ANSWERS

A7.4 Errors, range and precision

Data representation · A level · OCR H446 1.4.1, AQA 7517 4.5.4.5, Eduqas A500QS 2.3 · about 30 min

What this lesson is about

Rounding errors, absolute and relative error, range against precision, overflow and underflow, with a real sensor reading.

- Rounding errors
- Absolute and relative errors
- Range and precision
- Overflow and underflow
- A real reading

Questions

6 marks in all

1. A true value of 2.7 is stored as 2.625. What is the absolute error? [1 mark]

> **Answer:** 0.075 (accept within 0.0001). Absolute error is the size of the difference between the true value and the stored value: 2.7 - 2.625 = 0.075.

2. A true value of 2.7 is stored as 2.625. What is the relative error as a percentage, to 2 decimal places? [1 mark]

> **Answer:** 2.78 (accept within 0.01). Relative error is absolute error divided by the true value: 0.075 / 2.7 = 0.02778, about 2.78%.

3. A floating point format keeps 12 bits in total. The mantissa is given more bits and the exponent fewer. What changes? [1 mark]

- ☐ **A** Precision increases and range decreases
- ☐ **B** Range increases and precision decreases
- ☐ **C** Both increase
- ☐ **D** Neither changes

> **Answer:** A. More mantissa bits give more precision; fewer exponent bits give a smaller range.

4. What is underflow? [1 mark]

- ☐ **A** A result too close to zero for the format to represent
- ☐ **B** A result too large for the format
- ☐ **C** A negative result from adding two positive numbers
- ☐ **D** A loss of the carry bit

> **Answer:** A. Underflow needs an exponent more negative than the format allows, so the value is stored as zero.

5. Why can 0.1 not be stored exactly in binary floating point? [1 mark]

- ☐ **A** Its binary fraction repeats for ever, so it must be cut off
- ☐ **B** It is an irrational number
- ☐ **C** It is too small for the exponent
- ☐ **D** Binary cannot store fractions

> **Answer:** A. 0.1 in binary is 0.000110011001100..., which never ends, so any fixed number of bits rounds it. It is rational, not irrational.

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

```
a = 0.1 + 0.2
print(a == 0.3)
print(abs(a - 0.3) < 1e-9)
```

> **Answer:**
>
> ```
> False
> True
> ```
>
> Because of rounding errors the stored sum is not exactly the stored 0.3, so real numbers should be compared within a tolerance.

The task: store a reading

The robot faces a wall. Read `distance()` once (it returns centimetres) and convert it to metres. Store the metres in a byte of unsigned fixed point with all 8 bits after the point: the stored whole number is `int(metres * 256)`, which truncates. Then print exactly these five lines, calculating every value: - `reading: <metres> m`, the metres as Python prints the float - `stored: <bits>`, the stored whole number as 8 binary digits (you may use `format(q, "08b")`) - `stored value: <value> m`, the stored whole number divided by 256 - `absolute error: <error> m`, the absolute error, then `round(error, 7)` - `relative error: <percent>%`, the relative error as a percentage, then `round(percent, 2)`, with no space before the `%`

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

metres = distance() / 100
print("reading:", metres, "m")
```

The hint students can ask for: Scaling by 256 moves the binary point eight places, so the whole part of metres times 256 is the stored pattern. Dividing the stored whole number by 256 gives back the value the byte really holds. The absolute error is the gap between the two; the relative error is that gap as a fraction of the true reading.

A solution

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

metres = distance() / 100
print("reading:", metres, "m")
q = int(metres * 256)
print("stored:", format(q, "08b"))
stored = q / 256
print("stored value:", stored, "m")
absolute = abs(metres - stored)
print("absolute error:", round(absolute, 7), "m")
print("relative error: " + str(round(absolute / metres * 100, 2)) + "%")
```

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