IQM3010 A2 Teacher marking exercise

Computer Science Assessment 2 Paper 2 Written Exam Sample 2026 2026 S

The selected synthetic extracts below form an unmarked exercise. Apply the supplied assessment criteria, record evidence and propose marks or bands before consulting the separate teacher reference.

Internal synthetic response pack for IQM3010 Computer Science Assessment 2, Paper 2, Sample 2026 (set ID 2026-S), current source version 2.0. Source: the combined Candidate Question And Answer Paper v2 and Staff Mark Scheme and Assessment Objective 2 Marking Map v2. The question paper is closed-book and prohibits computer execution; code below is written Python 3 only. This internal material is not an authentic examination, candidate submission, marker action, approval or grade decision. The exact source paper and mark scheme remain controlled.

Selected short response extracts are synthetic and unmarked. They reproduce source contexts for staff practice only; no actual candidate is identified.

Unmarked selections for source point practice

Use each full extract with its source prompt. Identify which numbered source features are demonstrated, note exact evidence and propose an item mark. Do not infer execution or actual candidate performance.

Extract A — S2 A4

Source question context: A program repeatedly creates Score objects. Each object needs its own initially empty list of attempts. Where should this list be created, and why?

Fictional response excerpt: Put self.attempts = [] in __init__, because that runs when each object is made and gives it a list.

Extract B — S2 B2

Source question context: This SeatPool should accept a positive integer request when enough seats remain, subtract that request once and return True. Otherwise it should return False without changing the state. Assume capacity is a non-negative integer and requests are integers. (a) Identify and correct three independent errors in the book method; explain each correction. [6] (b) A fresh pool has capacity 5. Give expected return values and remaining seats after the sequential requests 0, 5 and 1. [3] (c) Give one further test that checks rejection of negative requests. [1]

```
class SeatPool:
    def __init__(self, capacity):
        self.remaining = capacity
    def book(self, count):
        if count < 0 or count >= self.remaining:
            return False
        self.remaining = count
        return True
```

Fictional response excerpt: The method should use count <= 0, compare count > remaining so a request equal to the remaining seats is accepted, and set remaining to remaining minus count. My corrected code and trace are below. Starting at 5: 0 returns False and leaves 5; 5 returns True and leaves 0; 1 returns False and leaves 0. A fresh pool tested with -1 should return False and remain at 5.

```
class SeatPool:
    def __init__(self, capacity):
        self.remaining = capacity
    def book(self, count):
        if count <= 0 or count > self.remaining:
            return False
        self.remaining = self.remaining - count
        return True
```

Extract C — S3 C2

Source question context: Use the KitPool interface described in C1. You may assume KitPool is implemented correctly even if you did not finish C1. Write a Python 3 subclass LimitedPool. Its constructor takes code, capacity and limit, calls the parent constructor, then stores limit. Assume limit is a positive integer. Override checkout(self, count): reject counts above limit by returning False without changing any state; otherwise delegate to the parent checkout and return its result. Inherit return_kits and on_loan unchanged. [10] Write code that creates a LimitedPool with code "L1", capacity 8 and limit 3, then stores the results of checking out 4 kits and checking out 3 kits in first and second. [3] State first, second and available after these calls. [2]

Fictional response excerpt: LimitedPool checks the limit first and otherwise asks the parent to process the checkout. I did not set up all of the parent fields.

```
class LimitedPool(KitPool):
    def __init__(self, code, capacity, limit):
        self.limit = limit
    def checkout(self, count):
        if count > self.limit:
            return False
        return super().checkout(count)

limited = LimitedPool("L1", 8, 3)
first = limited.checkout(4)
second = limited.checkout(3)
```

I expect first = False; second = True; available = 5.

Extract D — S1 C3

Source question context: A list pools contains KitPool and LimitedPool objects with unique codes. Use the interfaces in C1–C2; assume they are implemented correctly. Write a function request_kits(pools, code, count) that searches the list in order for the given code. If found, it calls that object’s checkout(count) exactly once and returns the resulting Boolean. If no code matches, it returns False. It must work for an empty list, must not change the list and must not directly change object attributes. [10] Then write a function total_on_loan(pools) that returns the sum of on_loan() over all objects, returning 0 for an empty list. [5] No input, print, external libraries or separate type checks are required.

Fictional response excerpt: The search returns the matching object’s checkout result immediately, exactly once; if no code matches, the loop ends and returns False. The second function accumulates each object’s on_loan() value and returns zero for an empty list. Neither function changes the list or directly mutates attributes.

```
def request_kits(pools, code, count):
    for pool in pools:
        if pool.code == code:
            return pool.checkout(count)
    return False

def total_on_loan(pools):
    total = 0
    for pool in pools:
        total += pool.on_loan()
    return total
```