1.6 Files and errors

Name: _____ Class: _____ Date: _____ **Total: 21 marks**

KEY WORDS file • exception • ValueError

Objective

Read and write text files with `with open(...)`, handle the expected exceptions by name with a sensible response, and let the unexpected ones crash loudly.

1 Key knowledge and worked examples

■ The with-block owns the file

A **file** outlives the program. `with open(path) as f:` opens for reading and closes on exit whatever happens – the file handle never leaks.

```
with open("log.txt", "w") as f:      # "w" writes AND EMPTIES first
    f.write("opened\n")

with open("log.txt") as f:
    for line in f:
        print(line.rstrip())         # rstrip drops the trailing newline
```

The classic accident is opening an existing file with `"w"` meaning to append – `"a"` appends, `"w"` rewrites from empty.

■ Worked example: settings with a fallback

```
try:
    with open("settings.txt") as f:
        setting = f.read().strip()
except FileNotFoundError:
    setting = "default"
print("Using", setting)
```

If the file exists, its first content is used (stripped of whitespace); if not, the program continues with a stated default instead of dying. The response matches the failure: a missing optional file deserves a fallback.

exercises.lol exercises.com.cn

Exceptions you will actually meet

- `FileNotFoundError` – the path is wrong.
- `ValueError` – `int("seven")`: the text is not a number.
- `KeyError` – a dictionary lookup for a key that is not there.
- `TypeError` – mixing types, like `"3" + 4`.

Catch the type you expect, where you can respond:

```
try:
    count = int(input("How many? "))
except ValueError:
    print("Whole numbers only.")
    count = 0
```

A bare `except:` catches everything – including the typo errors you need to see. Never use it.

2 Practice

2.1 Write two lines that append the line `session closed` to `events.txt` without erasing what is there. [2]

2.2 For each failure, name the exception: (a) `int("3.5")`; (b) opening `missing.txt` for reading; (c) `d["x"]` when `d` has no key `"x"`; (d) `"a" - "b"`. [4]

2.3 The settings example catches `FileNotFoundError`. State one reason it should NOT also catch `PermissionError` silently. [2]

3 Exam-style questions

3.1 A program reads exam marks from `marks.txt`, one whole number per line, and prints the average. Some lines may be blank; the file may be missing.

Write the program: handle a missing file with the message `No marks yet` and exit with average 0; skip blank lines; convert and average the rest. Give the output for a file containing `72`, “, `85`, `61`. [7]

3.2 Explain, with one worked example each: (a) why the `with` statement is preferred over `open` without it; (b) why the blank-line case is handled with a condition rather than an exception; (c) what happens if a line contains `7.5` and how you would handle it deliberately. [6]

exercises.lol exercises.com.cn

Solutions

2.1

- `with open("events.txt", "a") as f:`
- `f.write("session closed\n")`

2.2

- (a) `ValueError`.
- (b) `FileNotFoundError`.
- (c) `KeyError`.
- (d) `TypeError`.

2.3

- A permission problem means the file EXISTS but the program is misconfigured – continuing silently would hide a real fault.
- It should crash loudly (or be caught separately with a distinct response).

3.1

```
try:
    with open("marks.txt") as f:
        lines = f.readlines()
except FileNotFoundError:
    print("No marks yet")
    lines = []

marks = []
for line in lines:
    if line.strip():
        marks.append(int(line))

average = sum(marks) / len(marks) if marks else 0
print("Average:", average)
```

- Missing file handled with the message and no crash.
- Blank lines skipped by condition.
- Conversion and collection of the valid marks.
- Zero-division guarded (empty case → 0).
- For 72, blank, 85, 61 → `Average: 72.66666666666667`.

3.2

- (a) `with` closes the file on exit even if an exception fires mid-read – bare `open` leaks the handle if anything goes wrong between open and close.
- (b) A blank line is expected data, not a failure: `if line.strip()` states the intent; wrapping expected cases in try/except hides the program's logic.
- (c) `int("7.5")` raises `ValueError` – use `float(line)` and decide, or catch `ValueError` per line and report it.