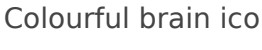
Pseudocode and algorithm reasoning

Inputs, processes and outputs



Colourful brain icon

Icon: OpenMoji, CC BY-SA 4.0

Most algorithms can be understood by identifying what data enters, what happens to it and what result is produced.

- **Input:** What data enters the algorithm?
- **Process:** What calculations, decisions or repetitions occur?
- **Output:** What must be displayed, stored or returned?

Identify the three parts

A roadside safety system receives a vehicle's speed from a sensor. It compares that speed with the road's speed limit. If the vehicle is travelling too quickly, the system displays a warning to the driver.

Copy and complete the following:

- **Input:** ...
- **Process:** ...
- **Output:** ...

Check your answer

- **Input:** the vehicle's speed from the sensor and the road's speed limit.
- **Process:** compare the vehicle's speed with the speed limit.
- **Output:** display a warning when the vehicle's speed exceeds the limit.

Writing clear pseudocode

Pseudocode communicates the logic of an algorithm without requiring the exact syntax of a programming language. Its steps must still be ordered, consistent and unambiguous.

```
BEGIN
INPUT temperature
SET status = "safe"
```

IF temperature > 40 THEN SET status = "warning" ENDIF

OUTPUT status END

- Give variables names that reveal their job.
- Indent code so its structure is easy to see.
- Give totals, counters, lists and flags a starting value.
- Close structures with `ENDIF`, `ENDFOR` or `ENDWHILE`.
- Avoid syntax that belongs to one programming language, such as JavaScript, PHP or Python.

Selection and iteration

Selection chooses a path

```
IF condition THEN
    instructions
ELSE
    alternative instructions
ENDIF
```

Iteration does it again

```
FOR EACH item IN items DO
    process item
ENDFOR
```

WHILE condition DO repeat instructions ENDWHILE

Use `FOR EACH` when processing every item in a collection. Use `WHILE` when repetition depends on a condition.

Check for understanding

A registration form cannot predict how many times a user will enter an invalid password. It must keep requesting another password while the entered password contains fewer than 12 characters.

Analyse the situation. Choose either a `FOR EACH` loop or a `WHILE` loop, then explain why it suits this repetition.

Check your answer

A `WHILE` loop is appropriate because the number of attempts is unknown. Repetition continues while the password contains fewer than 12 characters.

```
INPUT password
WHILE LENGTH(password) < 12 DO
    OUTPUT "Password must contain at least 12 characters"
    INPUT password
ENDWHILE
```

Translate criteria into algorithm steps

Before writing pseudocode, break the problem into smaller jobs. Every criterion must be represented by one or more working steps in the algorithm.

Example criteria

- Read every sensor record.
- Identify readings above 80.
- Store the location of each high reading.
- Display the number of high readings.

Connect each criterion to a step

1. Input the collection of sensor records.
2. Initialise an empty list and a counter.
3. Iterate through every record.
4. Select records whose reading is above 80.
5. Store their locations and increment the counter.
6. Output the counter and stored locations.

Check for understanding

A review app stores a list of numerical ratings for each product. It must calculate the product's average rating. Some new products have an empty ratings list, so the algorithm must not attempt to divide by zero.

Analyse the requirement below before writing any pseudocode:

> Display the average rating, or display “No ratings” when there are no reviews.

Write:

1. the three variables needed to total, count and average the ratings
2. the calculation used to find the average
3. the condition that decides whether the calculation is safe.

Check your answer

- Variables: `totalRating`, `numberOfRatings` and `averageRating`.
- Calculation: `averageRating = totalRating / numberOfRatings`.
- Condition: calculate the average if `numberOfRatings > 0`; otherwise display `No ratings`.

Lists and objects

A list is an ordered collection. An object groups related values under named keys. Digital solutions often store objects inside lists and lists inside objects.

```
stations = [
    {"name": "North", "readings": [24, 31, 29]},
    {"name": "South", "readings": [35, 38]}
]
```

- `stations` is a list.
- Each station is an object.
- `station["name"]` accesses a named value.
- `station["readings"]` accesses a nested list.

Accessing nested data

```
FOR EACH station IN stations DO
    SET total = 0
```

```
FOR EACH reading IN station["readings"] DO
    SET total = total + reading
ENDFOR

SET average = total / LENGTH(station["readings"])
OUTPUT station["name"], average
```

ENDFOR

The outer loop picks a station. The inner loop opens that station's readings and handles them one at a time.

Check for understanding

The current algorithm divides `total` by `LENGTH(station["readings"])`. Consider what happens when a station has no readings and the list length is zero.

Identify the error that could occur. Then **recommend** an `IF` statement that prevents it and displays `No readings` instead.

Check your answer

The algorithm could attempt to divide by zero. A suitable refinement is:

```
IF LENGTH(station["readings"]) > 0 THEN
    SET average = total / LENGTH(station["readings"])
    OUTPUT station["name"], average
ELSE
    OUTPUT station["name"], "No readings"
ENDIF
```

Modularisation and library functions

Modularisation divides a larger problem into smaller, focused jobs. Library functions provide existing operations that can be reused when their documented inputs, outputs and purpose match the problem.

```
SET congestion = CalculateCongestion(sensorReadings)
SET safestRoute = FindSafestRoute(start, destination, hazards)
```

Check a function's inputs, output and purpose. A function called `FindSafestRoute` will not send an alert unless its description explicitly says so.

Useful comments

```
# Criterion: warn the user about unavailable routes
IF safestRoute = NONE THEN
    OUTPUT "No safe route is available"
ENDIF
```

A useful comment explains intent or connects code to a criterion. A comment such as `# Add one to count` only repeats the instruction and adds little value.

Worked example: choosing suitable rooms

A venue needs to place a group in a suitable room. The algorithm must display every room big enough for the group, then count how many of those rooms are accessible.

```
BEGIN
INPUT rooms
INPUT groupSize
SET accessibleCount = 0
```

FOR EACH room IN rooms DO IF room["capacity"] >= groupSize THEN OUTPUT room["name"]

```
    IF room["accessible"] = TRUE THEN
        SET accessibleCount = accessibleCount + 1
    ENDIF
ENDIF
```

ENDFOR

OUTPUT "Accessible suitable rooms:", accessibleCount END

Check for understanding

The venue algorithm is tested with a group of 18 people and the following room data:

```
rooms = [
    {"name": "A1", "capacity": 20, "accessible": TRUE},
    {"name": "B2", "capacity": 12, "accessible": TRUE},
    {"name": "C3", "capacity": 30, "accessible": FALSE}
]
```

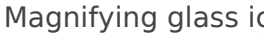
groupSize = 18

1. **Explain** why `accessibleCount` is initialised before the loop.
2. **Explain** why the accessibility selection is nested inside the capacity selection.
3. **Determine** every value displayed by the algorithm.

Check your answer

1. The counter needs a known starting value before it can be incremented. Initialising it before the loop also prevents it from being reset for every room.
2. Only rooms with enough capacity are suitable. Nesting ensures that accessibility is counted only for suitable rooms.
3. The algorithm displays `A1`, `C3` and `Accessible suitable rooms: 1`.

Desk checking an algorithm



Magnifying glass icon

Icon: OpenMoji, CC BY-SA 4.0

A desk check traces an algorithm manually using supplied test data and a table. Record every variable change, decision and output in sequence rather than jumping directly to the expected result.

Iteration	room	capacity condition	accessible condition	accessibleCount	output
Initial	—	—	—	0	—
1	A1	True	True	1	A1
2	B2	False	Not tested	1	—
3	C3	True	False	1	C3
Final	—	—	—	1	Accessible suitable rooms: 1

> **Key idea:** The final answer is not enough. The trail of intermediate values is what exposes the bug.

Independent practice: event check-in

A community event uses a check-in app. Staff need to see who has arrived and receive an alert when a checked-in attendee is under 18. Attendees who have not checked in must not be displayed or included in the count.

The app receives these attendee records:

```
attendees = [
    {"name": "Ari", "checkedIn": TRUE, "age": 17},
    {"name": "Mika", "checkedIn": FALSE, "age": 22},
    {"name": "Sam", "checkedIn": TRUE, "age": 31}
]
```

Available library function: `SendAlert(message)` sends a message to event staff.

Criteria

- Process every attendee.
- Display the name of each checked-in attendee.
- Count checked-in attendees who are under 18.
- If at least one checked-in attendee is under 18, use the library function to alert staff.
- Display the final under-18 count.

Develop and **symbolise** a complete algorithm in pseudocode. Add comments showing where each criterion is addressed. Then desk check the variables and output using the supplied data.

Expected output

Submit:

1. one complete pseudocode algorithm
2. one desk-check table showing the value of `under18Count` during every iteration
3. the names, alert and final count produced as output.

View a sample solution

```
BEGIN
INPUT attendees
SET under18Count = 0
```

Criterion: process every attendee

FOR EACH attendee IN attendees DO # Criterion: display each checked-in attendee IF attendee["checkedIn"] = TRUE THEN OUTPUT attendee["name"]

```
    # Criterion: count checked-in attendees under 18
    IF attendee["age"] &lt; 18 THEN
        SET under18Count = under18Count + 1
    ENDIF
ENDIF
```

ENDFOR

Criterion: alert staff when required

IF under18Count > 0 THEN SendAlert("A checked-in attendee is under 18") ENDIF

Criterion: display the final count

OUTPUT "Checked-in attendees under 18:", under18Count END

Iteration	name	checkedIn	age < 18	under18Count	output/action
Initial	—	—	—	0	—
1	Ari	True	True	1	Ari
2	Mika	False	Not tested	1	—

Iteration	name	checkedIn	age < 18	under18Count	output/action
3	Sam	True	False	1	Sam
After loop	—	—	—	1	`SendAlert(...)`
Final	—	—	—	1	Checked-in attendees under 18: 1

Revision #6
Created 2026-10-06 00:37:55 UTC by Mr Napper
Updated 2026-10-06 01:31:52 UTC by Mr Napper