TDD vs BDD

Explain now tdd and bdd testing differ. Explain in simple terms. Provide a pytest example of both

Test-Driven Development (TDD) and Behavior-Driven Development (BDD) are both software testing approaches, but they differ in their focus and the way tests are written.

TDD (Test-Driven Development):
TDD is a development approach where developers write tests before writing the actual code. The process typically follows these steps:

1. Write a small test case that defines the desired behavior of a specific piece of code.
2. Run the test, which should fail initially because the code hasn't been implemented yet.
3. Write the code to make the test pass.
4. Refactor the code if needed, while ensuring the test still passes.

Here's a simple Python pytest example for TDD:

```
# Test case for adding two numbers
def test_addition():
    result = add(2, 3)
    assert result == 5

# Code to make the test pass
def add(a, b):
    return a + b
```

BDD (Behavior-Driven Development):
BDD is a testing approach that focuses on the behavior and interactions of the system from the user's perspective. It often uses natural language specifications and is more oriented towards collaboration between developers, testers, and non-technical stakeholders. In BDD, you write tests in a human-readable format, often using "Given-When-Then" scenarios to describe system behavior.

Here's a simple Python pytest example for BDD using the `pytest-bdd` library:

```
# Feature: Calculator
# Scenario: Add two numbers
    Given a calculator
    When I add 2 and 3
    Then the result should be 5
```

And the corresponding step definitions:

```
from pytest_bdd import given, when, then, parsers

@given("a calculator")
def calculator():
    return Calculator()

@when(parsers.parse("I add {a:d} and {b:d}"))
def add_numbers(calculator, a, b):
    calculator.result = calculator.add(a, b)

@then("the result should be {result:d}")
def check_result(calculator, result):
    assert calculator.result == result
```

In summary, TDD focuses on writing tests for individual code units to ensure they work correctly, while BDD emphasizes the behavior and interactions of the system as a whole from a user's perspective, often using more natural language specifications. Both approaches have their place in software development, and the choice between them depends on your project's requirements and team dynamics.