Chief Reader Report on Student Responses:

2023 AP® Computer Science Principles Performance Task

- Number of Students Scored 164,505
- Number of Readers 499
- Score Distribution

Exam Score	N	%At
5	18,925	11.50
4	33,834	20.57
3	51,094	31.06
2	33,699	20.49
1	26,953	16.38

- Global Mean 2.90

The following comments on the 2023 performance task for AP® Computer Science Principles were written by the Chief Reader, Tom Cortina, Carnegie Mellon University. They give an overview of the Computer Science Principles performance task and of how students performed on the task, including typical student errors. General comments regarding the skills and content that students frequently have the most problems with are included. Some suggestions for improving student preparation in these areas are also provided. Teachers are encouraged to attend a College Board workshop to learn strategies for improving student performance in specific areas.

Question 1

Task: Create Performance Task
Topic: Application from Ideas

	Max. Points:	Mean Score:
Row 1	1	0.55
Row 2	1	0.67
Row 3	1	0.34
Row 4	1	0.43
Row 5	1	0.44
Row 6	1	0.38

What were the responses to this question expected to demonstrate?

The Create Performance Task is designed to give students an opportunity to develop a program that solves a problem for the user or allows the pursuit of a creative interest. Students should be able to demonstrate the program running in a short video and explain its purpose, how it functions, and how it handles input and output of information as shown in the video.

Programs typically process collections of data to help the user gain insight and make decisions. This task also requires students to demonstrate their understanding of data abstraction, using at least one list (or equivalent collection type) to hold data that is critical to fulfilling the program's purpose. Students must explain how the list manages complexity in the program, by either explaining why their program could not function without the list or why their program would require a more complex implementation without the use of the list, to demonstrate the importance of using this abstraction when processing larger amounts of data.

Programs use procedures to break a larger computational task into smaller subtasks to make a program easier to develop and test. This task also requires students to use procedural abstraction to write a procedure with at least one explicit parameter that demonstrates the use of sequencing, selection, and iteration, along with a call to this procedure. The student should be able to explain how the procedure works in detail, what the procedure does in summary, and how the procedure contributes to the overall functionality of the program. Finally, the student should be able to explain how to test the procedure for correctness using its parameter(s), using two examples that cause different behavior and results to occur.

How well did the responses address the course content related to this question? How well did the responses integrate the skills required on this question?

Program Purpose and Function

- Students were asked to develop a working program with a purpose to solve a problem for the user or pursue a creative interest. Students were also asked to create a video that demonstrated some functionality of their program that requires user input and shows program output. In general, most students were able to write a working program and create a video that demonstrated the program's functionality, input, and output. In addition, most students were able to accurately describe the functionality, input, and output in their written response.

- Some students were able to state the purpose of the program by describing a broader problem that is being solved for the user or the creative or artistic interest being explored, whereas other students incorrectly stated the functionality as the purpose. In the case of game programs, some students incorrectly stated the object of the game (i.e., how to win) as the purpose of the program.

- Some students based their solutions on example programs provided within the course (e.g., by content providers). As a result, some of their responses were considerably weaker than those who created their own program from scratch since they did not spend as much time working on the code development.

Data Abstraction and Managing Complexity

- Students were asked to provide two code segments from their program, the first showing the initialization of a list (or other collection type) of data and the second showing how that list is used in their program to reduce complexity. Most responses used lists, while a few used an alternate collection type such as a dictionary or a 2D array. Most students were able to identify a list in their program and provide two code segments that demonstrated the initialization and use of the identified list. Some students confused creating an empty list with initializing the list with data. Some students confused an external data source or a procedure for a list. Some students accessed only one specific element of the list instead of multiple elements, although accessing a single random element of the list was allowed since this will allow for accessing multiple elements over time.

- Some students were able to use the list in a manner that reduces program complexity by generalizing list element access using an index or subscript to allow for arbitrary-sized lists and iteration over lists. A subset of these students were able to explain in their written response how the list managed complexity in their program. On the other hand, some students wrote about managing complexity with lists in general and did not describe how their own program managed complexity with a list. Some students inaccurately described that their program could not be completed without lists when the use of the list was simplistic and could be replaced with a few individual variables. In cases where portions of the students' code that used a list were very similar to provider examples, students tended to have a harder time describing why the list managed complexity.

  Some students were unable to use a list to manage complexity in their program, but instead, they used a list when it was not needed (e.g., to store a small number of values where the code would have been more clear if they were each in named variables) or failed to leverage the structure of the list (e.g., by accessing each element with a separate statement rather than via a loop or other variable index value).

Procedural Abstraction, Algorithm Implementation, and Testing

- Students were asked to provide two code segments from their program, the first showing a procedure using at least one explicit parameter required to perform its function and the second which shows a call to this procedure with argument(s) for the parameter(s). Students were also asked to write about what the procedure does and how it contributes to the overall program. Most students were able to provide the two code segments and describe what the procedure does. Some students were also able to separately describe how the procedure contributes to the overall functionality of the program. However, some students had trouble explaining how the procedure contributes to the overall functionality of the program, particularly when the selected procedure performed almost the entirety of the overall functionality of the program. Some students stated how the procedure contributed to the overall program by describing its functionality, but the scoring guidelines require two descriptions, one for its function and one for how that procedure contributes to the overall program (e.g., what role that procedure plays in the program). Some students submitted procedures with no explicit parameters (e.g., using global variables to "pass" data to the procedure) or code fragments not enclosed in a procedure. These cases showed a lack of understanding of procedural abstraction.

- Students were asked to demonstrate sequencing, selection, and iteration in the code that was included in their identified procedure. While the requirements state that the algorithm should be included in a procedure, an exception is made to allow students to earn credit for demonstrating their ability to write an algorithm with sequencing, selection, and iteration even if it isn't included in a procedure with parameters. Additionally, students were asked to write a description of how the algorithm shown in the code segment works in enough detail so that someone else could recreate it. In general, most students were able to present a code segment that used sequencing, selection, and iteration, but some failed to explain the algorithm represented by the code at an appropriate level of detail. Some students provided an explanation at too high a level, explaining the results of the code but not how the algorithm works. Some students were unable to provide an algorithm that included iteration. Other students used a built-in `timedLoop` function as an iteration instead of using a built-in iteration command with a loop condition.

- Students were asked to describe two calls to the code segment representing their procedure, with each call passing different argument(s), explicit or implicit, that cause(s) a different segment of code to execute in their procedure. In each case, students were asked to describe what condition was being tested by each call and what resulted from each call. Some students were able to describe different argument(s) that caused separate paths to execute in the procedure, resulting in unique results. On the other hand, some students described calls to two different procedures, or they described calls to the same procedure with different arguments and different results but that followed the same sequence of instructions in the procedure. While the requirements state that the algorithm should have one or more parameters explicitly shown, an exception is made to allow students to earn credit if the response uses an implicit parameter to simulate the passing of an argument to a procedure by setting a global variable to be used by the procedure just prior to calling the procedure.

What common student misconceptions or gaps in knowledge were seen in the responses to this question?

Common Misconceptions/Knowledge Gaps	*Responses that Demonstrate Understanding*
Video and Written Response 3a: Program Purpose and Function	
Row 1 • Confusing purpose and function when describing the program illustrated in the video component. Responses often explain what the program does or how the program works, which is not its purpose. For example, "The purpose of my program is to play a game of hangman."	Row 1 • High-scoring responses stated a purpose for the program that went beyond the function of the program itself to explain *why* the program was written and for whom. These responses indicated how the program would be used to solve a problem for the user or improve their lives in some way; that is, a reason why someone would use the program. For example, "The purpose of this program is to help people gain skills towards fluency in a foreign language to be able to communicate with more people who may not speak English."
• Not being clear about what comprises the input to and output of the program illustrated in the video component. Although this misconception was less frequent, some responses did not clearly indicate what was considered input to and output of the program. For example, "The input is the questions displayed to the user at the start of the program. The output is the results of the game."	• High-scoring responses clearly described the data the program needed to perform its function and labeled this data as input. These responses also described the data produced by the program and labeled this data as output. For example, "The input is the user's typed responses to the game's questions. The output is the questions displayed on the screen and whether or not the answer was correct."

Written Response 3b: Data Abstraction and Managing Complexity	
Row 2 • Providing several lists and either naming a list that is not being used or not showing how any of the lists are being used. For example, the first code segment shows a list named `cities` being initialized with data. The second code segment shows the list `citiesOutput` being accessed via a for loop, and it is not the case that `citiesOutput` is a parameter that could feasibly refer to the same list as `cities`. • Confusing a database with a named list in the program. For example, “The list used in my program is 100 birds of the world,” where “100 birds of the world” refers to a file that contains information about birds from which data is extracted and stored in named lists in the program. • Showing an empty list being created instead of data being stored in a list.	Row 2 • High-scoring responses identify only one list and show how data is added to that list in the first code segment and how multiple elements of the list are accessed in the second code segment. • When using a database, high-scoring responses identify the name of the list variable that stores data from that database to be used within the program. • High-scoring responses show the list being initialized to contain data or show data being added to an empty list after creating it.
Row 3 • Explaining how the use of the list manages complexity by stating how the program would be rewritten in a generic way that does not reference the submitted program. For example, “The program would be written differently had I not used the list because I would have had to use more variables and a lot more if/else if statements. This would make my program a lot longer and more complex.” • Explaining how the use of the list manages complexity by stating how the program could not be written without the list even when it would be possible. For example, for a program that uses a list of four elements, stating that it would not be possible to implement the program without the list, when it would be possible to use a separate variable for each element in the list.	Row 3 • High-scoring responses describe how the program would need to be rewritten in a more complex manner without the list, referencing specific instructions and functionality in the submitted program. For example, “The list stores a set of 30 colors to test against, so without the list, I would have to test each of the colors individually in a long sequence of 30 if statements to update the score.” • High-scoring responses precisely identify why an alternate solution would not be possible. For example, “It would not be possible to implement my program without a list because the user can input an unknown number of scores, so it would not be possible for the program to know how many variables it would need to store these inputs.”

• Providing an alternate solution without clearly explaining why the alternate would be more complex. For example, "If I did not use the list, each state name would have to be stored in a separate variable," without explaining why these separate variables would make the code more complex.	• High-scoring responses clearly explain why the alternate response adds complexity to the program. For example, "If I were to write this code without the list, I would need to write a series of if statements for each character like this: `var firstHiraganaCharacter;` `if(firstNumber == 0){` `  firstHiraganaCharacter = ...;` `}else if(firstNumber == 1){` `  firstHiraganaCharacter = ...;` `}else if...` A new variable would be declared called `firstHiraganaCharacter` with nothing stored. This variable will store characters instead of the list … Basically, without a list, an if/else statement would be needed for each of the seventy-one Japanese characters used in my program, three times. This would create an unnecessarily long series of code, making it harder to read and debug."
• Using a list in a manner that does not manage complexity, such as: ○ Storing elements in a list, but only showing the use of the first element or the sum of the elements in the list. In these cases, it would be simpler to use a single variable. ○ Using a series of if statements instead of a loop to access each element in the list. In these cases, the code is not shorter due to the use of the list because the data contained in the list could be directly accessed in each if statement. ○ Using a list as a counter, storing arbitrary data in the list just to determine its length. For example, storing five copies of the number 1 in a list and then reporting the score is 5 because the length of the list is 5.	• High-scoring responses display lists that are used to manage complexity by storing a variable number of data values that need to be processed using a loop or iterator. For example, a list may contain a sequence of temperatures of variable length, and the program will display those temperatures that are greater than the average of the temperatures in the list. For example: `sum = 0` `for t in temps:` `   sum = sum + t` `avg = sum / len(temps)` `for t in temps:` `   if t > avg:` `      print t`

Written Response 3c: Procedural Abstraction and Algorithm Implementation	
Row 4	Row 4
• Not including both what the procedure does at a high level AND how it contributes to the functionality of the program. For example, “This function contributes to my program’s purpose because it filters the NHL team names, cities, and logos based on the division.” This type of response often occurs when the response includes a procedure that carries out most or all of the functionality of the program, making it difficult to distinguish between the procedure’s functionality and how the procedure contributes to the whole program since it is essentially the whole program already.	• High-scoring responses include two parts to the response, stating the overall function of the procedure in a sentence and then also stating how it contributes to the overall program by describing when it is executed, what subsequent code is run in the program as a result of this procedure having been executed, or how the procedure’s functionality ties into the rest of the program’s functionality. For example, “The function `creategroup4` uses the list of names and selects a random element from it with each iteration. This contributes to the overall functionality of the program by creating the fourth group which is required when the user selects to make four or more groups.”
• Failing to use one or more explicit parameters in the submitted procedure. In these cases, some responses select procedures that use implicit parameters (e.g., global variables or text boxes which are set just prior to the procedure being called), while others simply prompt for the data the procedure requires from within the procedure itself.	• High-scoring responses include a procedure that takes one or more explicit parameters that are necessary for the procedure to perform its function and that does not rely on external data from global variables or user interface elements.

• Including a parameter that has no effect on the procedure either by ignoring the parameter or initializing it to another value. For example,
`function rps(userchoice):`
`   userchoice = input("Enter your`
`                guess: ")`
`   ...`
	• High-scoring responses include a procedure that uses the parameter value(s) in the procedure body without overwriting the argument being passed into the procedure.

• Including an event handler or other built-in construct as a student-developed procedure.	• High-scoring responses include a custom procedure written by the student that is not part of the language itself. In programs where an event handler is used, the code within the event handler calls this student-developed procedure.
• Including code that is not encapsulated in a procedure.	• High-scoring responses include one clearly defined procedure, including the procedure’s header with explicit parameters, in the first code segment.

Row 5	Row 5
• Including an algorithm that does not contain iteration or have meaningful iteration. Some responses included extensive nested selection (if/else if/else if/…) but no iteration. Some responses included a loop that only ran through one iteration; for example, a loop where a break statement will always execute on the loop's first iteration.	• High-scoring responses included a loop that contributed meaningfully to the algorithm. These loops often repeated more than three times or ran until a specific condition was met (e.g., by using a while loop).
• Providing a very vague and brief description that does not include enough detail for someone else to recreate the algorithm. For example, "This procedure divides each 'vibe' list into gender-specified lists which are split into a 'Male' and 'Female' category."	• High-scoring responses include enough detail of each step of the algorithm to allow someone to reasonably recreate the algorithm.
• Providing an explanation of each line of the code rather than a description of the algorithm. For example, "Line 62 has an input statement. Then on line 63 there is a for loop with `i = 1` until `userChoice`. Lines 64-68 show another if statement for when the input is 'Yes'…" These descriptions do not abstract far enough away from the code for the description of the algorithm to be clear enough that it could be recreated.	• High-scoring responses summarize several lines at a time but include all the functionality that the code implements. For example, "The `help` function decides if the user would like to see either state information (by inputting a one) or a program outline (by inputting a two). If the user were to select state information, a short description would be displayed describing the purpose of the states and tax ranges. The user is then asked if they would like to see a list of the states abbreviations by inputting a 1 for yes and a 2 for no. If they select yes, then a for loop will iterate through a list of the state's names and appropriate abbreviations and print them out. Otherwise, the user can select to see the program outline, describing the ability to choose state of residency, and choosing between inputting a single, daily, or weekly purchase to be tracked."

• Providing a description of the algorithm that does not match the included program code. Often these responses explained steps out of order or failed to include portions of the algorithm. For example, describing the lines:
 	
`if (PlaceLocation[i] ==`
`"California" && Location ==`
`"Spring") {`
` appendItem(FilteredPlaceName,`
`            PlaceName[i]);`
` appendItem(FilteredPlaceImage,`
`            PlaceImage[i]);`
`}`
 	
with the text "When the if statement comes out to be `true` these lines will traverse the list going through the names and images of all the national parks and displaying the ones that correspond to the location given." This response is conflating what happens in the overall for loop that encloses these statements with what happens in the specific if statement.
	• High-scoring responses accurately described how the algorithm works, describing the steps in the correct order without omitting any important pieces.

Written Responses 3d: Testing	
Row 6	Row 6
• Describing calls to two different procedures rather than two calls to the same procedure with different arguments. For example, `move_left(2)` and `move_right(2)`. • Describing two different parts of the program and what behavior occurs during these parts.	• High-scoring responses included two calls to the same procedure with different arguments that caused different behavior to occur in the procedure. • High-scoring responses identify the single procedure given in response 3c and provide two different sets of values for the parameters that cause different behavior to occur inside the procedure.
• Describing two calls to the same procedure with different arguments that cause the same sequence of code to execute in both cases. A common misconception in this situation is that a procedure that has two different return values must be executing different instructions inside. For example, this situation occurs when the two calls cause a for loop in the procedure to execute a different nonzero number of times to produce a different return value.	• High-scoring responses provided two different arguments that would cause a different sequence of code to execute in each case, leading to a unique result (output or return value). Frequently these responses would include arguments where each would cause two different branches of a conditional statement to execute.

• Setting the value of an implicit parameter inside the procedure instead of immediately before the procedure is called. For example, using `choice` as the implicit parameter,
 	
`function select()`
`{`
`   choice = getText("Name");`
`   print(choice);`
`   ...  // choice determines path`
`}`
	• High-scoring responses that used implicit parameters set the value of the parameter immediately before the call to the procedure to simulate parameter passing and then described two different paths being taken through the procedure. For example, using `choice` as the implicit parameter,
	
	`choice = getText("Name");`
	`select();`
	`...`
	`function select()`
	`{`
	`   print(choice);`
	`   ...  // choice determines path`
	`}`

• Not identifying specific argument values passed in the two calls to the procedure, but instead describing which part of the code will run with each call. For example, "The first call is when the first if statement is true and the element is found in the list. The second call is when the else statement runs because the element is not found in the list."	• High-scoring responses clearly identify the values of specific argument(s) that are passed to the procedure in each call.
• Providing two calls with different arguments where the difference between the segments of code that execute in each case is caused by some other factor (e.g., user input received within the procedure) and not by the value of the arguments.	• High-scoring responses included procedures where the branches in the code were determined by the argument value(s) and not some other factor within the procedure.

• Providing arguments that cause different segments of code to execute, but these segments are identical to one another. For example, using the arguments `1` and `2` and the following procedure:
 	
`function win_message(choice) {`
`   if (choice == 1) {`
`      print("YOU WIN!");`
`   }`
`   if (choice == 2) {`
`      print("YOU WIN!");`
`   }`
`}`
	• High-scoring responses included arguments that caused different segments of code to execute that performed at least some action differently to illustrate a difference between using each of the argument values in testing.

• Including code that is not encapsulated in a procedure.	• High-scoring responses that addressed two separate testing cases included one clearly defined procedure with explicit or implicit parameters in the first code segment.

Based on your experience at the AP® Reading with student responses, what advice would you offer teachers to help them improve the student performance on the exam?

In general, some students have trouble understanding what is required for each prompt in the performance task. Provide several completed tasks for students to review and analyze, asking them to determine whether each requirement was met or not, and why. As a teacher, review the high-quality examples to make sure you understand the nuances of the scoring criteria so that you can score the shorter student examples accurately.

Students need explicit instruction and experience taking screen captures of code segments and incorporating them into their responses. Code submitted for scoring should be as clear as possible (not blurry), and text should be at least 10-point font size.

If a student wishes to use an unconventional programming language for the Create Performance Task, evaluate its ability to clearly address the requirements of the task and advise the student accordingly.

The following bulleted list gives more specific advice for each part of the Create Performance Task.

Responses 2 and 3a: Program Purpose and Function

- Have students review high-quality examples of the Create Performance Task to become familiar with the difference between function and purpose. Give students additional examples of computer programs and ask them what the purpose of each program is to see if they can identify the problem being solved by the program or the creative or artistic pursuit. Ask students to think about “why” the program exists or “how” it might help the user solve some larger problem as opposed to “what” the program does or “how” to win the computer game.

- Ensure that students have access and opportunity to practice using computational video tools to capture their program features. Integrate the use of computational tools such as screen capture and creating short videos into multiple assignments. Assist students in learning how to make sure any text in the video is clearly visible and readable for scoring.

- Give students examples of computer programs and ask them to identify what explicit data are being input to the program and what explicit data are being output to the user. For input data, have students identify specific types of data like numbers, letters, images, mouse clicks, etc. For output data, have students identify specific types of data like text messages, colors, sounds, movement of objects, etc.

- Make it clear to students that while it is OK to base their program on a sample program used in class, they must make significant changes to the sample program by adding additional functionality. The program code used for their written responses should be newly student-developed program code, and their answers should address their newly written code.

Response 3b: Data Abstraction and Managing Complexity

- Give students examples of program code that initializes and uses a list, highlighting the difference between initializing a list and creating an empty list. Have students identify in the code where the initialization and use of the list are happening.

- Give examples that use lists that can be of arbitrary length to illustrate the power of using lists to store a collection of data. Compare these to examples where the lists are of fixed length. Have students write code to process elements of an arbitrary length list by using an index (e.g., `numlist[j]`) rather than hard-coding access (e.g., `numlist[1]`).

- Provide practice using lists with a large number of elements that will require students to write code that is more abstract and able to handle a change in the number of elements more easily than if the code was written in a more hardcoded way with a list containing just a few elements.
- Discuss why a well-designed list makes the code less complex by showing what would happen if the list were not present. Have students explain in their own words why the list is necessary by referencing the code. Show students examples of lists that can be replaced easily without making the code more complex (e.g., a list containing data only to later determine how many items are in the list, which can be replaced with a counter variable).
- Remind students that even if their code segment uses multiple lists, they should clearly identify one list and respond to the prompts based on this one identified list only. The list they identify and describe must be one they create, not one that they are given, such as a data table from a third-party provider.
- If a student passes a list to a function to access the elements of the list, the student should be reminded to make it clear that the list inside the function as a parameter may have a different name but is referencing the same list that was identified for the prompt.

Response 3c: Procedural Abstraction and Algorithm Implementation

- Give students examples of program code that contain procedures with explicit parameter(s), and have the student identify what code makes up the procedure and where the parameter(s) get their values. Show how using global variables is not the same communication mechanism as using parameters to pass data into a procedure: explain that when a program uses a global variable, this may be modified incorrectly by any part of the program, making debugging harder, whereas passing data as a parameter gives the procedure a local copy to use.
- Encourage students to use explicit parameters over implicit parameters since this will make their code easier to debug and easier to explain for the Create Performance Task. Remember that points are not awarded in row 4 if only implicit parameters are used.
- Remind students to identify one procedure with explicit parameter(s) to focus their response and to only include this procedure and a call to this procedure in response 3c. If the procedure calls additional student-developed procedures, students can include them as well in the first code segment, but they should be listed after the first procedure, and the student should be sure to focus their response on the first procedure. Students should not include multiple procedures and calls in their response.

Response 3d: Testing

- Give examples of testing a single procedure with different arguments, and trace the path taken in the code to generate each result to show that the paths are different or that different parts of the code are being tested.
- Give an example where a response shows two different procedures, with one test case per procedure, and explain why this response does not address the requirements for the task.

- Remind students that each identified path through the procedure must do something that is distinct from the other path and should not just be the same code duplicated with two if statements in order to artificially make two paths in the procedure.

What resources would you recommend to teachers to better prepare their students for the content and skill(s) required on this question?

- AP Classroom has scaffolded Create Performance Task prompts that can be paired with the programs you are already having your students complete. Pairing these shorter scaffolded writing opportunities with all of the programs students need to complete will give students extra practice without being as overwhelming as having to complete an entire practice task. These scaffolded questions help to build student stamina with and confidence in answering the written response prompts for the Create Performance Task.

- The College Board webinar "AP Computer Science Principles: How to Avoid Plagiarism and Exam Violations" discusses plagiarism and exam violation policies, shows examples of student plagiarism, and covers instructions to ensure students' compliance with the policies. It also answers many frequently asked student questions. (https://www.youtube.com/watch?v=m8dpz32HlB0)

- The College Board webinar "Evaluating the Create PT as an AP Reader" provides teachers with insight on how each row of the scoring guidelines is applied to the samples and was conducted in a similar manner to how readers are trained. (https://www.youtube.com/watch?v=mCM3cFBBJvo)

- The College Board will be offering additional webinars this fall to explain some changes that will be made to the administration of the Create Performance Task and help teachers and students prepare for the 2024 exam.