class Complex

A complex number can be represented as a paired real number with imaginary unit; a+bi. Where a is real part, b is imaginary part and i is imaginary unit. Real a equals complex a+0i mathematically.

You can create a Complex object explicitly with:

- A complex literal.

You can convert certain objects to Complex objects with:

- Method `Complex`.

`Complex` object can be created as literal, and also by using `Kernel#Complex`, `Complex::rect`, `Complex::polar` or `to_c` method.

```
2+1i                  #=> (2+1i)
Complex(1)            #=> (1+0i)
Complex(2, 3)         #=> (2+3i)
Complex.polar(2, 3)   #=> (-1.9799849932008908+0.2822400161197344i)
3.to_c                #=> (3+0i)
```

You can also create complex object from floating-point numbers or strings.

```
Complex(0.3)          #=> (0.3+0i)
Complex('0.3-0.5i')   #=> (0.3-0.5i)
Complex('2/3+3/4i')   #=> ((2/3)+(3/4)*i)
Complex('1@2')        #=> (-0.4161468365471424+0.9092974268256817i)

0.3.to_c              #=> (0.3+0i)
'0.3-0.5i'.to_c       #=> (0.3-0.5i)
'2/3+3/4i'.to_c       #=> ((2/3)+(3/4)*i)
'1@2'.to_c            #=> (-0.4161468365471424+0.9092974268256817i)
```

A complex object is either an exact or an inexact number.

```
Complex(1, 1) / 2    #=> ((1/2)+(1/2)*i)
Complex(1, 1) / 2.0  #=> (0.5+0.5i)
```

Constants

I

The imaginary unit.

Public Class Methods

polar(abs[, arg]) → complex

Returns a complex object which denotes the given polar form.

```
Complex.polar(3, 0)            #=> (3.0+0.0i)
Complex.polar(3, Math::PI/2)   #=> (1.836909530733566e-16+3.0i)
Complex.polar(3, Math::PI)     #=> (-3.0+3.673819061467132e-16i)
Complex.polar(3, -Math::PI/2)  #=> (1.836909530733566e-16-3.0i)
```

rect(real[, imag]) → complex
rectangular(real[, imag]) → complex

Returns a complex object which denotes the given rectangular form.

```
Complex.rectangular(1, 2)  #=> (1+2i)
```

rect(real[, imag]) → complex
rectangular(real[, imag]) → complex

Returns a complex object which denotes the given rectangular form.

```
Complex.rectangular(1, 2)  #=> (1+2i)
```

Public Instance Methods

cmp * numeric → complex

Performs multiplication.

```
Complex(2, 3)  * Complex(2, 3)   #=> (-5+12i)
Complex(900)   * Complex(1)      #=> (900+0i)
Complex(-2, 9) * Complex(-9, 2)  #=> (0-85i)
Complex(9, 8)  * 4               #=> (36+32i)
Complex(20, 9) * 9.8             #=> (196.0+88.2i)
```

cmp ** numeric → complex

Performs exponentiation.

```
Complex('i') ** 2              #=> (-1+0i)
Complex(-8) ** Rational(1, 3)  #=> (1.0000000000000002+1.7320508075688772i)
```

cmp + numeric → complex

Performs addition.

```
Complex(2, 3)  + Complex(2, 3)   #=> (4+6i)
Complex(900)   + Complex(1)      #=> (901+0i)
Complex(-2, 9) + Complex(-9, 2)  #=> (-11+11i)
Complex(9, 8)  + 4               #=> (13+8i)
Complex(20, 9) + 9.8             #=> (29.8+9i)
```

cmp - numeric → complex

Performs subtraction.

```
Complex(2, 3)  - Complex(2, 3)   #=> (0+0i)
Complex(900)   - Complex(1)      #=> (899+0i)
Complex(-2, 9) - Complex(-9, 2)  #=> (7+7i)
Complex(9, 8)  - 4               #=> (5+8i)
Complex(20, 9) - 9.8             #=> (10.2+9i)
```

-cmp → complex

Returns negation of the value.

```
-Complex(1, 2)  #=> (-1-2i)
```

cmp / numeric → complex
quo(numeric) → complex

Performs division.

```
Complex(2, 3)  / Complex(2, 3)   #=> ((1/1)+(0/1)*i)
Complex(900)   / Complex(1)      #=> ((900/1)+(0/1)*i)
Complex(-2, 9) / Complex(-9, 2)  #=> ((36/85)-(77/85)*i)
Complex(9, 8)  / 4               #=> ((9/4)+(2/1)*i)
Complex(20, 9) / 9.8             #=> (2.0408163265306123+0.9183673469387754i)
```

cmp <=> object → 0, 1, -1, or nil

If `cmp` 's imaginary part is zero, and `object` is also a real number (or a `Complex` number where the imaginary part is zero), compare the real part of `cmp` to object. Otherwise, return nil.

```
Complex(2, 3)  <=> Complex(2, 3)   #=> nil
Complex(2, 3)  <=> 1               #=> nil
Complex(2)     <=> 1               #=> 1
Complex(2)     <=> 2               #=> 0
Complex(2)     <=> 3               #=> -1
```

cmp == object → true or false

Returns true if cmp equals object numerically.

```
Complex(2, 3)  == Complex(2, 3)   #=> true
Complex(5)     == 5               #=> true
Complex(0)     == 0.0             #=> true
Complex('1/3') == 0.33            #=> false
Complex('1/2') == '1/2'           #=> false
```

abs → real

Returns the absolute part of its polar form.

```
Complex(-1).abs         #=> 1
Complex(3.0, -4.0).abs  #=> 5.0
```

Also aliased as: magnitude

abs2 → real

Returns square of the absolute value.

```
Complex(-1).abs2          #=> 1
Complex(3.0, -4.0).abs2   #=> 25.0
```

angle → float

Returns the angle part of its polar form.

```
Complex.polar(3, Math::PI/2).arg  #=> 1.5707963267948966
```

Alias for: *arg*

arg → float

Returns the angle part of its polar form.

```
Complex.polar(3, Math::PI/2).arg  #=> 1.5707963267948966
```

Also aliased as: *angle*, *phase*

conj → complex

Returns the complex conjugate.

```
Complex(1, 2).conjugate  #=> (1-2i)
```

Alias for: *conjugate*

conjugate -> complex

Returns the complex conjugate.

```
Complex(1, 2).conjugate  #=> (1-2i)
```

Also aliased as: *conj*

denominator → integer

Returns the denominator (lcm of both denominator - real and imag).

See numerator.

`fdiv(numeric) → complex`

Performs division as each part is a float, never returns a float.

```
Complex(11, 22).fdiv(3)  #=> (3.6666666666666665+7.333333333333333i)
```

`finite? → true or false`

Returns `true` if `cmp`'s real and imaginary parts are both finite numbers, otherwise returns `false`.

`hash()`

`imag → real`

Returns the imaginary part.

```
Complex(7).imaginary      #=> 0
Complex(9, -4).imaginary  #=> -4
```

Alias for: imaginary

`imaginary -> real`

Returns the imaginary part.

```
Complex(7).imaginary      #=> 0
Complex(9, -4).imaginary  #=> -4
```

Also aliased as: imag

`infinite? → nil or 1`

Returns `1` if `cmp`'s real or imaginary part is an infinite number, otherwise returns `nil`.

```
For example:

   (1+1i).infinite?                   #=> nil
   (Float::INFINITY + 1i).infinite?   #=> 1
```

inspect → string

Returns the value as a string for inspection.

```
Complex(2).inspect                       #=> "(2+0i)"
Complex('-8/6').inspect                  #=> "((-4/3)+0i)"
Complex('1/2i').inspect                  #=> "(0+(1/2)*i)"
Complex(0, Float::INFINITY).inspect      #=> "(0+Infinity*i)"
Complex(Float::NAN, Float::NAN).inspect  #=> "(NaN+NaN*i)"
```

magnitude → real

Returns the absolute part of its polar form.

```
Complex(-1).abs         #=> 1
Complex(3.0, -4.0).abs  #=> 5.0
```

Alias for: abs

numerator → numeric

Returns the numerator.

```
    1   2       3+4i  <-  numerator
    - + -i  ->  ----
    2   3        6    <-  denominator

c = Complex('1/2+2/3i')  #=> ((1/2)+(2/3)*i)
n = c.numerator          #=> (3+4i)
d = c.denominator        #=> 6
n / d                    #=> ((1/2)+(2/3)*i)
Complex(Rational(n.real, d), Rational(n.imag, d))
                         #=> ((1/2)+(2/3)*i)
```

See denominator.

phase → float

Returns the angle part of its polar form.

```
Complex.polar(3, Math::PI/2).arg  #=> 1.5707963267948966
```

Alias for: arg

polar → array

Returns an array; [cmp.abs, cmp.arg].

```
Complex(1, 2).polar  #=> [2.23606797749979, 1.1071487177940904]
```

cmp / numeric → complex
quo(numeric) → complex

Performs division.

```
Complex(2, 3)  / Complex(2, 3)   #=> ((1/1)+(0/1)*i)
Complex(900)   / Complex(1)      #=> ((900/1)+(0/1)*i)
Complex(-2, 9) / Complex(-9, 2)  #=> ((36/85)-(77/85)*i)
Complex(9, 8)  / 4               #=> ((9/4)+(2/1)*i)
Complex(20, 9) / 9.8             #=> (2.0408163265306123+0.9183673469387754i)
```

rationalize([eps]) → rational

Returns the value as a rational if possible (the imaginary part should be exactly zero).

```
Complex(1.0/3, 0).rationalize  #=> (1/3)
Complex(1, 0.0).rationalize    # RangeError
Complex(1, 2).rationalize      # RangeError
```

See to_r.

real → real

Returns the real part.

```
Complex(7).real      #=> 7
Complex(9, -4).real  #=> 9
```

Complex(1).real? → false
Complex(1, 2).real? → false

Returns false, even if the complex number has no imaginary part.

rect → array

Returns a complex object which denotes the given rectangular form.

```
Complex.rectangular(1, 2)  #=> (1+2i)
```

Alias for: [rectangular]

rectangular -> array

Returns an array; [cmp.real, cmp.imag].

```
Complex(1, 2).rectangular  #=> [1, 2]
```

Also aliased as: [rect], rect

to_c → self

Returns self.

```
Complex(2).to_c       #=> (2+0i)
Complex(-8, 6).to_c   #=> (-8+6i)
```

to_f → float

Returns the value as a float if possible (the imaginary part should be exactly zero).

```
Complex(1, 0).to_f    #=> 1.0
Complex(1, 0.0).to_f  # RangeError
Complex(1, 2).to_f    # RangeError
```

to_i → integer

Returns the value as an integer if possible (the imaginary part should be exactly zero).

```
Complex(1, 0).to_i    #=> 1
Complex(1, 0.0).to_i  # RangeError
Complex(1, 2).to_i    # RangeError
```

to_r → rational

Returns the value as a rational if possible (the imaginary part should be exactly zero).

```
Complex(1, 0).to_r    #=> (1/1)
Complex(1, 0.0).to_r  # RangeError
Complex(1, 2).to_r    # RangeError
```

See rationalize.

to_s → string

Returns the value as a string.

```
Complex(2).to_s                      #=> "2+0i"
Complex('-8/6').to_s                 #=> "-4/3+0i"
Complex('1/2i').to_s                 #=> "0+1/2i"
Complex(0, Float::INFINITY).to_s     #=> "0+Infinity*i"
Complex(Float::NAN, Float::NAN).to_s #=> "NaN+NaN*i"
```