ANSWERS

A15.4 Pseudocode in the exam languages

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

What this lesson is about

OCR's exam reference language and AQA's pseudo-code side by side with Python, the traps in translating, and a stack class that brings the robot home.

- The two languages side by side
- The traps in translating
- Reading a class in the exam reference language

Questions

5 marks in all

1. How many times does the AQA pseudo-code loop FOR i ← 1 TO 10 run? [1 mark]

- ☐ **A** 10
- ☐ **B** 9
- ☐ **C** 11
- ☐ **D** It depends on the language

> **Answer:** A. Both ends are included, so i takes the values 1 to 10. In Python this is range(1, 11).

2. Which Python matches the OCR loop do ... until reading < 30? [1 mark]

- ☐ **A** while True: with the body, then if reading < 30: break
- ☐ **B** while reading < 30: with the body
- ☐ **C** for reading in range(30): with the body
- ☐ **D** if reading < 30: with the body

> **Answer:** A. A post-condition loop runs the body at least once and stops when its condition becomes true.

3. In AQA pseudo-code, what does x ← 3 do? [1 mark]

- ☐ **A** Assigns 3 to x
- ☐ **B** Checks whether x equals 3
- ☐ **C** Outputs 3
- ☐ **D** Moves x three places left

> **Answer:** A. AQA uses the arrow for assignment and = for comparison.

4. In OCR's exam reference language, what is the name of a class's constructor? [1 mark]

> **Answer:** new. OCR writes public procedure new(...), called with the keyword new, as in myStack = new Stack().

5. This is a stack from the exam reference language, translated. What does it print? [1 mark]

```
items = [None] * 5
top = -1
for move in ["forward", "right", "left"]:
    top = top + 1
    items[top] = move
while top != -1:
    print(items[top])
    top = top - 1
```

Answer:

```
left
right
forward
```

The pointer version still behaves as a stack: the last item pushed is the first popped.

The task: there and back with a stack

Translate the `Stack` class above into Python, then use it to bring the robot home. Write `class Stack` with: - `__init__(self)` : an empty stack; - `push(self, item)` : puts `item` on the top; - `pop(self)` : removes and returns the top item; - `is_empty(self)` : returns `True` if there are no items, otherwise `False` . `route` is a list of moves in order. Each move is a tuple `(direction, cm)` : `direction` is one of `"forward"` , `"backward"` , `"left"` or `"right"` , and `cm` is a whole number of centimetres. Drive each move in order at speed 50, pushing it onto a stack after it is driven. The robot then reaches the charger: play a note and print `arrived` . To come home, loop `while not <stack>.is_empty()` : pop a move, print `undo <direction> <cm>` , and drive the **opposite** direction (forward and backward are opposites, as are left and right) the same distance at speed 50. After the loop print `home` . Do not reverse the list yourself: the stack must do it.

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

route = [("forward", 30), ("right", 20), ("forward", 15), ("left", 10)]
```

The hint students can ask for: Translate the pseudocode one method at a time, keeping its structure. Drive each move of the route and push it as you go. To come back, pop a move, drive the opposite way the same distance, and repeat until the stack is empty: the last move out is the first undone.

A solution

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

class Stack:
    def __init__(self):
        self.__items = []

    def push(self, item):
        self.__items.append(item)

    def pop(self):
        if self.is_empty():
            raise IndexError("pop from an empty stack")
        return self.__items.pop()

    def is_empty(self):
        return len(self.__items) == 0

OPPOSITE = {"forward": "backward", "backward": "forward", "left": "right", "right": "left"}
MOVES = {"forward": forward, "backward": backward, "left": left, "right": right}
route = [("forward", 30), ("right", 20), ("forward", 15), ("left", 10)]

done = Stack()
for direction, cm in route:
    MOVES[direction](50, distance=cm)
    done.push((direction, cm))
tone(880, 0.3)
print("arrived")
while not done.is_empty():
    direction, cm = done.pop()
    print("undo", direction, cm)
```

```
    MOVES[OPPOSITE[direction]](50, distance=cm)
print("home")
```

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