ANSWERS

A11.8 Capturing, managing and exchanging data

Databases and big data · A level · OCR H446 1.3.2, AQA 7517 4.9.4.10, Eduqas A500QS 2.4 · about 55 min

What this lesson is about

How data gets in, how it is selected and managed, and how systems swap it as CSV, JSON and XML. The robot exports its own readings.

- Capturing data
- Selecting data
- Managing data
- Exchanging data

Questions

6 marks in all

1. Exam answer sheets are scanned and the positions of shaded boxes are read. Which capture method is this? [1 mark]

- ☐ **A** Optical mark recognition (OMR)
- ☐ **B** Optical character recognition (OCR)
- ☐ **C** RFID
- ☐ **D** A data logger

> **Answer:** A. OMR detects marks in fixed positions; OCR reads the shapes of letters and digits.

2. A run time of 7.2 s is typed as 2.7 s. Which check could catch this? [1 mark]

- ☐ **A** Verification, such as double entry
- ☐ **B** A range check
- ☐ **C** A presence check
- ☐ **D** A format check

> **Answer:** A. 2.7 s is a valid time, so validation passes it; only comparing with the source finds the error.

3. What does a data dictionary hold? [1 mark]

- ☐ **A** Data about the data: tables, fields, types, lengths, keys and validation rules
- ☐ **B** Every record in the database, sorted
- ☐ **C** A list of the words used in text fields
- ☐ **D** Backups of deleted records

> **Answer:** A. The data dictionary is metadata that the DBMS uses to manage the data.

4. Which are true of JSON compared with XML? [1 mark]

Tick every answer that is true.

- ☐ **A** JSON is usually more compact
- ☐ **B** JSON has types such as numbers, true and false
- ☐ **C** Only XML can represent nested data
- ☐ **D** JSON requires every value to be wrapped in opening and closing tags

> **Answer:** A, B. Both can nest data; it is XML that wraps values in tags, which makes it more verbose.

5. What does this print? [1 mark]

```
import json
text = '{"robot": "Ada", "readings": [61, 51, 41], "ok": true}'
data = json.loads(text)
print(data["robot"], len(data["readings"]), data["readings"][-1] + 1, data["ok"])
```

> **Answer:**
>
> ```
> Ada 3 42 True
> ```
>
> JSON arrays become lists, numbers become numbers and true becomes True.

6. What does this print? [1 mark]

```
import csv, io
rows = list(csv.reader(io.StringIO("name,cm\nAda,42\nBolt,38\n")))
print(rows[1])
print(rows[1][1] + rows[2][1])
```

> **Answer:**
>
> ```
> ['Ada', '42']
> 4238
> ```
>
> CSV has no types: every value is text, so + joins the strings.

The task: three formats, one set of readings

The robot starts facing a wall. Take four readings, numbered `step` 0 to 3. For each: record `step` (a whole number), `y` (the second value of `position()`, rounded to 1 decimal place) and `cm` (the value of `distance()`), then drive forward 10 cm at speed 40 before the next reading (no drive after step 3). Then, from that one list of readings: 1. Write `readings.csv`: the header line `step,y,cm`, then one line per reading, such as `1,10.0,51.0`. 2. Write `readings.json` with `json.dump`: a list of four objects, each with the keys `step`, `y` and `cm`. 3. Read `readings.json` back with `json.load` and print `json readings: <n>`, the number of objects in it. 4. Print the readings as XML: a line `<readings>`, then one line per reading exactly in the form `<reading step="1" y="10.0" cm="51.0"/>` (indenting is allowed), then `</readings>`.

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

import json

readings = []
```

The hint students can ask for: Collect the readings once, as a list of dictionaries, and write all three formats from that one list. For the CSV, the header comes once, before the loop. Reading the JSON back is the receiving program's side of the exchange.

A solution

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

import json

readings = []
for step in range(4):
    x, y = position()
    readings.append({"step": step, "y": round(float(y), 1), "cm": distance()})
    if step < 3:
        forward(40, distance=10)

with open("readings.csv", "w") as f:
    f.write("step,y,cm\n")
    for r in readings:
        f.write(f"{r['step']},{r['y']},{r['cm']}\n")

with open("readings.json", "w") as f:
    json.dump(readings, f, indent=2)

with open("readings.json") as f:
    received = json.load(f)
print("json readings:", len(received))

print("<readings>")
for r in readings:
    print(f'  <reading step="{r["step"]}" y="{r["y"]}" cm="{r["cm"]}"/>')
print("</readings>")
```

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