Chief Reader Report on Student Responses:

2021 AP® Computer Science Principles Performance Task

• Number of Students Scored	116,466		
• Number of Readers	488		
• Score Distribution	Exam Score	N	%At
	5	14,432	12.4
	4	25,223	21.7
	3	37,797	32.5
	2	23,151	19.9
	1	15,863	13.6
• Global Mean	2.99		

The following comments on the 2021 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.

Task: Create Performance Task | **Topic:** Application from Ideas

	Max. Points:	**Mean Score:**
Row 1	1	0.47
Row 2	1	0.66
Row 3	1	0.35
Row 4	1	0.44
Row 5	1	0.40
Row 6	1	0.36

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. The student should be able to demonstrate the program running and explain its purpose, how it functions and how it handles input and output of information.

Programs typically process collections of data of the same type 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 to hold data that is critical to fulfilling the program's purpose. Students must explain why the list manages complexity in the program 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 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).

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 express themselves creatively (i.e. in an artistic sense). Students were also asked to create a video that demonstrated some functionality of their program that requires user input and program output. In general, most students were able to write a working program and create a video that accurately demonstrated functionality, input and output that they describe in their written response.
- Some students were able to state the purpose of the program and describe the broader problem that is being solved for the user, whereas other students 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 did not explicitly state what information was going into the program as input and what data was coming out of the program as output, instead simply describing how the program runs.

Data Abstraction and Managing Complexity

- Students were asked to provide two code segments from their program, the first showing the initialization of a list of data and the second showing how that list is used in their program to reduce complexity. Some responses used other acceptable representations of lists, including: array, dictionary (associative array), or a tree. Most students were able to identify a list in their program and provide two code segments that demonstrated the creation and use of the identified list. Some students confused creating an empty list with initializing the list with data.
- Some students were able to use the lists in a manner that reduces complexity by generalizing list element access using an index or subscript to allow for arbitrary-sized lists and iteration over lists. In general, 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.

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. Students who described what the procedure does at a higher level, rather than giving a line-by-line explanation, found it easier to connect the procedure to the overall function of the program. Most students were able to provide the two code segments and describe what the procedure does. Some students had trouble explaining how the procedure contributed to the overall program, particularly when the selected procedure performed almost the entirety of the overall functionality of 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. When students submit multiple procedure, the score is based on the first procedure and those that it calls. Some students submitted multiple procedures that weren’t related, and the first procedure didn’t meet the requirements.
- Students were asked to demonstrate sequencing, selection, and iteration in their code segment, whether it was included in a procedure with parameters or not. Additionally, students were asked to write a description of how their 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 code in enough detail to allow someone else to write a similar code segment.
- Students were asked to describe two calls to the code segment representing their procedure, each call passing a different argument(s), explicit or implicit, that cause 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 results 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 described calls to the same procedure with different arguments and different results, but that followed the same sequence of instructions in 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. The function of the program is not its purpose. For example, "the purpose of the program is to let the user choose a country and get facts about that country." If the program is a game, the purpose is not the object of the game. For example, "guess the colors that the program selected in the least amount of tries possible." • Describing input or output of the program obliquely or not at all. For example, "The user clicks on the blue target and the game is over." This response does not identify the output. Another example, "The user enters their data and the program outputs the result." This response does not specify what specific data is input or output.	Row 1 • High-scoring responses stated a purpose for the program that went beyond the function of the program itself. These responses indicated how the program would be used to solve a larger problem for the user or improve their lives in some way. For example: "The overall purpose of this app is to aid with language barriers for those who find difficulty in communicating with others in a language foreign to them." • High-scoring responses also identified the input into the program and output of the program clearly. For example, "the input is … when the user clicks on one of the six boxes and enters a valid character. The output is the color displayed below the boxes." Another example, "The user will input their classes along with what time each class ends. The user will also enter the current time. The program will then output the calculated time for each class to end and the time for the school day to end."
Written Response 3b: Data Abstraction and Managing Complexity	
Row 2 • Identifying the list as the table from which they built the list, but then answering the prompts based on the list they built. For example, `Names = getColumn("Fullnames", "country");` and "The name of my list is `Fullnames`" (instead of `Names`). • Initializing the list as an empty list without showing any data stored in the list. For example, `mylist = [ ];` • Providing many lists and not identifying which list is being described for the prompt.	Row 2 • High-scoring responses that used a third-party provider API to access a data table identified the name of the list being created by the table method, not the table itself. • High-scoring responses provided code that clearly stored data in the identified list. For example, for the list `guessList`, the student showed data being added to the list with `guessList.append(nextGuess)` Others initialized their lists with a loop variable to control the list position. For example, using loop variable `i`, `vals[i] = newValue;` • High-scoring responses identified only one list and answered the prompts based on this single list.

• Not indicating that a list parameter is the same as the identified list. For example, when the list `x` is passed as an argument to the parameter `myl`, the response does not explicitly mention that the two names, e.g., `x` and `myl`, refer to the same list. • Failing to specify what the data in the list represented.	• High-scoring responses that use a procedure to process the list identify that the list parameter is the same list as the one initialized in the prior code segment. • In addition, most responses with correct list usage included correct statements that indicated what the data in the list represented.
Row 3 • Explaining how the use of the list manages complexity by solely using generalizations about how lists could manage complexity in a program. For example, "This list manages complexity in the program because if it weren't for the lists, each individual variable would have had to be written out and given its own name." This response does not relate this response to their own list. • Using a list when a list was not necessary. Some examples include: ○ a list with only 2 elements can be replaced with two variables; ○ a list of strings that is only output in the same order could be stored as a single string with concatenation; ○ a list that is used only for its length, where the data inside does not matter, can be replaced with a counter variable. • Building a list with a constant size that cannot be changed without rewriting a significant portion of the program code. For example, the program only works if the list has 4 elements in it.	Row 3 • High-scoring responses explained how their specific list managed complexity by stating how it would have to be rewritten without lists. For example, "Without using `randomWordList`, I would've had to store each random word in a variable with the variable name being a number assigned to each random word (Ex: `randomWord1`). … Writing my code this way would make my code unnecessarily longer and complex because with my list I'm already able to select a word from my list in one line of code. … My list manages complexity as I can store as many words as I want inside a single list versus having hundreds of variables." • High-scoring responses also used lists in appropriate ways to hold a collection of related data. For example, "vacation destinations in the Western hemisphere". • High-scoring responses also used lists to store an arbitrary number of data elements and loops to process this data which reduced the complexity of their code. These submissions set their loops to run a number of times equal to the length of the list as given by a length function.

Written Response 3c: Procedural Abstraction and Algorithm Implemenation	
Row 4 • Failing to use one or more explicit parameters in the submitted procedure, passing data to the procedure using global variables instead. • Using an explicit parameter for the submitted procedure, but then immediately overriding its value inside the procedure, making the parameter passing irrelevant. • Including a larger segment of code that contains multiple procedures that either aren't related to each other through calls or the response confused which procedure was the primary one to describe. • Describing the function of the procedure and the overall contribution to the program using the same response, i.e. not understanding that the function and the contribution are two separate concepts. This occurred most often when the procedure selected performed almost all of the computation for the entire program. • Identifying a code fragment that is not encapsulated as a procedure.	Row 4 • High-scoring responses included a procedure that had one or more explicit parameters that contributed to the functionality of the procedure. • High-scoring responses used the parameter value(s) in the procedure body without overwriting the argument being passed into the procedure. • High-scoring responses included one clearly defined procedure with explicit parameters and a single procedure call with relevant argument(s). • High-scoring responses chose procedures that performed some smaller subset of the overall computation, but not the entire computation. In this way, students were able to distinguish between what the procedure does and how it contributes to the overall functionality of the program. • High-scoring responses contained a full procedure, including a procedure header with the procedure name, parameter(s), and return type (if needed).
Row 5 • Using selection in a trivial manner or not at all. For example, "if `y > 0` …" when y is always greater than 0. Some responses included extensive nested selection (if/else if/else if/…) but no iteration. • Including an algorithm that does not contain iteration or have meaningful iteration. For example, some responses highlighted an algorithm that includes a loop that only iterates once or that iterates multiple times but only the final iteration has an effect on the functionality of the program. • Providing not enough detail in their description of the algorithm to recreate the algorithm by someone else. Their description is very vague and brief.	Row 5 • High-scoring responses included reasonable uses of selection with conditions that were non-trivial and caused different parts of the algorithm to execute. • High-scoring responses included a loop that contributed meaningfully to the algorithm. These loops often repeated a number of times greater than 3 or ran until a specific condition was met (e.g., a while loop). • High-scoring responses included significant detail of each step of the algorithm to allow someone to reasonably recreate the algorithm. Some responses included line numbers that were referenced in the response to make their description easier to follow.

• Describing an algorithm that does not correspond to the code segment that was given first in their response in 3c. Some responses provided multiple procedures or code segments and did not describe how the first of these segments worked in detail.	• High-scoring responses included just one algorithm (code segment) or procedure and described the algorithm in reasonable detail. These responses often identified the name of the procedure if a procedure was given to make it clear what algorithm was being described.
Written Responses 3d: Testing	
Row 6 • Describing calls to two different procedures rather than two calls to the same procedure with different arguments. For example, `is_prime(6)` and `is_perfect(6)` • Describing two calls to the same procedure that caused the same sequence of code to execute in both cases. A misconception here is that a procedure that has two different outcomes must be doing something differently inside with respect to the computation. For example, the parameter indicates the number of times the loop repeats, but the exact same loop body is executed in both cases. • Setting the value of an implicit parameter inside the procedure instead of immediately before the procedure is called. • Identifying input arguments and output values but not describing what conditions are being tested in each case. • In some cases, descriptions were implausible given the code supplied in response 3c. For example, a response indicates that an argument will cause a specific output after a loop is completed, but the argument actually causes an infinite loop.	Row 6 • High-scoring responses included two calls to the same procedure with different arguments. • 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). For example, a linear search on a list which has two outcomes if one argument matches a list element, and the other argument does not match a list element. In the first case, the body of an if statement is executed, and in the second case, the body of the if statement is never executed, causing a different outcome. • High-scoring responses that used implicit parameters set the value(s) of the parameter(s) before the described procedure call in anticipation of calling the procedure. The procedure then used these (global) variables to perform its computation, behaving like an argument that has been passed. • High-scoring responses included a description of each call, highlighting what code is being tested in each case and how these tests are unique. • High-scoring responses accurately described what conditions were being tested for each test case without making statements that were inconsistent or implausible with the submitted procedure.

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, give students several opportunities to complete a practice Create Performance task of shorter duration to gain a better understanding of the learning objectives and skills required for the task. Score these against the established scoring criteria to help students improve their understanding. 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 must be clear and legible to be evaluated. Incorporate a few exercises where students just capture and store code images to test their understanding of the tools necessary for the Create Performance task.

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 broader problem being solved by the program. Ask students to think about "why" the program exists 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 into the program and what explicit data are being output to the user.
- Make it clear to students that while it is okay to base their program on a program used in class, they must make significant changes to the program by adding additional functionality. The program code used for their written responses should be newly student-developed program code.

Response 3b: Data Abstraction and Managing Complexity

- Give students examples of computer code that initializes and uses a list; highlighting the difference between initialization of 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 hardcoding 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.
- 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.
- 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.

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.
- 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.

- 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, but they should be listed as subsequent to the first procedure and the student should be sure to focus their response on the first procedure.

Response 3d: Testing

- Give examples of testing for 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.

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

- Throughout the school year, pair the Create Performance Task topic questions in AP Classroom with smaller projects. In the AP Question Bank, filter the question type to "PT: Formative" and add up to four of these scaffolded and aligned written response prompts to a topic quiz and assign along with your current programming project. The prompts are designed to scaffold the students writing as well as programming skills. At the start of the school year, you may assign the prompt that requires students to write about a variable first and later assign the one about lists. As they build their content understanding of sequencing, selection, and iteration, you will find there are topic questions that ask them to write about just one of these three things instead of combining them together into a more complex algorithm. There are 22 formative Create Performance Task topic questions available for you to mix and match and assign alongside your programming projects. Because the projects are different, you can feel free to reuse prompts for additional practice to measure student growth.

- The following AP Daily Videos and corresponding Topic Questions can be found in AP Classroom to support the Create Performance task.

  Responses 2 and 3a: Program Purpose and Function
  - Topic 1.2, especially videos 1 and 2

  Response 3b: Data Abstraction and Managing Complexity:
  - Topics 3.2 and 3.10

  Response 3c: Procedural Abstraction and Algorithm Implementation
  - For procedures – topics 3.13 and 3.12
  - For algorithms – topics 3.6, 3.7, 3.8, and 3.9

  Response 3d: Testing
  - Topic 1.4