Pearson

International Advanced Subsidiary in Computer Science

Unit 2: Practical Programming and Problem-solving

Mark Scheme

General marking guidance

- All candidates must receive the same treatment. Examiners must mark the first candidate in exactly the same way as they mark the last.
- Mark schemes should be applied positively. Candidates must be rewarded for what they have shown they can do rather than penalised for omissions.
- Examiners should mark according to the mark scheme not according to their perception of where the grade boundaries may lie.
- There is no ceiling on achievement. All marks on the mark scheme should be used appropriately.
- All the marks on the mark scheme are designed to be awarded. Examiners should always award full marks if deserved, i.e. if the answer matches the mark scheme. Examiners should also be prepared to award zero marks if the candidate's response is not worthy of credit according to the mark scheme.
- Where some judgement is required, mark schemes will provide the principles by which marks will be awarded and exemplification may be limited.
- When examiners are in doubt regarding the application of the mark scheme to a candidate's response, the team leader must be consulted.

This mark scheme includes colour coded text. Centres must ensure that all hard copies are printed in full colour so that assessment guidance is accurately conveyed.

Question number	Mark Point	Appx. Line	Answer Award marks as shown	Additional Guidance	Mark
1	1.1	10	A comma is added after 4th occurrence of None `None, None, None, None, None, None, None, None, None, None`		12
	1.2	20	Multiplication by 1/5 (* 0.5) is removed from range argument `range(STACK_DEPTH):`		
	1.3	23	The -9 is changed to -1 `top_stack = -1`		
	1.4	29	Item is converted from None to string `str(item)`		
	1.5	33	A semicolon is added to end of definition line `def is_empty():`		
	1.6	45	The ! symbol is removed from right of = symbol `status = True`		
	1.7	54	The +2 is changed to + 1 `top_stack + 1`		

Question number	Mark Point	Appx. Line	Answer Award marks as shown	Additional Guidance	Mark
1 *(continued)*	1.8	62	The global keyword is added for top_stack `global top_stack`		12
	1.9	65	Not is added to negate returned value `if (not is_empty()):`		
	1.10	91	The identifer name is corrected by removing capitalisation `clean_stack()`		
	1.11	94	A closing closing double quote is added to string `"Banana"`		
	1.12	95	The argument "Apple" is removed `show_stack()`		

```
# ------------------------------------------------------------------------------
# Constants
# ------------------------------------------------------------------------------
STACK_SIZE = 10            # Number elements that can be held

# ------------------------------------------------------------------------------
# Global variables
# ------------------------------------------------------------------------------
# Fixed-length list
my_stack = [None, None, None, None, None, None, None, None, None, None]

top_stack = -1           # No items in the list; tracking indexes

# ------------------------------------------------------------------------------
# Subprograms
# ------------------------------------------------------------------------------
def clean_stack():
    global top_stack

    for index in range(STACK_SIZE):
        my_stack[index] = None

    top_stack = -1

def show_stack():
    out_string = ""

    for item in my_stack:
        out_string = out_string + str(item) + " "
    print(out_string)
    print("top = " + str(top_stack))

def is_empty():
    status = False

    if(top_stack == -1):
        status = True

    return(status)

def is_full():
    status = False

    if(top_stack == STACK_SIZE - 1):
        status = True

    return(status)

```

```
49  def push(p_item):
50      global top_stack
51      status = None
52
53      if(not is_full()):
54          top_stack = top_stack + 1
55          my_stack[top_stack] = p_item
56      else:
57          status = False          # Can't push
58
59      return(status)
60
61  def pop():
62      global top_stack
63      status = "Empty"
64
65      if(not is_empty()):
66          item = my_stack[top_stack]
67          top_stack = top_stack - 1
68          return(item)
69      else:
70          return(status)
71
72  # --------------------------------------------------------------------------
73  # Main program
74  # --------------------------------------------------------------------------
75  print("-"*50)
76  print ("Logic Test 1: Before the code is corrected, the top of stack is reporte
77  clean_stack()
78  show_stack()
79
80  # ----------------------
81  print("-"*50)
82  print ("Logic Test 2: Before the code is corrected, the stack is empty so nothi
    pointer is adjusted in error")
83  clean_stack()
84  show_stack()
85  print (pop())
86  show_stack()
87
88  # ----------------------
89  print("-"*50)
90  print ("Logic Test 3: Before the code is corrected, positions in the stack are
91  clean_stack()
92  show_stack()
93  push("Apple")
94  push("Banana")
95  show_stack()
96
```

Question number	Mark Point	Appx. Line	Answer Award marks as shown	Additional Guidance	Mark
2			**Check for Palindrome**	• Do not award marks in this section, where the order of the lines matches the original student code	15
	2.1	17	Definition line is the first line in the subprogram `def is_palindrome (p_string):`		
	2.2	18	Length is calculated before any other instruction `length = len(p_string)`		
	2.3	19 21 23	The selection statement keywords are placed in order as: 1. If 2. Elif 3. Else	• Must have at least one instruction indented under each keyword, i.e. no empty subblock • Ignore accuracy of indented code	

	2.4	20	The return (True) is placed inside the if selection sublock
			
			`if (length < 2):`
			`  return (True)`		

	2.5	21
		22
			The return (recursive call) is placed inside the elif sublock
			
			`elif (p_string[0] == p_string[length - 1]):`
			`  return (is_palindrome(p_string[1:length - 1]))`		

Question number	Mark Point	Appx. Line	Answer Award marks as shown	Additional Guidance	Mark

2
(continued)
	2.6	23
		24
			The return (False) is placed inside the else sublock
			
			`else:`
			`    return (False)`
					15

			Prepare String		
	2.7	29	The correct line has the # symbol removed `def prepare_string(p_string):`	• 2nd in sequence of four	
	2.8	36	The correct line has the # symbol removed `for index in range(len(p_string)):`	• 1st in sequence of four	
	2.9	46	The correct line has the # symbol removed `if (char.isalnum()):`	• 4th in sequence of four	
	2.10	51	The correct line has the # symbol removed `new_string = new_string + char`	• 3rd in sequence of four	
	2.11	58	The correct line has the # symbol removed `new_string = new_string.lower()`	• 4th in sequence of four	

Question number	Mark Point	Appx. Line	Answer Award marks as shown	Additional Guidance	Mark
2 *(continued)*			**Main Program**	• Do not award marks in this section, where the order of the lines matches the original student code	15

	2.12	68	Input, while, input loop repeats until empty string entered
			
			`input_string = input("Enter a word or string: ")`
			`while (input_string != ""):`
			`. . .`
			`    input_string = input("Enter a word or string: ")`
				• While immediately follows input AND input is last instruction INSIDE while loop
				• Ignore intervening lines
				• Ignore over indentation	

	2.13	70
		74
			String must be prepared before checking to see if it is a palindrome
			
			`the_string = prepare_string(input_string)`
			`. . .`
			`if (is_palindrome(the_string)):`
				• Strict order only
				• Ignore intervening lines
				• Ignore indentation errors	

			Overall		
	2.14		The solution: • is a reasonable attempt to solve the problem • must translate without syntax errors	• For this mark, some amendments to the code must have been attempted • Indentation errors are syntax errors • Ignore functionality	

Question number	Mark Point	Appx. Line	Answer Award marks as shown	Additional Guidance	Mark
2 *(continued)*	2.15		Fully functional	Works with test data given in the question paper	15

Input	Output
civic	Original: civic Prepared: civic is a palindrome
CiVic	Original: CiVic Prepared: civic is a palindrome
No melon, no lemon	Original: No melon, no lemon Prepared: nomelonnolemon is a palindrome
Coffee	Original: Coffee Prepared: coffee is not a palindrome
34$543	Original: 34$543 Prepared: 34543 is a palindrome
<empty>	Program terminates

```
# -------------------------------------------------------------------------
# Constants
# -------------------------------------------------------------------------
SPACE = " "

# -------------------------------------------------------------------------
# Global variables
# -------------------------------------------------------------------------
input_string = ""
the_string = ""

# -------------------------------------------------------------------------
# Subprograms
# -------------------------------------------------------------------------

# =====> Rearrange these lines
def is_palindrome(p_string):
    length = len(p_string)
    if (length < 2):
        return (True)                # Base case
    elif (p_string[0] == p_string[length - 1]):
        return (is_palindrome(p_string[1:length - 1]))    # Next two from each end match
    else:
        return (False)               # Keep moving along
# End rearranging

# =====> Choose the correct line for the subprogram definition
#def prepare_string(new_string):
def prepare_string(p_string):
#def prepare_string(the_string):
#def prepare_string(input_string):

    new_string = ""

    # =====> Choose the correct line for the loop
    for index in range(len(p_string)):
    #for index in range(1, len(p_string)):
    #for index in range(1, len(p_string) - 1):
    #for index in range(len(p_string) - 1):
        char = p_string[index]

        # =====> Choose the correct line to deal with spaces and punctuation
        #if (char.isspace()):
        #if (char.isalpha()):
        #if (char.strip(SPACE)):
        if (char.isalnum()):

            # =====> Choose the correct line to create the
            #new_string = char + [new_string]
            #new_string = str(char), new_string
            new_string = new_string + char
            #new_string = [char, new_string]

    # =====> Choose the correct line to produce the required case
    #new_string = new_string.islower()
    #new_string = new_string.isupper()
    #new_string = not new_string.upper()
    new_string = new_string.lower()

    return (new_string)
# End choosing the correct line

# -------------------------------------------------------------------------
# Main program
# -------------------------------------------------------------------------

# =====> Rearrange these lines
input_string = input("Enter a string: ")
while (input_string != ""):  # Loop until empty string inputted
    the_string = prepare_string(input_string)
    if (the_string == ""):       # Validate input
        print("Invalid string.  Must be only letters and digits.  Must not be blank")
    else:                        # Process and report
        if (is_palindrome(the_string)):
            print("Original: " + input_string + " Prepared: " + the_string + " is a palindrome")
        else:
            print("Original: " + input_string + " Prepared: " + the_string + " is not a palindrome")
    input_string = input("Enter a string: ")
# End rearranging

```

Question number	Mark Point	Appx. Line	Answer Award marks as shown	Additional Guidance	Mark
3			**Anywhere within the code**	• Double quotes and single quotes are interchangeable, as long as they are paired accurately • Ignore omission of ; • Ignore lowercase keywords • Ignore quotation marks, except where noted	15

	3.1	40, 42
		49, 51
		68, 72
		75, 78, 80
			The line to execute a query is completed correctly
			
			`db_cur.execute(db_query)`
			
			The line to retrieve query results is completed correctly
			
			`db_result = db_cur.fetchall()`
				• Both lines required
				• Award even if query is incorrect
				• Award only one instance
				• Allow fetchone() for fetchall() for this mark	

	3.2	65 85	The line to call the show_result is added `show_result(db_result)`	• Award even if query fails • Award one instance only	
	3.3	46 55	An output message that is fit for purpose is displayed `print("Product 9 has sold " + str(db_result[0][0]) + " times.")`	Only one instance required for this mark Meaningful for the user and informative	

Question number	Mark Point	Appx. Line	Answer Award marks as shown	Additional Guidance	Mark
3 *(continued)*			**Number of different products**		15
	3.4	40	The line is completed to identify the keywords for a query and specify a table `"SELECT COUNT (*) FROM tbl_products;"`	• Award red only	
	3.5	40	The line is completed to find the total number `"SELECT COUNT (*) FROM tbl_products;"`	• Award red only • Award field name for *	
			Number of Product ID 9 sold		
	3.6	49	The line is completed to find the sum `"SELECT SUM (quantity) FROM tbl_order_products WHERE product_ID = 9;"`	• Award red only	
	3.7	49	The line is completed to find the number of product number 9 sold `"SELECT SUM (quantity) FROM tbl_order_products WHERE product_ID = 9;"`	• Award red only	

Question number	Mark Point	Appx. Line	Answer Award marks as shown	Additional Guidance	Mark
3 *(continued)*			**Customers in postcodes 608087%**		15
	3.8	58	The line is completed to find all the customers with a similar postcode `"SELECT * FROM tbl_customers WHERE postcode LIKE '608087%';"`	• Award red only	
	3.9	58	The line is completed to set the postcode pattern to match `"SELECT * FROM tbl_customers WHERE postcode LIKE '608087%';"`	• Award red only • Clause must be accurate with single or double quotes around postcode and with single % symbol at the end	
			Add the employee table		
	3.10	68	The line is completed to create the employee table `'CREATE TABLE "tbl_employees" ("employee_ID" INTEGER NOT NULL, "address_1" TEXT, "address_2" TEXT, "postcode" TEXT, PRIMARY KEY("employee_ID"));`	• Award red only	

Question number	Mark Point	Appx. Line	Answer Award marks as shown	Additional Guidance	Mark
3 *(continued)*	3.11	68	The line is completed to set the primary key for the employee table `'CREATE TABLE "tbl_employees" ("employee_ID" INTEGER NOT NULL,  "address_1" TEXT, "address_2" TEXT, "postcode" TEXT, PRIMARY KEY("employee_ID"));`	• Award red only	15
			Who are the wood workers?		
	3.12	75	The line is completed with the keywords, fields, and table name required for the query `"SELECT trading_name FROM tbl_makers"`	• Award red only	

	3.13	76
		77
			The inner join is completed with the two table names for each side of the join
			
			`"INNER JOIN tbl_crafts ON tbl_makers.maker_id = tbl_crafts.maker_id"`
			
			`"INNER JOIN tbl_specialisms ON tbl_specialisms.specialism_id = tbl_crafts.specialism_id"`
				• Award red only
				• Same field on two tables
				• Same field on two tables
				• Table names must be supplied or the field names will conflict	

Question number	Mark Point	Appx. Line	Answer Award marks as shown	Additional Guidance	Mark
3 *(continued)*	3.14	78	The line is completed to specify the condition for the query `"WHERE tbl_specialisms.description = 'Woodworking';"`	• Award red only • Must have double or single quote around 'Woodworking'	15
	Functionality and refinement				
	3.15		The outputs are accurate	• Ignore formatting	

Test Data

Question	Query	Expected output
How many different products are there?	`"SELECT COUNT (*) FROM tbl_products;"`	There are 12 different products.
How many of product_ID 9 have been sold?	`"SELECT SUM (quantity) FROM tbl_order_products WHERE product_ID = 9;"`	Product 9 has sold 5 times.
Who are all the customers in postcodes that start with: 608087	`"SELECT * FROM tbl_customers WHERE postcode LIKE '608087%';"`	The customers who have postcodes that start with 608087 are: (327675, 'Evans', 'Erin', '6080873745') (534980, 'Adams', 'Aspen', '6080878902') (571526, 'Franks', 'Faye', '6080872932')

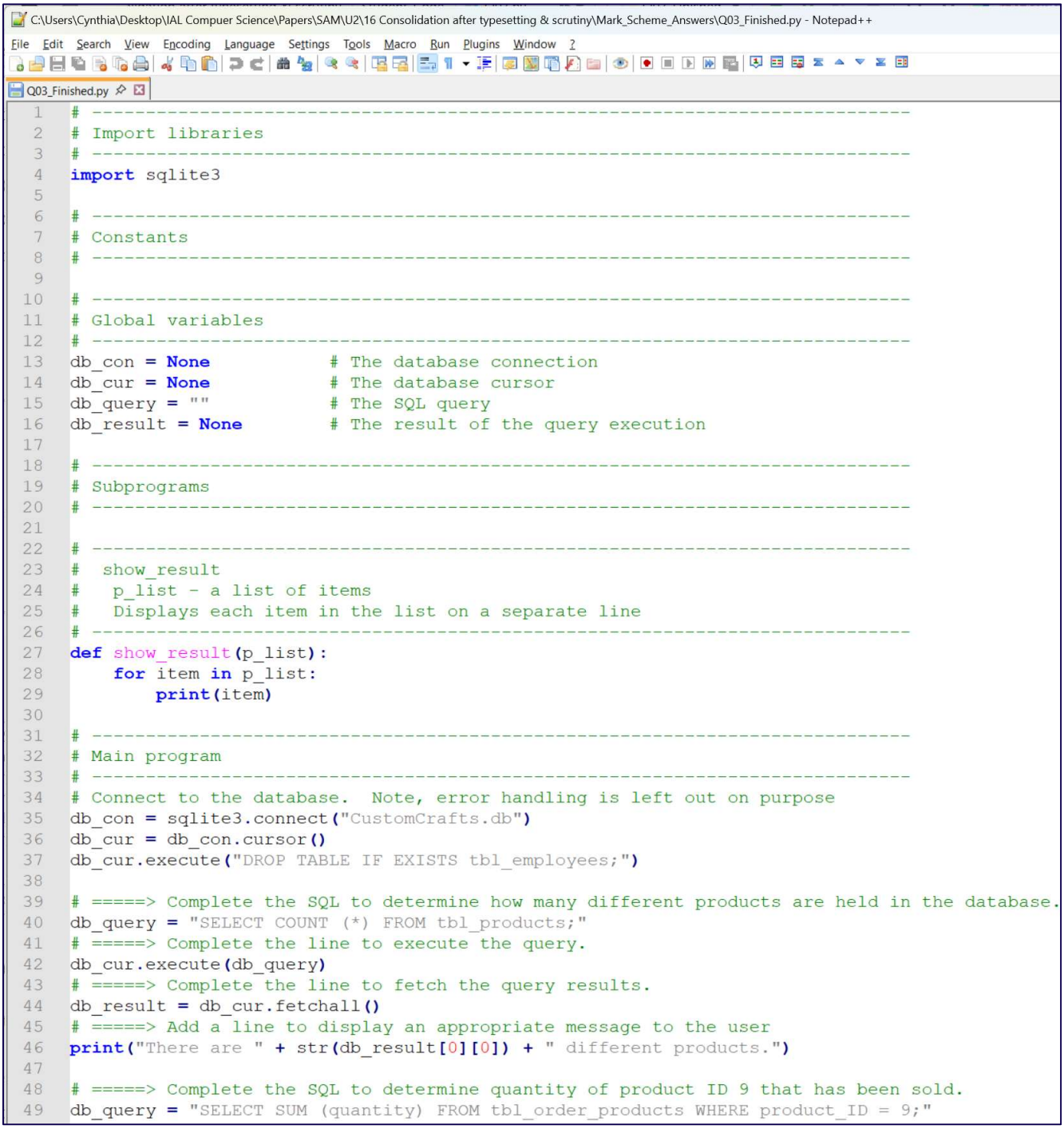
Question	Query	Expected output

Add a table for the employees	`'CREATE TABLE "tbl_employees" ("employee_ID" INTEGER NOT NULL, ' + \`
	`    '"address_1" TEXT, "address_2" TEXT, "postcode" TEXT, ' + \`
	`    'PRIMARY KEY("employee_ID"));'`
		No output to the screen

Who are the woodworkers?	`"SELECT trading_name FROM tbl_makers " + \`
	`    "INNER JOIN tbl_crafts ON tbl_makers.maker_id = tbl_crafts.maker_id " + \`
	`    "INNER JOIN tbl_specialisms ON tbl_specialisms.specialism_id = tbl_crafts.specialism_id " + \`
	`    "WHERE tbl_specialisms.description = 'Woodworking';"`
		The trading names of the makers who specialise in Woodworking are:
		('Natural Materials',)
		('Delicate Designs',)

To see if new table is added, examiner copies this code to the bottom of the file and execute.	`db_query = 'SELECT name FROM sqlite_master WHERE type="table"'`
	`db_cur.execute(db_query)`
	`db_result = db_cur.fetchall()`
	`show_result(db_result)`
	`db_query = 'PRAGMA table_info ("tbl_employees")'`
	`db_cur.execute(db_query)`
	`db_result = db_cur.fetchall()`
	`show_result(db_result)`
		('tbl_specialisms',)
		('tbl_makers',)
		('tbl_customers',)
		('tbl_order_products',)
		('tbl_orders',)
		('tbl_products',)
		('tbl_crafts',)
		('tbl_employees',)
		(0, 'employee_ID', 'INTEGER', 1, None, 1)
		(1, 'address_1', 'TEXT', 0, None, 0)
		(2, 'address_2', 'TEXT', 0, None, 0)
		(3, 'postcode', 'TEXT', 0, None, 0)

tbl_employees		CREATE TABLE "tbl_employees" ("employee_ID" INTEGER NOT NULL, "address_1" TEXT, "address_2" TEXT, "postcode" TEXT, PRIMARY KEY("employee_ID"))
employee_ID	INTEGER	"employee_ID" INTEGER NOT NULL
address_1	TEXT	"address_1" TEXT
address_2	TEXT	"address_2" TEXT
postcode	TEXT	"postcode" TEXT




```python
# ------------------------------------------------------------------------------
# Import libraries
# ------------------------------------------------------------------------------
import sqlite3

# ------------------------------------------------------------------------------
# Constants
# ------------------------------------------------------------------------------

# ------------------------------------------------------------------------------
# Global variables
# ------------------------------------------------------------------------------
db_con = None           # The database connection
db_cur = None           # The database cursor
db_query = ""           # The SQL query
db_result = None        # The result of the query execution

# ------------------------------------------------------------------------------
# Subprograms
# ------------------------------------------------------------------------------

# ------------------------------------------------------------------------------
#  show_result
#   p_list - a list of items
#   Displays each item in the list on a separate line
# ------------------------------------------------------------------------------
def show_result(p_list):
    for item in p_list:
        print(item)

# ------------------------------------------------------------------------------
# Main program
# ------------------------------------------------------------------------------
# Connect to the database.  Note, error handling is left out on purpose
db_con = sqlite3.connect("CustomCrafts.db")
db_cur = db_con.cursor()
db_cur.execute("DROP TABLE IF EXISTS tbl_employees;")

# =====> Complete the SQL to determine how many different products are held in the database.
db_query = "SELECT COUNT (*) FROM tbl_products;"
# =====> Complete the line to execute the query.
db_cur.execute(db_query)
# =====> Complete the line to fetch the query results.
db_result = db_cur.fetchall()
# =====> Add a line to display an appropriate message to the user
print("There are " + str(db_result[0][0]) + " different products.")

# =====> Complete the SQL to determine quantity of product ID 9 that has been sold.
db_query = "SELECT SUM (quantity) FROM tbl_order_products WHERE product_ID = 9;"
```

```
# =====> Complete the line to execute the query.
db_cur.execute(db_query)
# =====> Complete the line to fetch the query results.
db_result = db_cur.fetchall()
# =====> Add a line to display an appropriate message to the user.
print("Product 9 has sold " + str(db_result[0][0]) + " times.")

# =====> Complete the SQL to determine all the customers who have postcodes that start with: 60808
db_query = "SELECT * FROM tbl_customers WHERE postcode LIKE '608087%';"
# =====> Complete the line to execute the query.
db_cur.execute(db_query)
# =====> Complete the line to fetch the query results.
db_result = db_cur.fetchall()
print ("The customers who have postcodes that start with 608087 are: ")
# =====> Add a line to call the show_result subprogram to display the results.
show_result(db_result)

# =====> Complete the SQL to add the employee table to the database
db_query = 'CREATE TABLE "tbl_employees" ("employee_ID" INTEGER NOT NULL, ' + \
    '"address_1" TEXT, "address_2" TEXT, "postcode" TEXT, ' + \
    'PRIMARY KEY("employee_ID"));'
# =====> Complete the line to execute the query.
db_cur.execute(db_query)

# =====> Complete the SQL to determine the trading names of the makers who specialise in Woodworki
db_query = "SELECT trading_name FROM tbl_makers " + \
    "INNER JOIN tbl_crafts ON tbl_makers.maker_id = tbl_crafts.maker_id " + \
    "INNER JOIN tbl_specialisms ON tbl_specialisms.specialism_id = tbl_crafts.specialism_id " + \
    "WHERE tbl_specialisms.description = 'Woodworking';"
# =====> Complete the line to execute the query.
db_cur.execute(db_query)
# =====> Complete the line to fetch the query results.
db_result = db_cur.fetchall()
print ("The trading names of the makers who specialise in Woodworking are: ")
# =====> Add a line to call the show_result subprogram to display the results.
show_result(db_result)

# ------------------------------------------------------------------------
# For examiners only to display all the table names to check if
# the new employee table has been added.
print("="*80)
db_query = 'SELECT name FROM sqlite_master WHERE type="table"'
db_cur.execute(db_query)
db_result = db_cur.fetchall()
show_result(db_result)
db_query = 'PRAGMA table_info ("tbl_employees")'
db_cur.execute(db_query)
db_result = db_cur.fetchall()
show_result(db_result)
```

Question number	Mark Point	Appx. Line	Answer Award marks as shown	Additional Guidance	Mark
4			**Libraries**		18
	4.1	5	Random library is imported `import random`	• Allow any method	
			insertion_sort Subprogram	• No return value required, as this subprogram should be a procedure, but ignore if the sorted list is returned. Full marks still available.	
	4.2	26	A for in range() is used for the outer loop `for outer_index in range()):`		
	4.3	26	The range() function is accurately bound by 1 and the length of the list parameter `range(1, len(p_list))`		
	4.4	29	First item is acknowledged as already sorted, by setting the target to the item at index 1 `target = p_list[outer_index]`		

Question number	Mark Point	Appx. Line	Answer Award marks as shown	Additional Guidance	Mark
4 *(continued)*	4.5	35	The first comparator is set to the second item `inner_index = outer_index - 1`	• See additional guidance below	18
	4.6	43	A while loop is used to implement the inner loop `while ((inner_index >= 0) and (p_list[inner_index] > target)):`	• Award red only	
	4.7	43	The tests for the while loop are both accurate and joined with a Boolean operator `while ((inner_index >= 0) and (p_list[inner_index] > target)):`	• Award red only • See additional guidance below • Condition 1: inner_index >= 0 • Condition 2: pList[inner_index] > item	
	4.8	47	The value which is greater than the target is shifted to the right `p_list[inner_index + 1] = p_list[inner_index]`	• See additional guidance below	
	4.9	50	The inner_index is decremented `inner_index = inner_index - 1`		
	4.10	55	The item is moved to the correct position `p_list[inner_index + 1] = target`	• See additional guidance below	

Question number	Mark Point	Appx. Line	Answer Award marks as shown	Additional Guidance	Mark
4 *(continued)*			**display_list_matrix Subprogram**		18
	4.11	60	A loop is used to processes every item in the inputted list `for index in range(len(p_list)):`	• Allow any method • Use of for <index> in range anticipated • Allow use of for each, with additional counter for column	
	4.12	64	A mechanism for determining when the MAX_WIDTH position has been reached is implemented `if (((index + 1) % MAX_WIDTH) == 0):`	• Allow any method • Use of % is expected • Use of MAX_WIDTH required	

	4.13	65
		67
			A mechanism for adding a space, when not at MAX_WIDTH position, and adding a LF, when at MAX_WIDTH position, is implemented
			
			`print(p_list[index])`
			`print(str(p_list[index]) + " ", end = '')`
				• Allow any method	

	4.14		Outputs for the number of rows, with MAX_WIDTH number of items each are displayed	• Do not award if displaying full list notation, e.g. with commas and square brackets	

Question number	Mark Point	Appx. Line	Answer Award marks as shown	Additional Guidance	Mark
4 *(continued)*			**Main program**		18
	4.15	76 77 78	Main program: • prints little_list • calls insertion sort subprogram with little_list as an argument • prints the sorted little_list	• No formatting required for little list, so can be in full list notation • Award even if insertion sort does not fully work	

	4.16	82	One hundred random integers between zero and 100 are generated and appended to big_list
			
			`for count in range(100):`
			`  bigList.append(random.randint(0, 100))`
				• Allow any method for loop	

	4.17	88	Main program:
			• calls display_list_matrix() with big_list as an argument
			• calls insertion_sort(), with big_list as an argument
			• calls display_list_matrix(), with big_list as an argument
			
			`display_list_matrix(big_list)`
			`insertion_sort(big_list)`
			`display_list_matrix(big_list)`
				• Award for sequence of accurate calls
				• Award even if the display list matrix subprogram does not fully work
				• Award even if the insertion_sort subprogram does not fully work	

Question number	Mark Point	Appx. Line	Answer Award marks as shown	Additional Guidance	Mark
4 *(continued)*			**Functionality**		18
	4.18		The outputs are accurate for both little_list and big_list	• Must have called insertion sort with little_list and big_list as a parameter • Do not award for sort method other than the user-devised insertion sort	

Mark point	Guidance

MP 4.5
MP 4.7
MP 4.8
MP 4.10
	The inner loop and the location of the insertion can be done by tracking the location of the vacancy on each shift. Either solution is awardable.
	
	`for i in range(1,len(pList)):`
	`  key = pList[i]`
	`  curPos = i`
	`  while key < pList[curPos-1] and curPos > 0:`
	`  pList[curPos] = pList[curPos-1]`
	`  curPos -= 1`
	`  pList[curPos] = key`

```
# ------------------------------------------------------------------
# Import libraries
# ------------------------------------------------------------------
# =====> Import the required library
import random

# ------------------------------------------------------------------
# Constants
# ------------------------------------------------------------------
MAX_WIDTH = 20

# ------------------------------------------------------------------
# Global variables
# ------------------------------------------------------------------
little_list = [44, 22, 11, 33, 55]
big_list = []

# ------------------------------------------------------------------
# Subprograms
# ------------------------------------------------------------------
def insertion_sort(p_list):

    # =====> Create an outer FOR loop starting at the index position
    #        of the second item in the list that goes right in the
    #        list, one item at a time, until there are no more items
    for outer_index in range(1, len(p_list)):

        # =====> Create a variable that holds the target value
        target = p_list[outer_index]

        # =====> Create a variable to hold the index value for
        #        the inner loop that is one to the left of
        #        the outer index, because it moves along the
        #        sorted list going left
        inner_index = outer_index - 1

        # =====> Create an inner WHILE loop that handles shifting.
        #        The WHILE loop repeats as long as
        #        the inner index is greater than or equal to 0
        #        AND
        #        the value in the list at the inner index
        #        is greater than the target value
        while ((inner_index >= 0) and (p_list[inner_index] > target)):

            # =====> Shift the value at the inner index to the right
            #        by one position
            p_list[inner_index + 1] = p_list[inner_index]

            # =====> Adjust the inner index one to the left
            inner_index = inner_index - 1

```

```
        # =====> A location is identified.  Insert the target value
        #        into the gap that is one to the right
        #        of the inner index
        p_list[inner_index + 1] = target


def display_list_matrix(p_list):
    # =====> Set up a loop to process every item in the inputted list
    for index in range(len(p_list)):
        # =====> Check if the item is at the MAX_WIDTH location
        #        If it is, then print with the line feed
        #        Otherwise, print the item followed by a SPACE
        if (((index + 1) % MAX_WIDTH) == 0):
            print(p_list[index])
        else:
            print(str(p_list[index]) + " ", end = '')

# ---------------------------------------------------------------------
# Main program
# ---------------------------------------------------------------------

# =====> Display the unsorted little list (no formatting required),
#        Call the insertion sort subprogram to sort the little list
#        Display the sorted little list (no formatting required)
print(little_list)
insertion_sort(little_list)
print(little_list)
print("\n")

# =====> Fill the big list with 100 random integers between 0 and 100
for count in range(100):
    big_list.append(random.randint(0, 100))

# =====> Display the unsorted big list in rows of MAX_WIDTH items
#        Call the insertion_sort to sort the big list
#        Display the sorted big list in rows of MAX_WIDTH items
display_list_matrix(big_list)
print("\n")
insertion_sort(big_list)
display_list_matrix(big_list)

```

```
# ------------------------------------------------------------------
# Examiner testing only
# ------------------------------------------------------------------

# Copy this line to immediately follow the creation of the big_list of random numbers
# It makes a copy that can be used to double check that the code is working
unsortedBigList = big_list.copy()

print("\n")
print("-"*70)
sortedBigList = sorted(unsortedBigList)
# sortedBigList[0] = 888          # Show a real error, just for debug
mismatches = [item for item in big_list if item not in sortedBigList] + \
             [item for item in sortedBigList if item not in big_list]
if (len(mismatches) != 0):
    print("Error - Two big lists do not match.  Mismatches: ")
    print(mismatches)
else:
    print("Pass - Two big lists match")

```

Question number	Mark Point	Appx. Line	Answer Award marks as shown	Additional Guidance	Mark
5			**Libraries**		20
	5.1	5	Import datetime library `import datetime`	• Allow any method	
			Main loop		
	5.2	117	A loop is used to process every row in the data, bounded by the length of the array `while (index < len(data)):`	• Allow any method • Loop to be bound on length of data structure	

	5.3	119
		120
			Two-dimensional indexing is used to access the data
			
			`record = data[index]`
			`sensor_ID = record[ID]`
				• Award for/while for first dimension with indexing for second dimension	

			Identifying invalid readings		

	5.4	76	A range check is used to determine if a reading is valid
			
			`p_record[READING] < MIN_READING) or`
			`(p_record[READING] > MAX_READING)`
				• Allow reverse relational operator if the test is for invalid
				• Allow literals instead of constants	

Question number	Mark Point	Appx. Line	Answer Award marks as shown	Additional Guidance	Mark

5
(continued)
	5.5	125
		126
		127
			A data structure is used to track bad records
			
			`if sensor_ID not in bad_sensor_IDs:`
			`  bad_sensor_IDs.add(sensor_ID)`
			`  bad_sensor_records.append(record)`
				• 1D or 2D list anticipated
				• Only for bad records
				• Ignore tracking bad IDs
					20

			A user-defined subprogram		
	5.6	73 81	One subprogram definitions is connestructed accurately with parameters `def is_valid_reading(p_record):`	• Likely choices for subrograms are validation and reporting `def report_faulty_sensors(p_data):`	
	5.7	123 131	One user-devised subprogram is called and the calling code passes in an argument `if (not is_valid_reading(record)):`	• Likely location for these calls is in the main program `report_faulty_sensors(bad_sensor_records)`	
			Output (display and file)		
	5.8	82	The correct file is opened for writing only and closed, as required `out_file = open(OUT_FILE, "w")`	• Allow literal file name • Allow 'with open' if used accurately	

Question number	Mark Point	Appx. Line	Answer Award marks as shown	Additional Guidance	Mark

5 *(continued)*	5.9	107	More than one line containing relevant information, is written to the file
			
			`for index in range(len(p_data)):`
			`  out_file.write(`
			`  f"{record[ID]:^5}`
			`  {record[LOCATION]:^22}`
			`  {my_date_time:`
			`  %Y-%m-%d %H:%M:%S}`
			`  {record[READING]:>7.1f}\n")`
				• Allow even if information is not accurate or is inaccurately formatted
				• Do not award for headers and separators only
					20

	5.10	105	Invalid readings are displayed in the required format
			
			`print(f"{record[ID]:^5}`
			`  {record[LOCATION]:^22}`
			`  {my_date_time:%Y-%m-%d %H:%M:%S}`
			`  {record[READING]:>7.1f}")`
				• Headings and separator displayed
				• Centre alignment for integers and text
				• Right alignment for real numbers
				• Allow different column widths	

	5.11	103	Functions from the datetime library are used to convert timestamp to a human-readable format
			
			`record = p_data[index]`
			`my_date_time =`
			`datetime.datetime.fromtimestamp(record[TIMESTAMP])`		

Question number	Mark Point	Appx. Line	Answer Award marks as shown	Additional Guidance	Mark
5 *(continued)*			**Design**		20
	5.12		Tabular output meets alignment requirements and is fit for purpose		
	5.13		A single pass through the data is implemented		
	5.14		Reduction of side effects is demonstrated by original data structure remaining unchanged for the duration of the program	For award of this mark, there must be code that attempts to access the dataset	
			Clean code		
	5.15		Clean code techniques are used: • Comments that aid understanding, but are not excessive • White space that delimits blocks of logic, but is not excessive • Consistent notation conventions and meaningful identifiers	• There must be some attempt to solve the question, with enough student-supplied code to base a decision on • For this mark, there must be at least one example of each of informative commenting, white space, consistent notation and meaningful identifiers	
	5.16		Supplied constants are used throughout	• There should be no literals in code written by the student where a constant is provided	

Question number	Mark Point	Appx. Line	Answer Award marks as shown	Additional Guidance	Mark
5 *(continued)*	5.17		Clear understanding of scope is demonstrated by parameter identifiers that indicate parameters	• Do not award where global variable names are used as parameter definitions	20
			Functionality		
	5.18		The solution: • is a reasonable attempt to solve the problem • must translate without syntax errors • must execute without runtime errors	• All three required for this mark • Minimum requirement of functionality for this mark: o Every item in the dataset is processed o Invalid readings are identified o Formatted output is attempted	
	5.19		Some of the requirements must be met and some parts of the code must function as required	• Minimum requirement of functionality for this mark: o Sensors with invalid readings are tracked o Invalid sensors are reported to the display	

Question number	Mark Point	Appx. Line	Answer Award marks as shown	Additional Guidance	Mark
	5.20		The outputs are accurate	• The displayed outputs match requirements • The outputs stored in the file match requirements • Date/Time UTC displayed and stored in the file in correct format	

Output

```
There are 4 faulty sensors


 ID           Location             Date/Time       Reading
-----------------------------------------------------------------------
 16       Hyde Park Street     2024-11-05 02:50:00   -33.7
 18          Abbey Wood        2024-11-01 00:10:00    55.7
 47        Greenwich Park      2024-11-05 06:30:00     0.0
 125        Green Park         2024-11-03 04:50:00   122.8
```

```
# -----------------------------------------------------------------------------
# Import libraries
# -----------------------------------------------------------------------------
# =====> Write your code here
import datetime

# -----------------------------------------------------------------------------
# Constants
# -----------------------------------------------------------------------------
OUT_FILE = "Q05_Out.txt"
LF = "\n"

MIN_READING = 10.0          # Minimum valid reading
MAX_READING = 51.9          # Maximum valid reading

ID = 0                      # Field indexes in data record
LATITUDE = 1
LONGITUDE = 2
LOCATION = 3
TIMESTAMP = 4
READING = 5

# -----------------------------------------------------------------------------
# Global variables
# -----------------------------------------------------------------------------
# sensor id, latitude, longitude, location, timestamp, reading
data = [[16, 51.51273, -0.16709, "Hyde Park Street", 1730775000, -33.7],
        [16, 51.51273, -0.16709, "Hyde Park Street", 1730775600, 19.1],
        [16, 51.51273, -0.16709, "Hyde Park Street", 1730776200, 31.5],
        [16, 51.51273, -0.16709, "Hyde Park Street", 1730776800, 29.4],
        [16, 51.51273, -0.16709, "Hyde Park Street", 1730777400, 13.9],
        [18, 51.49087, 0.12227, "Abbey Wood", 1730419200, 23.8],
        [18, 51.49087, 0.12227, "Abbey Wood", 1730419800, 55.7],
        [18, 51.49087, 0.12227, "Abbey Wood", 1730420400, -2.5],
        [18, 51.49087, 0.12227, "Abbey Wood", 1730992800, 22.2],
        [18, 51.49087, 0.12227, "Abbey Wood", 1730993400, 42.9],
        [47, 51.47229, 0.00359, "Greenwich Park", 1730419200, 51.9],
        [47, 51.47229, 0.00359, "Greenwich Park", 1730419800, 19.8],
        [47, 51.47229, 0.00359, "Greenwich Park", 1730420400, 49],
        [47, 51.47229, 0.00359, "Greenwich Park", 1730788200, 0],
        [47, 51.47229, 0.00359, "Greenwich Park", 1730788800, 96.5],
        [54, 51.524, -0.14714, "Regent's Park", 1730492400, 39.9],
        [54, 51.524, -0.14714, "Regent's Park", 1730493000, 29.7],
        [54, 51.524, -0.14714, "Regent's Park", 1730493600, 40],
        [54, 51.524, -0.14714, "Regent's Park", 1730494200, 46.7],
        [61, 51.50972, -0.0758, "Tower of London", 1730647200, 11.8],
        [61, 51.50972, -0.0758, "Tower of London", 1730647800, 25],
        [61, 51.50972, -0.0758, "Tower of London", 1730648400, 41.7],
        [61, 51.50972, -0.0758, "Tower of London", 1730649000, 36.7],
        [61, 51.50972, -0.0758, "Tower of London", 1730649600, 26.3],
        [125, 51.50406, -0.14733, "Green Park", 1730609400, 122.8],
        [125, 51.50406, -0.14733, "Green Park", 1730610000, 0],
        [125, 51.50406, -0.14733, "Green Park", 1730610600, 12.1],
        [125, 51.50406, -0.14733, "Green Park", 1730611200, 40.4],
        [144, 51.51077, 0.03525, "London City Airport", 1730517600, 37],
        [144, 51.51077, 0.03525, "London City Airport", 1730518200, 34],
        [144, 51.51077, 0.03525, "London City Airport", 1730518800, 30.1],
        [144, 51.51077, 0.03525, "London City Airport", 1730519400, 22.7],
        [144, 51.51077, 0.03525, "London City Airport", 1730520000, 49.2]]

```

```
# =====> Write your code here
bad_sensor_IDs = set()            # Just tracks sensors with bad readings
bad_sensor_records = []           # For the whole first record of a bad sensor

# General use
index = 0

# ------------------------------------------------------------------------
# Subprograms
# ------------------------------------------------------------------------
# =====> Write your code here

def is_valid_reading(p_record):
    valid = True            # Assume good reading

    if ((p_record[READING] < MIN_READING) or (p_record[READING] > MAX_READING)):
        valid = False

    return (valid)

def report_faulty_sensors(p_data):
    out_file = open(OUT_FILE, "w")

    # Report the number of bad sensors found to display and to file
    print(LF)
    print("There are " + str(len(bad_sensor_IDs)) + " faulty sensors")

    out_file.write("There are " + str(len(bad_sensor_IDs)) + " faulty sensors\n")

    # Report the header information to display and to file
    print(LF)
    print(f"{'ID':<5}  {'Location':^22} {'Date/Time':^20} {'Reading':^7}")
    print("-" * 60)

    out_file.write(LF)
    out_file.write(f"{'ID':<5}  {'Location':^22} {'Date/Time':^20} {'Reading':^>7}\n")
    out_file.write("-" * 60)
    out_file.write(LF)

    # Report sensor information for each faulty sensor to display and to file
    for index in range(len(p_data)):
        record = p_data[index]  # More readable
        my_date_time = datetime.datetime.fromtimestamp(record[TIMESTAMP])

        print(f"{record[ID]:^5} {record[LOCATION]:^22} {my_date_time:%Y-%m-%d
        %H:%M:%S} {record[READING]:>7.1f}")

        out_file.write(
            f"{record[ID]:^5} {record[LOCATION]:^22} {my_date_time:%Y-%m-%d %H:%M:%S}
            {record[READING]:>7.1f}\n")

    out_file.close()
```

```
111 # ------------------------------------------------------------------------
112 # Main program
113 # ------------------------------------------------------------------------
114 # =====> Write your code here
115
116 # Look at all the records in the data
117 while (index < len(data)):
118
119     record = data[index]
120     sensor_ID = record[ID]       # Track to reduce indexing later
121
122     # Range check for out of bounds readings
123     if (not is_valid_reading(record)):
124         # If this bad sensor is not seen before, track it
125         if sensor_ID not in bad_sensor_IDs:
126             bad_sensor_IDs.add(sensor_ID)
127             bad_sensor_records.append(record)
128
129     index = index + 1                   # Move to the next record
130
131 report_faulty_sensors(bad_sensor_records)
132
```