Solutions

Marking: **M1** a correct method, **A1** the correct answer, **B1** an independent mark.

Each answer shows the steps, then the **result**.

2.1

arrange: `acc = BankAccount("Ana")`; act: `acc.deposit(50)`

assert: `assert acc.balance == 50`

2.2

branch coverage: **2** tests, e.g. `(8, True)` and `(30, False)`

path coverage: $2^2 = \mathbf{4}$

2.3

long method: extract smaller functions

duplicated code: one shared function called from each place

magic number: a named constant, e.g. `FINE_PER_DAY`

2.4

the result depends on the day the test runs, so a test can pass today and fail tomorrow

fix: `def fine(due, today):` and pass the date in; the program passes `date.today()`

3.1

(a) e.g.

```
def test_normal():
    l = Loan(14); l.renew(7); assert l.days == 21
def test_exactly_28():
    l = Loan(14); l.renew(14); assert l.days == 28
def test_past_28_raises():
    l = Loan(14)
    with pytest.raises(ValueError):
        l.renew(15)
def test_unchanged_after_error():
    l = Loan(14)
    with pytest.raises(ValueError):
        l.renew(15)
    assert l.days == 14
```

normal case; the boundary 28; `pytest.raises` past 28; state checked after the error

(b) yes: the normal tests take the "no raise" branch and the error tests take the "raise" branch

(c) the exactly-28 test: with `>=`, a total of 28 now raises, so the boundary test fails and the mutation is caught

3.2

(a) "3 days late" changes meaning every day, because `today` is the real date

(b) `def fine(due, today): return max((today - due).days, 0) * 0.20`: `today` as a parameter; the same rule inside

(c) `assert fine(date(2026, 11, 1), date(2026, 11, 4)) == pytest.approx(0.60)`: fixed dates 3 days apart; `approx` for the float

3.3

(a) one function or class per job: `validate(order)`, `price(order)`, `reserve_stock(order)`, `send_confirmation(order)`; `process_order` just calls them in order; the e-mail sender is passed in, so tests can use a fake

(b) pricing tests, on the boundaries of each discount and tax rule, because that is where the bugs are

(c) coverage shows which lines *ran*, not that the results were checked; the 15% not run may be the risky branches, and branch coverage may be much lower; so it is not enough: judge tests by the bugs they would catch, e.g. with mutation testing on the pricing code

exercises.lol exercises.com.cn