

Paul Razvan Berg

PRBMath: A Modern Smart Contract Library for Fixed-Point Math

Web3



code.talks

Who am I?

- Co-founder of Sablier, an Ethereum-based payment protocol with >\$100M in volume
- Smart contract developer since 2018, focused on Solidity
- Active open-source developer and contributor



What is Solidity? 🧑💻

- The most popular programming language for writing Ethereum smart contracts
- Smart contracts are immutable scripts that target the Ethereum Virtual Machine ("EVM")
- High-level and object-oriented
- Influenced by C++, Python and JavaScript



What is the Problem? 🤔

- Solidity has neither fixed-point nor floating-point types
- Solidity also doesn't have any built-in advanced math functions (e.g. logs)
- Implementing advanced math functions from scratch is hard
- Few math libraries take advantage of the latest v0.8 features

Example Projects

- Percentages and proportionalities
- Interest rates
- Derivative pricing
- Automated Market Makers (AMM)
- Custom trading algorithms



How can I represent decimal values in Solidity?

Asked 6 years, 5 months ago Modified 2 years, 11 months ago Viewed 35k times

▲

39

▼

★

7

🕒

I just started writing some solidity and I noticed that there are no decimal values such as double or float.

What do you guys do when you need to return send to a wallet let's say 1.57 ether? Or when you need to calculate $1/2 = 0.5$?

I'm very confused - I tried doing something like `var a = 0.5;` but it gives me an error that says

Invalid literal value.

solidity contract-design contract-development Edit tags

Share Edit Follow Close Flag

edited Jan 29, 2018 at 18:41

 Community Bot

1

asked Apr 14, 2016 at 10:33

 sfrj

712 1 8 12

Can't do any integer division

Asked 5 years, 5 months ago Modified 11 months ago Viewed 18k times

▲ I'm building a new smart contract but really can't figure out how to perform integer division. I know that fixed point numbers haven't been implemented yet but it should be possible to perform $100/3 = 33$, at least.

▼ I hope someone can help me. I have seen multiple examples where they talk about rounded integer division but can't seem to be able to perform it.

2 Cheers, Hugo

solidity data-types integers math fixed-point Edit tags

Share Edit Follow Close Flag

asked Apr 10, 2017 at 17:56

 Hugo Moreau

223 1 2 4

Is there any efficient way to compute the exponentiation of a fraction and an integer?

Asked 5 years, 9 months ago Modified 1 year, 4 months ago Viewed 3k times

▲ Mind the following JavaScript operation:

13

```
function exp(n){
  return Math.floor(Math.pow(1.01, n));
}
```

▼ Where `r` is a fraction (such as `1.01`) and `n` is an integer. Is there any efficient way to emulate that operation efficiently on Ethereum? That is, a function:

```
function exp(uint n) returns (uint) {
  return ...
}
```

Which returns similar values to the former? Such a function would have many use cases; for example, computing compound interest rates.

solidity dapp-development Edit tags

Share Edit Follow Close Flag

edited Dec 1, 2016 at 23:12

asked Dec 1, 2016 at 17:22

 MaiaVictor

3,147 2 15 36

The Overflow Bl

- ✍ The many pr Single Sign-
- ✍ Hackathons Stack Overfl Ambassador

Featured on Met

- 📄 Google Anal
- 📄 Planned mai Wednesday, 03:00 UTC..

✉ Love this si

Get the weekly

- The week's t
- Important co
- Questions th

Solution: PRBMath

So you don't have to reinvent the wheel

- Modern fixed-point math library that operates with 18 decimals
- Leverages the latest features added in Solidity v0.8
 - User-defined value types
 - Custom errors
- Feature rich: logarithms, exponentials, square roots and more
- Gas efficient, but still user-friendly
- Well documented and commented

Basic Features

- avg (average)
- div (division)
- inv (inversion)
- gm (geometric mean)
- mul (multiplication)
- powu (raise to whole power)

Advanced Features

- exp (raise e to some value)
- exp2 (raise 2 to some value)
- ln (natural logarithm)
- log2 (binary logarithm)
- log10 (common logarithm)
- pow (raise some value to some value)
- sqrt (square root)

Simple Numbers

- Let's define the following numbers using PRBMath's UD60x18:
 - 0.009
 - 1.25
 - PI
- UD60x18 = number with 60 digits and 18 decimals



```
function simpleNumbers() external pure {  
    UD60x18 x = UD60x18.wrap(0.009e18);  
    UD60x18 y = UD60x18.wrap(1.25e18);  
    UD60x18 z = UD60x18.wrap(3_141592653589793238);  
}
```

Percentage

- Let's calculate what is 3.14% of 1729
- The result should be 54.2906
- Fun fact: 1729 is the smallest number expressible as the sum of two cubes in two different ways



```
pragma solidity >=0.8.13;

import { SCALE, UD60x18 } from "@prb/math/UD60x18.sol";

contract Percentage {
    UD60x18 internal constant x = UD60x18.wrap(3.14e18);
    UD60x18 internal constant y = UD60x18.wrap(1729e18);

    function foo() external pure returns (UD60x18 percentage) {
        percentage = x.mul(SCALE).div(y);
    }
}
```

Yield Space AMM

$$\left(\frac{y}{x}\right)^t = \left(\frac{\left(x_{start}^{1-t} + y_{start}^{1-t} - x^{1-t}\right)^{\frac{1}{1-t}}}{x}\right)^t$$



```
pragma solidity >=0.8.13;

import { SCALE, UD60x18 } from "@prb/math/UD60x18.sol";

contract YieldSpace {
    UD60x18 internal constant k = UD60x18.wrap(7927447996);
    UD60x18 internal constant g = UD60x18.wrap(1052631578947368421);

    function trade(
        UD60x18 x_s,
        UD60x18 y_s,
        UD60x18 x,
        UD60x18 ttm
    ) external pure returns (UD60x18 y) {
        UD60x18 t = k.mul(ttm);
        UD60x18 a = SCALE.uncheckedSub(g.mul(t));
        UD60x18 base = x_s.pow(a).uncheckedAdd(y_s.pow(a)).uncheckedSub(x.pow(a));
        y = base.pow(a.inv());
    }
}
```

Solidity v0.8 Features

The cherries on top

- Custom errors
 - More gas efficient and more elegant
 - The future of Solidity error reporting
- User-defined value types
 - Type safety
- Unchecked arithmetic
 - Increased gas efficiency



```
error PRBMathUD60x18__AddOverflow(uint256 x, UD60x18 y);
error PRBMathUD60x18__CeilOverflow(UD60x18 x);
error PRBMathUD60x18__ExpInputTooBig(UD60x18 x);
error PRBMathUD60x18__Exp2InputTooBig(UD60x18 x);
error PRBMathUD60x18__GmOverflow(UD60x18 x, UD60x18 y);
error PRBMathUD60x18__LogInputTooSmall(UD60x18 x);
error PRBMathUD60x18__SqrtOverflow(UD60x18 x);
error PRBMathUD60x18__SubUnderflow(UD60x18 x, UD60x18 y);
error PRBMathUD60x18__ToUD60x18Overflow(uint256 x);
```

Developer Experience 🙌

- NatSpec comments, everywhere.
 - NatSpec ~ JSDoc
- Explanatory comments in the function bodies
- Clear installation and usage guides



frankie
@FrankielsLost

Replying to @0xTomoyo

Highly recommend PRBMath for a cleanly written and commented fixed point arithmetic library. They use the same algo but include an explanation for the bitshifts: setting an initial guess to the closest power of 2 greater than x github.com/hifi-finance/p...

```
// Set the initial guess to the closest power of two that is higher than x.
uint256 xAux = uint256(x);
result = 1;
if (xAux >= 0x100000000000000000000000000000000) {
    xAux >>= 128;
    result <<= 64;
}
if (xAux >= 0x10000000000000000) {
    xAux >>= 64;
    result <<= 32;
}
if (xAux >= 0x100000000) {
    xAux >>= 32;
    result <<= 16;
}
if (xAux >= 0x10000) {
    xAux >>= 16;
    result <<= 8;
}
if (xAux >= 0x100) {
    xAux >>= 8;
}
```

Testing and Security

- ~1,400 tests written
 - Domain boundaries
 - Inflection points
 - Edge cases
- ~96% test coverage
- Switching to Foundry soon™



Thanks for Coming 🙏

- QR code gets you to the source code
- Let's be social!
 - @PaulRBerg on Twitter 🐦
 - @PaulRBerg on GitHub 🧑💻



SCAN ME