IQM3010 A2 Teacher marking reference

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

This reference supports discussion of the selected exercise extracts. Marks and bands are proposed documentary judgements, subject to human subject review.

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.

Reference decisions are proposed feature-point allocations against the current Staff Mark Scheme v2 and remain subject to human review.

Extract specific source point reference

Use the current source criteria below. Credit the quoted demonstrated features only; do not infer execution. The exercise extracts are written synthetic material.

Extract A — S2 A4 — source prompt: A program repeatedly creates Score objects. Each object needs its own initially empty list of attempts. Where should this list be created, and why?

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

Point	Published source feature (one mark)	Extract evidence and proposed decision
1	Create self.attempts = [] within __init__.	Award 1. Evidence: Put self.attempts = [] in __init__, because that runs when each object is made and gives it a list..
2	1 mark: This creates a separate list for each object, avoiding an unintentionally shared mutable class list.	Award 1. Evidence: Put self.attempts = [] in __init__, because that runs when each object is made and gives it a list..

Proposed extract mark: 2/2; earned source points: 1, 2.

Extract B — S2 B2 — source prompt: 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
```

Exact extract response: 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
```

Point	Published source feature (one mark)	Extract evidence and proposed decision
1	1 mark: Change count < 0 to count <= 0.	Award 1. Evidence: if count <= 0 or count > self.remaining:
2	1 mark: Explain this rejects zero, which is not positive.	Award 1. Evidence: The method should use count <= 0
3	1 mark: Change count >= self.remaining to count > self.remaining.	Award 1. Evidence: if count <= 0 or count > self.remaining:
4	1 mark: Explain equality must be accepted when exactly enough seats remain.	Award 1. Evidence: compare count > remaining so a request equal to the remaining seats is accepted
5	1 mark: Change self.remaining = count to self.remaining = self.remaining - count (or -= count).	Award 1. Evidence: self.remaining = self.remaining - count
6	1 mark: Explain this subtracts booked seats from existing capacity rather than replacing capacity with the request.	Award 1. Evidence: set remaining to remaining minus count
7	1 mark: Request 0: False, remaining 5.	Award 1. Evidence: Starting at 5: 0 returns False and leaves 5;
8	1 mark: Request 5: True, remaining 0.	Award 1. Evidence: 5 returns True and leaves 0;
9	1 mark: Request 1: False, remaining 0.	Award 1. Evidence: 1 returns False and leaves 0.
10	1 mark: For example, fresh capacity 5; request -1: False with remaining 5.	Award 1. Evidence: A fresh pool tested with -1 should return False and remain at 5.

Proposed extract mark: 10/10; earned source points: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10.

Extract C — S3 C2 — source prompt: 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]

Exact extract response: 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.

Point	Published source feature (one mark)	Extract evidence and proposed decision
1	1 mark: Declares class LimitedPool(KitPool).	Award 1. Evidence: class LimitedPool(KitPool):
2	1 mark: Constructor signature accepts code, capacity and limit with self.	Award 1. Evidence: def __init__(self, code, capacity, limit):
3	1 mark: Calls parent __init__ on current object.	Award 0. Evidence: This selected extract does not demonstrate this source feature.
4	1 mark: Passes code and capacity correctly.	Award 0. Evidence: This selected extract does not demonstrate this source feature.
5	1 mark: Stores self.limit = limit.	Award 1. Evidence: self.limit = limit
6	1 mark: Overrides checkout with self and count.	Award 1. Evidence: def checkout(self, count):
7	1 mark: Tests count > self.limit.	Award 1. Evidence: if count > self.limit:
8	1 mark: Returns False without mutation when above limit.	Award 1. Evidence: if count > self.limit: return False
9	1 mark: Otherwise calls parent checkout with count, avoiding recursive self.checkout.	Award 1. Evidence: return super().checkout(count)

Point	Published source feature (one mark)	Extract evidence and proposed decision
10	1 mark: Returns result of parent call.	Award 1. Evidence: return super().checkout(count)
11	1 mark: Creates limited = LimitedPool("L1", 8, 3).	Award 1. Evidence: limited = LimitedPool("L1", 8, 3)
12	1 mark: first = limited.checkout(4).	Award 1. Evidence: first = limited.checkout(4)
13	1 mark: second = limited.checkout(3).	Award 1. Evidence: second = limited.checkout(3)
14	1 mark: first is False and second is True (both required).	Award 1. Evidence: I expect first = False; second = True; available = 5.
15	1 mark: limited.available is 5.	Award 1. Evidence: I expect first = False; second = True; available = 5.

Proposed extract mark: 13/15; earned source points: 1, 2, 5–15 except 3–4. The written False/True/available trace earns points 14–15 independently of the incomplete parent-constructor call.

Extract D — S1 C3 — source prompt: 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.

Exact extract response: 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
```

Point	Published source feature (one mark)	Extract evidence and proposed decision
1	1 mark: Defines request_kits with three parameters.	Award 1. Evidence: def request_kits(pools, code, count):
2	1 mark: Iterates over supplied list.	Award 1. Evidence: for pool in pools:

Point	Published source feature (one mark)	Extract evidence and proposed decision
3	1 mark: Compares each object's code to requested code.	Award 1. Evidence: if pool.code == code:
4	1 mark: Calls checkout on matching object.	Award 1. Evidence: return pool.checkout(count)
5	1 mark: Passes count unchanged.	Award 1. Evidence: return pool.checkout(count)
6	1 mark: Calls checkout exactly once for match.	Award 1. Evidence: if pool.code == code: return pool.checkout(count)
7	1 mark: Returns checkout Boolean, including False on rejection.	Award 1. Evidence: return pool.checkout(count)
8	1 mark: Does not return False prematurely after a non-match.	Award 1. Evidence: return pool.checkout(count) return False
9	1 mark: Returns False after exhausting list, including empty list.	Award 1. Evidence: return False
10	1 mark: Preserves list and uses method interface without direct state mutation.	Award 1. Evidence: if pool.code == code: return pool.checkout(count) return False
11	1 mark: Defines total_on_loan with pools parameter.	Award 1. Evidence: def total_on_loan(pools):
12	1 mark: Initialises numeric total to 0.	Award 1. Evidence: total = 0
13	1 mark: Visits all objects and calls each on_loan().	Award 1. Evidence: for pool in pools: total += pool.on_loan()
14	1 mark: Adds returned values to total once per object.	Award 1. Evidence: total += pool.on_loan()
15	1 mark: Returns sum after iteration, including 0 for empty list.	Award 1. Evidence: return total

Proposed extract mark: 15/15; earned source points: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15.