ANSWERS

A14.4 System design

Software development, law and ethics · A level · OCR H446 1.2.3, AQA 7517 4.13.1.2, Eduqas A500QS 1.6 · about 40 min

What this lesson is about

Structure charts, data flow diagrams, designing data, algorithms and the user interface, and printing a structure chart with recursion.

- What a design contains
- Top-down design and structure charts
- Data flow diagrams
- Designing the data
- Designing the interface
- Designing the algorithms

Questions

6 marks in all

1. In a structure chart, what does a small arrow with an open circle show? [1 mark]

- ☐ **A** A data couple: data passed between two modules
- ☐ **B** A loop
- ☐ **C** A decision
- ☐ **D** An external entity

> **Answer:** A. Data couples show parameters passed down and return values passed up; a filled circle is a control flag.

2. In a data flow diagram, which of these flows breaks the rules? [1 mark]

- ☐ **A** Data flowing directly from one data store to another
- ☐ **B** Data flowing from an external entity to a process
- ☐ **C** Data flowing from a process to a data store
- ☐ **D** Data flowing from a process to an external entity

> **Answer:** A. Every data flow must start or end at a process, since only a process can move or change data.

3. Which of these should a design plan, according to AQA? [1 mark]

Tick every answer that is true.

- ☐ **A** The data structures for the data model
- ☐ **B** The algorithms
- ☐ **C** The modular structure
- ☐ **D** The human user interface
- ☐ **E** The exam board's mark scheme

> **Answer:** A, B, C, D. AQA lists data structures, algorithms, a modular structure and the user interface.

4. The robot's screen shows errors only as red text. Which change best improves accessibility? [1 mark]

- ☐ **A** Add a symbol and a clear message, so colour is not the only way the error is shown
- ☐ **B** Make the red text brighter
- ☐ **C** Remove the error messages
- ☐ **D** Use a smaller font to fit more on screen

> **Answer:** A. Colour alone excludes users who cannot distinguish it; a symbol and words carry the meaning for everyone.

5. What name is given to design that starts with the whole problem and repeatedly breaks it into smaller sub-problems? [1 mark]

> **Answer:** top-down design. Top-down design, or stepwise refinement, is what a structure chart records.

6. A structure chart is stored as a dictionary. What does this print? [1 mark]

```
chart = {"robot": ["sense", "act"], "act": ["drive", "beep"]}
def show(module, depth):
    print("-" * depth + module)
    for sub in chart.get(module, []):
        show(sub, depth + 1)
show("robot", 0)
```

> **Answer:**
>
> ```
> robot
> -sense
> -act
> --drive
> --beep
> ```
>
> Each module is printed, then its sub-modules one level deeper, so drive and beep come after act with two dashes.

The task: print the structure chart

`chart` is a dictionary describing a structure chart: each key is a module name (a string), and its value is the list of that module's sub-modules, in left-to-right order. A module that is not a key has no sub-modules. Write a **recursive** procedure `show(module, depth)` that prints the module's name after `2 * depth` spaces, then calls itself for each of the module's sub-modules with `depth + 1`. Call it as `show("delivery robot", 0)`. While it runs, count every module printed and collect the **leaves** (modules with no sub-modules) in the order they are printed. After the chart, print `modules: <n>` and then `leaves: <names>`, with the names separated by `, `. The robot stays still.

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

chart = {
    "delivery robot": ["plan route", "drive route", "report"],
    "plan route": ["read map", "shortest path"],
    "drive route": ["drive leg", "check gap"],
    "drive leg": ["set motors", "read heading"],
    "report": ["format message", "send message"],
}
```

The hint students can ask for: A module's line is its name after some spaces that depend on its depth. After printing it, do the same for each of its sub-modules one level deeper. A module that is not a key in the dictionary has no sub-modules, so it is a leaf.

A solution

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

chart = {
    "delivery robot": ["plan route", "drive route", "report"],
    "plan route": ["read map", "shortest path"],
    "drive route": ["drive leg", "check gap"],
    "drive leg": ["set motors", "read heading"],
    "report": ["format message", "send message"],
}

count = 0
leaves = []

def show(module, depth):
    global count
    print("  " * depth + module)
    count += 1
    if module not in chart:
        leaves.append(module)
    for sub in chart.get(module, []):
        show(sub, depth + 1)

show("delivery robot", 0)
print("modules:", count)
print("leaves:", ", ".join(leaves))
```

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