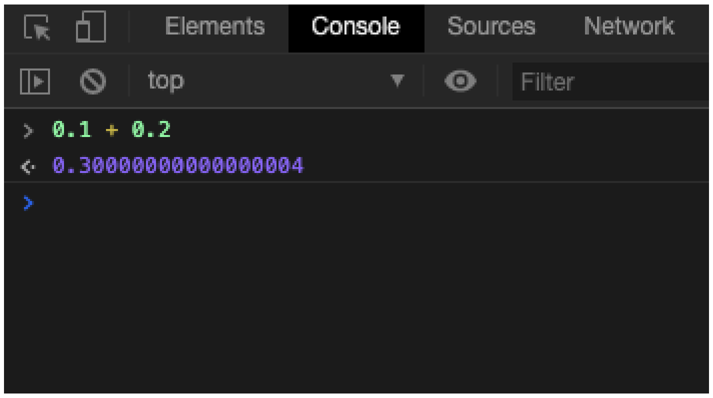
Why is 0.1 + 0.2 Not Equal to 0.3 in Most Programming Languages?

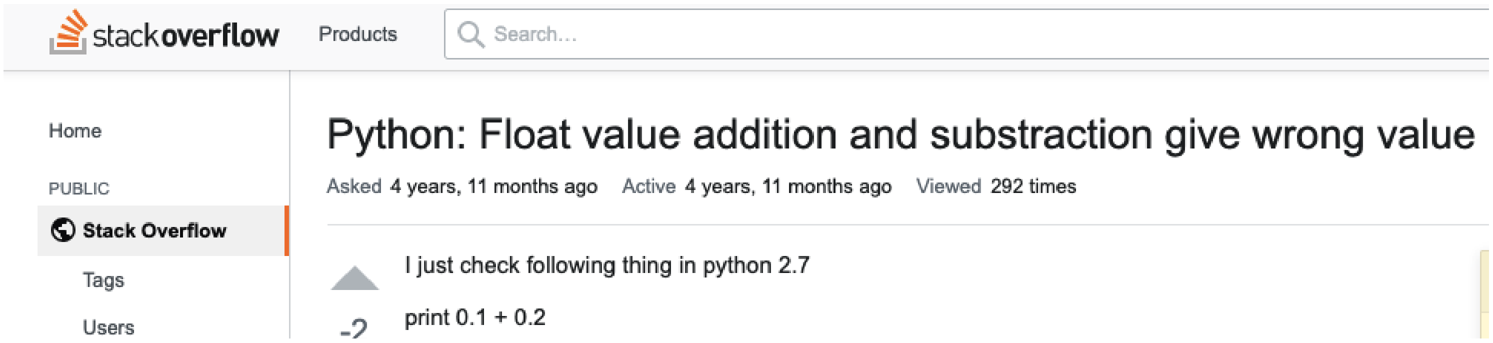
Parul Malhotra Follow
Sep 30, 2019 · 4 min read ★



```
> 0.1 + 0.2
<· 0.30000000000000004
>
```

Ever since childhood, we have been taught that 0.1 + 0.2 equals 0.3. However, in the baffling world of computing, things work pretty differently.

I recently started to code in JavaScript, and while reading about data types, I noticed the strange behavior of 0.1 + 0.2 not being equal to 0.3. I resorted to Stack Overflow for help and found a couple of posts that helped. Have a look below:



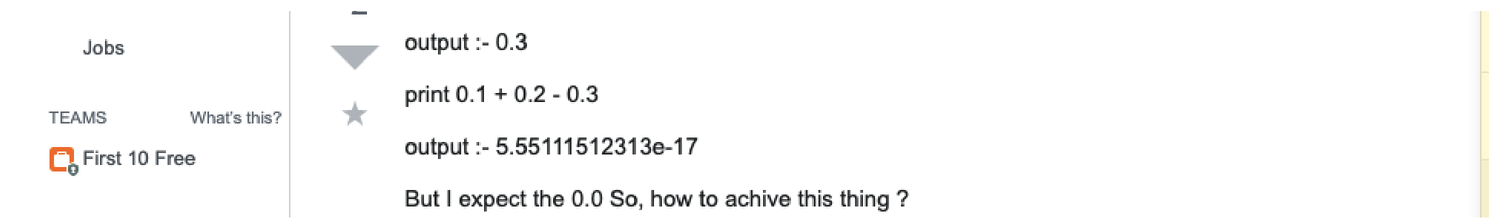
Python: Float value addition and substraction give wrong value

Asked 4 years, 11 months ago Active 4 years, 11 months ago Viewed 292 times

-2

I just check following thing in python 2.7

print 0.1 + 0.2

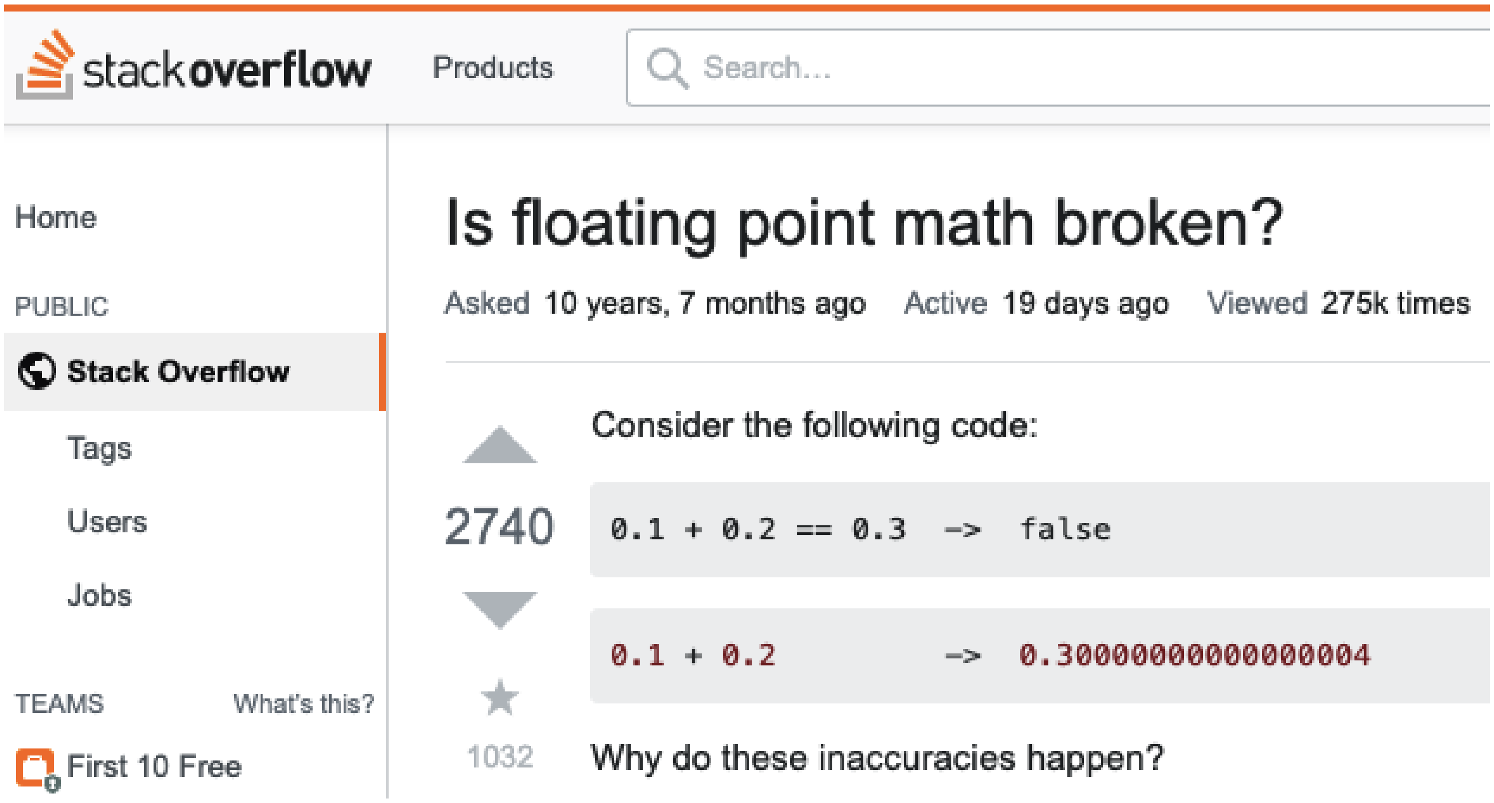


```
output :- 0.3

print 0.1 + 0.2 - 0.3

output :- 5.55111512313e-17
```

But I expect the 0.0 So, how to achive this thing ?



Is floating point math broken?

Asked 10 years, 7 months ago Active 19 days ago Viewed 275k times

Consider the following code:

```
0.1 + 0.2 == 0.3  ->  false
```

```
0.1 + 0.2         ->  0.30000000000000004
```

Why do these inaccuracies happen?

Stack Overflow reference images

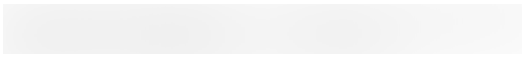
After doing a lot of research and math, I concluded this is not an error. This is math: floating-point arithmetic. Let's dig deeper to understand what's happening behind the scenes.

Problem statement: How is it that `0.1 + 0.2 = 0.30000000000000004` ?

Well, if you have done programming in languages like Java or C, you must be aware of different data types used to store values. The two data types we would be considering in the discussion ahead are *integer* and *float*.

Integer data types store whole numbers, while float data types store fractional numbers.

Before we proceed, let's understand one small concept: How are numbers represented for computational purposes? Very small and very large numbers are usually stored in scientific notation. They are represented as:



Also, a number is normalized when it is written in scientific notation with one nonzero decimal digit before the decimal point. For example, a number 0.0005606 in scientific notation and normalized will be represented as:

Significant is the number of significant digits which do not include zeroes, and *base* represents the base system used — which is decimal(10) here. *Exponent* represents the number of places the radix point needs to be moved left or right to be represented correctly.

Now, there are two ways to display numbers in floating point arithmetic: single precision and double precision. Single precision uses 32 bits, and double precision uses 64 bits for floating-point arithmetic.

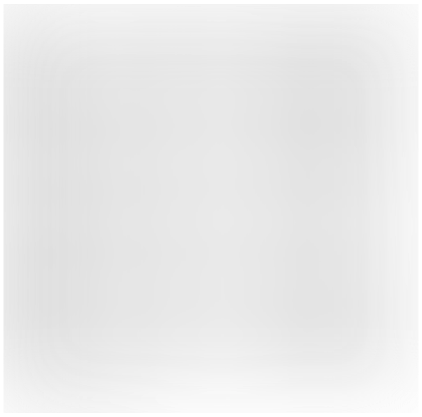
Unlike many other programming languages, JavaScript does not define different types of numeric data types and always stores numbers as double precision floating point numbers, following the international IEEE 754 standard.

This format stores numbers in 64 bits, where the number (the fraction) is stored in bits 0 to 51, the exponent in bits 52 to 62, and the sign in bit 63.

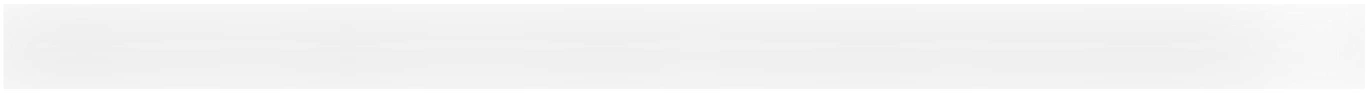
IEEE754 Double Precision Standard

Let’s represent 0.1 in 64 bit following the IEEE754 standard.

The first step is converting `(0.1)base 10` to its binary equivalent `(base 2).`
To do so, we will start by multiplying `0.1` by `2` and will separate out the digit before the decimal to get the binary equivalent.

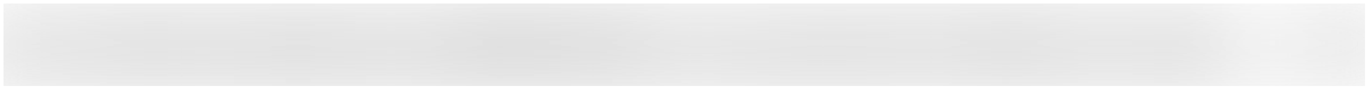


On repeating this for 64 bits, we are going to arrange them in ascending order to get our mantissa, which we are going to round off to 52 bits as per the double precision standard.



Mantissa

Representing it in scientific form and rounding off to the first 52 bits will yield:



The mantissa part is ready. Now for the exponent we shall use the below calculation:

Here, `11` represents the number of bits we are going to use for 64 bit representation of the exponent, and `-4` represents the exponent from the scientific notation.

The final representation of the number 0.1 is :

Similarly, 0.2 would be represented as:

Adding the two after making the exponents same for both would give us:

When represented in floating point, this becomes:

This is represented by `0.1 + 0.2`.

That is precisely the reason behind getting `0.1 + 0.2 = 0.30000000000000004`.

Programming JavaScript Python Binary 100daysofcode