LEARN PYTHON PROGRAMMING

Python's modulo operator and floor division

Phil Best
4 min read · Mar 26

In addition to the common operators for addition, subtraction, multiplication, and division, the Python standard library includes some arithmetic operators you may be less familiar with. In this post I'm going to talk about the modulo (`%`) and floor division (`//`) operators, and some common pitfalls associated with them.

What do modulo and floor division do?

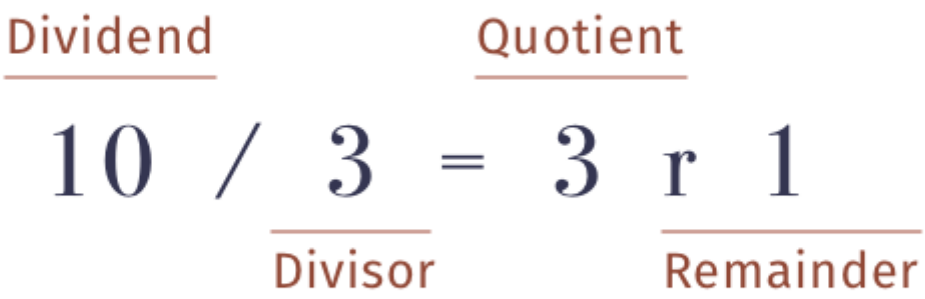
Modulo

The modulo operator is used to carry out Euclidean division. You almost certainly learnt about Euclidean division in school, even if you didn't know what it was called.

When performing Euclidean division, you start with a **dividend** (a number you want to divide), and a **divisor** (the number you want to divide the dividend by). Let's say `10` and `3` .

In what we might think of as "standard division" the result of `10 / 3` is `3.333` recurring, or `3` and a third. In Euclidean division, we don't

whole number result of the division (in this case `3`), which is called the **quotient**. The amount left over after the division is called the **remainder**, which in this case is `1`.



To give you another example, `11 / 3` gives us a quotient of `3` and a remainder of `2`.

In Python, the modulo operator simply yields the remainder:

```
>>> 10 % 3
1
>>> 11 % 3
2
```

Floor division

Floor division is also used to carry out Euclidean division, but unlike the modulo operator, floor division yields the **quotient**, not the remainder.

Let's look at a couple of examples:

```
>>> 10 // 3
3
>>> 9 // 2
4
```

We can create a small function to print out the full result of Euclidean division like so:

```
def euclidean_division(x, y):
    quotient = x // y
    remainder = x % y
    print(f"{quotient} remainder {remainder}")

euclidean_division(10, 3)  # 3 remainder 1
euclidean_division(11, 3)  # 3 remainder 2
```

There is a problem, however. What happens when we call our function with negative numbers?

```
>>> euclidean_divison(10, -3)
-4 remainder -2
```

That doesn't look right. `-3` multiplied by `-4` equals `12`, which is higher than `10`. The result we should have gotten was `-3` remainder `1`. So what's going on?

It's floor division's fault

The picture I painted for you earlier about Euclidean division is not entirely accurate. For positive integers, floor division and modulo

when it comes to negative numbers.

The reason for this is to do with rounding. If we look at the Python documentation for arithmetic operators we find a clue about what is going on:

> [T]he result [of floor division] is that of mathematical division with the 'floor' function applied to the result.

Okay, so let's look at the documentation for the `floor` function.

> Return the floor of x, the largest integer less than or equal to x.

And there's the problem. `-4` is considered less than `-3.333` recurring. It's further left on the number line. This means that floor division will always round away from zero for negative numbers, but towards zero for positive numbers.

```
>>> 10 // 3
3    # 3.333 rounds towards zero
>>> 10 // -3
-4    # -3.333 rounds away from zero
>>> 9 // 2
4    # 4.5 rounds towards zero
>>> 9 // -2
-5    # -4.5 rounds away from zero
```

So what's the problem with modulo?

We've figured out why we're getting unexpected results from floor division and negative numbers, but our remainder was also not what we expected. So why is that?

operators, which you may have read if you looked a little further in the documentation. Modulo and floor division are connected by the following identity: `x = (x // y) * y + (x % y)`.

This means that the result of floor division has a direct impact on the result of a modulo operation. We can rearrange the above like so: `(x % y) = x - (x // y) * y`.

Let's try one of our earlier examples, `10 % -3`:

```
10 % -3 = 10 - (-3) * (10 // -3)
10 % -3 = 10 - (-3) * (-4) # Remember, here floor division rounds aw
10 % -3 = 10 - 12
10 % -3 = -2
```

And we get the exact same result we got before: `10 % -3 = -2`.

A few final notes

In this final section, I just want to mention a few other interesting things about modulo and floor division. Before I go into that, however, I think it's important to note that the implementation of floor division and modulo in Python isn't a bug. However, I do think it's important to understand when and how implementations like this differ from our intuitive understanding, because then we can use this implementation as a tool, and it stops being a source of perhaps frustrating bugs.

Type conversion

One of the interesting things about floor division in Python is that it doesn't necessarily produce a floating point number. In fact, in all of

to standard division in Python, which always yields a float.

The modulo operator works the same way, yielding an integer if both operands were integers.

If either operand is a float, both modulo and floor division will yield a floating point number.

Using floats

While the examples in this post have all been integers, it's perfectly possible to use floating point numbers for either operand, and they work exactly the same, they simply yield a floating point number.

An example from the documentation is `3.14 % 0.7`, which will yield `0.34`.

Complex numbers

Complex numbers are numbers with a real and imaginary part. They're only worth noting here because neither modulo nor floor division can accept a complex number as an operand. Trying to use a complex number for floor division or modulo operations will raise a `TypeError`.

Recap

- For positive numbers floor division and modulo work like Euclidean division, yielding the quotient and remainder respectively.
- For negative numbers, both operations will yield slightly unexpected results.
- Floor division always rounds away from zero for negative numbers, so `-3.5` will round to `-4`, but towards zero for positive numbers, so `3.5` will round to `3`.

`x = (x // y) * y + (x % y)`, which is why modulo *also* yields unexpected results for negative numbers, not just floor division.

I hope you learnt something new, and if you're looking to upgrade your Python skills even further, check out our Complete Python Course.

Phil Best
I'm a freelance developer, mostly working in web development. I also create course content for Teclado!

Read More

Learn Python Programming

Introduction to Object-Oriented Programming in Python →

Dictionary Merge and Update Operators in Python 3.9 →

Working with Python virtual environments: the complete guide →

→ See all 144 posts

LEARN PYTHON PROGRAMMING

Rounding in Python

Take a deep dive into Python's rounding functions! Learn about floor, ceil, trunc, and round, and avoid common rounding errors in Python.

Phil Best
3 min read · Mar 31

Share this