WORKSHEET

A1.3 Exception handling

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

BugBotLab

Name _____ Class _____ Date _____

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"))
```

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

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

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)
```

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

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")
```

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

Do it on the robot

www.bugbotlab.com/learn/a1-3-exception-handling/

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

Challenges

1. Add a second `except` for `TypeError`. Can `read_distance` ever raise one, when `input` always returns a string?
2. Give up after three rejected answers by raising an exception of your own, and catch it in the main program.
3. Put `print(1 / 0)` inside the `try` block that drives. Does `motors stopped` still print? Does the program carry on afterwards?