ANSWERS

F4.4 Decomposition and abstraction

Functions and structured code · GCSE · OCR J277 2.1.1, AQA 8525 3.1.1, Edexcel 1CP2 1.1.1 · about 20 min

What this lesson is about

Structure diagrams, and hiding the details of a job behind a function.

- Decomposition: breaking a problem down
- Abstraction: hiding the details
- Building from the bottom up, testing as you go

Questions

5 marks in all

1. What is decomposition? [1 mark]

- ☐ **A** Breaking a problem down into smaller, more manageable sub-problems
- ☐ **B** Removing unnecessary detail
- ☐ **C** Putting code in alphabetical order
- ☐ **D** Deleting unused code

> **Answer:** A. Each sub-problem can then be solved, and often written as a subprogram, on its own.

2. What is abstraction? [1 mark]

- ☐ **A** Removing unnecessary detail to focus on what matters
- ☐ **B** Breaking a problem into parts
- ☐ **C** Writing code in capitals
- ☐ **D** Testing a program

> **Answer:** A. A map of a bus route is an abstraction; so is a function that hides how it does its job behind a name.

3. A simulator's mat keeps the walls and distances but not the carpet colour. Which idea is that? [1 mark]

- ☐ **A** Abstraction
- ☐ **B** Decomposition
- ☐ **C** Iteration
- ☐ **D** Casting

> **Answer:** A. It leaves out detail that does not matter for the robot's problem.

4. What does a structure diagram show? [1 mark]

- ☐ **A** A problem broken down into smaller sub-problems, top to bottom
- ☐ **B** The order lines run in
- ☐ **C** The values of variables as a program runs
- ☐ **D** How data moves over a network

> **Answer:** A. The whole problem at the top, the parts beneath it, each part broken down again.

5. Why keep a delivery route as a list of stops, separate from the code that drives? [1 mark]

- ☐ **A** The route can change without changing the functions
- ☐ **B** Lists run faster than functions
- ☐ **C** Python requires it
- ☐ **D** Functions cannot use numbers

> **Answer:** A. Data separate from code is abstraction: the functions do not need to know which stops they visit.

The task: three stops

Write `go_to(x, y)` and `signal()`, then use them to visit three stops in order, signalling at each: **A** at (30, 10), **B** at (30, 60) and **C** at (0, 60), all in cm from where the robot starts. The robot starts in the bottom left of the mat.

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

def go_to(x, y):
    """Drive to (x, y), in cm from where the robot started."""
    here_x, here_y = position()

def signal():
    """Light and beep at a stop."""
    led("green")
```

The hint students can ask for: Work out where you are, then how far you still have to go in each direction, and drive that difference. Sideways and forwards are separate moves. Do the signal at each stop.

A solution

```
from bugbot import *
connect()
def go_to(x, y):
    """Drive to (x, y), in cm from where the robot started."""
    here_x, here_y = position()
    across = x - here_x
    up = y - here_y
    if across > 0:
        right(60, distance=across)
    elif across < 0:
        left(60, distance=-across)
    if up > 0:
        forward(60, distance=up)
    elif up < 0:
        backward(60, distance=-up)

def signal():
    """Light and beep at a stop."""
    led("green")
    tone(784, 0.3)
    led("off")

for x, y in [(30, 10), (30, 60), (0, 60)]:
    go_to(x, y)
    signal()
```

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