

International Baccalaureate · IB Diploma · Computer Science · HL

Source-supported focus teaching; remaining qualification requirements are held

; Original teaching and practice · exercises.lol

Contents

1. Computer fundamentals · supported focus	2
2. Networks · supported focus	5
3. Databases · supported focus	8
4. Machine learning · supported focus	11
5. Computational thinking · supported focus	14
7. Object-oriented programming · supported focus	17
8. Abstract data types (HL only) · supported focus	20
9. Case study · supported focus	23

Computer fundamentals · supported focus

International Baccalaureate · IB Diploma · Computer Science · HL

Scope and prerequisites

Supported HL focus. First assessment 2027 target; official PDF returns 403; older acquired brief is final assessment 2026. Remaining guide, assessment and practical requirements retain their recorded holds.

Prerequisites: read the stated quantities and units, use arithmetic and the model conditions below. Each lesson develops its own method before independent transfer.

These are original or explicitly fictional teaching examples, not actual measurements or completed assessed learner investigations.

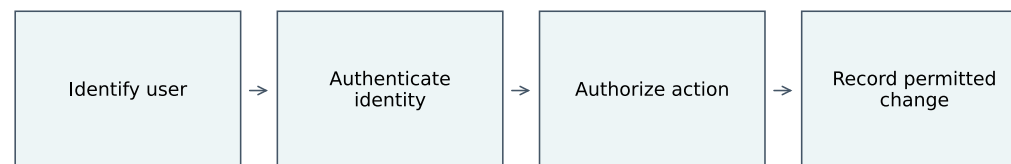
Computing systems, networks and requirements

■ What would explain this observation?

- A school booking system can calculate correctly and still fail if users lose access or records are disclosed. Success includes the system context.
- Start with a prediction. State the quantities or features you would compare, then decide what evidence could distinguish two explanations.

■ Build the model

- A computer system combines hardware, software, data and people. A network enables communication using protocols. Requirements should distinguish function from constraints such as availability, security and accessibility.
- **authentication** : Checking an asserted identity; **authorization** : Determining permitted actions.



■ Choose evidence that can test it

- Separate authentication from authorization. Authentication checks identity; authorization determines permitted actions. A threat model connects a valuable asset, a possible attack and an appropriate control.
- Use a fictitious school dataset to define users and permissions. Draw data flows, compare validation and verification, and specify tests for normal, boundary and invalid input. Do not use real credentials or student records in exercises.

■ Work from known quantities

- State the known values and their units. Choose the relation because its assumptions fit this case, then rearrange before substitution.
- Known: a file contains 12 megabytes, where this example defines one megabyte as one million bytes. $\text{Bits} = \text{bytes} \times 8 = 12 \times 1,000,000 \times 8 = 96,000,000$ bits. At 8,000,000 bits per second, ideal time = size/rate = 12 s, excluding overhead.

Example:

A 40 million bit file transfers at 5 million bits per second. Find ideal time. Use the same sequence: known quantities → model → relation → substitution → unit and interpretation.

■ Check the conclusion and its limits

- Bandwidth is not actual end-to-end throughput. Encryption does not by itself ensure correct authorization or remove every security risk.
- Return to the original observation. Explain what the result supports, which conditions it assumes, and one way to test a competing explanation.

Warn:

Encryption guarantees that every user has appropriate access permissions. This claim is false: Bandwidth is not actual end-to-end throughput. Encryption does not by itself ensure correct authorization or remove every security risk.

Key:

Computing systems, networks and requirements: Separate authentication from authorization. Authentication checks identity; authorization determines permitted actions. A threat model connects a valuable asset, a possible attack and an appropriate control.

■ Runnable trace and boundary

```
size_MB, rate_Mbps = 20, 10
bits = size_MB * 1_000_000 * 8
if rate_Mbps <= 0:
    raise ValueError("Rate must be positive")
seconds = bits / (rate_Mbps * 1_000_000)
print(seconds)
print(0 / (rate_Mbps * 1_000_000))
```

Expected output:

```
16.0
0.0
```

Decimal MB means one million bytes; multiply by eight before dividing by bits per second. Zero data has zero ideal transmission duration. A nonpositive rate is rejected; actual overhead is outside this arithmetic model.

Networks · supported focus

International Baccalaureate · IB Diploma · Computer Science · HL

Scope and prerequisites

Supported HL focus. First assessment 2027 target; official PDF returns 403; older acquired brief is final assessment 2026. Remaining guide, assessment and practical requirements retain their recorded holds.

Prerequisites: read the stated quantities and units, use arithmetic and the model conditions below. Each lesson develops its own method before independent transfer.

These are original or explicitly fictional teaching examples, not actual measurements or completed assessed learner investigations.

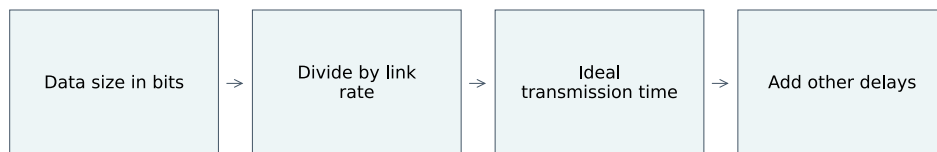
Networks: latency, throughput and layered delivery

■ What would explain this observation?

- A small message can arrive late on a high-bandwidth link. Capacity and delay measure different properties.
- Start with a prediction. State the quantities or features you would compare, then decide what evidence could distinguish two explanations.

■ Build the model

- Packets carry addressed data through a network. A layered model separates responsibilities such as application meaning, transport delivery and network routing. Bandwidth describes capacity; **throughput** is the achieved data rate; **latency** is delay.
- throughput: Achieved rate of useful data transfer; latency: Delay experienced in communication.



■ Choose evidence that can test it

- Transmission time depends on data size and rate. Total delay may also include propagation, processing and queueing. Encryption protects content under its assumptions but does not remove congestion or every metadata exposure.
- Trace a message route using a documented local model. Record payload size, units and measured time. Use school-approved networks and synthetic messages; do not scan or intercept another user traffic.

■ Work from known quantities

- State the known values and their units. Choose the relation because its assumptions fit this case, then rearrange before substitution.
- Known: an 8 megabit file crosses a 2 megabit/s link. Ideal transmission time = $\text{size}/\text{rate} = 8/2 = 4$ s. Protocol overhead, other users and latency can make observed completion slower.

Example:

A 12 megabit file crosses a 3 megabit/s link. Calculate ideal transmission time. Use the same sequence: known quantities → model → relation → substitution → unit and interpretation.

■ Check the conclusion and its limits

- A megabyte is eight megabits before considering overhead. A faster rated link does not guarantee low latency or secure endpoints.
- Return to the original observation. Explain what the result supports, which conditions it assumes, and one way to test a competing explanation.

Warn:

A high-bandwidth connection guarantees zero latency. This claim is false: A megabyte is eight megabits before considering overhead. A faster rated link does not guarantee low latency or secure endpoints.

Key:

Networks: latency, throughput and layered delivery: Transmission time depends on data size and rate. Total delay may also include propagation, processing and queueing. Encryption protects content under its assumptions but does not remove congestion or every metadata exposure.

■ Runnable trace and boundary

```
size_Mb, rate_Mbps, propagation_s = 12, 3, 0.10
for rate in [rate_Mbps, 2 * rate_Mbps]:
    transmission_s = size_Mb / rate
    print(round(transmission_s + propagation_s, 2))
```

Expected output:

```
4.1
2.1
```

Measure from transmission start to last-bit arrival. Serial transmission time and propagation delay add under this ideal model. Increasing bandwidth changes only the first term; store-and-forward hops and queueing would add further terms.

Databases · supported focus

International Baccalaureate · IB Diploma · Computer Science · HL

Scope and prerequisites

Supported HL focus. First assessment 2027 target; official PDF returns 403; older acquired brief is final assessment 2026. Remaining guide, assessment and practical requirements retain their recorded holds.

Prerequisites: read the stated quantities and units, use arithmetic and the model conditions below. Each lesson develops its own method before independent transfer.

These are original or explicitly fictional teaching examples, not actual measurements or completed assessed learner investigations.

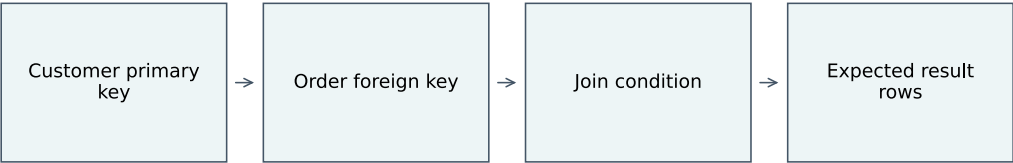
Relational databases: a key and a relationship

■ What would explain this observation?

- Duplicating a user address in every order can produce conflicting records when the address changes.
- Start with a prediction. State the quantities or features you would compare, then decide what evidence could distinguish two explanations.

■ Build the model

- A relational table contains rows and attributes. A **primary key** identifies a row; a **foreign key** links to a referenced key in another table. Normalization can reduce avoidable duplication and update anomalies while preserving meaningful relationships.
- primary key: Attribute set uniquely identifying a table row; foreign key: Attribute set referencing a key in another table.



■ Choose evidence that can test it

- A join combines rows according to a specified condition. The result count depends on relationship cardinality and filters. A foreign key constraint enforces a relationship rule; it does not automatically encrypt personal data.
 - Design a small synthetic user-order dataset with no real personal information. Declare keys, test duplicate and missing-reference inserts, then check a query result against a hand-worked expected table.
-

■ Work from known quantities

- State the known values and their units. Choose the relation because its assumptions fit this case, then rearrange before substitution.
- Known: four customers have respectively 2, 0, 3 and 1 orders. An inner join of customer to order returns $2+0+3+1 = 6$ rows. A left join also retains the customer with no order as one null-matched row, giving 7 here.

Example:

Three customers have 1, 3 and 2 orders. How many rows does the customer-order inner join return? Use the same sequence: known quantities $\rightarrow$ model $\rightarrow$ relation $\rightarrow$ substitution $\rightarrow$ unit and interpretation.

■ Check the conclusion and its limits

- A primary key need not be a person name. A join is not a simple concatenation of tables. Use parameterized queries for untrusted input rather than building SQL from raw strings.
- Return to the original observation. Explain what the result supports, which conditions it assumes, and one way to test a competing explanation.

Warn:

A foreign key automatically encrypts every row. This claim is false: A primary key need not be a person name. A join is not a simple concatenation of tables. Use parameterized queries for untrusted input rather than building SQL from raw strings.

Key:

Relational databases: a key and a relationship: A join combines rows according to a specified condition. The result count depends on relationship cardinality and filters. A foreign key constraint enforces a relationship rule; it does not automatically encrypt personal data.

■ Runnable trace and boundary

```
import sqlite3
db = sqlite3.connect(":memory:")
db.executescript("""CREATE TABLE c(id INTEGER PRIMARY KEY);
CREATE TABLE o(id INTEGER PRIMARY KEY, cid INTEGER);
INSERT INTO c VALUES(1),(2),(3);
INSERT INTO o VALUES(10,1),(11,1),(12,3);""")
for kind in ["INNER", "LEFT"]:
    sql = f"SELECT COUNT(*),COUNT(o.id) FROM c {kind} JOIN o ON
    ↪ c.id=o.cid"
    print(kind, db.execute(sql).fetchone())
db.close()
```

Expected output:

```
INNER (3, 3)
LEFT (4, 3)
```

The left join retains customer 2 with null order fields. COUNT(*) includes that row, while COUNT(o.id) excludes null. Primary keys distinguish records; customer count is not order count. The interpolated join keyword comes from a fixed internal list, not user input.

Machine learning · supported focus

International Baccalaureate · IB Diploma · Computer Science · HL

Scope and prerequisites

Supported HL focus. First assessment 2027 target; official PDF returns 403; older acquired brief is final assessment 2026. Remaining guide, assessment and practical requirements retain their recorded holds.

Prerequisites: read the stated quantities and units, use arithmetic and the model conditions below. Each lesson develops its own method before independent transfer.

These are original or explicitly fictional teaching examples, not actual measurements or completed assessed learner investigations.

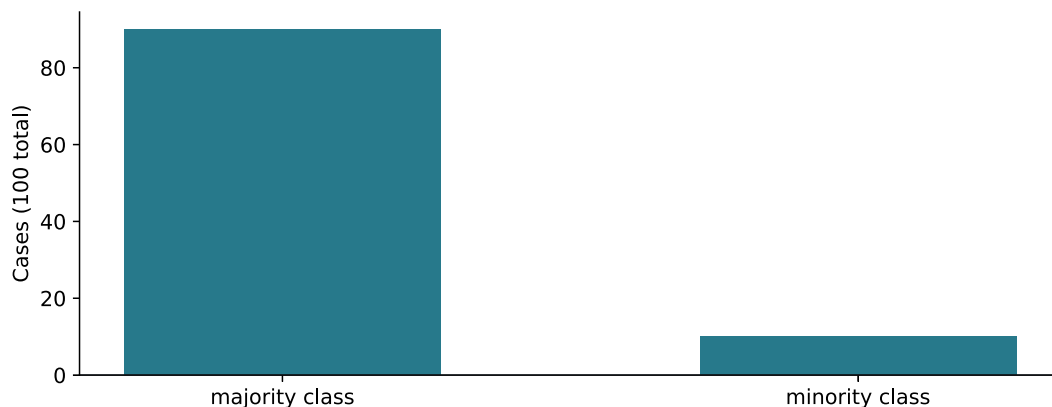
Data models and responsible machine learning

■ What would explain this observation?

- A model can score well by learning a shortcut in the training data. Evaluation must test the intended task on appropriate unseen examples.
- Start with a prediction. State the quantities or features you would compare, then decide what evidence could distinguish two explanations.

■ Build the model

- A relational database uses tables, keys and relationships to organize data. Machine learning estimates patterns from data. Training and evaluation data serve different roles.
- **primary key** : An attribute set uniquely identifying a record; **data leakage** : Use of information unavailable at the intended prediction time.



■ Choose evidence that can test it

- A primary key uniquely identifies a record; a foreign key relates records. In prediction, leakage can expose information unavailable at the real decision time. Accuracy alone may hide an unbalanced target distribution.
- Use fictional booking records to identify entities, attributes and relationships. For a learning exercise, use a public non-sensitive dataset, separate training and test data, and describe who may be affected by errors. The current 2027 CS objective scope awaits the acquired guide.

■ Work from known quantities

- State the known values and their units. Choose the relation because its assumptions fit this case, then rearrange before substitution.
- Known: a classifier makes 90 correct predictions among 100 cases. Accuracy = $\text{correct}/\text{total} \times 100 = 90\%$. If 90 cases belong to one class, always predicting that class gives the same accuracy and can still fail every minority-class case.

Example:

A model makes 72 correct predictions out of 80. Find accuracy percentage. Use the same sequence: known quantities → model → relation → substitution → unit and interpretation.

■ Check the conclusion and its limits

- High accuracy is not evidence of fairness or causal understanding. A database primary key is not simply whichever field looks important.
- Return to the original observation. Explain what the result supports, which conditions it assumes, and one way to test a competing explanation.

Warn:

A classifier with 90% accuracy must treat every class equally well. This claim is false: High accuracy is not evidence of fairness or causal understanding. A database primary key is not simply whichever field looks important.

Key:

Data models and responsible machine learning: A primary key uniquely identifies a record; a foreign key relates records. In prediction, leakage can expose information unavailable at the real decision time. Accuracy alone may hide an unbalanced target distribution.

■ Runnable trace and boundary

```
tp, fn, fp, tn = 4, 1, 10, 85
total = tp + fn + fp + tn
accuracy = (tp + tn) / total
recall = tp / (tp + fn) if tp + fn else None
print(accuracy, recall)
positives = 0
print(0 / positives if positives else "UNDEFINED")
```

Expected output:

```
0.89 0.8
UNDEFINED
```

Accuracy divides correct predictions by all cases; positive recall divides true positives by actual positives. With no positive cases, recall is undefined rather than zero. The dataset is fictional, with no trained model or personal data.

Computational thinking · supported focus

International Baccalaureate · IB Diploma · Computer Science · HL

Scope and prerequisites

Supported HL focus. First assessment 2027 target; official PDF returns 403; older acquired brief is final assessment 2026. Remaining guide, assessment and practical requirements retain their recorded holds.

Prerequisites: read the stated quantities and units, use arithmetic and the model conditions below. Each lesson develops its own method before independent transfer.

These are original or explicitly fictional teaching examples, not actual measurements or completed assessed learner investigations.

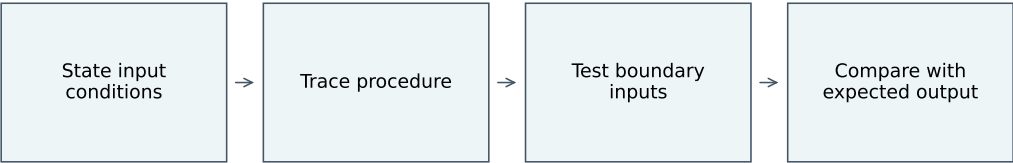
Algorithms, traces and correctness evidence

■ What would explain this observation?

- An **algorithm** can work for one example and fail at a boundary. Testing should be designed from the specification, not only the happy path.
- Start with a prediction. State the quantities or features you would compare, then decide what evidence could distinguish two explanations.

■ Build the model

- An algorithm is a finite, unambiguous procedure for a task. A trace records state changes. A loop invariant describes a property preserved by each iteration and helps justify correctness.
- algorithm: A finite procedure solving a stated task; **boundary test** : A test at a limit of the allowed input range.



■ Choose evidence that can test it

- State input conditions and expected outputs. Use boundary cases, empty collections where allowed, duplicates and invalid values. Distinguish a wrong algorithm from a wrong implementation or an incomplete requirement.
- Trace a search over a small fictional sorted list. State the indexing convention. For binary search, update bounds so the remaining interval shrinks and reject unsorted input unless sorting is part of the task.

■ Work from known quantities

- State the known values and their units. Choose the relation because its assumptions fit this case, then rearrange before substitution.
- Known: a linear search of an eight-item list can require eight comparisons when the sought item is last or absent. Doubling the list length doubles the worst-case comparison count under this model. Binary search reduces the interval by roughly half each step but requires a suitable ordered structure.

Example:

A linear search scans all 14 items without a match. How many item comparisons occur? Use the same sequence: known quantities → model → relation → substitution → unit and interpretation.

■ Check the conclusion and its limits

- A successful sample test does not prove correctness for all valid inputs. Do not import Cambridge-specific pseudocode syntax into an IB course without a course source.
- Return to the original observation. Explain what the result supports, which conditions it assumes, and one way to test a competing explanation.

Warn:

One successful sample test proves an algorithm correct for every valid input. This claim is false: A successful sample test does not prove correctness for all valid inputs. Do not import Cambridge-specific pseudocode syntax into an IB course without a course source.

Key:

Algorithms, traces and correctness evidence: State input conditions and expected outputs. Use boundary cases, empty collections where allowed, duplicates and invalid values. Distinguish a wrong algorithm from a wrong implementation or an incomplete requirement.

■ Runnable trace and boundary

```
def first_index(values, target):  
    for index, value in enumerate(values):  
        if value == target:  
            return index  
    return -1  
print(first_index([4, 7, 4], 4))  
print(first_index([4, 7, 4], 9))  
print(first_index([], 4))
```

Expected output:

```
0  
-1  
-1
```

The loop tests index 0 first, stops on a match, and returns -1 only after exhausting the list. Duplicates therefore return the first index. An empty list executes no loop body.

Object-oriented programming · supported focus

International Baccalaureate · IB Diploma · Computer Science · HL

Scope and prerequisites

Supported HL focus. First assessment 2027 target; official PDF returns 403; older acquired brief is final assessment 2026. Remaining guide, assessment and practical requirements retain their recorded holds.

Prerequisites: read the stated quantities and units, use arithmetic and the model conditions below. Each lesson develops its own method before independent transfer.

These are original or explicitly fictional teaching examples, not actual measurements or completed assessed learner investigations.

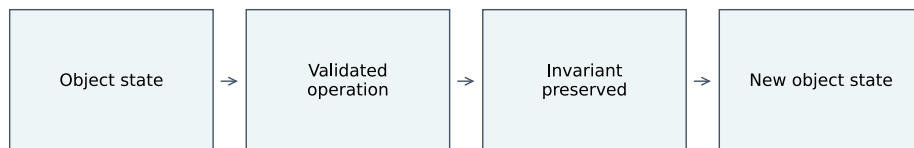
Objects: state, behaviour and an interface

■ What would explain this observation?

- Two bank-account model objects can respond to the same deposit operation while holding different balances.
- Start with a prediction. State the quantities or features you would compare, then decide what evidence could distinguish two explanations.

■ Build the model

- A class defines a type with state and behaviour. An object is an **instance**. **Encapsulation** controls access through an interface so operations can preserve invariants. Inheritance models an appropriate type relationship; composition models an object containing or using another object.
- encapsulation: Controlling access to state through an interface; instance: An individual object of a class.



■ Choose evidence that can test it

- A method call acts on a particular instance. State changes should satisfy preconditions and postconditions. Polymorphism lets code use a common interface with different implementations when the contract is respected.
- Implement a small synthetic account or inventory model. Test two independent instances, rejected invalid operations and boundary values. Keep the model away from real financial accounts and credentials.

■ Work from known quantities

- State the known values and their units. Choose the relation because its assumptions fit this case, then rearrange before substitution.
- Known: object A starts at 40 units and receives 15; object B starts at 20 and receives 5. Final balances are 55 and 25. Their total is 80, but neither individual object balance is 80.

Example:

An instance starts at 12 units, adds 8 and removes 3. Find its final value. Use the same sequence: known quantities → model → relation → substitution → unit and interpretation.

■ Check the conclusion and its limits

- A class is not the same thing as one instance. Inheritance is not automatically better than composition. Merely hiding a field does not prove that all operations preserve valid state.
- Return to the original observation. Explain what the result supports, which conditions it assumes, and one way to test a competing explanation.

Warn:

Two instances of a class must always share one balance. This claim is false: A class is not the same thing as one instance. Inheritance is not automatically better than composition. Merely hiding a field does not prove that all operations preserve valid state.

Key:

Objects: state, behaviour and an interface: A method call acts on a particular instance. State changes should satisfy preconditions and postconditions. Polymorphism lets code use a common interface with different implementations when the contract is respected.

■ Runnable trace and boundary

```
class Account:
    def __init__(self, balance):
        self.balance = balance
    def withdraw(self, amount):
        if amount < 0 or amount > self.balance:
            return False
        self.balance -= amount
        return True
a, b = Account(15), Account(20)
print(a.withdraw(30), a.balance, b.balance)
print(a.withdraw(15), a.balance, b.balance)
```

Expected output:

```
False 15 20
True 0 20
```

Rejection returns before mutation. With nonnegative initial balances, validated withdrawals preserve $\text{balance} \geq 0$. This small teaching class exposes attributes; a production design must control all mutation paths, not assume the method alone enforces encapsulation.

Abstract data types (HL only) · supported focus

International Baccalaureate · IB Diploma · Computer Science · HL

Scope and prerequisites

Supported HL focus. First assessment 2027 target; official PDF returns 403; older acquired brief is final assessment 2026. Remaining guide, assessment and practical requirements retain their recorded holds.

Prerequisites: read the stated quantities and units, use arithmetic and the model conditions below. Each lesson develops its own method before independent transfer.

These are original or explicitly fictional teaching examples, not actual measurements or completed assessed learner investigations.

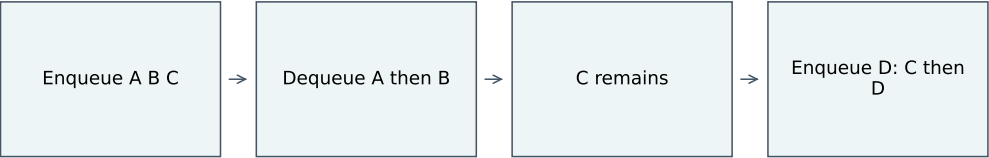
Abstract data types: an operation contract

■ What would explain this observation?

- A waiting-line simulation needs the oldest pending item first. A **stack** gives the most recent item first, so the operation contract matters.
- Start with a prediction. State the quantities or features you would compare, then decide what evidence could distinguish two explanations.

■ Build the model

- An abstract data type specifies values and operations independently of a particular implementation. A stack is last in, first out; a **queue** is first in, first out. A collection interface needs defined behaviour for empty and invalid operations.
- stack: A last-in-first-out abstract data type; queue: A first-in-first-out abstract data type.



■ Choose evidence that can test it

- Choose the structure for the required access pattern. Complexity depends on implementation and assumptions: removing the first item from a shifting array differs from advancing a head pointer in a queue.
 - Trace operation sequences by hand, then compare a program with the expected states. Test empty, singleton and repeated operations. State overflow behaviour if capacity is fixed.
-

■ Work from known quantities

- State the known values and their units. Choose the relation because its assumptions fit this case, then rearrange before substitution.
- Known: enqueue A, B and C, then dequeue twice. A and B leave; C remains. After enqueue D there are two pending items, C then D. The same pushes on a stack would remove C then B.

Example:

A queue starts empty, receives 7 items and removes 4. Find the number remaining. Use the same sequence: known quantities → model → relation → substitution → unit and interpretation.

■ Check the conclusion and its limits

- An ADT is not a specific memory layout. A queue does not sort by priority unless its contract defines a priority queue.
- Return to the original observation. Explain what the result supports, which conditions it assumes, and one way to test a competing explanation.

Warn:

Every queue automatically sorts items by priority. This claim is false: An ADT is not a specific memory layout. A queue does not sort by priority unless its contract defines a priority queue.

Key:

Abstract data types: an operation contract: Choose the structure for the required access pattern. Complexity depends on implementation and assumptions: removing the first item from a shifting array differs from advancing a head pointer in a queue.

■ Runnable trace and boundary

```
from collections import deque
queue = deque()
queue.extend(["A", "B"])
print(queue.popleft())
queue.append("C")
print(queue.popleft())
print(list(queue))
queue.clear()
print(queue.popleft() if queue else "EMPTY")
```

Expected output:

```
A
B
['C']
EMPTY
```

Removing from the left implements FIFO. The explicit empty guard defines the boundary result; deque.popleft without it would raise IndexError. A stack instead removes from the right.

Case study · supported focus

International Baccalaureate · IB Diploma · Computer Science · HL

Scope and prerequisites

Supported HL focus. First assessment 2027 target; official PDF returns 403; older acquired brief is final assessment 2026. Remaining guide, assessment and practical requirements retain their recorded holds.

Prerequisites: read the stated quantities and units, use arithmetic and the model conditions below. Each lesson develops its own method before independent transfer.

These are original or explicitly fictional teaching examples, not actual measurements or completed assessed learner investigations.

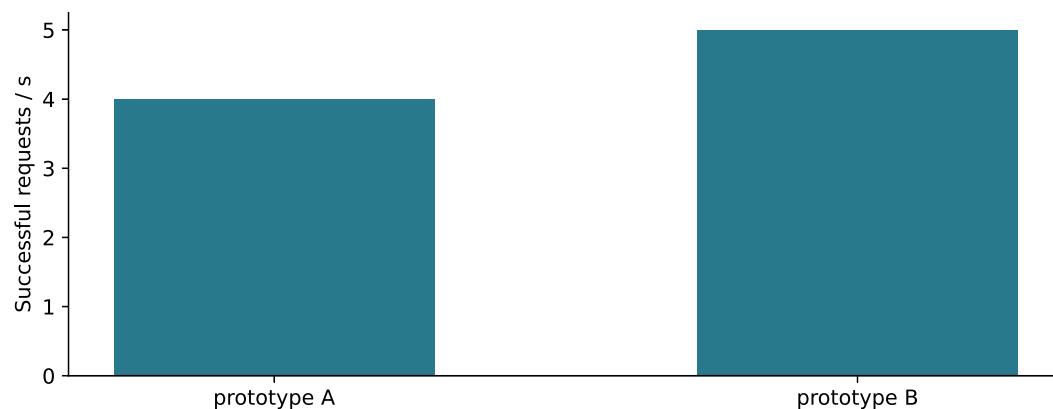
Computing case analysis: evaluate a proposed system

■ What would explain this observation?

- A proposed school booking system looks efficient, but its recommendation depends on users, constraints and evidence. A plausible technical term alone is not a justified decision.
- Start with a prediction. State the quantities or features you would compare, then decide what evidence could distinguish two explanations.

■ Build the model

- A case study places computational choices within a stated scenario. Identify stakeholders, requirements, available evidence and constraints before recommending an architecture or algorithm. Distinguish a scenario fact from an assumption and a prediction. Trade-offs can involve performance, reliability, accessibility, privacy, maintenance and cost.
- **throughput** : Completed work per unit time under a stated workload;
acceptance test : A test of whether a specified user requirement is met.



■ Choose evidence that can test it

- A recommendation should connect a requirement to a mechanism and an observable test. For example, a concurrency problem requires a strategy that prevents conflicting updates, plus tests demonstrating the invariant. A claim about faster response requires comparable workload measurements rather than only a complexity label. Evaluate alternatives under the same stated conditions.
- Use an original school-approved scenario and synthetic data. Build a matrix of claim, supporting scenario evidence, technical explanation, limitation and acceptance test. Mark missing facts explicitly and show how the recommendation would change if an assumption failed. This prepares case-based reasoning without reproducing an unavailable IB assessment case study or inventing its examination rubric.

■ Work from known quantities

- State the known values and their units. Choose the relation because its assumptions fit this case, then rearrange before substitution.
- Known: a fictional prototype processes 120 successful requests during a 30-second observation. Throughput is $120/30=4$ successful requests/s. A second prototype processes 150 in 30 seconds, or 5/s. The second has 25% greater recorded throughput, but no conclusion about latency, failure rate or performance under a larger workload follows without those measurements.

Example:

A fictional system successfully processes 210 requests in 30 s. Calculate throughput. Use the same sequence: known quantities → model → relation → substitution → unit and interpretation.

■ Check the conclusion and its limits

- Throughput is not response time. A strong recommendation states the conditions in which it is expected to work and the evidence that could disconfirm it. Current official CS case-study requirements must be checked against the applicable assessment version.
- Return to the original observation. Explain what the result supports, which conditions it assumes, and one way to test a competing explanation.

Warn:

Higher observed throughput automatically proves lower response time for every request. This claim is false: Throughput is not response time. A strong recommendation states the conditions in which it is expected to work and the evidence that could disconfirm it. Current official CS case-study requirements must be checked against the applicable assessment version.

Key:

Computing case analysis: evaluate a proposed system: A recommendation should connect a requirement to a mechanism and an observable test. For example, a concurrency problem requires a strategy that prevents conflicting updates, plus tests demonstrating the invariant. A claim about faster response requires comparable workload measurements rather than only a complexity label. Evaluate alternatives under the same stated conditions.

■ Runnable trace and boundary

```
for successful, attempted, seconds in [(240, 300, 30), (270, 300, 30)]:
    if seconds <= 0 or attempted <= 0:
        raise ValueError("Positive exposure required")
    throughput = successful / seconds
    failure_fraction = (attempted - successful) / attempted
    print(throughput, failure_fraction)
```

Expected output:

```
8.0 0.2
9.0 0.1
```

Success throughput and failure fraction use different denominators. Zero time or zero attempts cannot support those rates. These fictional summaries do not supply latency, cost or representative-load evidence.