ANSWERS

# F6.3 Testing and test data

Robust programs · GCSE · OCR J277 2.3.2, AQA 8525 3.2.11, Edexcel 1CP2 6.1.6 · about 15 min

## What this lesson is about

Iterative and final testing; normal, boundary, invalid and erroneous data; test plans in code.

- When to test
- Choosing test data
- A test plan
- Tests the program runs itself
- Testing a robot

## Questions

**5 marks in all**

**1.** A speed must be from 0 to 100. Which is boundary test data? [1 mark]

- ☐ **A** 100
- ☐ **B** 55
- ☐ **C** fast
- ☐ **D** 250

> **Answer:** A. Boundary data sits at the edges of what is allowed, such as 0 and 100 (and just outside them).

**2.** A speed must be from 0 to 100. Which is erroneous test data? [1 mark]

- ☐ **A** fast
- ☐ **B** 100
- ☐ **C** 55
- ☐ **D** 0

> **Answer:** A. Erroneous data is the wrong type altogether.

**3.** What is iterative testing? [1 mark]

- ☐ **A** Testing each part as it is built, and fixing it before moving on
- ☐ **B** Testing once the whole program is finished
- ☐ **C** Testing with random data
- ☐ **D** Testing by a different person

> **Answer:** A. Final (terminal) testing is the check at the end.

**4.** When should the expected result in a test plan be worked out? [1 mark]

- ☐ **A** Before running the test, from the requirements
- ☐ **B** After seeing what the program does
- ☐ **C** Only if the test fails
- ☐ **D** It is not needed

> **Answer:** A. Otherwise it is too easy to decide that whatever happened was right.

**5.** What does this program print? [1 mark]

```
def valid(s):
    return 0 < s < 100

print(valid(0), valid(50), valid(100))
```

**Answer:**

```
False True False
```

< excludes both ends, so 0 and 100 are rejected: a boundary bug.

## The task: fix it with tests

The test program below has a boundary bug. Fix `valid_speed` so every test prints `pass`, without changing the tests. Then add two more tests of your own: one normal, one erroneous. All tests must pass.

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

def valid_speed(text):
    """True if text is a whole number from 0 to 100."""
    return text.isdigit() and 0 < int(text) < 100

tests = [("50", True), ("0", True), ("100", True), ("101", False), ("-5", False), ("fast",
False)]

for number, (data, expected) in enumerate(tests, start=1):
    actual = valid_speed(data)
    result = "pass" if actual == expected else "FAIL"
    print(f"test {number}: {data!r} expected {expected}, got {actual}: {result}")
```

**The hint students can ask for:** The boundary is wrong: the lowest and highest allowed values should pass. Then add tests of your own, each a pair of an input and the answer you expect, including one that should be refused.

## A solution

```
from bugbot import *
connect()
def valid_speed(text):
    """True if text is a whole number from 0 to 100."""
    return text.isdigit() and 0 <= int(text) <= 100

tests = [("50", True), ("0", True), ("100", True), ("101", False), ("-5", False), ("fast",
False)]
tests.append(("75", True))
tests.append(("12.5", False))

for number, (data, expected) in enumerate(tests, start=1):
    actual = valid_speed(data)
    result = "pass" if actual == expected else "FAIL"
    print(f"test {number}: {data!r} expected {expected}, got {actual}: {result}")
```

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