Overview

This lesson introduces `approx()` from the `pytest` testing library. You will learn why decimal numbers can behave unexpectedly in programs and how `approx()` helps you write reliable tests when working with decimal values.

Important Information

Computers do not store decimal numbers the same way humans write them. Inside a computer, numbers are stored using binary (base 2). Many simple decimals, such as `0.1` or `0.2`, cannot be represented exactly in binary. Instead, the computer stores the closest possible approximation.

Because of this, math that looks simple can produce surprising results. For example, adding `0.1` and `0.2` does not result in exactly `0.3`, but instead `0.30000000000000004`. This is not a Python bug, it is a limitation of how numbers are stored in hardware.

When writing tests, this becomes a problem. A direct equality check (`==`) expects values to match exactly. If two decimal values are only slightly different due to rounding, a test can fail even though the calculation is logically correct.

The `approx()` function from `pytest` solves this by checking whether two values are *close enough* to each other instead of exactly equal. It allows for a very small margin of error, which is appropriate when testing decimal math.

When `approx()` Is Not Needed

If you are comparing integers or decimals that are known to be exact, you can safely use `==`.

```
def test_addition():
    result = 2 + 3
    assert result == 5
```

This test is safe because integers are stored exactly in binary.

When `approx()` Is Needed

When calculations involve decimals, especially results of division or repeated arithmetic, exact equality is risky.

```
def test_decimal_math():
    result = 0.1 + 0.2
    assert result == 0.3  # This test will fail
```

Even though this looks correct, the test fails because `result` is actually something like `0.30000000000000004`.

Using `approx()` fixes this problem.

```
from pytest import approx

def test_decimal_math():
    result = 0.1 + 0.2
    assert result == approx(0.3)
```

Here, `approx(0.3)` tells `pytest` to accept values that are extremely close to `0.3`, which makes the test both accurate and realistic.

Testing is about checking correctness, not punishing tiny rounding errors that humans do not care about. When decimals are involved, `approx()` makes your tests match how computers actually work, rather than how numbers look on paper.