ANSWERS

A1.6 Programming paradigms and procedural programming

Programming techniques and object-oriented programming · A level · OCR H446 1.2.4, AQA 7517 4.1.2.1, Eduqas A500QS 1.4 · about 20 min

What this lesson is about

Procedural, object-oriented, declarative and low-level paradigms, the structured approach, and hierarchy charts.

- The main paradigms
- Procedural programming
- Hierarchy charts

Questions

5 marks in all

1. Which describes a declarative language? [1 mark]

- ☐ **A** You state what result you want, and the language works out how to get it
- ☐ **B** You give step-by-step instructions that change the program's state
- ☐ **C** You write instructions for one particular processor
- ☐ **D** You organise the program into classes and objects

> **Answer:** A. SQL and Prolog are declarative; procedural and object-oriented languages are imperative.

2. Which are features of the structured approach to programming? [1 mark]

Tick every answer that is true.

- ☐ **A** Only sequence, selection and iteration are used
- ☐ **B** Problems are designed top-down and split into subroutines
- ☐ **C** Each block has one entry and one exit point
- ☐ **D** GOTO statements jump between parts of the program

> **Answer:** A, B, C. The structured approach removes jumps such as GOTO.

3. In a hierarchy chart, what does a line from box A down to box B mean? [1 mark]

- ☐ **A** Subroutine A calls subroutine B
- ☐ **B** B runs before A
- ☐ **C** A and B share a variable
- ☐ **D** B is called exactly once

> **Answer:** A. A hierarchy chart shows what calls what, not the order or number of calls.

4. Which paradigm would the Little Man Computer instruction set belong to? [1 mark]

- ☐ **A** Low-level (assembly)
- ☐ **B** Object-oriented
- ☐ **C** Functional
- ☐ **D** Logic

> **Answer:** A. LMC mnemonics such as LDA and ADD are a simple assembly language.

5. Which are advantages of the structured approach? [1 mark]

Tick every answer that is true.

- ☐ **A** Subroutines can be tested individually
- ☐ **B** Different programmers can work on different subroutines
- ☐ **C** A change is often confined to one subroutine
- ☐ **D** The program never needs testing as a whole

> **Answer:** A, B, C. Structure makes testing and maintenance easier, but the whole program must still be tested.

The task: structure it from the hierarchy chart

The starter drives a square and a triangle with every move typed out, then plays three notes. Restructure it to match the hierarchy chart above, with no global variables: - `side(length)` drives forward `length` cm. - `corner(sides)` turns right by the angle for a regular polygon with `sides` sides: a full turn, 360 degrees, divided by `sides`. Work it out; do not type 90 or 120. - `polygon(sides, length)` repeats `side` then `corner` once for each side, then prints `polygon <sides> done`. - `fanfare()` plays the notes 523, 659 and 784 Hz for 0.15 seconds each. The main program calls `polygon(4, 20)`, then `polygon(3, 20)`, then `fanfare()`, and nothing else.

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

forward(60, distance=20)
turn_right(30, angle=90)
forward(60, distance=20)
turn_right(30, angle=90)
forward(60, distance=20)
turn_right(30, angle=90)
forward(60, distance=20)
turn_right(30, angle=90)
print("polygon 4 done")
forward(60, distance=20)
turn_right(30, angle=120)
forward(60, distance=20)
turn_right(30, angle=120)
forward(60, distance=20)
turn_right(30, angle=120)
print("polygon 3 done")
tone(523, 0.15)
tone(659, 0.15)
tone(784, 0.15)
```

The hint students can ask for: Write the subroutines at the bottom of the chart first, and test each one alone. Each box becomes one subroutine, and each line down from a box is a call inside it. The main program at the top only calls the boxes directly under it.

A solution

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

def side(length):
    forward(60, distance=length)

def corner(sides):
    turn_right(30, angle=360 / sides)

def polygon(sides, length):
    for i in range(sides):
        side(length)
        corner(sides)
    print("polygon", sides, "done")
```

```
def fanfare():
    for note in [523, 659, 784]:
        tone(note, 0.15)

polygon(4, 20)
polygon(3, 20)
fanfare()
```

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