Solutions

Each answer shows the steps, then the **result**.

2.1

- `APPEND` **keeps** the existing contents and adds after them; `WRITE` **overwrites** (erases) the file

2.2

- an **error or unexpected condition** that occurs **during execution** (e.g. file not found, divide by zero)

2.3

- code that **always runs** (whether or not an exception happened) — used for **cleanup** such as closing files

2.4

- any one: forgetting to **close** a file (data lost); opening `FOR WRITE` instead of `APPEND` (overwrites); reading past `EOF`; hard-coded paths

3.1

- open, loop until found or EOF, then close and report

```
found ← FALSE
OPENFILE "people.txt" FOR READ
WHILE NOT EOF("people.txt") AND NOT found DO
    READFILE "people.txt", Line
    IF Line = Target THEN
        found ← TRUE
    ENDIF
ENDWHILE
CLOSEFILE "people.txt"
IF found THEN
    OUTPUT "Found"
ELSE
    OUTPUT "Not found"
ENDIF
```

3.2

- `TRY … ENDTRY` with the file read inside and an `EXCEPT` handler

```
TRY
    OPENFILE "data.txt" FOR READ
    READFILE "data.txt", Line
    CLOSEFILE "data.txt"
EXCEPT FileNotFound
    OUTPUT "Sorry, the file does not exist."
ENDTRY
```

3.3

- programs face errors that **cannot be prevented up front** (missing files, bad input)
- handling them lets the program **recover/inform the user** instead of **crashing**, and keeps the normal code clean

3.4

- test `b = 0` and `RAISE`; otherwise return the quotient

```
FUNCTION Divide(a : INTEGER, b : INTEGER) RETURNS INTEGER
    IF b = 0 THEN
        RAISE DivideByZero
    ENDIF
    RETURN a DIV b
ENDFUNCTION
```

3.5

- `SEEK` moves to a given **record number / position** in the file
- `PUTRECORD` **writes a record** at that position