ANSWERS

F7.5 Project: the logbook

Files and databases · GCSE · about 25 min

What this lesson is about

Log every run to a file, load the file into a table, and find the best run with SQL.

- The brief
- The logbook file
- Plan
- Step 1: read the log and find the next run number
- Step 2: load the file into a table
- Step 3: put it together

Questions

4 marks in all

1. In the logbook, why is the file used as well as the database table? [1 mark]

- ☐ **A** The file keeps the runs between sessions; the table is rebuilt from it to query
- ☐ **B** The table cannot hold numbers
- ☐ **C** SQL cannot read files
- ☐ **D** Files are required by SQL

> **Answer:** A. The in-memory table disappears when the program stops; the file does not.

2. logbook.csv has a header line and four runs. How many lines does the file have? [1 mark]

> **Answer:** 5. The header plus four records, which is why the next run number is the number of lines.

3. Which query finds the fastest run? [1 mark]

- ☐ **A** SELECT run FROM runs ORDER BY cm / seconds DESC
- ☐ **B** SELECT run FROM runs ORDER BY cm ASC
- ☐ **C** SELECT COUNT(*) FROM runs
- ☐ **D** SELECT run FROM runs WHERE seconds = 0

> **Answer:** A. Speed is distance divided by time; descending puts the largest first.

4. What happens to the logbook if the program opens logbook.csv with mode "w" to add a run? [1 mark]

- ☐ **A** Every earlier run is lost
- ☐ **B** The run is added at the end
- ☐ **C** The file is read
- ☐ **D** Nothing changes

> **Answer:** A. "w" empties the file first; appending needs "a".

The task: the logbook

Build the program from the brief. Print `runs logged: <n>` using `SELECT COUNT(*)`, and `fastest: run <n> at <speed> cm/s` with the speed rounded to 1 decimal place, using `ORDER BY`. The task types `28`, and starts from the four-run logbook above every time.

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

import sqlite3
```

The hint students can ask for: Read the file to find the next run number; drive and time; append; CREATE TABLE and INSERT every line; SELECT COUNT(*); SELECT run, cm / seconds ... ORDER BY cm / seconds DESC.

A solution

```
from bugbot import *
connect()
import sqlite3

far = float(input("How far? "))

with open("logbook.csv") as f:
    lines = f.read().splitlines()
next_run = len(lines)

start = clock()
forward(50, distance=far)
seconds = round(clock() - start, 1)
with open("logbook.csv", "a") as f:
    f.write(f"{next_run},{far:g},{seconds}\n")

db = sqlite3.connect(":memory:")
db.execute("CREATE TABLE runs (run INTEGER PRIMARY KEY, cm REAL, seconds REAL)")
with open("logbook.csv") as f:
    f.readline()
    for row in f:
        run, cm, secs = row.strip().split(",")
        db.execute("INSERT INTO runs VALUES (?, ?, ?)", (int(run), float(cm), float(secs)))

count = db.execute("SELECT COUNT(*) FROM runs").fetchone()[0]
print("runs logged:", count)
run, speed = db.execute("SELECT run, cm / seconds FROM runs ORDER BY cm / seconds
DESC").fetchone()
print(f"fastest: run {run} at {round(speed, 1)} cm/s")
```

Any program that meets the task's checks is marked correct in the simulator; this is one way, not the only way.