ANSWERS

# A1.3 Exception handling

Programming techniques and object-oriented programming · A level · AQA 7517 4.1.1.9 · about 20 min

BugBotLab

## What this lesson is about

try, except, else and finally, raising your own exceptions, and keeping the motors safe when something goes wrong.

- An unhandled exception
- try and except
- else, finally and as
- Raising your own
- Exceptions or checks?

## Questions

**5 marks in all**

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

```
def half(text):
    try:
        n = int(text)
    except ValueError:
        return -1
    else:
        return n // 2
    finally:
        print("checked", text)

print(half("9"))
print(half("nine"))
```

**Answer:**

```
checked 9
4
checked nine
-1
```

finally runs before each return completes, so checked prints first; 9 // 2 is 4, and "nine" makes int raise ValueError, giving -1.

**2.** When does a finally block run? [1 mark]

- ☐ **A** Every time the try statement finishes, whether or not an exception was raised
- ☐ **B** Only when an exception is raised
- ☐ **C** Only when no exception is raised
- ☐ **D** Only when the exception is not caught

**Answer:** A. finally is for clean-up that must always happen, such as stopping motors.

**3.** Why is a bare except: that catches every exception usually a bad idea? [1 mark]

- ☐ **A** It also hides errors you did not expect, such as a misspelt variable name
- ☐ **B** It makes the program run more slowly
- ☐ **C** Python does not allow it
- ☐ **D** It stops finally blocks from running

> **Answer:** A. Catch only the exceptions you can deal with, so unexpected bugs still show up.

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

```
def check(cm):
    if cm > 50:
        raise ValueError("too far")
    return cm

total = 0
for cm in [20, 80, 30]:
    try:
        total = total + check(cm)
    except ValueError as error:
        print(error)
print(total)
```

> **Answer:**
>
> ```
> too far
> 50
> ```
>
> 80 raises ValueError, so its addition never happens; the loop carries on and the total is 20 + 30.

**5.** Which Python keyword is used to signal an exception yourself, for example when a value is out of range? [1 mark]

> **Answer:** raise. raise creates the exception; a caller's try and except can then handle it.

## The task: a distance that cannot crash the program

Write a function `read_distance(text)` that takes a string `text` and returns it as an integer. It must raise a `ValueError` in two cases: when `text` is not a whole number (let `int` raise it) and when the number is below 1 or above 100 (raise it yourself with `raise ValueError`). In the main program, keep asking `Distance (1 to 100)?` with `input` until `read_distance` succeeds. Each time it raises `ValueError`, print `rejected <text>`, where `<text>` is exactly what was typed. The task types `twenty`, `250`, `-5` and then `25`. When you have a valid distance, drive forward that many cm inside a `try` block whose `finally` block calls `stop()` and prints `motors stopped`.

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

cm = int(input("Distance (1 to 100)? "))
forward(50, distance=cm)
print("motors stopped")
```

**The hint students can ask for:** Put the conversion and the range check inside one function, so both kinds of bad input end up as the same exception. The loop keeps asking until a call succeeds. Think about which code must run however the driving ends.

## A solution

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

def read_distance(text):
    cm = int(text)
    if cm < 1 or cm > 100:
        raise ValueError("out of range")
    return cm

while True:
    text = input("Distance (1 to 100)? ")
    try:
        cm = read_distance(text)
        break
    except ValueError:
        print("rejected", text)

try:
    forward(50, distance=cm)
finally:
    stop()
    print("motors stopped")
```

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