

Solutions

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

2.1

- IF ... ELSE ... ENDIF with the pass/fail condition

```
IF Mark >= 50 THEN
    OUTPUT "Pass"
ELSE
    OUTPUT "Fail"
ENDIF
```

2.2

- CASE OF ... ENDCASE with two value branches and OTHERWISE

```
CASE OF Grade
    "A": OUTPUT "Excellent"
    "B": OUTPUT "Good"
    OTHERWISE: OUTPUT "Try again"
ENDCASE
```

2.3

- WHILE (pre-condition) may run **zero** times
- REPEAT ... UNTIL (post-condition) runs **at least once**

2.4

- FOR ... STEP 2 from 2 to 20

```
FOR i ← 2 TO 20 STEP 2
    OUTPUT i
NEXT i
```

- (*outputting i only when $i \bmod 2 = 0$ also scores*)

3.1

- CASE OF ... ENDCASE with three branches, the right outputs, and OTHERWISE

```
CASE OF Choice
    1: OUTPUT "New"
    2: OUTPUT "Open"
    3: OUTPUT "Save"
    OTHERWISE: OUTPUT "Invalid"
ENDCASE
```

3.2

- a **post-condition** (REPEAT ... UNTIL) loop, because the password must be entered **at least once** before it can be tested

```

REPEAT
    INPUT Password
UNTIL Password = "OPEN"

```

3.3

- start Total at 0; loop while it is under 100; add each input

```

Total ← 0
WHILE Total < 100 DO
    INPUT n
    Total ← Total + n
ENDWHILE

```

3.4

- B doubles on each pass until A reaches 5

A	B	OUTPUT
1	1	
2	2	
3	4	
4	8	
5	16	16

- stops when $A = 5$; OUTPUT **16**

3.5

(a)

```

PROCEDURE Tick()
    SS ← SS + 1
    IF SS = 60
        THEN
            SS ← 0
            MM ← MM + 1
            IF MM = 60
                THEN
                    MM ← 0
                    HH ← HH + 1
                    IF HH = 24 THEN
                        HH ← 0
                    ENDIF
                ENDIF
            ENDIF
        ENDIF
    ENDPROCEDURE

```

- increment seconds; reset seconds and carry into minutes; reset minutes and carry into hours; hours wrap at 24; the carries nested, not independent; correct pseudocode syntax

(b) call 1: 10 59 59

- call 2: 11 00 00
- call 3: 11 00 01

(c) the minutes can only roll over **because** the seconds just did, and the hours only because the minutes did

- if the three were tested independently, MM would be checked against 60 before it had been incremented on this tick, so the rollover would be a second late
- testing hours first would let the clock show 11:59:60 for one tick

(d) any global variable can be changed by **any** part of the program, so a fault anywhere can corrupt the time and be hard to trace

- pass the three values in as parameters and return the updated time — or hold them in one record passed by reference — so only `Tick()` can change them

(e) a **post-condition** loop, REPEAT ... UNTIL Stopped, because the clock must tick at least once before the user has any chance to stop it