WORKSHEET

A15.6 Pre-release material and skeleton programs

Exam preparation · A level · OCR H446 2.2.1, AQA 7517 4.4.1.2, Eduqas A500QS 1.8 · about 45 min

Name _____ Class _____ Date _____

What this lesson is about

Getting to know a skeleton program before the exam, the questions asked about it, and making exam-style changes to one.

- What a skeleton program is
- Studying the skeleton before the exam
- The questions it brings
- Reading unfamiliar code quickly

Questions

5 marks in all

1. A question asks you to change a skeleton program so a robot cannot move into a shelf, and there is already an is_free method. What is the best approach? [1 mark]

- ☐ **A** Call is_free in the move method before changing the position
- ☐ **B** Write a new check in the main program after the move
- ☐ **C** Delete the shelves from the grid
- ☐ **D** Rewrite the whole Warehouse class

2. What is usually needed for a skeleton program modification question? [1 mark]

Tick every answer that is true.

- ☐ **A** The changed program code
- ☐ **B** Evidence of testing with the data the question gives
- ☐ **C** A copy of the whole skeleton program
- ☐ **D** A flowchart of the change

3. Which is the best way to prepare for questions on a skeleton program? [1 mark]

- ☐ **A** Map its classes and subroutines, trace its main loop, and practise likely changes under time
- ☐ **B** Memorise its code line by line
- ☐ **C** Run it once to see what it does
- ☐ **D** Wait to read it in the exam

4. Part of a changed skeleton. What does it print? [1 mark]

```
class Robot:
    MOVES = {"N": (-1, 0), "S": (1, 0)}
    def __init__(self):
        self.row = 0
    def move(self, d):
        if d not in Robot.MOVES:
            raise ValueError("unknown direction")
        self.row += Robot.MOVES[d][0]
bot = Robot()
for d in ["S", "Q", "S"]:
    try:
        bot.move(d)
        print(d, bot.row)
    except ValueError as e:
        print(f"{d}: {e}")
```

---

5. In the warehouse skeleton, MOVES is defined inside the class but outside any method. What kind of attribute is it? [1 mark]

- ☐ **A** A class attribute, shared by every Robot object
- ☐ **B** A local variable of move
- ☐ **C** A private attribute of one object
- ☐ **D** A global variable

The task: change the skeleton program

The skeleton below simulates a warehouse robot on a grid. `Warehouse(rows, cols)` makes a grid of `"."` squares; `add_shelf(row, col)` puts a shelf `"#"` on a square; `is_free(row, col)` returns `True` if a square on the grid has no shelf; `show(robot)` prints the grid with the robot as `R`. A `Robot` starts at row 0, column 0 with a whole-number `battery`. Row numbers increase going south, and columns increase going east. As in the exam, it has weaknesses. Make these changes. 1. Change `move(self, direction, warehouse)` so that it does these checks, in this order, before anything changes: - if `direction` is not one of the keys of `Robot.MOVES`, raise `ValueError("unknown direction")`; - if `battery` is less than 5, print `battery low` and return without moving; - if the new square would be off the grid, or `warehouse.is_free` says it is not free, print `blocked` and return without moving, using no battery; - otherwise move to the new square and reduce `battery` by 5. 2. Change the main program so that a `ValueError` from `move` is caught: print `<command>: <message>` (for example `X: unknown direction`) instead of the position line, and carry on with the next command. Do not change `Warehouse`, the shelves, the starting battery or the list of commands. The robot on the mat stays still.

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

class Warehouse:
    def __init__(self, rows, cols):
        self.rows = rows
        self.cols = cols
        self.grid = [["." for c in range(cols)] for r in range(rows)]

    def add_shelf(self, row, col):
        self.grid[row][col] = "#"

    def is_free(self, row, col):
        return self.grid[row][col] == "."

    def show(self, robot):
        for r in range(self.rows):
            line = ""
            for c in range(self.cols):
                if r == robot.row and c == robot.col:
                    line += "R"
                else:
                    line += self.grid[r][c]
            print(line)


class Robot:
    MOVES = {"N": (-1, 0), "S": (1, 0), "E": (0, 1), "W": (0, -1)}

    def __init__(self, battery):
        self.row = 0
        self.col = 0
        self.battery = battery

    def move(self, direction, warehouse):
        d_row, d_col = Robot.MOVES[direction]
        self.row += d_row
        self.col += d_col
        self.battery -= 5
```

```
warehouse = Warehouse(4, 5)
for row, col in [(1, 1), (1, 2), (2, 3)]:
    warehouse.add_shelf(row, col)
bot = Robot(20)
for command in ["E", "S", "E", "N", "N", "E", "E", "S", "X", "S"]:
    bot.move(command, warehouse)
    print(command, bot.row, bot.col, bot.battery)
warehouse.show(bot)
```

Plan your program here, then type it in and press Run.

Do it on the robot

www.bugbotlab.com/learn/a15-6-skeleton-programs/

The simulator checks it and tells you when it passes. Nothing to install, no account.

Challenges

1. Add a method `charge(self, amount)` that adds to the battery but never takes it above 100, and test it with 30 and then 90.
2. Answer as if in the exam: "State the name of a class attribute in the skeleton program" and "Explain why `is_free` is a method of `Warehouse` rather than of `Robot`."
3. Add the command `"D"` to drop a parcel on the square the robot is on, recording it in the grid as `"P"`. Where must the change be made so that later moves treat it as blocked?