Solutions

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

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

2.1

`Copy`: `_out` is always exactly `True` or `False`

`BankAccount`: `_balance` is never negative

2.2

`@property def celsius(self): return self._c`

`@celsius.setter def celsius(self, v):` with `if v < -273.15: raise ValueError(...)`

then `self._c = v`; `__init__` sets `self.celsius = c` so the check runs there too

2.3

`get_items` returns the cart's own list

the caller's `.clear()` empties the cart itself

fix: `return tuple(self._items)` or `list(self._items)`

2.4

the balance is changed before the check, so after the error it stays negative

fix: `if amount > self._balance: raise ValueError(...)` first, then subtract

3.1

(a)

```
class Loan:
    MAX = 28
    def __init__(self, days):
        if not 1 <= days <= Loan.MAX:
            raise ValueError("days out of range")
        self._days = days

    def renew(self, extra):
        if extra < 1 or self._days + extra > Loan.MAX:
            raise ValueError("renewal too long")
        self._days += extra

    @property
    def days(self):
        return self._days
```

checks in `__init__`; checks in `renew` on the new total; private field; a read-only `days` property

(b) $14 + 20 = 34 > 28$, so `ValueError`; `days` is still 14

(c) so a failed call leaves the object as it was; no undo is needed

3.2

(a) 30; 1

(b) `obj.balance = x` now calls the setter, so the same caller code runs the new check

(c) name mangling: Python renames it `_ClassName__balance`, which stops accidental use from outside or from a subclass

3.3

(a)

```
@dataclass(frozen=True)
class Money:
    pence: int
    currency: str
    def add(self, other):
        if other.currency != self.currency:
            raise ValueError("currency mismatch")
        return Money(self.pence + other.pence, self.currency)
```

`frozen=True` with the two fields; the currency check; returns a new object

(b) with aliasing, a change through one order would change the price of all of them; an immutable value can be shared safely

(c) no: a million small objects take well under a second, and most are freed at once

(d) `ISBN` and `Money` are values: immutable; a `Copy` changes state (in or out), so it stays mutable behind its wall

exercises.lol exercises.com.cn