Examiners' Report
Principal Examiner Feedback

Summer 2023

Pearson Edexcel International GCSE
In Computer Science (4CP0)
Paper 02 Application of Computational Thinking

Summer 2023
Publications Code 4CP0_02_2306_ER
All the material in this publication is copyright
© Pearson Education Ltd 2023

SPECIFIC COMMENTS

Written response questions

- There was one multiple-choice questions included in the question paper, 2(a)(ii). This question was answered well with most candidates knowing that a function returns a result.

- Single mark questions were also generally well answered.
    - In question 1(b) most pupils got most or all available marks. The most common errors were pupils describing what a comment was for 1(b)(i), rather than giving the specific comment from the code, and not being able to identify a logical operator for 1(b)(ii). Pupils scored well in 1(b)(v) indicating that most pupils are aware of the difference between a parameter and an argument.

    - 1(c) was not as well answered. Although most candidates knew what a constant was for 1(c)(i), most could not articulate a suitable reason for using a constant, often simply repeating their previous answer and so not gaining a mark for 1(c)(ii).

    - 2(a)(iii) was well answered with most candidates understanding the scope of a local variable.

- Multiple mark questions were generally less well done with many answers lacking the detail required for an expanded response.
    - In 1(a) many candidates lost the mark for identifying a process. Some just gave incorrect answers such as '*debit – credit*', with many giving a vague response such as 'a *calculation'* or '*adding*'. The question was looking for a specific process that occurred in the table, such as *'balance + credit*'. A description would also have been acceptable such as '*adding a credit to the balance'*.

    - For question 2(a)(i), most candidates knew what a subprogram library was and had some idea about why to use them, with most scoring at least 1 mark. For the few who scored no marks, answers were generally vague benefits such as '*it is faster'*. Many candidates didn't score the second mark as their second answer was a rephrasing of their first response e.g. *'..saves time as it is already coded..'* and *'..you don't have to code it yourself..'*.

- 2(b)(i) was very well answered with most candidates able to identify valid, boundary, and invalid test data to gain all 3 marks. Where marks were lost it was most often due to candidates not properly reading the question, which asked for 'numerical data', and giving text or another data type as invalid data.

- Question 3(a) was the least well answered question in the paper. While responses indicated that candidates knew what a trace table was, most were not able to identify the characteristics of one, with many responses stating the benefits or purpose of a trace table. Some candidates did confuse trace tables with truth tables.

- Question 3(b) was very well answered as most candidates could follow the given algorithm to correctly identify the outputs. A few candidates lost marks by simply stating a number e.g. '*2*' rather than the full output '*State 2'*, but this was rare. The most commonly lost mark seemed to be with the first output which many candidates gave as 'State 3'.

- Candidates were comfortable completing Caesar Cypher encodings, with most scoring both marks in 4(a)(i) and 4(a)(ii). While 4(a)(ii) was also quite well done, with over half of candidates getting both marks, most marks were lost in this part of the question. Most candidates could identify where the error had occurred; the letter Y being incorrectly encoded, but many failed to sufficiently explain the reason. Candidates were expected to identify that the error had been caused by not handling the end of the alphabet correctly, i.e. should have moved to letter a. Instead, many assumed that there was simple human error e.g. '*they did a -3 instead of a +5*'.

- Question 5 was the most difficult written-response question, and most candidates struggled to score well in this question. Many candidates seemed unsure of how to present their response for 5(a). Candidates needed to show the merging of groups of elements in their response. Many candidates failed to show any kind of grouping and simply showed a single progressively sorted list but often with no indication why positions were changing. Most did show some sort of grouping but lost a mark by sorting 3 individual elements being grouped together in one step, rather than the 2 steps that would be required in a merge sort.

- 5(b) was also not well answered by most candidates. It is difficult to ascertain whether candidates were not well versed in the use of a temporary variable for swapping the values inside two existing variables, or if it was simply too difficult to work out which line could be changed without the need to change a second line. Some candidates did identify a change in lines 12 or 13 that could have been part of a solution, but line 11 was the only line that could be changed on its own to give a full solution.

Coding response questions

Examiners found that candidates responded very well to the coding challenges presented in the question paper. There were many candidates who scored close to full marks on these questions. Most candidates made some attempt at every question.

2(b)(ii) Most candidates scored most or all marks in this question. Many inventive ways were found of correcting the errors in the code. The answers in the mark scheme are indicative of expected responses, but any response that solved the given problem was awarded the marks.
Some candidates reversed the entire selection statement rather than reversing the condition that compared the stock level to the required level.
Many candidates divided the required stock calculation by -1000 rather than swap the values in the calculation i.e. *'weightNeed = (inStock - requiredWeight) /* -1000', to get both the third and fourth mark-points.
Where marks were lost, it was often as a result of candidates removing the division by zero, most likely as this caused a run-time error.

3(c) Although this question was generally well answered, candidates seem to have found it more difficult to implement the logic of a flowchart compared to similar questions in previous papers which use pseudocode. Many candidates followed the order of the flowchart and added the code for each element as they came to it, rather than looking at the algorithm as a whole. This resulted in many candidates missing the fact that loops were required, and so only included selection statements. Relatively few candidates had solutions that successfully repeated until a zero was entered as the base value. For those that did, most did not have the terminating condition on the while statement itself (*while base !=0)*, but would use '*while True'* followed by a selection statement which would '*break'* the loop '*if base* == *0'*. Both methods were acceptable for the marks available.
Many solutions lost marks as candidates did not use an appropriate error message, with many just being '*error'*. Where error messages are requested in coding tasks, candidates should give some indication to the user of what the problem is. In this instance an appropriate error message would have indicated that the exponent must not be a negative number e.g. '*The exponent cannot be a negative value'*.
Examiners noted that many responses held the user inside a loop until a positive exponent value was entered. While this was not the logic of the flowchart, it was encouraging to see effective use of loops here.
Another common mistake came from candidates not properly reading the question. The second bullet point in the program requirements instructed candidates that the base and exponent, as well as the answer, should be included in the final output. Many candidates only output the answer and omitted the base and exponent values.

4(b) Another question that was generally well answered. Candidates are comfortable with taking inputs and most were able to cast them to the correct data types. Most candidates were able to reverse the letters of the input word which was almost always achieved through string indexing. Many candidates were also able to get the whole number part of the float by casting it to an int, however some did lose this mark by leaving the input as a string and using string indexing to take only the first digit rather than the whole number. This would not have worked on floats of 10 or higher. About half of the candidates successfully cast the number parts back to strings and used concatenation to join the parts of the key together. Some candidates left all three parts as strings from input and so still got the marks for concatenating these parts together.
Many candidates were not able to correctly check for a word of exactly two characters. Most candidates did get the length of the word but many didn't seem to be familiar with the not operator (similar to its omission in Q3(c)) so relied on a more complex condition e.g. '*if len(word) <= 2 and len(word) > 1*' rather than '*if len(word) != 2*'. This additional complexity often introduced logical errors which resulted in candidates losing marks.

5(c) Although many candidates did get most or all marks in this question, most candidates found this question too difficult and gained very few marks as a result. Most candidates were able to open the file for writing (append mode was also acceptable) however many forgot to close the file after processing. The provided data, tbl_tuna, was a two-dimensional array and most candidates were not able to use 2D indexing to access the individual elements inside the array. It is unclear whether candidates were not aware of the structure of a csv file, or if they were hampered by this inability to access the elements of a two-dimensional array.
Most candidates did use a for loop to iterate over the table, but many candidates didn't use the 'in' keyword, instead using 'in range' set to the length of the array, which added to the difficulty of achieving a working program. Most responses that wrote anything to a file consisted of a number followed by one or more of the 1D arrays inside tbl_tuna.
For those who were able to perform 2D indexing, almost all were able to correctly construct each line, including commas and line break, and successfully write these to the file.

6 It was good to see that almost all candidates made a reasonable attempt at completing this programming task, although a few of those attempts amounted to little more than hard-coding a series of output statements. This question required candidates to iterate over several one-dimensional arrays to combine the data for calculations and outputting. A peculiarity noted by many examiners was that many candidates started by combining the data from each of these arrays into a single two-dimensional array before doing the requested processing on the 2D array rather than the original arrays.

Most candidates were able to iterate over the arrays, calculate daily milk volumes per breed and output the required data in rows. Most were also able to calculate the total daily milk yield either as a running total or using the sum function on the completed array. Many candidates were not able to append data to an array, although some created a new array and appended the data to it rather than in the empty array that was provided. Most candidates struggled to create suitable code for selecting the best breed. Some did identify the correct breed, but their code would not work on alterative data sets e.g. some hard coded the care rating as 1 and only checked for this value rather than finding the lowest number in the array. Others hard coded the count-controlled loop rather than finding the lengths of the given arrays.

A number of methods were used for finding the best breed including comparisons joined with logical operators, comparisons within nested selection statements, and values being appended to new arrays and sorted. For the more capable candidates who were able to create working solutions, marks were often lost in the levels-based part of the mark scheme. This was for two main reasons. Most candidates created a new loop to complete each part of the program rather than completing each of the tasks within the one loop. We permitted up to two separate loops while still allowing full marks in that section.

The second common issue was the lack of commenting. It was surprising to see how many candidates who obtained most marks in this question but had no comments at all. A very small number of candidates overused commenting, making code more difficult to read. Good commenting should be used to separate and explain blocks of code.