International Advanced Level in Computer Science

Unit 4: Advanced 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	5	The regular expression library is imported `import re`	• Allow any method	12
	1.2	15	Closing square bracket] added to close declaration and initialisation of list `"4. UpDown 15 GB 12 months 120 minutes 50 texts £4.44 per month"]`		
	1.3	25	The logic error is fixed by adding the missing string between quotes `regex = r"no contract"`		
	1.4	29	The attribute error is fixed by correcting splot to search `match = re.search(regex, offer)`	• Award red only	
	1.5	29	The name error is fixed by correcting regextra to reg `match = re.search(regex, offer)`	• Award red only	
	1.6	36	The logic error is corrected by suppling the missing text for the expression to the left of minutes `regex = r"[uU]nlimited minutes"`	• Allow any method `[unlimited|Unlimited]`	

Question number	Mark Point	Appx. Line	Answer Award marks as shown	Additional Guidance	Mark
1 *(continued)*	1.7	45	The logic error is corrected by suppling the additional parts of the expression and the curly braces `regex = r"^[a-zA-Z]{2}"`	• Allow `"^[a-zA-Z][a-zA-z]"`	12
	1.8	54	The logic error is corrected by replacing the contents of the regex expression for digits `regex = r"^[a-zA-Z]{2}[0-9]{3}[a-zA-Z]{2}$"`	• Award red only • One mark for the three digits, wherever they appear	
	1.9	54	The logic error is corrected by replacing the contents of the regex expression for digits `regex = r"^[a-zA-Z]{2}[0-9]{3}[a-zA-Z]{2}$"`	• Award red only • One mark for one instance of the two letters, wherever it appears	
	1.10	65	The logic error is corrected by adding the missing parts of the day number `regex = r"^(0[1-9]|[12][0-9]|3[01])$"`	• **One** mark for each of the following up to a maximum of **two** marks: ○ Days: 01-09 ○ Days: 30, 31 ○ Days: 10-19, 20-29	
	1.11	65			
	1.12	65	The logic error is corrected by adding the missing parts of the day number for all three cases (purple, turquoise, red) `regex = r"^(0[1-9]|[12][0-9]|3[01])$"`	• Completely accurate expression including use of ^, |, and $ • Sub-expressions between the ORs could be in different orders	

Mark points	Guidance
MP1.10 MP1.11	Do not award either mark, where a sequence of all the numbers 01-31 is supplied using a simple OR (|), e.g.: (01|02|03|04|05|06|07|08|09|10|11|12|13|14|15|16|17|18|19|20|21|22|23|24|25|26|27|28|29|30|31)

```
# ------------------------------------------------------------------------
# Import libraries
# ------------------------------------------------------------------------
# =====> Fix the syntax error
import re

# ------------------------------------------------------------------------
# Global variables
# ------------------------------------------------------------------------
# ====> Fix the syntax error
offers = [
    "1. Seven mobile 24 months 3 GB unlimited minutes Unlimited texts £5.00
    per month",
    "2. Able network no contract 1 GB 200 minutes Unlimited texts £3.95 per
    month",
    "3. MobileTalk 1 month 30 GB Unlimited minutes unlimited texts £6.95 per
    month",
    "4. UpDown 15 GB 12 months 120 minutes 50 texts £4.44 per month"]

keys = ["AB123CD", "1A123", "&", "Ef987", "123", "bc010", "WX345YZA", "WX3456"]

# ------------------------------------------------------------------------
# Main program
# ------------------------------------------------------------------------
# Find all offers with no contract
print("----- No contract -----")
# ====> Fix the logic error
regex = r"no contract"
for offer in offers:
    match = re.search(regex, offer)
    if (match is not None):
        print(offer)

# Find all offers with unlimited minutes
print("----- Unlimited minutes -----")
# =====> Fix the logic error
regex = r"[uU]nlimited minutes"     # regex = r"[unlimited|Unlimited] minutes"
for offer in offers:
    match = re.search(regex, offer)
    if (match is not None):
        print(offer)

```

```
# Find all keys starting with two alphabetic characters
print("----- Keys starting with two alphabetic characters -----")
# =====> Fix the logic error
regex = r"^[a-zA-Z]{2}"     # regex = r"^[a-zA-Z][a-zA-z]"
for key in keys:
    match = re.search(regex, key)
    if (match is not None):
        print(key)

# Find all keys ending with three non-zero digits
print("----- Keys ending with three non-zero digits -----")
# =====> Fix the logic error
regex = r"[1-9]{3}$"
for key in keys:
    match = re.search(regex, key)
    if (match is not None):
        print(key)

# Find all keys with the pattern LLNNNLL
print("----- Keys with pattern LLNNNLL")
# =====> Fix the logic error
regex = r"^[a-zA-Z]{2}[0-9]{3}[a-zA-Z]{2}$"
for key in keys:
    match = re.search(regex, key)
    if (match is not None):
        print(key)

# Validate the day part of a date
# Two digits exactly
# Leading zeros required
print("----- Day numbers with format of DD -----")
# =====> Fix the logic error
regex = r"^(3[01]|[12][0-9]|0[1-9])$"
in_string = input("Enter a day number: ")
while (in_string != ""):
    match = re.search(regex, in_string)
    if (match is not None):
        print(in_string, "Pass")
    else:
        print(in_string, "Fail")
    in_string = input("Enter: ")

```

Question number	Mark Point	Appx. Line	Answer Award marks as shown	Additional Guidance	Mark
2			**add_member Function**		13
	2.1	18	Parameter for last name, first name, year and month are added to definition header `def add_member(p_members_list, p_last, p_first, p_month, p_year)`	• Award red only • Allow any order • Ignore any other changes to the line	
	2.2	21	Calculation of new key is implemented using the input parameters `new_key = p_year + (p_year % p_month)`		
	2.3	24	A new record is created for the table `new_record = [new_key, p_last, p_first, p_month, p_year]`	• Award red only • Allow any order for fields	
	2.4	24	The order of the fields in the records is correct `new_record = [new_key, p_last, p_first, p_month, p_year]`	• Award field order only • Ignore names	
	2.5	30	The new record is added to the original member list parameter `new_members_list = p_members_list + [new_record]`	• Award red only • New list construction	

Question number	Mark Point	Appx. Line	Answer Award marks as shown	Additional Guidance	Mark
2 *(continued)*	2.6	30	A new 2D list (table) is generated `new_members_list = p_members_list + [new_record]`	• Award red only • Allow any meaningful identifier for the new list • Assignment only	13
	2.7	33	The new list is returned `return(new_members_list)`	• Ignore brackets • Variable name must match variable created by appending	
			lazy_fives Function		
	2.8	38	The line is completed to generate numbers from zero to p_num `num in range(p_num):`		
	2.9	42	The line is completed to determine if an integer is evenly divisible by five `if (num % 5 == 0):`	• Award red only • Relational test only	
	2.10	45	A line is added to ensure lazy evaluation `yield(num)`		

Question number	Mark Point	Appx. Line	Answer Award marks as shown	Additional Guidance	Mark
2 *(continued)*			**Main Program**		13
	2.11	61	The add_member pure function is called with arguments in the order to match the parameters `newList = add_member(members, "Wilson", "Wilma", birth_month, birth_year)`	• Order of arguments must match order of parameters for `add_member() [appx. Line 18]`	
	2.12	71	The lazy_fives function is assigned to a callable name, i.e. the generator function `gen_function = lazy_fives(30)`	• Assignment only • Allow any function name	

	2.13	74	The next available value available from the generator functions is retrieved by using next(callable name)
			
			`print(next(gen_function), next(gen_function), next(gen_function))`
				• Allow any method
				
				`for count in range(3):`
				`  print (next(gen_function), end = " ")`
				
				`print(str(next(gen_function)) + " " + str(next(gen_function)) + " " + str(next(gen_function)))`
				
				`print("`
				`".join([str(next(gen_function))`
				`for i in range(3)]))`	

```
from pprint import pprint           # For debugging only

# -----------------------------------------------------------------
# Global variables
# -----------------------------------------------------------------
# Key, last name, first name, month, year
members = [[1996, "Adams", "Amanda", 1, 1996],
           [2002, "Brown", "Brenda", 6, 2000],
           [2000, "Collins", "Christopher", 3, 1999]]

birth_year = 0
birth_month = 0

# -----------------------------------------------------------------
# Subprograms
# -----------------------------------------------------------------
# =====> Complete the definition for the pure function
def add_member(p_members_list, p_last, p_first, p_month, p_year):

    # =====> Complete the calculation to produce a new key
    new_key = p_year + (p_year % p_month)

    # =====> Complete the line to create a new record for the member
    new_record = [new_key, p_last, p_first, p_month, p_year]

    # =====> Complete the line to create a new list by adding the
    #        new record to the end of the member list.
    #        Changing the values in the input list would produce
    #        a side effect
    new_members_list = p_members_list + [new_record]

    # =====> Complete the line to return the new list
    return(new_members_list)

```

```
35  def lazy_fives(p_num):
36
37      # =====> Complete the line to generate integers from zero to p_num
38      for num in range(p_num):
39
40          # =====> Complete the line to determine if an integer
41          #        is evenly divisible by five
42          if (num % 5 == 0):
43
44              # =====> Add a line to ensure lazy evaluation of the
                    function
45              yield(num)                  # Causes the evaluation to be
                    lazy
46
47  # ---------------------------------------------------------------------
48  # Main program
49  # ---------------------------------------------------------------------
50  birth_month = int(input("Enter your birth month (1 is January): "))
51  birth_year = int(input("Enter your birth year in four digits: "))
52
53  # Debugging only - for highlighting side effects
54  print("\nOriginal members before add_member call")
55  pprint(members)
56
57  # =====> Complete the line to call the pure function to add a
58  #        new member with a last name of Wilson, a first name
59  #        of Wilma, a birth month as entered, and a birth year
60  #        as entered
61  newList = add_member(members, "Wilson", "Wilma", birth_month, birth_year
    )
62
63  # Debugging only - For highlighting side effects
64  print("\nOriginal members after add_member call")
65  pprint(members)
66  print("\nNew list of members after add_member call")
67  pprint(newList)
68  print("\n")
69
70  # =====> Complete the line to assign the output of lazyFives to a
    generator function
71  gen_function = lazy_fives(30)
72
73  # =====> Add a line to display the first three values generated by the
    call to lazyFives, all on a single line.
74  print(next(gen_function), next(gen_function), next(gen_function))
75
```

Question number	Mark Point	Appx. Line	Answer Award marks as shown	Additional Guidance	Mark
3			**Person Class**		17
	3.1	12	The last name is returned from the get_last() function `return(self.__last_name)`		
	3.2	16	The first name returned from get_first() function `return(self.__first_name)`		
	3.3	21	A string is constructed consisting of last name, space, first name constructed in the get_info() method `person_info = self.__last_name + " " + self.__first_name`	• Allow any method • Allow any order of fields	
	3.4	22	The string value is returned from get_info() method `return(person_info)`	• Allow construction of string inside return	
			Employee Class		
	3.5	26	Person is added to class definition header `class Employee(Person):`		

Question number	Mark Point	Appx. Line	Answer Award marks as shown	Additional Guidance	Mark
3 *(continued)*	3.6	32	The base class constructor is called `super().__init__(p_last, p_first)`	• Two arguments required • Ignore order of arguments • Do not award any other method, as this MP is not for functionality	17
	3.7	38	The get_info method in the base class is called to get last names and first names `employee_string = super().get_info() + " " + self.__start_date`	• Award red only • Award concatenation by any method • Do not award any other method, as this MP is not for functionality	
	3.8	38	The private attribute __start_date is appended to output of get_info() separated by a space `employee_string = super().get_info() + " " + self.__start_date`	• Award red only • Award concatenation by any method	
			Department Class		
	3.9	46	None is changed to [] `self.__emp_list = []`	• Ignore any attempt to dimension the list	
	3.10	51	Public method add_employee header is implemented `def add_employee(self, p_employee):`	• Allow any name for second parameter	

Question number	Mark Point	Appx. Line	Answer Award marks as shown	Additional Guidance	Mark
3 *(continued)*	3.11	52	The input parameter is appended to list `self.__emp_list.append(p_employee)`		17
			Main Program		
	3.12	64	Person class is used to create an instance `person1 = Person("Ansel", "Alice")`	• Allow any variable name • Allow reversal of arguments	
	3.13	66	The get_info() method of the instance is called `print(person1.get_info())`	• Output to display required	

	3.14	71
		77
			Instiantiation is implemented using Employee class constructor
			
			`employee1 = Employee("Banks", "Barbara", "30/04/2015")`
			`employee2 = Employee("Cash", "Connie", "29/04/2020")`
				• At least one instance
				• Allow any variable name
				• Allow reversal of arguments	

	3.15	73
		79
			The get_info() method in the instianted object is called to retrieve the employee info
			
			`print(employee1.get_info())`
			`print(employee2.get_info())`
				• At least one instance
				• Output to display required	

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

3
(continued)
	3.16	85
		86
			The add_employee method of the Department instance is called
			
			`the_dept.add_employee (employee1)`
			`the_dept.add_employee (employee2)`
				• At least one instance
				• Ignore functionality of method add_employee()
					17

	3.17	88	All the employees are accurately displayed by calling the show method of department `the_dept.show_dept()`	• Matches output given in paper • Order of employees may be reversed, depending on order added to the employee list	

Anticipated output

```
========== Create and show a person ==========
Person 1
Ansel Alice
========== Create and show employees ==========
Employee 1
Banks Barbara 30/04/2015
Employee 2
Cash Connie 29/04/2020
========== Create and show the department ==========
The Department
Banks Barbara 30/04/2015
Cash Connie 29/04/2020
```

```
# ------------------------------------------------------------------------------
# Classes
# ------------------------------------------------------------------------------
class Person:
    def __init__(self, p_last, p_first):
        # Private attributes
        self.__last_name = p_last
        self.__first_name = p_first

    def get_last(self):
        # =====> Complete the getter method
        return(self.__last_name)

    def get_first(self):
        # =====> Complete the getter method
        return(self.__first_name)

    def get_info(self):
        # =====> Complete the get_info() method to return a string containing
        #        __last_name and __first_name separated by a single space.
        person_info = self.__last_name + " " + self.__first_name
        return(person_info)

# ------------------------------------------------------------------------------
# =====> Amend Employee class to inherit Person class
class Employee(Person):
    def __init__(self, p_last, p_first, p_startdate):
        # Private attributes
        self.__start_date = p_startdate

        # =====> Amend the Employee constructor to call the Person constructor
        super().__init__(p_last, p_first)

    # =====> Complete the get_info() method to extend the results from
    #        the Person get_info() method to append a single space and
    #        the Employee attribute __start_date and return the result
    def get_info(self):
        employee_string = super().get_info() + " " + self.__start_date
        return(employee_string)

```

```
41  # ------------------------------------------------------------------------------
42  class Department:
43      def __init__(self):
44          # Private attributes
45          # =====> Amend the line to assign an empty list to __emp_list
46          self.__emp_list = []
47
48      # =====> Add a public method named add_employee that takes
49      #        an Employee object as a parameter and adds it to the
50      #        list of employees, but returns no value
51      def add_employee(self, p_employee):
52          self.__emp_list.append(p_employee)
53
54      def show_dept(self):
55          for employee in self.__emp_list:
56              print(employee.get_info())
57
58  # ------------------------------------------------------------------------------
59  # Main program
60  # ------------------------------------------------------------------------------
61  print("="*10 + " Create and show a person " + "="*10)
62  print("Person 1")
63  # =====> Create a Person object named person1 with values:  "Ansel", "Alice"
64  person1 = Person("Ansel", "Alice")
65  # =====> Call the get_info() method to display person1.
66  print(person1.get_info())
67
68  print("="*10 + " Create and show employees " + "="*10)
69  print("Employee 1")
70  #  =====> Create an Employee object named employee1 with values: "Banks",
    "Barbara", "30/04/2015"
71  employee1 = Employee("Banks", "Barbara", "30/04/2015")
72  # =====> Call the get_info() method to display employee1.
73  print(employee1.get_info())
74
```

Question number	Mark Point	Appx. Line	Answer Award marks as shown	Additional Guidance	Mark
4			**Insert procedure**		18
	4.1	31	An insert subprogram header is defined `def insert(p_node):`	• Award red only	
	4.2	31	The insert subprogram is implemented with a new node parameter `def insert(p_node):`	• Award red only • Award this mark where the root node is also passed as an input parameter	
	4.3	33	Manipulation of scope is demonstated by the use of global keyword `global root`	• Award additional input parameter, which also exemplifies understanding of scope `def insert(p_node, p_root):`	
	4.4	40	A while loop is used to locate insertion point `while (not found):`	• Allow recursive solution	

Question number	Mark Point	Appx. Line	Answer Award marks as shown	Additional Guidance	Mark
4 *(continued)*	4.5	42	If/else is used with a relational test to determine wheter the left or the right poiniter is to be followed `if (p_node.name < pointer.name):`	• Accurate use of < to follow left/right with else. • Do not award two if statements • Tests could be reversed, as in if/else clauses may be swapped • Ignore inclusion of =, as there are no duplicates in the data	18

	4.6	44
		51
			The new node is inserted by assigning left or right pointer to the new node
			
			`pointer.left = p_node`
			`pointer.right = p_node`
				• At least one instance	

	4.7	55	Inserting a node is implemented as a procedure, that has no return value	• Do not award this mark point if there is a return statement provided, even if mal-formed • Allow recursive solution	
			Post-order traversal		
	4.8	58	The post order traversal header definition takes a node as a parameter `def post_order(p_node):`		

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

4
(continued)
	4.9	59
		61
			The node's pointer is checked to see if it is accessible, e.g is not equal to None
			
			`(p_node.left is not None):`
			`(p_node.right is not None):`
				• Allow any method
				• At least one instance
					18

	4.10	60
		62
			The tree is accessed, firstly, in the left direction, and, secondly, in the right direction
			
			`post_order(p_node.left)`
			`post_order(p_node.right)`
				• Award red only	

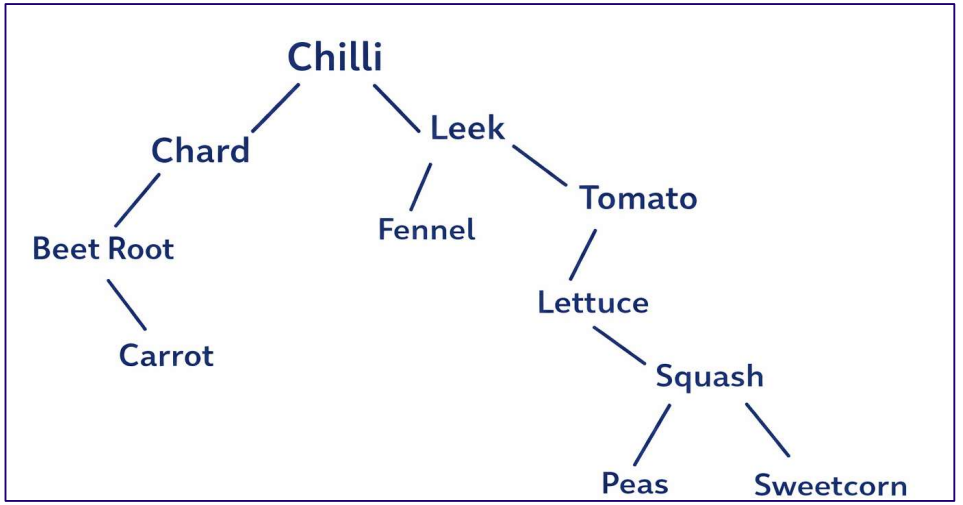
	4.11	60
		62
			Recursive calls are used to pass in each subtree
			
			`post_order(p_node.left)`
			`post_order(p_node.right)`
				• Ignore accuracy of traversal
				• At least one instance of a recursive call with an argument of a node pointer	

	4.12	63	Display of node name is done as last line of code in the procedure, i.e. on the unwinding of the recursion `print(p_node.name)`		
			Main program		

	4.13	79	A loop is used to process every item in the food table
			
			`for index in range(1,`
			`len(food)):`
				• Allow any method bounded by the length of the food list	

Question number	Mark Point	Appx. Line	Answer Award marks as shown	Additional Guidance	Mark
4 *(continued)*	4.14	82	A new node is created, using Node class, from the food located in the list `new_node = Node(food[index])`	• Allow creation of node inside call to insert subprogram	18
			Overall		
	4.15		A minimisation of storage space is demonstrated by using no additional copies of data structures	• No copies of data structures, nodes or tree, are required	
	4.16		An understanding of scope isolation is demonstrated by minimising the use of global variables	• The root is the only global variable required	
			Functionality		
	4.17		The solution: • is a reasonable attempt to solve the problem • must translate without syntax errors • must execute without runtime errors	• At least one of insert or post order traversal works correctly • Where the insert puts the nodes in the wrong order, the traversal can be awarded this mark, provided it accurately reports the state of the tree • Where the traversal is inaccurate, but the insert has put the nodes in the correct order, the insert can be awarded this mark. This may require memory inspection of the tree.	

Question number	Mark Point	Appx. Line	Answer Award marks as shown	Additional Guidance	Mark
4 *(continued)*	4.18		The outputs are accurate	• Both the insert and the post-order traversal are accurate for the full binary tree	18

Output	Completed tree
========== Building tree ========== ========== Tree complete ========== ========== Post Order Traversal ========== Carrot Beet Root Chard Fennel Peas Sweet Corn Squash Lettuce Tomato Leek Chilli	

```
# ------------------------------------------------------------------------------
# Global variables
# ------------------------------------------------------------------------------
# Seed name
food = ["Tomato",
        "Lettuce",
        "Fennel",
        "Beet Root",
        "Squash",
        "Sweet Corn",
        "Peas",
        "Carrot"
        ]

root = None             # Root of the binary search tree

# ------------------------------------------------------------------------------
# Classes
# ------------------------------------------------------------------------------
class Node(object):
    def __init__(self, p_name):
        self.name = p_name
        self.right = None           # No right element
        self.left = None            # No left element

# ------------------------------------------------------------------------------
# Subprograms
# ------------------------------------------------------------------------------
# =====> Create a procedure to insert a node into the binary search tree.
#        The node is passed as a parameter
def insert(p_node):

    global root
    found = False       # Loop control

    # Create pointer to search the tree
    pointer = root

    # Find the place to insert the new node
    while (not found):
        # New record has a name to left of current pointer
        if (p_node.name < pointer.name):
            if (pointer.left is None):
                pointer.left = p_node
                found = True        # Have inserted to left
            else:
                pointer = pointer.left
        # New record has a name to the right of the current pointer
        else:
            if (pointer.right is None):
                pointer.right = p_node
                found = True        # Have inserted to right
            else:
                pointer = pointer.right

```

```
# =====> Create a procedure that implements a recursive post-order traversal
#        of the binary search tree to display the food name in each node
def post_order(p_node):
    if (p_node.left is not None):
        post_order(p_node.left)
    if (p_node.right is not None):
        post_order(p_node.right)
    print(p_node.name)

# ------------------------------------------------------------------------------
# Main program
# ------------------------------------------------------------------------------
# Setup up the binary search tree - Do Not amend
node1 = Node("Chard")
node2 = Node("Leek")
root = Node("Chilli")
root.right = node2
root.left = node1
# End setup

print("========== Building tree ==========")
# =====> Create a new node for each food in the list of food

for index in range(0, len(food)):
    # =====> Insert the new node into the binary search tree,
    #        using the insert procedure
    new_node = Node(food[index])
    insert(new_node)
print("========== Tree complete ==========")

print("========== Post Order Traversal ==========")
# =====> Call the post-order traversal procedure
post_order(root)

```

Question number	Mark Point	Appx. Line	Answer Award marks as shown	Additional Guidance	Mark
5			**Calculating scores**		20

	5.1	160
		161
			The median of three numbers for diver 1 and/or diver 2 is found
			
			`median_diver1 =`
			`(sorted(this_record[DIVER_1:DIVER_1 + 3]))[1]`
			`median_diver2 =`
			`(sorted(this_record[DIVER_2:DIVER_2 + 4]))[1]`
				• Allow any method
				• At least one instance required
				• Anticipated method: pick out the fields from the record with a slice, sort the slice, index the sorted slice	

	5.2	162	The median of five numbers (synchro) is found
			
			`synchro_score =`
			`(sorted(this_record[SYNCHRO:SYNCHRO + 5]))[1:4]`
				• Allow any method
				• Anticipated method: pick out the fields from the record with a slice, sort the slice, index the sorted slice	

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

5
(continued)
	5.3	163
		166
		169
		172
			The formula to calculate the total score of each dive is translated accurately
			
			`raw_score = median_diver1 + median_diver2 + sum(synchro_score)`
			`raw_score = raw_score * SCALE_ADJ`
			`round_score = raw_score * this_record[DIFFICULTY]`
			`total_score = round(round_score, 2)`
				• Ignore resulting number if error made with medians
				• Allow literal for constant (SCALE_ADJ)
				• Allow equivalent arithmetical expressions
				• ((Median 1 + Median 2 + Synchro) * scale adjustment * difficulty) rounded to two places
				• ((Median 1 + Median 2 + (Synchro 1 + Synchro 2 + Synchro 3)) * scale adjustment * difficulty) rounded to two places
					20

	5.4	175 142- 147	The scores are stored in a 2D data structure with team number (leader_board)	• Allow any method • May require a loop to search the leader board for the team number to find storage location	

Question number	Mark Point	Appx. Line	Answer Award marks as shown	Additional Guidance	Mark
5 *(continued)*			**Leader board**		20

	5.5	114
		115
			The data structure, holding the scores, is sorted in reverse order, prior to display
			
			`index_total = len(out_board[0]) - 1`
			`out_board.sort(key = itemgetter(index_total), reverse = True)`
				• Allow any method	

	5.6	99	The supplied dictionary is used to access team name based on team number `out_board[out_board_row][1] = teams.get(team[0])`	• Allow any method of accessing the supplied dictionary • Dictionary is supplied	

	5.7	131
		132
			A mechanism to handle no score symbol being displayed in the output
			
			`for count in range(NUM_ROUNDS - num_dives):`
			`  layout_row = layout_row + f"{NO_SCORE:^8} "`
				• Allow any method
				• Anticipate completely filling display board with NO_SCORE prior to filling with output data
				• Alternatively, the NO_SCORE may be added one at a time when dynamically building an f-string	

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

5
(continued)
	5.8	102
		103
		104
			Total team score is calculated correctly from individual round scores
			
			`for count in range(1, num_dives + 1):`
			`  out_board[out_board_row][count + 1] = team[count]`
			`  total = total + team[count]`
				• Allow any method
				• Anticipate a loop or use of sum() across a slice
					20

	5.9	119	Column headers, with appropriate field widths, and separator are displayed
			
			`print(f"{titles[0]:^8}`
			`  {titles[1]:^10} {titles[2]:^8} {titles[3]:^8}`
			`  {titles[4]:^8} {titles[5]:^8} {titles[6]:^8}`
			`print("-"*62)`
				• Allow <string>.format() or f-string
				• Do not award literal strings or attempts to manually space out the titles	

	5.10	130
		133
			Round scores and total score displayed with two digits after decimal point
			
			`layout_row= layout_row + f"{team[2 + count]:^8.2f}"`
			`layout_row = layout_row + f"{team[6]:^8.2f}"`
				• Formatting attempt required for this mark
				
				`{^8.2f}`	

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

5
(continued)
	5.11	73	Minimisation of side effects is demonstrated by keeping the team scores in an order disconnected to the display or ranking order
			
			Keeps all the scores in order of team:
			`leader_board = [[17],…]`
			
			Keeps the output that is to be displayed:
			`out_board = []`
				• Anticipated methods:
				• Single structure to keep the scores associated with each team that is updated with scores for each round, given in the code
				• Separate structure to hold the sorted scores for displaying.
				• This means that each time a score is updated, the whole data structure doesn't have to be searched or sorted
				• Allow deep copy or separately defined data structure
					20

			Design		

	5.12		Modularity demonstrated by the use of subprograms
			
			`def show_leader_board(p_leaders):`
			`def update_scores(p_team, p_score, p_leaders):`
			`def calc_dive_score(p_dive_data, p_leaders):`
				• Modularity enforced with single entry and single exit points (blocks, loops, subprograms, etc.)	

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

5 *(continued)*	5.13		Parameters to reduce the need for global variables are used in subprograms
			
			`def show_leader_board(p_leaders):`
			`def update_scores(p_team, p_score, p_leaders):`
			`def calc_dive_score(p_dive_data, p_leaders):`
				• A solution can be written without the need for global variables, other than those provided in the student code	20

	5.14		Minimisation of storage space is demonstrated by a design that does not require copies of the data structures	• A solution can be written without the need to copy any data structure.	
			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 score in at least one round is processed o The calculations for a round are attempted 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 The round calculations o The total score calculations o Leader board sorting	

Question number	Mark Point	Appx. Line	Answer Award marks as shown	Additional Guidance	Mark
5 *(continued)*	5.20		The outputs are accurate	• Allow different column widths and alignment, if fit for purpose	20

```
# ------------------------------------------------------------------------
# Import libraries
# ------------------------------------------------------------------------
from operator import itemgetter     # Can use with <list>.sort() to sort display

# ------------------------------------------------------------------------
# Constants
# ------------------------------------------------------------------------
TEAM = 0                    # Index of team number
DIFFICULTY = 1              # Index of degree difficulty
DIVER_1 = 2                 # Index of Diver 1 scores
DIVER_2 = 5                 # Index of Diver 2 scores
SYNCHRO = 8                 # Index of Synchro scores
NUM_ROUNDS = 4              # Number of rounds in competition
NUM_TEAMS = 5               # Number of teams
NO_SCORE = "-"              # No score on leader board
SCALE_ADJ = 0.6             # To bring synchronised score in line with individual

# ------------------------------------------------------------------------
# Global variables
# ------------------------------------------------------------------------
teams = {17: "Pirates",
         23: "Hawks",
         28: "Warriors",
         45: "Chargers",
         83: "Orcas"}

# Field widths for titles {:6} {:10} {:8} {:8} {:8} {:8} {:8}
titles = ["Team", "Name", "Round 1", "Round 2", "Round 3", "Round 4", "Total"]

# For storing the leader board
leader_board = [[17],
                [23],
                [28],
                [45],
                [83]]

```

```
# Team, Difficulty, 3 x Diver 1, 3 x Diver 2, 5 x Synchronisation
round1 = [[17, 2.0, 8.5, 9.0, 8.5, 9.0, 8.5, 9.5, 8.5, 9.5, 9.0, 9.0, 9.5],
          [23, 2.0, 9.0, 8.0, 9.0, 8.0, 9.0, 9.0, 8.5, 8.5, 8.0, 8.0, 8.5],
          [28, 3.3, 9.5, 9.0, 9.0, 9.0, 9.5, 9.0, 8.5, 9.5, 8.0, 8.5, 9.5],
          [45, 3.0, 9.5, 9.5, 9.5, 8.0, 9.5, 8.5, 8.5, 9.0, 8.0, 9.5, 9.0],
          [83, 3.0, 9.0, 9.5, 8.0, 8.0, 8.0, 9.0, 9.5, 9.5, 8.5, 9.0, 9.5]
         ]

round2 = [[17, 2.7, 8.5, 8.5, 9.5, 8.0, 9.0, 9.0, 8.5, 9.0, 8.5, 9.5, 8.5],
          [23, 3.0, 8.5, 9.5, 8.0, 8.5, 8.5, 8.5, 8.0, 9.0, 8.5, 9.5, 8.0],
          [28, 3.2, 9.0, 8.5, 9.5, 8.5, 9.0, 9.5, 9.5, 9.5, 9.0, 8.0, 9.5],
          [45, 2.8, 8.5, 8.0, 9.5, 8.0, 8.5, 8.0, 9.0, 9.0, 8.5, 8.0, 9.5],
          [83, 2.4, 9.0, 8.0, 8.5, 9.5, 9.5, 8.5, 8.5, 8.0, 8.0, 9.5, 9.5]
         ]

round3 = [[17, 2.2, 9.5, 8.5, 9.5, 8.5, 9.5, 8.5, 8.0, 8.0, 8.0, 8.5, 9.0],
          [23, 2.7, 9.0, 8.5, 9.0, 9.5, 9.5, 8.5, 8.0, 9.5, 8.0, 9.5, 9.5],
          [28, 3.4, 8.0, 9.0, 8.0, 9.5, 9.0, 9.5, 9.5, 8.0, 8.5, 8.0, 9.0],
          [45, 3.2, 8.0, 9.0, 9.0, 9.0, 8.0, 9.0, 8.5, 8.5, 9.5, 9.0, 8.5],
          [83, 2.1, 8.5, 8.5, 9.0, 9.5, 8.0, 8.5, 9.0, 9.0, 8.5, 8.5, 9.0]
         ]

round4 = [[17, 3.1, 8.5, 9.5, 9.5, 9.5, 9.0, 9.0, 8.5, 8.5, 8.0, 9.5, 9.0],
          [23, 2.2, 9.0, 8.5, 8.5, 9.5, 8.5, 9.5, 9.0, 9.0, 8.0, 9.0, 8.5],
          [28, 1.7, 8.0, 9.0, 9.5, 8.5, 8.0, 9.0, 8.0, 8.5, 9.0, 9.5, 8.5],
          [45, 1.9, 8.0, 9.5, 8.0, 8.0, 8.5, 8.5, 9.5, 8.0, 9.5, 8.5, 9.0],
          [83, 2.7, 8.5, 8.5, 9.0, 9.0, 9.0, 9.0, 9.0, 8.5, 9.5, 9.0, 8.5]
         ]


# ------------------------------------------------------------------------
# Subprograms
# ------------------------------------------------------------------------
# =====> Write your code here
def show_leader_board(p_leaders):
    out_board = []

    # Initialise an empty displayable score board with all blanks
    for count in range(len(p_leaders)):
        out_board_row = list(NO_SCORE)        # For team number
        out_board_row.append(NO_SCORE)        # For team name
        out_board_row.append(NO_SCORE)        # For the total

        # One column for each of the dives in the competition
        for columns in range(NUM_ROUNDS):
            out_board_row.append(NO_SCORE)
        out_board.append(out_board_row)

    # Figure out how many dives have scores (between 1 and 5)
    # First field is not a dive, it's a team number
    num_dives = len(p_leaders[0]) - 1

    # Note - the contents of the out_board are not yet sorted in order
    # Pick up information from the input parameter list holding
    # the actual scores and put them into the empty score board
    # Add up the total of existing dives per team for last column
    out_board_row = 0                     # Start at the top
```

```
    for team in p_leaders:
        total = 0                                   # Reset each team
        out_board[out_board_row][0] = team[0]       # Team number

        # Look up team name in dictionary
        out_board[out_board_row][1] = teams.get(team[0])

        for count in range(1, num_dives + 1):
            out_board[out_board_row][count + 1] = team[count]      # Dive score
            total = total + team[count]         # Run team total

        # Put total in last column
        out_board[out_board_row][-1] = total

        # Next slot on board
        out_board_row = out_board_row + 1

    # Sort on last column in reverse, to get winner at top
    # Can use <list>.sort() because in place is fine on this copy
    index_total = len(out_board[0]) - 1
    out_board.sort(key = itemgetter(index_total), reverse = True)

    # Display in score board headers
    print("\n")
    print(f"{titles[0]:^6} {titles[1]:^10} {titles[2]:^8} {titles[3]:^8} {titles[
    4]:^8} {titles[5]:^8} {titles[6]:^8}")

    print("-"*62)
    # Display the information for each row in the sorted score board
    # Build up the formatting string for team number and team name
    # Only for the number of dives with scores, pick up the score
    # For remaining dives put in the NO_SCORE symbol
    # Finish with the total, which all teams will have
    for team in out_board:
        layout_row = f"{team[0]:^6} {team[1]:^10} "         # Integer team
        number and team name
        for count in range(num_dives):                      # Floating point
        scores
            layout_row = layout_row + f"{team[2 + count]:^8.2f} "
        for count in range(NUM_ROUNDS - num_dives):         # Hyphen
            layout_row = layout_row + f"{NO_SCORE:^8} "
        layout_row = layout_row + f"{team[6]:^8.2f}"        # Total
        print(layout_row)

```

```
136 def update_scores(p_team, p_score, p_leaders):
137     found = False
138     index = 0
139 
140     # Find the team in the leader board
141     # Add the new score to the end
142     while ((index < len(p_leaders)) and (not found)):
143         if (p_leaders[index][0] == p_team):
144             p_leaders[index].append(p_score)
145             found = True
146         else:
147             index = index + 1
148 
149 def calc_dive_score(p_dive_data, p_leaders):
150     count = 0
151 
152     while (count < len(p_dive_data)):          # For each of five teams
153 
154         # Work with one dive at a time
155         this_record = p_dive_data[count]
156 
157         # Calculate raw score from all 11 judges
158         # Sort each set of judges scores, then pick up the middle
159         # one for diver1 and diver2, and the middle three for synchro
160         median_diver1 = (sorted(this_record[DIVER_1:DIVER_1 + 3]))[1]
161         median_diver2 = (sorted(this_record[DIVER_2:DIVER_2 + 3]))[1]
162         synchro_score = (sorted(this_record[SYNCHRO:SYNCHRO + 5]))[1:4]
163         raw_score = median_diver1 + median_diver2 + sum(synchro_score)
164 
165         # Multiply by a scale adjustment
166         raw_score = raw_score * SCALE_ADJ
167 
168         # Adjust for the degree of difficulty
169         round_score = raw_score * this_record[DIFFICULTY]
170 
171         # Round to two places
172         total_score = round(round_score, 2)
173 
174         # Update leader board with team id and new score
175         update_scores(this_record[TEAM], total_score, p_leaders)
176 
177         count = count + 1
178 
179 # ------------------------------------------------------------------------
180 # Main program
181 # ------------------------------------------------------------------------
182 # =====> Write your code here
183 calc_dive_score(round1, leader_board)
184 show_leader_board(leader_board)
185 
186 calc_dive_score(round2, leader_board)
187 show_leader_board(leader_board)
188 
189 calc_dive_score(round3, leader_board)
190 show_leader_board(leader_board)
191 
192 calc_dive_score(round4, leader_board)
193 show_leader_board(leader_board)
194 
195 
```