ANSWERS

A14.9 Project: the delivery robot

Software development, law and ethics · A level · OCR H446 1.2.3, AQA 7517 4.13.1.1, Eduqas A500QS 1.5 · about 60 min

What this lesson is about

A small project taken through analysis, design, implementation, testing and evaluation, the way the NEA is written up.

- 1. Analysis
- 2. Design
- 3. Implementation
- 4. Testing
- 5. Evaluation
- Law and ethics in the project

Questions

5 marks in all

1. In a project write-up, what should the evaluation be based on? [1 mark]

- ☐ **A** Each success criterion from the analysis, with evidence of whether it was met
- ☐ **B** A description of every line of code
- ☐ **C** The number of hours spent on the project
- ☐ **D** The programming language chosen

> **Answer:** A. The criteria written in analysis are what testing and evaluation check.

2. Why is it good practice to unit test gap_ok before the robot moves? [1 mark]

- ☐ **A** It is a pure function, so its boundaries can be checked quickly and repeatably without the hardware
- ☐ **B** Unit tests make the robot drive faster
- ☐ **C** The robot cannot move until a test fails
- ☐ **D** It removes the need for final testing

> **Answer:** A. Testing small modules first finds errors early, when they are cheap to fix, before integration.

3. AQA's A level project is marked out of how many marks? [1 mark]

> **Answer:** 75. AQA's NEA is 75 marks and worth 20% of the A level.

4. Put the stages of a project write-up in order. [1 mark]

Number the lines 1 to 5 to put them in the right order.

___ `Implementation`
___ `Analysis`
___ `Design`
___ `Testing`
___ `Evaluation`

Answer:

```
Analysis
Design
Implementation
Testing
Evaluation
```

Each stage follows from the one before: the design meets the analysis, and the evaluation checks the analysis's criteria.

5. The evaluation counts the criteria met. What does this print? [1 mark]

```
results = [("SC1", True), ("SC2", False), ("SC3", True)]
met = 0
for name, ok in results:
    print(name, "met" if ok else "not met")
    met += ok
print(f"criteria met: {met} of {len(results)}")
```

Answer:

```
SC1 met
SC2 not met
SC3 met
criteria met: 2 of 3
```

True counts as 1 and False as 0, so two criteria are met out of three.

The task: the delivery robot

Build the designed program. The robot starts facing the charging wall, 71 cm away. The delivery bay is to the right of where it parks. Write these subroutines, and use them in this order: - `gap_ok(gap)` : `gap` is a number of cm; returns `True` if it is from `GAP_LOW` to `GAP_HIGH` inclusive, otherwise `False` . - `run_unit_tests()` : tests `gap_ok` with exactly these five values, in this order: 16.9 (expected `False`), 17 (`True`), 20 (`True`), 23 (`True`) and 23.1 (`False`). For each, print `test gap_ok(<value>): pass` or `test gap_ok(<value>): FAIL` , then print `unit tests: <passed> of 5 passed` . - `approach(target)` : `target` is the gap in cm to park at; drive towards the wall and return the gap read with `distance()` once stopped. Call it as `approach(20)` . - `sidestep(cm)` : move right `cm` centimetres. Call it as `sidestep(30)` . - `signal()` : LED green and an 880 Hz note. - `evaluate(gap, elapsed)` : `gap` is the value `approach` returned and `elapsed` is `clock()` read after `signal()` . Print `SC1 gap <gap> cm: <result>` using `gap_ok` , `SC2 in the bay after <elapsed> s: <result>` (met if `elapsed` is at most `TIME_LIMIT`), and `SC3 signal: met` , where each `<result>` is `met` or `not met` . Then print `criteria met: <n> of 3` , counting the criteria met. The whole program prints exactly ten lines. SC4 is checked by watching the run, so the program does not print it: do not touch the wall.

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

GAP_LOW, GAP_HIGH = 17, 23     # SC1, in cm
TIME_LIMIT = 30                # SC2, in seconds

def gap_ok(gap):
    pass
```

The hint students can ask for: Build it in the order of the design: write gap_ok and test it before the robot moves, then approach, then the sidestep and signal, and evaluate last. Keep a count of passed tests and of criteria met as you print each line, so the totals come from the counts.

A solution

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

GAP_LOW, GAP_HIGH = 17, 23
TIME_LIMIT = 30

def gap_ok(gap):
    return GAP_LOW <= gap <= GAP_HIGH

def run_unit_tests():
    tests = [(16.9, False), (17, True), (20, True), (23, True), (23.1, False)]
    passed = 0
    for value, expected in tests:
        ok = gap_ok(value) == expected
        passed += ok
        print(f"test gap_ok({value}): " + ("pass" if ok else "FAIL"))
    print(f"unit tests: {passed} of {len(tests)} passed")

def approach(target):
    while distance() > target + 10:
        forward(60, distance=5)
```

```
    while distance() > target + 1:
        forward(30, distance=1)
    return distance()

def sidestep(cm):
    right(50, distance=cm)

def signal():
    led("green")
    tone(880, 0.3)

def evaluate(gap, elapsed):
    results = [
        (f"SC1 gap {gap} cm", gap_ok(gap)),
        (f"SC2 in the bay after {elapsed} s", elapsed <= TIME_LIMIT),
        ("SC3 signal", True),
    ]
    met = 0
    for text, ok in results:
        print(text + ": " + ("met" if ok else "not met"))
        met += ok
    print(f"criteria met: {met} of {len(results)}")

run_unit_tests()
gap = approach(20)
sidestep(30)
signal()
evaluate(gap, clock())
```

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