

# 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.

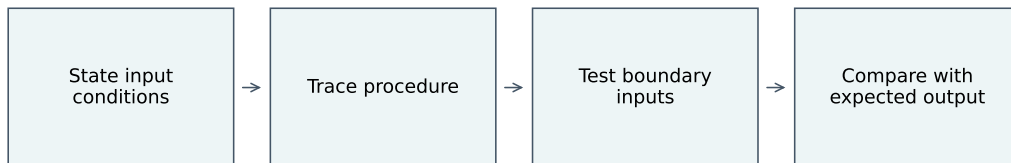
## Algorithms, traces and correctness evidence

### ■ 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.