAngle*.
- **tan(doubleAngle)**
  Returns the trigonometric tangent of the angle specified by *doubleAngle*.
- **tanh(decimalAngle)**
  Returns the hyperbolic tangent of *decimalAngle*. The hyperbolic tangent of *decimalAngle* is defined to be $(e^x - e^{-x})/(e^x + e^{-x})$ where *e* is Euler's number. In other words, it is equivalent to `sinh(x)/cosinh(x)`. The absolute value of the exact `tanh` is always less than 1.
- **tanh(doubleAngle)**
  Returns the hyperbolic tangent of *doubleAngle*. The hyperbolic tangent of *doubleAngle* is defined to be $(e^x - e^{-x})/(e^x + e^{-x})$ where *e* is Euler's number. In other words, it is equivalent to `sinh(x)/cosinh(x)`. The absolute value of the exact `tanh` is always less than 1.

abs(decimalValue)

Returns the absolute value of the specified Decimal.

Signature

```
public static Decimal abs(Decimal decimalValue)
```

Parameters

decimalValue

Type: Decimal

Return Value

Type: Decimal

abs(doubleValue)

Returns the absolute value of the specified Double.

Signature

```
public static Double abs(Double doubleValue)
```

Parameters

doubleValue

Type: Double

Return Value

Type: Double

```
public static Integer abs(Integer integerValue)
```

Parameters

integerValue

Type: Integer

Return Value

Type: Integer

Example

```
Integer i = -42;
Integer i2 = math.abs(i);
system.assertEquals(i2, 42);
```

abs(longValue)

Returns the absolute value of the specified Long.

Signature

```
public static Long abs(Long longValue)
```

Parameters

longValue

Type: Long

Return Value

Type: Long

acos(decimalAngle)

Returns the arc cosine of an angle, in the range of 0.0 through *pi*.

Signature

```
public static Decimal acos(Decimal decimalAngle)
```

Parameters

decimalAngle

Type: Decimal

Return Value

Type: Decimal

acos(doubleAngle)

Returns the arc cosine of an angle, in the range of 0.0 through *pi*.

Signature

```
public static Double acos(Double doubleAngle)
```

Return Value

Type: Double

asin(decimalAngle)

Returns the arc sine of an angle, in the range of -*pi*/2 through *pi*/2.

Signature

```
public static Decimal asin(Decimal decimalAngle)
```

Parameters

decimalAngle

Type: Decimal

Return Value

Type: Decimal

asin(doubleAngle)

Returns the arc sine of an angle, in the range of -*pi*/2 through *pi*/2.

Signature

```
public static Double asin(Double doubleAngle)
```

Parameters

doubleAngle

Type: Double

Return Value

Type: Double

atan(decimalAngle)

Returns the arc tangent of an angle, in the range of -*pi*/2 through *pi*/2.

Signature

```
public static Decimal atan(Decimal decimalAngle)
```

Parameters

decimalAngle

Type: Decimal

Return Value

Type: Decimal

atan(doubleAngle)

Returns the arc tangent of an angle, in the range of -*pi*/2 through *pi*/2.

Signature

```
public static Double atan(Double doubleAngle)
```

Return Value

Type: Double

atan2(xCoordinate, yCoordinate)

Converts rectangular coordinates (*xCoordinate* and *yCoordinate*) to polar (*r* and *theta*). This method computes the phase *theta* by computing an arc tangent of *xCoordinate*/*yCoordinate* in the range of *-pi* to *pi*.

Signature

```
public static Decimal atan2(Decimal xCoordinate, Decimal yCoordinate)
```

Parameters

xCoordinate

Type: Decimal

yCoordinate

Type: Decimal

Return Value

Type: Decimal

atan2(xCoordinate, yCoordinate)

Converts rectangular coordinates (*xCoordinate* and *yCoordinate*) to polar (*r* and *theta*). This method computes the phase *theta* by computing an arc tangent of *xCoordinate*/*yCoordinate* in the range of *-pi* to *pi*.

Signature

```
public static Double atan2(Double xCoordinate, Double yCoordinate)
```

Parameters

xCoordinate

Type: Double

yCoordinate

Type: Double

Return Value

Type: Double

cbrt(decimalValue)

Returns the cube root of the specified Decimal. The cube root of a negative value is the negative of the cube root of that value's magnitude.

Signature

```
public static Decimal cbrt(Decimal decimalValue)
```

Parameters

decimalValue

Type: Decimal

cbrt(doubleValue)

Returns the cube root of the specified Double. The cube root of a negative value is the negative of the cube root of that value's magnitude.

Signature

```
public static Double cbrt(Double doubleValue)
```

Parameters

doubleValue

Type: Double

Return Value

Type: Double

ceil(decimalValue)

Returns the smallest (closest to negative infinity) Decimal that is not less than the argument and is equal to a mathematical integer.

Signature

```
public static Decimal ceil(Decimal decimalValue)
```

Parameters

decimalValue

Type: Decimal

Return Value

Type: Decimal

ceil(doubleValue)

Returns the smallest (closest to negative infinity) Double that is not less than the argument and is equal to a mathematical integer.

Signature

```
public static Double ceil(Double doubleValue)
```

Parameters

doubleValue

Type: Double

Return Value

Type: Double

cos(decimalAngle)

Returns the trigonometric cosine of the angle specified by *decimalAngle*.

Signature

```
public static Decimal cos(Decimal decimalAngle)
```

Parameters

Type: Decimal

cos(doubleAngle)

Returns the trigonometric cosine of the angle specified by *doubleAngle*.

Signature

```
public static Double cos(Double doubleAngle)
```

Parameters

doubleAngle

Type: Double

Return Value

Type: Double

cosh(decimalAngle)

Returns the hyperbolic cosine of *decimalAngle*. The hyperbolic cosine of *d* is defined to be $(e^x + e^{-x})/2$ where *e* is Euler's number.

Signature

```
public static Decimal cosh(Decimal decimalAngle)
```

Parameters

decimalAngle

Type: Decimal

Return Value

Type: Decimal

cosh(doubleAngle)

Returns the hyperbolic cosine of *doubleAngle*. The hyperbolic cosine of *d* is defined to be $(e^x + e^{-x})/2$ where *e* is Euler's number.

Signature

```
public static Double cosh(Double doubleAngle)
```

Parameters

doubleAngle

Type: Double

Return Value

Type: Double

exp(exponentDecimal)

Returns Euler's number *e* raised to the power of the specified Decimal.

Signature

Type: Decimal

Return Value

Type: Decimal

exp(exponentDouble)

Returns Euler's number *e* raised to the power of the specified Double.

Signature

```
public static Double exp(Double exponentDouble)
```

Parameters

exponentDouble

Type: Double

Return Value

Type: Double

floor(decimalValue)

Returns the largest (closest to positive infinity) Decimal that is not greater than the argument and is equal to a mathematical integer.

Signature

```
public static Decimal floor(Decimal decimalValue)
```

Parameters

decimalValue

Type: Decimal

Return Value

Type: Decimal

floor(doubleValue)

Returns the largest (closest to positive infinity) Double that is not greater than the argument and is equal to a mathematical integer.

Signature

```
public static Double floor(Double doubleValue)
```

Parameters

doubleValue

Type: Double

Return Value

Type: Double

log(decimalValue)

Returns the natural logarithm (base *e*) of the specified Decimal.

decimalValue

Type: Decimal

Return Value

Type: Decimal

log(doubleValue)

Returns the natural logarithm (base *e*) of the specified Double.

Signature

```
public static Double log(Double doubleValue)
```

Parameters

doubleValue

Type: Double

Return Value

Type: Double

log10(decimalValue)

Returns the logarithm (base *10*) of the specified Decimal.

Signature

```
public static Decimal log10(Decimal decimalValue)
```

Parameters

decimalValue

Type: Decimal

Return Value

Type: Decimal

log10(doubleValue)

Returns the logarithm (base *10*) of the specified Double.

Signature

```
public static Double log10(Double doubleValue)
```

Parameters

doubleValue

Type: Double

Return Value

Type: Double

max(decimalValue1, decimalValue2)

Returns the larger of the two specified Decimals.

decimalValue1

Type: Decimal

decimalValue2

Type: Decimal

Return Value

Type: Decimal

Example

```
Decimal larger = math.max(12.3, 156.6);
system.assertEquals(larger, 156.6);
```

max(doubleValue1, doubleValue2)

Returns the larger of the two specified Doubles.

Signature

```
public static Double max(Double doubleValue1, Double doubleValue2)
```

Parameters

doubleValue1

Type: Double

doubleValue2

Type: Double

Return Value

Type: Double

max(integerValue1, integerValue2)

Returns the larger of the two specified Integers.

Signature

```
public static Integer max(Integer integerValue1, Integer integerValue2)
```

Parameters

integerValue1

Type: Integer

integerValue2

Type: Integer

Return Value

Type: Integer

max(longValue1, longValue2)

Returns the larger of the two specified Longs.

longValue1

Type: Long

longValue2

Type: Long

Return Value

Type: Long

min(decimalValue1, decimalValue2)

Returns the smaller of the two specified Decimals.

Signature

```
public static Decimal min(Decimal decimalValue1, Decimal decimalValue2)
```

Parameters

decimalValue1

Type: Decimal

decimalValue2

Type: Decimal

Return Value

Type: Decimal

Example

```
Decimal smaller = math.min(12.3, 156.6);
system.assertEquals(smaller, 12.3);
```

min(doubleValue1, doubleValue2)

Returns the smaller of the two specified Doubles.

Signature

```
public static Double min(Double doubleValue1, Double doubleValue2)
```

Parameters

doubleValue1

Type: Double

doubleValue2

Type: Double

Return Value

Type: Double

min(integerValue1, integerValue2)

Returns the smaller of the two specified Integers.

integerValue1

Type: Integer

integerValue2

Type: Integer

Return Value

Type: Integer

min(longValue1, longValue2)

Returns the smaller of the two specified Longs.

Signature

```
public static Long min(Long longValue1, Long longValue2)
```

Parameters

longValue1

Type: Long

longValue2

Type: Long

Return Value

Type: Long

mod(integerValue1, integerValue2)

Returns the remainder of *integerValue1* divided by *integerValue2*.

Signature

```
public static Integer mod(Integer integerValue1, Integer integerValue2)
```

Parameters

integerValue1

Type: Integer

integerValue2

Type: Integer

Return Value

Type: Integer

Example

```
Integer remainder = math.mod(12, 2);
system.assertEquals(remainder, 0);

Integer remainder2 = math.mod(8, 3);
system.assertEquals(remainder2, 2);
```

```
public static Long mod(Long longValue1, Long longValue2)
```

Parameters

longValue1

Type: Long

longValue2

Type: Long

Return Value

Type: Long

pow(doubleValue, exponent)

Returns the value of the first Double raised to the power of *exponent*.

Signature

```
public static Double pow(Double doubleValue, Double exponent)
```

Parameters

doubleValue

Type: Double

exponent

Type: Double

Return Value

Type: Double

random()

Returns a positive Double that is greater than or equal to 0.0 and less than 1.0.

Signature

```
public static Double random()
```

Return Value

Type: Double

rint(decimalValue)

Returns the value that is closest in value to *decimalValue* and is equal to a mathematical integer.

Signature

```
public static Decimal rint(Decimal decimalValue)
```

Parameters

decimalValue

Type: Decimal

Return Value

Returns the value that is closest in value to *doubleValue* and is equal to a mathematical integer.

Signature

```
public static Double rint(Double doubleValue)
```

Parameters

doubleValue

Type: Double

Return Value

Type: Double

round(doubleValue)

Do not use. This method is deprecated as of the Winter '08 release. Instead, use `Math.roundToLong`. Returns the closest Integer to the specified Double. If the result is less than -2,147,483,648 or greater than 2,147,483,647, Apex generates an error.

Signature

```
public static Integer round(Double doubleValue)
```

Parameters

doubleValue

Type: Double

Return Value

Type: Integer

round(decimalValue)

Returns the rounded approximation of this Decimal. The number is rounded to zero decimal places using half-even rounding mode, that is, it rounds towards the “nearest neighbor” unless both neighbors are equidistant, in which case, this mode rounds towards the even neighbor. If the result is less than -2,147,483,648 or greater than 2,147,483,647, Apex generates an error.

Signature

```
public static Integer round(Decimal decimalValue)
```

Parameters

decimalValue

Type: Decimal

Return Value

Type: Integer

Usage

Note that this rounding mode statistically minimizes cumulative error when applied repeatedly over a sequence of calculations.

Example

```
Integer i2 = Math.round(d2);
System.assertEquals(6, i2);
```

roundToLong(decimalValue)

Returns the rounded approximation of this Decimal. The number is rounded to zero decimal places using half-even rounding mode, that is, it rounds towards the “nearest neighbor” unless both neighbors are equidistant, in which case, this mode rounds towards the even neighbor.

Signature

```
public static Long roundToLong(Decimal decimalValue)
```

Parameters

decimalValue

Type: Decimal

Return Value

Type: Long

Usage

Note that this rounding mode statistically minimizes cumulative error when applied repeatedly over a sequence of calculations.

Example

```
Decimal d1 = 4.5;
Long i1 = Math.roundToLong(d1);
System.assertEquals(4, i1);

Decimal d2 = 5.5;
Long i2 = Math.roundToLong(d2);
System.assertEquals(6, i2);
```

roundToLong(doubleValue)

Returns the closest Long to the specified Double.

Signature

```
public static Long roundToLong(Double doubleValue)
```

Parameters

doubleValue

Type: Double

Return Value

Type: Long

signum(decimalValue)

Returns the signum function of the specified Decimal, which is 0 if *decimalValue* is 0, 1.0 if *decimalValue* is greater than 0, -1.0 if *decimalValue* is less than 0.

Signature

Type: Decimal

Return Value

Type: Decimal

signum(doubleValue)

Returns the signum function of the specified Double, which is 0 if *doubleValue* is 0, 1.0 if *doubleValue* is greater than 0, -1.0 if *doubleValue* is less than 0.

Signature

```
public static Double signum(Double doubleValue)
```

Parameters

doubleValue

Type: Double

Return Value

Type: Double

sin(decimalAngle)

Returns the trigonometric sine of the angle specified by *decimalAngle*.

Signature

```
public static Decimal sin(Decimal decimalAngle)
```

Parameters

decimalAngle

Type: Decimal

Return Value

Type: Decimal

sin(doubleAngle)

Returns the trigonometric sine of the angle specified by *doubleAngle*.

Signature

```
public static Double sin(Double doubleAngle)
```

Parameters

doubleAngle

Type: Double

Return Value

Type: Double

sinh(decimalAngle)

Returns the hyperbolic sine of *decimalAngle*. The hyperbolic sine of *decimalAngle* is defined to be $(e^x - e^{-x})/2$ where e is Euler's number.

decimalAngle

Type: Decimal

Return Value

Type: Decimal

sinh(doubleAngle)

Returns the hyperbolic sine of *doubleAngle*. The hyperbolic sine of *doubleAngle* is defined to be (e^x - e^{-x})/2 where *e* is Euler's number.

Signature

```
public static Double sinh(Double doubleAngle)
```

Parameters

doubleAngle

Type: Double

Return Value

Type: Double

sqrt(decimalValue)

Returns the correctly rounded positive square root of *decimalValue*.

Signature

```
public static Decimal sqrt(Decimal decimalValue)
```

Parameters

decimalValue

Type: Decimal

Return Value

Type: Decimal

sqrt(doubleValue)

Returns the correctly rounded positive square root of *doubleValue*.

Signature

```
public static Double sqrt(Double doubleValue)
```

Parameters

doubleValue

Type: Double

Return Value

Type: Double

tan(decimalAngle)

Parameters

decimalAngle

Type: Decimal

Return Value

Type: Decimal

tan(doubleAngle)

Returns the trigonometric tangent of the angle specified by *doubleAngle*.

Signature

```
public static Double tan(Double doubleAngle)
```

Parameters

doubleAngle

Type: Double

Return Value

Type: Double

tanh(decimalAngle)

Returns the hyperbolic tangent of *decimalAngle*. The hyperbolic tangent of *decimalAngle* is defined to be $(e^x - e^{-x})/(e^x + e^{-x})$ where *e* is Euler's number. In other words, it is equivalent to `sinh(x)/cosinh(x)`. The absolute value of the exact `tanh` is always less than 1.

Signature

```
public static Decimal tanh(Decimal decimalAngle)
```

Parameters

decimalAngle

Type: Decimal

Return Value

Type: Decimal

tanh(doubleAngle)

Returns the hyperbolic tangent of *doubleAngle*. The hyperbolic tangent of *doubleAngle* is defined to be $(e^x - e^{-x})/(e^x + e^{-x})$ where *e* is Euler's number. In other words, it is equivalent to `sinh(x)/cosinh(x)`. The absolute value of the exact `tanh` is always less than 1.

Signature

```
public static Double tanh(Double doubleAngle)
```

Parameters

doubleAngle

Type: Double

Return Value

Let us know so we can improve!