Solutions

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

2.1

- a **syntax error** breaks the language's grammar and is caught at **translation** (the program won't run)
- a **logic error** lets the program run but produces the **wrong output**

2.2

- any one: divide by zero; array index out of range; file not found; invalid type conversion

2.3

- any valid set, e.g. normal `40`; boundary `1` or `120`; abnormal `0`, `121` or `"abc"`

2.4

- tracing the code **by hand on paper**, recording each variable's value (usually in a trace table)

3.1

(a) **syntax** error

(b) **run-time** error

(c) **logic** error

3.2

- any valid plan, e.g. normal `50` (inside the range); boundary (low) `0` (lowest accepted); boundary (high) `100` (highest accepted); abnormal `150` or `-5` (outside — must be rejected)

3.3

- **corrective** — fixing bugs found in use
- **adaptive** — changing it to keep working in a new environment (new OS, law)
- **perfective** — improving features or performance

3.4

- that a **change has not broken** existing, previously-working features

3.5

(a) a **logic error**

- the program translates and runs to completion, so the syntax is valid and nothing crashes, but the output is wrong

(b) any two, developed: **dry run / trace table** — work through the code by hand with sample values, recording each variable, which exposes a division by the wrong count

- **white-box testing with a debugger** — set a breakpoint and step through, watching the running total and the counter

(c) any two, developed: **modular design** — split the calculation into a small function that can be tested on its own

- **meaningful identifiers and comments**, so `Total / Count` is obviously wrong when `Count` is zero

(d) add a variable to hold the maximum, initialised to the **first mark** rather than zero

- inside the existing loop compare each mark and update it when the mark is larger
- extend the output statement
- then **re-run the existing tests** for the average — regression testing — to confirm the change has not broken what already worked

(e) it adds new **functionality** that the user has asked for, rather than fixing a fault (corrective) or responding to a change in hardware or OS (adaptive)