MODULE M PROGRAMMING · M22

Procedures: mark scheme

22 marks · OCR J277 spec 2.2.3 (the use of procedures; how to use sub programs to produce structured code), 2.3.1 (maintainability: use of sub programs), 2.1.2 (refining algorithms)

How to use. 1 mark per bullet up to the Max shown. **/** separates alternative answers; **//** separates alternative statements for the same mark. Words in (brackets) are not needed for the mark. **...** at the end of one point and the start of the next means the second depends on the first. Code answers are marked for logic, not exact syntax: accept OCR Exam Reference Language or any high-level language, and allow any logically valid method. Minor syntax slips are not penalised; a repeated slip is penalised once then follow through. Capitals in keywords are ignored. Credit any other valid answer.

QUESTION	ANSWER	MARKS	GUIDANCE
1(a) J277 2.2.3	Must have at least one parameter: **False** Can be called more than once: **True** Code only runs when the procedure is called: **True**	**3**	1 mark per correct row. Two ticks in a row: 0 for that row. Row 1: procedures can be written with or without parameters, e.g. `procedure agePass()` in the J277 spec.
1(b) J277 2.2.3	A function returns a value (to where it was called) but a procedure does not // a procedure does not return a value but a function does	**1**	Max 1. Accept "a function call can be used in an expression / assigned to a variable, but a procedure call cannot". NE: "a function does a calculation", "a procedure does a task" (both can). Do not accept "a procedure has no parameters" (wrong).

2(a) J277 2.2.3, 2.1.2	1 mark each to max 4:
	`----------` then `Spring Craft Fair` as the first two lines, in that order
	`Ticket 1` followed by `----------`
	`Ticket 2` followed by `----------`
	`Enjoy your visit` as the last line, output once only
	Full output:
	`----------`
	`Spring Craft Fair`
	`Ticket 1`
	`----------`
	`Ticket 2`
	`----------`
	`Enjoy your visit`
		4	Ignore the number of dashes. Ignore quotation marks round the output. Do not award a mark if the line it depends on is missing or in the wrong order (e.g. `Spring Craft Fair` before the first line of dashes = 0 for the first point). Output that includes `print(...)` or the code itself: penalise once.

QUESTION	ANSWER	MARKS	GUIDANCE
2(b) J277 2.2.3	1 mark each to max 2: The procedure `printHeader()` is called // the program jumps to (line 04,) `printHeader()` The code inside the procedure (lines 05 to 06) runs // it calls `printLine()` and then outputs `Spring Craft Fair` When the procedure ends, the program returns to / carries on from the line after the call (line 09)	**2**	NE: "it prints the header" alone (no idea of calling). Do not accept "the procedure is defined / created on line 08". Do not accept "the program returns to line 08 and runs it again" for the third point.

3(a)
J277 2.2.3
	1 mark each to max 3:
	Procedure header with the identifier `showMenu`: `procedure showMenu()` / `def showMenu():`
	Outputs `1 Borrow`, `2 Return` and `3 Renew`, in this order, inside the procedure
	End of the procedure shown: `endprocedure` / end of the indented block / `End Sub`
	Example:
	`procedure showMenu()`
	`  print("1 Borrow")`
	`  print("2 Return")`
	`  print("3 Renew")`
	`endprocedure`
		3	Accept the three lines as one output with line breaks (e.g. `\n`). Ignore small differences in spacing/case of the menu text. Missing quotation marks round the text: penalise once. Do not award the header mark for `function showMenu()` or a header with a parameter that is not used. A procedure that also returns a value: max 2. Python `def` with the body indented gets the end mark.

3(b) J277 2.2.3	`showMenu()`	**1**	Accept `call showMenu()` / `Call showMenu()` (VB). BOD `showMenu` without brackets. Do not accept the header `procedure showMenu()`, or `x = showMenu()` (a procedure does not return a value).

QUESTION	ANSWER	MARKS	GUIDANCE

4(a)
J277 2.2.3, 2.1.2
	1 mark each to max 4:
	Procedure `openGate()` defined, with an end (`procedure openGate()` ... `endprocedure` / `def openGate():` with indented body)
	... containing the three repeated outputs (`Gate opening`, `Please walk through`, `Gate closing`) in this order, and no other outputs
	`openGate()` called in the CARD branch **and** in the CODE branch (after the accepted message), replacing the repeated lines
	Rest of the program kept: input, selection, `Card accepted` / `Code accepted` / `Please ask at reception`, and the gate is **not** opened for any other input
	Example:
	`procedure openGate()`
	`    print("Gate opening")`
	`    print("Please walk through")`
	`    print("Gate closing")`
	`endprocedure`
	
	`method = input("Enter CARD or CODE")`
	`if method == "CARD" then`
	`    print("Card accepted")`
	`    openGate()`
	`elseif method == "CODE" then`
	`    print("Code accepted")`
	`    openGate()`
	`else`
	`    print("Please ask at reception")`
	`endif`
		4	The procedure may be written before or after the main program. Accept any logically equivalent structure, e.g. `if method == "CARD" OR method == "CODE"` with a nested `if` for the accepted message and one call to `openGate()`, as long as the output is the same for every input. Call made in only one branch: do not award the third mark. Call placed before the accepted message (changes the order of output): do not award the third mark. `openGate()` called in the `else` branch, or after `endif` for every input: do not award the fourth mark. Do not accept flowcharts, structured English or bullet points.

4(b) J277 2.3.1, 2.2.3	1 mark for each benefit, 1 mark for an explanation of it in this program, two benefits, max 4: The three lines are written only once ... // less repeated code ... ... so the program is shorter / quicker to write (the gate code is not typed out twice) Easier to maintain / update ... ... e.g. if a gate message changes it only needs changing in one place // both branches always give the same gate messages / no chance of changing one copy and forgetting the other Easier to test / debug ... ... the gate code can be tested once on its own // an error in the gate code only needs fixing in one place Easier to read / understand / more structured ... ... the name `openGate()` shows what that block of code does // the main program is shorter and easier to follow Can be reused ... ... e.g. called again if another way of entering is added (such as a phone pass) / used in other programs	**4**	Mark in pairs: the explanation mark needs its matching benefit. Max 2 if no answer refers to this program (the gate, its messages, the two branches). Do not award the same idea twice (e.g. "shorter" and "less code"). NE: "faster", "more efficient", "uses less memory" on their own. Do not accept "the program runs faster" (it does not).

Total: 22 marks