JavaScript Interval Library

(Interval Numbers)

Version 1.2

By Henrik Vestermark (hve@hvks.com)

Contents

Interval

Support for Interval number arithmetic in JavaScript

Constructor

new Interval(*lower*,*upper*,*type*) // Constructor
Interval(value) // Conversion

Arguments

lower The lower part of an Interval number
upper The optional upper part of an Interval number
type The optional interval type.

If the upper argument is omitted it is treated as the same as the lower argument
If there are no arguments, it is treated as an undefined Interval
If *Interval* is invoked as a conversion, the value parameter is converted to an *Interval number* and returned.
The interval type parameter indicates close, open or half open intervals.

Returns

Returns an Interval object initialize with the lower and upper value. If *Interval* is invoked as a conversion, the *value* parameter is converted to an *Interval number* and returned. If *value* is another Interval number then that is returned. If *value*

Internal format of an Interval Object

The interval is stored as an Object with the following fields:

Object.low Store the lower end of the interval.
Object.high Store the upper end of the interval.
Object.left Indicated if the left is closed (include the value of the lower endpoint, or Open exclude the value of lower endpoint.
Object.right Indicated if the left is closed (include the value of high, or Open exclude the value of high.

Closed interval is written in the notation within [] brackets. Open with (). If only one the side is open or closed, it is called a half open interval.

Notation		***Type Parameter***	***Explanation***

(a,b)	*a<x<b*	"()"	*An Open interval*
[a,b)	*a<=x<b*	"[)"	*Closed on left, Open on right also naned half open*
(a,b]	*a<x<=b*	"(]"	*Open on left, Closed on right also named half open*
[a,b]	*a<=x<=b*	"[]"	*A Closed interval*

The interval type when creating an interval is written as a two characters string. See above. If an Interval only contains a singleton value, by definition the (a,a), (a,a], [a,a) represent the empty set, where [a,a] represent the set a. You can convert an open set to the closed form by calling Interval.toClose() or the other way convert an interval to its open form by calling Interval.toOpen(). Note than an interval can also contains the *undefined* value or *NaN* for both the lower and the upper part of the interval.

The empty set

As previous mention the empty set can be represented by (a,a), (a,a], [a,a) or (a,b) where a>b. There is two other form of an empty set regardless if the set is open or half open. That is if the *lower* or *upper* range is either *undefined* or *NaN.* The Method isEmpty() can be used to test for empty sets.

Arithmetic Interval operations

There is the usual interval arithmetic's function like adding, subtracting, multiplying and dividing intervals. Notice all arithmetic or Interval Math functions always returned the result in a closed form.

Logical interval operations

There is also a number of logical operations like the union, intersection of intervals. Check for a subset, interior of intervals, if an interval precedes another interval or a number is within an interval range and the 6 comparison types: >,<,>=,<=,== or !=

Methods

abs()	Return the absolute value of Interval number
center()	Return the center (or midpoint) of the Interval number
in()	Test if a number is within the interval range. Return true or false.
isEmpty()	Test if the interval is empty or not. Return true or false
lower()	Return the lower part of the Interval number.
negate()	Return a new Negated Interval Object.
upper()	Return the upper part of the Interval number.
toNumber	Convert an interval number to a floating point using the midpoint of the interval number. Same as the center() method.

toExponential()	Converts a Interval number to a string using exponential notation with the specified number of digits after the Decimal place.
toFixed()	Converts a Interval number to a string that contains a specified number of digits after the decimal place.
toPrecision()	Convert a Interval number to a string using the specified number of precision digits. Uses exponential or fixed-point notation depending on the size of the number and the number of significant digits specified.
toString()	Convert an Interval number to a string using a specified radix(base)
valueOf()	The primitive lower number value of this Interval number object.
width()	Return the width of the interval number

Functions

abs()	Return the magnitude of an Interval number
add()	Return the addition of two Interval numbers
acos()	Return arc cosine of the Interval number
acosh()	Return the arc cosine hyperbolic of the Interval number
asin()	Return the arcsine of the Interval number
asinh()	Return the arc sine hyperbolic of the Interval number
atan()	Return the arc tangent of the Interval number
atan2()	Return the arc tangent of the quotient of its arguments
atanh()	Return the arctangent hyperbolic of the Interval number
cos()	Return cosine of the Interval number
cosh()	Return the cosine hyperbolic of the Interval number
div()	Return the division of two Interval numbers.
E	Return exp(1) as a constant interval.
equal()	Return the Boolean value (true, false) of the equality of two Interval numbers.
exp()	Return the Interval power of e.
greater()	Return the Boolean value (true,false) of the interval comparison
greaterequal()	Return the Boolean value (true,false) of the interval comparison
interior()	Return true if an interval is located within another interval, otherwise false
intersection()	Return the Intersection of two interval numbers. (and)
less()	Return the Boolean value (true,false) of the interval comparison
lessequal()	Return the Boolean value (true,false) of the interval comparison
LN2	Return Log(2) as a constant interval.
LN10	Return Log(10) as a constant interval.
log()	Return the Interval natural logarithm.
log10()	Return the Interval base 10 logarithm.
mul()	Return the product of two Interval numbers.
notequal()	Return the Boolean value (true, false) of the in-equality of two Interval numbers.
PI	Return π as a constant interval.
pow()	Return the Interval power of x^y.
preceeds()	Return true if an interval precede another interval, otherwise false

pred()	Return the predecessor number to a floating point number
sin()	Return the sine of the Interval number
sinh()	Return the sine hyperbolic of the Interval number
sub()	Return the difference between two Interval numbers.
subset()	Return true if an interval is a subset of another interval, otherwise false
sqrt()	Return the Interval square root.
succ()	Return the successor number to a floating point number
tan()	Return the tangent of the Interval number
tanh()	Return the tangent hyperbolic of the Interval number
toClose()	Convert an return a new closed interval object
toOpen()	Convert an return a new open interval object
union()	Return the union (or) of the two intervals

Constants

Interval.zero	Return a new Interval(0,0) object
Interval.one	Return a new Interval(1,1) object

Miscellaneous

parseInterval()	Parse an Interval float number string

API

Interval.abs()

Return the absolute value of the Interval number

Synopsis

Interval object.abs()

Returns

The absolute value of the Interval number is returned.

Example

```
var z = new Interval(3,-4);
z.abs()                          // result [3:4]
```

See Also

Interval.negate()

Interval.add()

Add two Interval numbers

Synopsis

Interval.add(a,b)

Arguments

a,b The Interval numbers to be added.

Returns

The result of the Interval addition.

Example

```
var z = new Interval(3,4);
var y=new  Interval(1,2);

Interval.add(z,y)                // result  [4:6]
```

See Also

Interval.div(), Interval.mul(), Interval.sub()

Interval.acos()

Return the arc cosine of the Interval number

Synopsis

Interval.acos(a)

Returns

Return the arc cosine of the Interval number. If a > 1 or a < -1 then it return Interval(*NaN*).

Example

```
var z = new Interval(0.5,0.6);
Interval.acos(z)                // result [0.9272952180016133: 1.0471975511965967]
Interval.acos(Interval(0.5));   // result [1.0471975511965967: 1.0471975511965992]
```

See Also

Interval.asin(), Interval.atan()

Interval.acosh()

Return the arc cosine hyperbolic of the Interval number

Synopsis

Interval.acosh(a)

Returns

Return the arc cosine hyperbolic of the Interval number.

Example

```
var z = new Interval(3,4);
Interval.acosh(z)               // result (1.7627471740390848: 2.0634370688955634]
Interval.acosh(Interval(0.5)); // result [0.9624236501192058: 0.9624236501192067]
```

See Also

Interval.asinh(), Interval.atanh()

Interval.asin()

Return the arcsine of the Interval number

Synopsis

Interval.asin(a)

Returns

Return the arc sine of the Interval number. If a > 1 or a < -1 then it return Interval(*NaN*).

Example

```
var z = new Interval(0.5,0.6);
Interval.asin(z)                // result [0.5235987755982974: 0.6435011087932853]
Interval.asin(Interval(0.5));   // result [0.5235987755982974: 0.5235987755982998]
```

See Also

Interval.acos(), Interval.atan()

Interval.asinh()

Return the arc sine hyperbolic of the Interval number

Synopsis

Interval.asinh(a)

Returns

Return the arc sine hyperbolic of the Interval number.

Example

```
var z = new Interval(3,4);
```

```
Interval.asinh(z)                // result [1.8184464592320593: 2.0947125472611066]
Interval.asinh(Interval(0.5));   // result [0.4812118250596029: 0.4812118250596051]
```

See Also

Interval.acosh(), Interval.atanh()

Interval.atan()

Return the arctangent of the Interval number

Synopsis

Interval.atan(a)

Returns

Return the arc tangent of the Interval number.

Example

```
var z = new Interval(3,4);
Interval.atan(z)                 // result [1.2490457723982522:1.3258176636680343]
Interval.atan(Interval(0.5));    // result [0.4636476090008055:0.4636476090008068]
```

See Also

Interval.acos(), Interval.asin(), Interval.atan2()

Interval.atan2()

Calculate atan2() of the Interval numbers

Synopsis

Interval.atan2(y,x)

Arguments

y	The Interval number of the Y coordinate of the point
x	The Interval number of the X coordinate of the point

Returns

The result of the Interval.atan2() which is an Interval number between –π and +π

Description

The Interval.atan2() function computes the arc tangent of the ratio y/x. The y argument can be considered the Y coordinate of a Interval, and the x argument can be considered the X coordinate of the Interval. Note the unusual order of the arguments to this function. The Y coordinate is passed before the X coordinate.

Example

```
var x=new Interval(1), y=new Interval(2);
Interval.atan2(y,x)              // result [1.1071487177940862:1.1071487177940942]
```

See Also

BigFloat.acos(), BigFloat.atan(), BigFloat.asin()

Interval.atanh()

Return the arc tangent hyperbolic of the Interval number

Synopsis

Interval.atanh(a)

Returns

Return the arc tanh hyperbolic of the Interval number.

Example

```
var z = new Interval(0.3,0.4);
Interval.atanh(z)              // result [0.30951960420311136: 0.42364893019360217]
Interval.atan(Interval(0.5));  // result [0.5493061443340536: 0.5493061443340561]
```

See Also

Interval.acosh(), Interval.asinh()

Interval.center()

Return the center or midpoint of the Interval number

Synopsis

Interval.center(a)

Returns

Return the center of the Interval number a.

Example

```
var z = new Interval(3,4);
Interval.center(z)              // result 3.5
```

See Also

Interval.width()

Interval.cos()

Return the cosine of the Interval number

Synopsis

Interval.cos(a)

Returns

Return the cosine of the Interval number.

Example

```
var z = new Interval(3,4);
Interval.cos(z)                 // result [-0.9899924966004484:-0.6536436208636089]
Interval.cos(Interval(0.5))     // result [0.8775825618903724: 0.8775825618903731]
```

See Also

Interval.sin(), Interval.tan()

Interval.cosh()

Return the cosine hyperbolic of the Interval number

Synopsis

Interval.cosh(a)

Returns

Return the cosine hyperbolic of the Interval number.

Example

```
var z = new Interval(3,4);
Interval.cosh(z)              // result [10.051926281038115: 27.3239685507563]
Interval.cos(Interval(0.5))   // result [1.1276259652063798:1.127625965206382]
```

See Also

Interval.sinh(), Interval.tanh()

Interval.div()

Divide two Interval numbers

Synopsis

Interval.div(a,b)

Arguments

a,b	The Interval numbers to be divided. Special calculation are made to prevent intermediate result to overflow.

Returns

The result of the Interval division a/b.

Example

```
var z = new Interval(3,4);
var y=new  Interval(1,2);

Interval.div(z,y)             // result [1.5:4]
```

See Also

Interval.add(), Interval.mul(), Interval.sub()

Interval.E

Return Interval exp (1)

Synopsis

Interval.E

Returns

The Interval constant E [2.718281828459045,2.7182818284590455]. It is more accurate to use Interval.E instead of the call Interval.exp(1)

Example

var z = Interval.E;

See Also

Interval.LN10, Interval.LN2

Interval.equal()

Compare two Interval numbers for equality

Synopsis

Interval.equal(a,b)

Arguments

a,b The Interval numbers to be compare for

Returns

The Boolean value of the equal comparison.

Example

```
var z = new Interval(3,4);
var y=new  Interval(1,2);

if( Interval.equal(z,y))…          // result false
if( Interval.equal(z,z))…          // result true
```

```
if( !Interval.equal(z,y))…          // result true ! to do “notequal” comparison
```

See Also

Interval.notequal(), Interval.lessequal(), Interval.less(), Interval.greater(), Interval.greaterequal()

Interval.exp()

Compute e^x for the Interval number

Synopsis

Interval.exp(x)

Arguments

x A Interval numbers to be used as the exponent

Returns

e^x, e raised to the power of the specified exponent x, where e is the base of the natural logarithm, with a value of approximately 2.71828.

Example

```
var z = new Interval(3,4);

Interval.exp(z)…          // result  [20.085536923187497:54.598150033144734]
Interval.exp(Interval(2)); // result  [7.389056098930634:7.389056098930683]
```

See Also

Interval.log(), Interval.log10(), Interval.pow()
‘

Interval.greater ()

Compare two Interval numbers for greater

Synopsis

Interval.greater(a,b)

Arguments

a,b The Interval numbers to be compare for

Returns

The Boolean value of the greater comparison.

Example

```
var z = new Interval(3,4);
var y=new  Interval(1,2);

if( Interval.greater(z,y))…          // result true
if( Interval.greater(y,z))…          // result false
if( Interval.greater(z,z))…          // result false
```

See Also

Interval.notequal(), Interval.equal(), Interval.lessequal(), Interval.less(), Interval.greaterequal()

Interval.greaterequal()

Compare two Interval numbers for greater or equal

Synopsis

Interval.greaterequal(a,b)

Arguments

a,b The Interval numbers to be compare for

Returns

The Boolean value of the greater equal comparison.

Example

```
var z = new Interval(3,4);
var y=new  Interval(1,2);
```

```
if( Interval.greaterequal(z,y))…        // result true
if( Interval.greaterequal(y,z))…        // result false
if( Interval.greaterequal(z,z))…        // result true
```

See Also

Interval.notequal(), Interval.equal(), Interval.less(), Interval.greater(), Interval.lessequal()

Interval.in()

Test if the argument is within the interval range and return the Boolean value.

Synopsis

Interval object.in(x)

Returns

Return true if x is within the Interval range otherwise false.

Example

```
var z = new Interval(3,4);
z.in(2);                    // return false
z.in(3);                    // returns true
z.in(3.5);                  // returns true
```

See Also

Interval.isEmpty()

Interval.interior()

Return true if a-interval is an interior of the b-interval, otherwise false

Synopsis

Interval.interior(a,b)

Arguments

a,b is the interval numbers.

Returns

The Boolean result if *a* is an interior of *b*. An interior is true if b.low < a.low and a.high < b.high otherwise false

Example

```
var z = new Interval(3,4);
var y=new  Interval(2,4);
var x=new Interval(3.5,3.9);

Interval.interior(z,y)          // result  false
Interval.interior(x,z)          // result  true
Interval.interior(z,z)          // result false
```

See Also

Interval.precedes()

Interval.intersection()

Create the common intervals of the two Interval numbers

Synopsis

Interval.intersection(a,b)

Arguments

a,b	The Interval numbers to create the intersection or and'ing together. It create and new interval that is the intersection of the two interval. If no intersection then the empty interval is returned.

Returns

The result of the Interval intersection.

Example

```
var z = new Interval(3,4);
var y=new  Interval(1,2);
var x=new Interval(2.5,3.5);

Interval.intersection(z,y)          // result  empty  interval []
Interval.intersection(x,z)          // result  [3:3.5]
Interval.intersection(z,z)          // result [3:4]
```

See Also

Interval.union()

Interval.isEmpty()

Test if the interval is empty and return the Boolean value.

Synopsis

Interval object.isEmpty()

Returns

Return true is the Interval is empty otherwise false.

Example

```
var z = new Interval(3,4);
z.isEmpty();                    // return false
z= new Interval();
z.isEmpty();                    // returns true
z=new Interval(3,"[)");
z.isEmpty();                    // returns true since [3) is an empty interval
```

See Also

Interval.in()

Interval.less ()

Compare two Interval numbers for less

Synopsis

Interval.less(a,b)

Arguments

a,b The Interval numbers to be compare for

Returns

The Boolean value of the less comparison.

Example

```
var z = new Interval(3,4);
var y=new  Interval(1,2);

if( Interval.less(z,y))…          // result false
if( Interval.less(y,z))…          // result true
if( Interval.less(z,z))…          // result false
```

See Also

Interval.notequal(), Interval.equal(), Interval.lessequal(), Interval.greater(), Interval.greaterequal()

Interval.lessequal()

Compare two Interval numbers for less or equal

Synopsis

Interval.lessequal(a,b)

Arguments

a,b The Interval numbers to be compare for

Returns

The Boolean value of the less equal comparison.

Example

```
var z = new Interval(3,4);
var y=new  Interval(1,2);

if( Interval.lessequal(z,y))…     // result false
if( Interval.lessequal(y,z))…     // result true
if( Interval.lessequal(z,z))…     // result true
```

See Also

Interval.notequal(), Interval.equal(), Interval.less(), Interval.greater(), Interval.greaterequal()

Interval.LN2

Return Interval log(2) of the natural logarithm

Synopsis

Interval.LN2

Returns

The Interval constant LN2 [0.6931471805599453:0.6931471805599454]. It is more accurate to use Interval.LN2 instead of the call Interval.log(2)

Example

var z = Interval.LN2;

See Also

Interval.LN10, Interval.E

Interval.LN10

Return Interval log(10) of the natural logarithm

Synopsis

Interval.LN10

Returns

The Interval constant LN10 [2.3025850929940455:2.302585092994046]. It is more accurate to use Interval.LN10 instead of the call Interval.log(10)

Example

var z = Interval.LN10;

See Also

Interval.LN2, Interval.E

Interval.lower()

Return or Set the lower part of the Interval number

Synopsis

Interval object.lower(i)

Arguments

i The optional lower value when setting the upper number to a new value. If omitted the call returns the actual lower value of the Interval number.

Returns

Return the lower part of the Interval number.

Example

```
var z = new Interval(3,4);
z.lower();                  // result 3
z.lower(5);                 // set lower to 2 and return it. Z is now [2:4]
```

See Also

Interval.upper()

Interval.log()

Compute the natural logarithm of x

Synopsis

Interval.log(x)

Arguments

x A Interval numbers greater than 0

Returns

Return log(x). If x is 0 then Interval(*–Infinity)* is returned. If x less than 0 then Interval(*NaN)* is returned.

Example

```
var z = new Interval(3,4);

Interval.log(z)…          // result [1.0986122886681071: 1.3862943611198921]
Interval.log(Interval(2)); // result [0.6931471805599435: 0.6931471805599461]
```

See Also

Interval.exp(), Interval.log10(),

Interval.log10()

Compute the base-10 logarithm of x

Synopsis

Interval.log10(x)

Arguments

x A Interval numbers not equal to zero

Returns

Return log10(x). If x is 0 then *–Infinity* is returned. If x less than 0 then *NaN* is returned.

Example

```
var z = new Interval(3,4);

Interval.log10(z)…          // result [0.4771212547196612: 0.6020599913279631]
Interval.log(Interval(2));  // result [0.30102999566398037: 0.30102999566398153]
```

See Also

Interval.exp(), Interval.log(),

Interval.mul()

Multiply two Interval numbers

Synopsis

Interval.mul(a,b)

Arguments

a,b The Interval numbers to be multiplied.

Returns

The result of the Interval multiplication.

Example

```
var z = new Interval(3,4);
var y=new  Interval(1,2);

Interval.mul(z,y)                    // result [3:8]
Interval.mul(Interval(2.5),Interval(0.1));       // result [0.25:0.25000000000000006]
```

See Also

Interval.add(), Interval.div(), Interval.sub()

Interval.negate()

Return the negated Interval number

Synopsis

Interval object.negate()

Returns

Return the negated Interval number.

Example

```
var z = new Interval(3,4);
z.negate()                    // result [-4:-3]
```

See Also

Interval.notequal()

Compare two Interval numbers for inequality

Synopsis

Interval.notequal(a,b)

Arguments

a,b The Interval numbers to be compare for

Returns

The Boolean value of the notequal comparison.

Example

```
var z = new Interval(3,4);
var y=new  Interval(1,2);

if( Interval.notequal(z,y))…        // result true
if( Interval.notequal(z,z))…        // result false
if( !Interval.noyequal(z,y))…       // result false ! to do “notequal” comparison
```

See Also

Interval.equal(), Interval.equal(), Interval.less(), Interval.lessequal(), Interval.greater(), Interval.greaterequal()

Interval.one

Return Interval one

Synopsis

Interval.one

Returns

The Interval constant one (1+i0).

Example

```
var z = Interval.one;          // z=[1:1]
```

See Also

Interval.zero

Interval.PI

Return Interval constant π

Synopsis

Interval.PI

Returns

The Interval constant PI [3.141592653589793:3.1415926535897936].

Example

```
var z = Interval.PI;
```

See Also

Interval.LN10, Interval.LN2, Interval.E

Interval.pow()

Compute $x^{\wedge y}$

Synopsis

Interval.pow(x,y)

Arguments

x	A Interval numbers to be raised to a power
y	A Interval power that x is raised to

Returns

X to the power of y. x^y

Example

```
var x = new Interval(3,4);
var y = new Interval(1,2);

Interval.pow(x,y)                // result [2.9999999999999787:16.000000000000167]
Interval.power(Interval(3),Interval(2)); // [8.999999999999872:9.00000000000011]
```

See Also

Interval.exp()

Interval.precedes()

Return true if a-interval precedes b-interval, otherwise false

Synopsis

Interval.precedes(a,b)

Arguments

a,b is the interval numbers.

Returns

The Boolean result if *a* precedes *b*. precedes is true if a.high < b.low otherwise false

Example

```
var z = new Interval(3,4);
var y=new  Interval(1,2);
var x=new Interval(3.5,3.9);

Interval.precedes(z,y)          // result  false
Interval.precedes(y,z)          // result  true
Interval.precedes(z,z)          // result false
```

See Also

Interval.interior()

Interval.pred()

Compute the previous representable number less than x

Synopsis

Interval.pred(x)

Arguments

x A floating point number.

Returns

The previous representable number less than x.

Example

```
var z = 1.5

Interval.pred(z);                    // result 1.4999999999999998
```

See Also

Interval.succ()

Interval.sin()

Return the sine of the Interval number

Synopsis

Interval.sin(a)

Returns

Return the sine of the Interval number.

Example

```
var z = new Interval(3,4);
Interval.sin(z)          // result [-0.7568024953079279: 0.1411200080598619]
```

Interval.sin(Interval(2)); // result [0.9092974268256793: 0.909297426825684]

See Also

Interval.cos(), Interval.tan()

Interval.sinh()

Return the sine hyperbolic sine of the Interval number

Synopsis

Interval.sinh(a)

Returns

Return the sine hyperbolic of the Interval number.

Example

```
var z = new Interval(3,4);
Interval.sinh(z)              // result [10.017874927409816: 27.289917197128002]
Interval.sinh(Interval(2));   // result [3.6268604078470106: 3.626860407847036]
```

See Also

Interval.cosh(), Interval.tanh()

Interval.sub()

Subtract two Interval numbers

Synopsis

Interval.sub(a,b)

Arguments

a,b The Interval numbers to be subtracted.

Returns

The result of the Interval subtraction.

Example

```
var z = new Interval(3,4);
var y=new  Interval(1,2);

Interval.sub(z,y)              // result [1:3]
```

See Also

Interval.add(), Interval.div(), Interval.mul()

Interval.sqrt()

Compute an Interval square root

Synopsis

Interval.sqrt(x)

Arguments

x	A Interval numbers to be square rooted. Special calculation are made to prevent intermediate result to overflow.

Returns

The square root of x. if x < 0 then it return Interval(NaN).

Example

```
var z = new Interval(3,4);

Interval.sqrt(z)…              // result [1.7320508075688774: 2]
Interval.sqrt(Interval(2));    // result [1.414213562373095: 1.4142135623730951]
```

See Also

Interval.succ()

Compute the next representable number bigger than x

Synopsis

Interval.succ(x)

Arguments

x A floating point number.

Returns

The next representable number bigger than x.

Example

```
var z = 1.5

Interval.succ(z);          // result 1.5000000000000002
```

See Also

Interval.pred()

Interval.tan()

Return the tangent of the Interval number

Synopsis

Interval.tan(a)

Returns

Return the tangent of the Interval number. If a==π/2 then it return Interval(*Infinity*). If a=3π/2 then it return Interval(–*Infinity*).

Example

```
var z = new Interval(3,4);
Interval.tan(z)              // result [-0.14254654307428194: 1.15782128234958]
Interval.tan(Interval(2));   // result [-2.185039863261552: -2.1850398632614847]
```

See Also

Interval.cos(), Interval.sin()

Interval.tanh()

Return the tangent hyperbolic of the Interval number

Synopsis

Interval.tanh(a)

Returns

Return the tanh hyperbolic of the Interval number.

Example

```
var z = new Interval(3,4);
Interval.tanh(z)              // result [0.13495454840125132: 7.368313124558387]
Interval.tanh(interval(2));   // result [0.9640275800758038: 0.96402758007583]
```

See Also

Interval.cosh(), Interval.sinh()

Interval.toClose()

Convert the interval to a close form of an interval.

Synopsis

Interval.toClose(a)

Returns

Return the Close interval form of a. If a is already in a closed form then a is returned. If a is half open on the left side then the closed form of [Interval.succ(a.lower()) : a.upper()] is returned. If a is half open on the right side then the closed form of [a.lower() : Interval.pred(a.upper())] is returned and if in open form then [Interval.succ(a.lower()) : Interval.pred(a.upper())] is returned.

Example

```
var z = new Interval(1.5,2,"(]");
var x;
x=Interval.toClose(z)         // result [1.5000000000000002: 2]
Interval.toClose(x);          // result [1.5000000000000002: 2]  or unchanged
                              // since already in closed form
```

See Also

Interval.toOpen()

Interval.toExponential()

Format a number using exponential notation

Synopsis

Interval.toExponential(digits)

Arguments

Digits The number of digits that will appear after the decimal point. This may be a value between 0 and 20, inclusive. If this argument is omitted, as many digits as necessary will be used. An Interval number is always formatted as:

[*lower_part* : *upper_part*]

The left or right bracket can be rounded bracket and then they represent an open or half open interval.

Returns

A string representations of the Interval number, in exponential notation, with one digit before the decimal place and *digits* digits after the decimal place. The fractional part of the Interval number is rounded or padded with zeros, as necessary, so that is has the specified length.

Example

```
var z = new Interval( 12345.6789, 12345.6789 );
z.toExponential(1);          // result [1.2e+4:1.2e+4]
z.toExponential(5);          // result [1.23457e+4:1.23457e+4]
z.toExponential(10);         // result [1.23456789000e+4:1.23456789000e+4]
z.toExponential();           // result [1.23456789e+4:1.23456789e+4]
```

See Also

Interval.toFixed(), Interval.toPrecision(), Interval.toString()

Interval.toFixed()

Format a number using fixed-point notation

Synopsis

Interval.toFixed(digits)

Arguments

Digits The number of digits that will appear after the decimal point. This may be a value between 0 and 20, inclusive. If this argument is omitted, it is treated as zero. An Interval number is always formatted as:

[*lower_part* : *upper_part*]

The left or right bracket can be rounded bracket and then they represent an open or half open interval.

Returns

A string representation of the Interval number, that does not used exponential notation and has exactly *digits* digits after the decimal point. The *Interval number* is rounded as necessary, and the fraction part is padded with zeros if necessary so that it has the specified length. If the *Interval number* is greater than 1e+21, this method simple calls *number*.toString() and return a string in exponential notation.

Example

```
var z = new Interval( 12345.6789, 12345.6789,"[]" );
z.toFixed(5);          // result [12345.7:12345.7]
z.toFixed(6);          // result [12345.678900:12345.678900]
z.toFixed();           // result [12346:1234.6]
```

See Also

Interval.toExponential(), Interval.toPrecision(), Interval.toString()

Interval.toOpen()

Convert the interval to an open form of an interval.

Synopsis

Interval.toOpen(a)

Returns

Return the Open interval form of a. If *a* is already in an open form then a is returned. If *a* is half open on the left side then the open form of (a.lower() : Interval.pred(a.upper())) is returned. If *a* is half open on the right side then the open form of (Interval.pred(a.lower()) : a.upper()) is returned and if in open form then (Interval.pred(a.lower()) : Interval.succ(a.upper())) is returned.

Example

```
var z = new Interval(1.5,1.5,"[)");
var x;
x=Interval.toOpen(z)        // result (1.4999999999999998:1.5)
Interval.toOpen(x);         // result (1.4999999999999998:1.5) or unchanged
                            // since already in an open form
```

See Also

Interval.toClose()

Interval.toPrecision()

Format the significant digits of a Interval number

Synopsis

Interval.toPrecision(digits)

Arguments

Digits The number of significant digits to appear in the returned string. This may be a value between 1 and 21, inclusive. If this argument is omitted, the toString() method is used instead to convert the Interval number to a base-10 value. An Interval number is always formatted as:

[lower_part : upper_part]

The left or right bracket can be rounded bracket and then they represent an open or half open interval.

Returns

A string representation of the *Interval number*, which contains *precisions* significant digits. If *precision* is large enough to include all the digits of the integer part of number, the returned string uses fixed-point notation. Otherwise, exponential notation is used with one digit before the decimal place and *precision* – 1 digits after the decimal place. The number is rounded or padded with zeros as necessary.

Example

```
var z = new Interval( 12345.6789, 12345.6789 );
z.toPrecision(1);              // result [1e+4:1e+4]
z.toPrecision(3);              // result [1.23e+4:1.2e+4]
z.toPrecision(5);              // result [12346:12346]
```

See Also

Interval.toExponential(), Interval.toFixed(), Interval.toString()

Interval.toString()

Format the significant digits of an Interval number

Synopsis

Interval.toString(radix)

Arguments

Radix If omitted the base 10 will be used to convert the Interval number to a string. Otherwise the radix will be used (2..36). An Interval number is always formatted as:

[*lower_part* : *upper_part*]

The left or right bracket can be rounded bracket and then they represent an open or half open interval.

Returns

A string representations of the *Interval number*, in the indicated radix.

Example

```
var z = new Interval( 12345.6789, 12345.6789 );
z.toString();              // result [1234.6789:1234.6789]
```

See Also

Interval.toExponential(), Interval.toFixed(), Interval.toPrecision()

Interval.union()

Create the Union of two Interval numbers

Synopsis

Interval.union(a,b)

Arguments

a,b The Interval numbers to be unionized or together. It create and interval from the lowest to the highest range. It conceptual similar to or'ring two intervals.

Returns

The result of the Interval union.

Example

```
var z = new Interval(3,4);
var y=new  Interval(1,2);

Interval.union(z,y)              // result  [1:4]
```

See Also

Interval.intersection()

Interval.upper()

Return or Set the upper part of the Interval number

Synopsis

Interval object.upper(*r*)

Arguments

r The optional upper value when setting the upper number to a new value. If omitted the call returns the actual upper value of the Interval number.

Returns

Return the upper part of the Interval number.

Example

```
var z = new Interval(3,4);
z.upper();                  // result 4
z.upper(5);                 //set the upper part to 5 and return the result 5
```

See Also

Interval.lower()

Interval.valueof()

Return the primitive number value

Synopsis

Interval object.valueof()

Returns

The primitive value of the *Interval number* is returned, which is the same as *Interval number*.lower().

Example

```
var z = new Interval(3,4);
z.valueOf()          // return 3
```

See Also

Interval.width()

Return the width of the Interval number

Synopsis

Interval.width(a)

Returns

Return the width of the Interval number a.

Example

```
var z = new Interval(3,4);
Interval.width(z)          // result 1
```

See Also

Interval.center()

Interval.zero

Return Interval zero

Synopsis

Interval.zero

Returns

The Interval constant zero [0:0].

Example

var z = Interval.zero;

See Also

Interval.one

parseInterval()

Convert a string to an Interval number

Synopsis

parseInterval(s)

Arguments

s The string to be parsed and converted to a *Interval number.*

Returns

parseInterval() parses and return a new Interval number contained in s. parseInterval() return a Interval NaN number if parsing fails. A Interval number can either be in the format:

[*lower_part : upper_part*]

Where either the *lower_part* or the *upper _part* can be missing but not both at the same time. parseInterval() can also parsed string omitting the leading and trailing parentheses or square brackets.

Example

```
var z = parseInterval( “(1.2:3.4E-5)”);      // result (1.2:3.4E-5) Open
z = parseInterval( “[1.2]” );                // result [1.2 :1.2]  Closed
z = parseInterval( “(1:2]” );                // result (1 : 2]  Half open
```

See Also

Example

Simplified version of the Interval.log() function. Exponent handling and argument reduction has been removed to simplify the example.

```
Interval.log=function(t)
        {var low,high;
        function intervallog(x)
                {var zn,zsq,i,k,sum,delta;
                if(x<0) {return new Interval(NaN,NaN);}
                if(x==0) {return new Interval(-Infinity,-Infinity);}
                if(x==1) {return new Interval(0);}
                zn=Interval(x);
                // Taylor series of log(x)
                // log(x)=2( z + z^3/3 + z^5/5 ...)
                // where z=(x-1)/(x+1)
                // Initialize the iteration
                zn=Interval.div(Interval.sub(zn,Interval.one),Interval.add(zn,Interval.one));
                zsq=Interval.mul(zn,zn); sum=zn;
                // Iterate using taylor series log(x) == 2( z + z^3/3 + z^5/5 ... )
                for(i=3;;i+=2)
                        {
                        zn=Interval.mul(zn,zsq);
                        delta=Interval.div(zn,Interval(i));
                        if(sum.center()+delta.center()==sum.center()) break;
                        sum=Interval.add(sum,delta);
                        }
                sum=Interval.mul(sum,Interval(2));
                return sum;
                }
        if(t.lower()==t.upper()) return intervallog(t.lower());
        low=intervallog(t.lower()).lower(); high=intervallog(t.upper()).upper();
        return new Interval(low,high);
        }
```