MODULE O ROBUST PROGRAMS · O1

Defensive design: anticipating misuse: mark scheme

21 marks · OCR J277 specification 2.3.1 (defensive design considerations: anticipating misuse; issues a programmer should consider to ensure a program caters for all likely input values; how to deal with invalid data in a program; practical experience of designing input validation), with 2.2.3 (arrays) in Q4

How to use. 1 mark per bullet up to the maximum shown. **/** separates alternative words or answers. Words in (brackets) are not needed for the mark. **...** means the mark depends on the one before. Code may be in OCR Exam Reference Language or any high-level language: mark the logic, and do not penalise minor syntax slips (e.g. a missing `then`, `endif`, colon or bracket); penalise a repeated slip once, then follow through. Credit any other valid answer that shows the same understanding.

QUESTION	ANSWER	MARKS	GUIDANCE
1(a) J277 2.3.1	Max 2 from: Thinking ahead about / predicting how users might use the program wrongly / enter unexpected / invalid data Considering all likely input values (including invalid ones) ... and designing / writing the program so that it caters for / handles them (e.g. rejects them, does not crash)	**2**	The third bullet can be given with either of the first two. Allow misuse that is accidental (mistakes) or deliberate (e.g. trying to break in / hack). Do not allow "stopping people misusing the program" on its own with no idea of planning for it. Do not allow answers about the Computer Misuse Act.
1(b) J277 2.3.1	1 mark for a method, 1 mark for the expansion. For example: Limit the number of password attempts ... ... e.g. lock the account / stop logins after 3 wrong attempts / until an administrator unlocks it OR Add a delay after each wrong attempt ... ... so guessing many passwords takes too long OR Ask for a second factor / code (sent to the staff member) ... ... so a guessed password alone does not give access	**2**	Allow any number of attempts as the example. Allow "require strong / long passwords" ... "so they are harder to guess" for 2. Allow "do not say whether the username or the password was wrong" ... "so the attacker learns less from each guess" for 2. Do not allow "use a firewall / anti-malware" or "encrypt the password": they do not stop repeated guessing.

QUESTION	ANSWER	MARKS	GUIDANCE
2(a) J277 2.3.1	1 mark per row: **Hours:** any value outside 1 to 8, e.g. 0 / 9 / −2 // a non-whole number, e.g. 2.5 // text, e.g. "three" // left blank **Type of bike:** any choice other than 1, 2 or 3, e.g. 4 / 0 // text, e.g. "tandem" / "E" // left blank **Surname:** left blank / nothing entered // far too long, e.g. 500 characters // only digits or symbols, e.g. "12345" / "??"	**3**	The answer must be an actual value or a clear description of one ("nothing entered", "a 500-letter name"). Do not allow vague answers such as "a wrong number", "invalid data" or "a mistake". Do not allow valid values (1 to 8 for hours; 1, 2 or 3 for bike type; a normal surname). Surname: do not allow a surname with a hyphen, apostrophe or space (e.g. O'Neil, Smith-Jones): these are real surnames the program should accept.
2(b) J277 2.3.1	Max 3 from: Check / validate the input (before it is used) Reject the value / do not accept / do not store / do not use it Output an error message (that says what is wrong / what is allowed, e.g. "Enter 1 to 8") Ask the customer to enter the value again / re-prompt Repeat (in a loop) until a valid value is entered The program should not crash / should keep running	**3**	Do not allow "it ignores it" on its own, or "the program stops / closes". Do not allow answers about choosing test data: this question is about how the running program deals with the value.
3(a) J277 2.3.1	0	**1**	With `count = 0` the loop does not run and line 07 divides by 0.
3(b) J277 2.3.1	Any one from: A negative number, e.g. −3 Text / a non-number, e.g. "five" A decimal / real number, e.g. 2.5 Nothing / left blank	**1**	Do not allow 0 (already given in (a)). Allow a very large number, e.g. 1 000 000, if the answer says it is unrealistic for the number of readings. Do not allow a positive whole number on its own (e.g. 4).

QUESTION	ANSWER	MARKS	GUIDANCE

3(c)
J277 2.3.1
	1 mark each:
	Input of `count` (stored as a whole number)
	Attempt at a loop that repeats the input
	... with a correct condition: repeats while `count < 1` / `count <= 0` // until `count >= 1` / `count > 0`
	Error message output AND count input again inside the loop
	Example answer:
	`count = int(input("How many?"))`
	`while count < 1`
	`  print("Must be 1 or more")`
	`  count = int(input("How many?"))`
	`endwhile`
	Alternative:
	`do`
	`  count = int(input("How many?"))`
	`  if count < 1 then`
	`  print("Must be 1 or more")`
	`  endif`
	`until count >= 1`
		4	A single `if` with one re-input (no loop) cannot get marks 2 or 3; it can get marks 1 and 4.
			Allow `count = input(...)` without `int()` (the question says assume a whole number).
			Do not allow ≤, ≥ or => written as maths symbols.
			Do not allow `count <= 1` / `count > 1` (these reject 1, which is valid).
			In a do-until answer, the error message must only show when the value is invalid.

4(a) J277 2.3.1, 2.2.3	Max 2 from: `slot - 1` is 6 / the program looks for `snacks[6]` The array only has indexes 0 to 5 / there is no element at index 6 / the index is outside the array (out of range) The value 7 was not anticipated / not checked before it was used	**2**	Allow "there are only 6 items in the array" for the second bullet. Do not allow "7 is not a valid number" on its own.

QUESTION	ANSWER	MARKS	GUIDANCE

4(b)
J277 2.3.1, 2.2.3
	1 mark each:
	Attempt at selection (`if`) checking the value of `slot`
	... correct condition for a valid slot: `slot >= 1 AND slot <= 6`
	Outputs `"You chose " + snacks[slot - 1]` when valid AND an error message otherwise
	Example answer:
	`if slot >= 1 AND slot <= 6 then`
	`  print("You chose " + snacks[slot - 1])`
	`else`
	`  print("Invalid slot")`
	`endif`
		3	Allow equivalent conditions: `slot > 0 AND slot < 7`; `slot - 1 >= 0 AND slot - 1 <= 5`; or the reverse test `slot < 1 OR slot > 6` with the outputs swapped.
			Do not allow a condition that checks only one end (e.g. `slot <= 6`): it still lets 0 through. Do not allow `slot >= 1 AND <= 6` (second half has no variable).
			Allow a loop that re-asks for the slot until it is valid, with an error message, followed by the output line, for all 3 marks.

Total: 21 marks