Ruby master - Feature #18179
Add Math methods to Numeric
09/20/2021 04:09 AM - ankane (Andrew Kane)

Status: Open
Priority: Normal
Assignee:
Target version:

Description
Hi, I wanted to get thoughts on adding class methods from Math as instance methods on Numeric.

x.sqrt # vs Math.sqrt(x)
x.log   # vs Math.log(x)

Rust takes this approach and it (subjectively) feels more intuitive/object-oriented. It also seems more consistent with methods like x.abs.

Related issues:
Has duplicate Ruby master - Feature #18477: Float#sqrt and Integer#sqrt

History
#1 - 09/20/2021 06:48 AM - duerst (Martin Dürst)
I support this. x.sqrt is indeed more object-oriented that Math.sqrt x. In an earlier discussion, it was pointed out that for Mathematicians, sqrt(x) is more natural than x.sqrt. Mathematicians can still use that notation, but also having the object-oriented notation in Ruby would indeed be great.

#2 - 09/22/2021 01:44 AM - mrkn (Kenta Murata)
I'm negative to this proposal. I don't think Math.sqrt is the behavior or the property of a Numeric object. It is the positive square root function that maps from/to the set of non-negative real numbers.

If we introduce Numeric#sqrt, we should expand its domain to negative numbers and Complex numbers. Also, it is better to consider other kinds of numbers, such as Quaternion.

#3 - 01/12/2022 12:56 AM - mame (Yusuke Endoh)
- Has duplicate Feature #18477: Float#sqrt and Integer#sqrt added

#4 - 03/24/2022 04:44 AM - duerst (Martin Dürst)
mrkn (Kenta Murata) wrote in #note-2:

I'm negative to this proposal. I don't think Math.sqrt is the behavior or the property of a Numeric object. It is the positive square root function that maps from/to the set of non-negative real numbers.

This is how Mathematicians think, and we don't want to take this possibility away from them. But Ruby is mainly for Ruby programmers. And most Ruby programmers may not think like Mathematicians. Ruby programmers should be able to use Ruby the Ruby way, even for such functionality.

For example, if I have an array a, and want to create an array of square roots from it, currently I have to write:

```ruby
a.map { |n| Math.sqrt n }
```

With this proposal, it would be possible to write:

```ruby
a.map(&:sqrt)
```

This expresses the intent (map the square root function) in a much more concise and functional way.

If we introduce Numeric#sqrt, we should expand its domain to negative numbers and Complex numbers. Also, it is better to consider other kinds of numbers, such as Quaternion.

Now that Complex is built-in, producing an error on Math.sqrt(-2) indeed doesn't look good. That's independent of this proposal, but it could be implemented together.

#5 - 03/24/2022 07:13 AM - Hanmac (Hans Mackowiak)
For the Complex Problem, maybe add an optional parameter like `raise_on_complex` or something so when you try:

`(-2).sqrt` it returns complex,
but `(-2).sqrt(raise_on_complex: true)` makes the old Exception?

---

#6 - 03/24/2022 02:54 PM - Eregon (Benoit Daloze)

Out of all Math methods:

- `acos`, `acosh`, `asin`, `asinh`, `atan`, `atan2`, `atanh`, `cbt`, `cos`, `cosh`, `erf`, `erf`, `exp`, `frexp`, `gamma`, `hypot`, `ldexp`, `lgamma`, `log`, `log10`, `log2`, `sin`, `sinh`, `sqrt`, `tan`, `tanh`

I think for me only `sqrt` would maybe feel natural as `num.sqrt`.

We already have `Integer.sqrt` (but no `#sqrt`), so that might be confusing. Math always deals with Float numbers, so that's consistent, but having it on numeric is less clear, should 5.sqrt be 2 (like `Integer.sqrt`) or 2.23606797749979 (like `Math.sqrt`)?

It's also not clear what should be `Rational(a, b).sqrt`, a Float, a `Rational`? The point of `Rational` is to be exact, converting to Float somewhat implicitly doesn't seem good.

At least I feel `num.sin`, `num.cos`, `num.acos`, `num.cosh`, etc, all look weird.
I would think many people would generally agree to that, but maybe I'm wrong.

Math.hypot seems also a good example why that should not be an instance method, because it takes two arguments as equal, there is no "receiver" and "operand" distinction.

Moving these methods to Float (instead of Numeric) would at least make it clear they use Float operands and return a Float, and sounds like a better change to me.
That would mean `integer.to_f.sqrt` if one wants the Float square root of an integer.

It seems Rust defines most of the Ruby Math methods on f64 (https://doc.rust-lang.org/std/primitive.f64.html) and much less (e.g., no `sqrt`) on i64 (https://doc.rust-lang.org/std/primitive.i64.html).