ANSWERS

A7.6 Error checking and correction

Data representation · A level · AQA 7517 4.5.5.3, Eduqas A500QS 2.8 · about 30 min

What this lesson is about

Parity bits, majority voting, checksums and check digits, with checksums on the robot's radio messages.

- Parity bits
- Majority voting
- Checksums
- Check digits
- Comparing the methods

Questions

6 marks in all

1. Even parity is used. What parity bit is added to the 7 bits 1101011? [1 mark]

> **Answer:** 1. There are five 1s, an odd number, so the parity bit must be 1 to make the total even.

2. Even parity is used. The byte 01101011 is received. What can the receiver conclude? [1 mark]

- ☐ **A** There is an error, because the byte has an odd number of 1s
- ☐ **B** The byte is correct
- ☐ **C** Bit 3 is wrong and can be corrected
- ☐ **D** Exactly two bits are wrong

> **Answer:** A. It has five 1s, which is odd, so at least one bit was flipped. Parity cannot say which bit.

3. Majority voting sends each bit three times. The groups 110 011 000 are received. Give the three data bits. [1 mark]

> **Answer:** 110. Take the value that appears most often in each group: 110 gives 1, 011 gives 1 and 000 gives 0.

4. Which of these methods can correct an error without the data being sent again? [1 mark]

Tick every answer that is true.

- ☐ **A** Majority voting
- ☐ **B** A parity bit
- ☐ **C** A checksum
- ☐ **D** A check digit

> **Answer:** A. Majority voting can outvote one flipped bit in a group. The others only detect errors.

5. A checksum adds the character codes of a message modulo 256. Which error does it always miss? [1 mark]

- ☐ **A** Two characters swapping places
- ☐ **B** One character changed
- ☐ **C** One bit flipped
- ☐ **D** A character missing

> **Answer:** A. Addition gives the same total in any order, so a swap leaves the checksum unchanged.

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

```
digits = [9, 7, 8, 1, 8, 6, 1, 9, 7, 8, 2, 7]
total = 0
for i in range(12):
    weight = 1 if i % 2 == 0 else 3
    total = total + digits[i] * weight
print(total, (10 - total % 10) % 10)
```

> **Answer:**
>
> ```
> 149 1
> ```
>
> This is the ISBN-13 check digit: weights alternate 1 and 3, and the check digit makes the total a multiple of 10.

The task: checksums on the radio

Write `checksum(text)`. The parameter `text` is a string. It returns the sum of the character codes (use `ord`) modulo 256, as a string of exactly two uppercase hex digits (you may use `format(n, "02X")`). 1. For each command in `GO 20`, `LEFT 90`, `STOP`, in that order, build the packet `<command>*<checksum>`, send it with `send()`, and print `sent <packet>`. Send the packet you built, not typed-in text. 2. For each packet in `received`, split it at the `*`, recalculate the checksum of the text part and print `<packet> ok` if it matches the checksum that arrived, or `<packet> corrupt` if it does not.

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

def checksum(text):
    return "00"

received = ["GO 20*18", "G0 20*18", "STOP*46", "LEFT 80*B4"]
```

The hint students can ask for: Add up the character code of every character and keep only the remainder after dividing by 256, so it fits in one byte. To check a packet, split it at the star, recalculate the checksum of the text part, and compare it with the one that arrived.

A solution

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

def checksum(text):
    total = 0
    for ch in text:
        total = total + ord(ch)
    return format(total % 256, "02X")

for command in ["GO 20", "LEFT 90", "STOP"]:
    packet = command + "*" + checksum(command)
    send(packet)
    print("sent", packet)

received = ["GO 20*18", "G0 20*18", "STOP*46", "LEFT 80*B4"]
for packet in received:
    text, check = packet.split("*")
    print(packet, "ok" if checksum(text) == check else "corrupt")
```

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