ANSWERS

F7.1 Reading and writing files

Files and databases · GCSE · OCR J277 2.2.3, Edexcel 1CP2 6.4.2 · about 15 min

What this lesson is about

Open, read, write, append and close: a run log in a CSV file.

- The files on this page
- Reading a file
- Line by line
- Writing a file
- The whole file as a list of records
- When files go wrong

Questions

6 marks in all

1. Put the steps for reading a file in order. [1 mark]

Number the lines 1 to 3 to put them in the right order.

___ `Open the file`

___ `Read from the file`

___ `Close the file`

Answer:

```
Open the file
Read from the file
Close the file
```

Open, use, close: the same in every language.

2. What does opening a file with mode "w" do to a file that already exists? [1 mark]

- ☐ **A** Empties it before writing
- ☐ **B** Adds to the end of it
- ☐ **C** Reads it
- ☐ **D** Nothing until close is called

Answer: A. "w" starts the file again; use "a" to add to the end.

3. Which mode adds new lines to the end of a file and keeps what was there? [1 mark]

- ☐ **A** "a"
- ☐ **B** "w"
- ☐ **C** "r"
- ☐ **D** "x"

Answer: A. a stands for append.

4. What does this program print? [1 mark]

```
line = "3,square,80,9.4\n"
fields = line.strip().split(",")
print(fields[1], float(fields[2]) + 1)
```

> **Answer:**
>
> ```
> square 81.0
> ```
>
> strip removes the newline, split makes a list of strings, and float converts the distance.

5. Why use `with open(...) as f:`? [1 mark]

- ☐ **A** The file is closed automatically at the end of the block
- ☐ **B** It reads the file faster
- ☐ **C** It makes the file bigger
- ☐ **D** It is the only way to write

> **Answer:** A. Even if an error happens inside the block, the file is closed.

6. Why store data in a file rather than only in variables? [1 mark]

- ☐ **A** Variables are lost when the program stops; a file keeps the data
- ☐ **B** Files are faster than variables
- ☐ **C** Variables cannot hold numbers
- ☐ **D** Files cannot be changed

> **Answer:** A. A file keeps data between runs of the program.

The task: log the run

Drive forward 25 cm, timing it with `clock()`. Then **append** a line to `runs.csv` for this run: run number `4`, task `wall`, `25` cm, and the seconds rounded to one decimal place, in the same format as the other lines. Finally read the file and print `4 runs logged`, counting the lines after the header rather than typing 4. The task starts from the original four-line `runs.csv` every time it runs, whatever earlier runs added to the copy above.

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

start = clock()
forward(50, distance=25)
```

The hint students can ask for: Time the run by taking a clock reading before and after. Open the file so that writing adds to the end rather than replacing it, and remember the header line when you count the rows afterwards.

A solution

```
from bugbot import *
connect()
start = clock()
forward(50, distance=25)
seconds = round(clock() - start, 1)
with open("runs.csv", "a") as f:
    f.write("4,wall,25," + str(seconds) + "\n")
with open("runs.csv") as f:
    lines = f.read().splitlines()
print(len(lines) - 1, "runs logged")
```

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