Chief Reader Report on Student Responses:

2024 AP® Computer Science Principles Performance Task Set 2

• Number of Students Scored	175,261		
• Number of Readers	680		
• Score Distribution	Exam Score	N	%At
	5	19,105	10.9
	4	34,979	20.0
	3	58,034	33.1
	2	35,542	20.3
	1	27,601	15.7
• Global Mean	2.90		

The following comments on the 2024 performance task for AP® Computer Science Principles were prepared by the Chief Reader, Thomas 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.

Create Performance Task

The written response prompts for the AP Computer Science Principles exam are centered around the Create Performance Task, in which students dedicate at least nine hours of class time to develop a computer program that addresses a need, concern, personal interest, or creative expression. Students may use any programming language, including text-based languages (e.g. Python, JavaScript) or block-based languages (e.g. Scratch, Snap!) to create their program. They are allowed to work with partner(s), use starter code, and use AI tools (with citation through comments in the program code component). Students individually create a short video that demonstrates the running of their program, illustrating input, functionality, and output. They also prepare a Personalized Project Reference sheet with screenshots of program code (without any comments). Their Personalized Project Reference includes a student-developed procedure definition with at least one explicit parameter and a call to the included procedure, as well as how data are stored in a list or collection and how the data in the list or collection are used in the program.

Students use the Personalized Project Reference sheet during the administration of the AP exam to answer four written response prompts about their program. The video, program code, and written response answers account for 30% of the total score for the AP Computer Science Principles exam.

Each prompt addresses at least one of the listed learning objectives for that prompt:

Written Response 1: Program Design, Function, and Purpose

- CRD-2.A: Describe the purpose of a computing innovation.
- CRD-2.B: Explain how a program or code segment functions.
- CRD-2.C: Identify input(s) to a program.,
- CRD-2.D: Identify output(s) produced by a program.
- CRD-2.E: Develop a program using a development process.
- CRD-2.F: Design a program and its user interface.
- CRD-2.G: Describe the purpose of a code segment or program by writing documentation.

Written Response 2(a): Algorithm Development

- CRD-2.B: Explain how a program or code segment functions.
- AAP-2.E.b: Evaluate expressions that use relational operators.
- AAP-2.F.b: Evaluate expressions that use logic operators.
- AAP-2.H.b: Determine the result of conditional statements.
- AAP-2.J: Express an algorithm that uses iteration without using a programming language.
- AAP-2.K.b: Determine the result or side effect of iteration statements.
- AAP-2.L: Compare multiple algorithms to determine if they yield the same side effect or result.
- AAP-2.M.a: Create algorithms.
- AAP-2.M.b: Combine and modify existing algorithms.

Written Response 2(b): Errors and Testing

- CRD-2.I.a: Identify the error.
- CRD-2.I.b: Correct the error.
- CRD-2.J: Identify inputs and corresponding expected outputs or behaviors that can be used to check the correctness of an algorithm or program.

Written Response 2(c): Data and Procedural Abstraction

- AAP-1.D.a: Develop data abstraction using lists to store multiple elements.
- AAP-1.D.b: Explain how the use of data abstraction manages complexity in program code.
- AAP-2.O.a: Write iteration statements to traverse a list.
- AAP-2.O.b: Determine the result of an algorithm that includes list traversals.
- AAP-3.B: Explain how the use of procedural abstraction manages complexity in a program.

Students should design their program carefully based on the Exam Information given in the Course and Exam Description so that it meets all the requirements. Meeting all the requirements in their program code gives students the best opportunity to answer the prompts given on the exam.

In the following sections, each question is addressed in more detail, including what is expected based on the requirements and written prompts. This is followed by examples of responses from the actual exam that show misconceptions compared with responses that demonstrate correct understanding of the requirements or written prompt. Suggested tips on helping teachers prepare to improve student performance and available resources are presented at the end of this report.

Question 1

Task: Video, Program Requirements, and Written Response 1
Topic: Course project and program design, function, and purpose

	Max Points:	**Mean Score**:
Video:	1	0.94
Program Requirements:	1	0.78
Written Response 1:	1	0.77

What were the responses to this question expected to demonstrate?

The responses to this question were expected to demonstrate that the student could:

- demonstrate the program input, functionality, and output in a short video (Course Project: Video),
- develop a working program that includes a student-developed procedure including sequencing, selection and iteration, and the creation and use of at least one list or collection (Course Project: Program Requirements), and
- explain at least one valid program input and how the program uses the input to perform its functionality (Written Response 1: Program Design, Function, and Purpose).

Students were asked to record a video demonstrating their program's functionality including input and output. Input could be user input (e.g. mouse clicks, text-entry) or file or database input. The source of the input can be verified by examining the program code if the video did not clearly capture the input.

The students were then asked to provide, on their Personalized Project Reference sheet, segments of code from their program that demonstrated a student-developed procedure which utilized selection and iteration appropriately, along with segments of code showing the creation of a list (or collection) and use of the same list (or collection) to contribute to the purpose of the program. Designing a program that includes these core features is critical to understanding basic programming in any language.

In Written Response 1, students were asked to identify the expected users of the program and to explain how the program addresses a concern or need of those users.

How well did the responses address the course content related to this question? How well did the responses integrate the skill(s) required on this question?

- Students were generally able to develop a working program and to demonstrate program input, functionality, and output in the submitted video responses. In some cases, the program code had to be referred to when the input was a touch screen or data source, but most students demonstrated user-generated input from a mouse click or text entered into a prompt box.
- Some students submitted programs based on example programs from content providers, but the stronger submissions were original programs coded by the student or in some cases, co-authored with a partner. Students were also permitted to use generative AI tools as supplementary resources. In these cases, students identified those sections of code in which they collaborated with partners or used AI in the program code component.
- Students were asked to provide code segments demonstrating a student-developed procedure which included sequencing, selection and iteration, and code segments demonstrating the creation and use of a list or another collection type. The response required that these be essential to the program's

functionality. The use of selection and iteration should not be trivial. Additionally, when implementing iteration, students should make sure there is a way of exiting the loop when some condition is met.

- Most students correctly developed and called a defined procedure that fulfilled expectations. For some students, particularly those using block-based programs, event handlers were often used instead of student-developed procedures.
- There was a variety of collection types used, and the use of lists and collections varied across programs. Some used lists as collections of data to be retrieved for a user, and some used lists to store user responses or scores in the case of game programs. Overall, most students used a list or collection in a way that was appropriate for the purpose of their program. Some students submitted code that could have been designed more efficiently if a list had been used instead of repeated conditional statements or submitted code that identified a set of individual variables as the list even though these were not stored in a list or collection. This suggests that their understanding of lists and their uses is still developing.
- Students were asked to identify the expected users of the program and to explain how their program addresses a concern or need of those users. Most students identified some group of users that would reasonably be using their program and what concern or issue that program addressed for these users. Some students identified a group of users, but then identified concerns of the programmer rather than those users. Some students identified the group of users as everyone, which is too broad an answer. Some students focused on the functionality of the program instead of how the program is useful for the users it is designed for.

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

Common Misconceptions/Knowledge Gaps	*Responses that Demonstrate Understanding*
Course Project Video	
• Not having input to the program while it is running. For example, a video shows the program running and characters interacting, but input to the program is not demonstrated in the video.	• High scoring responses show the program receiving and responding to graphical input such as mouse clicks or menu selections. For example, the video shows the user selecting Mercury from a dropdown menu and then clicking a button labeled “get information,” at which point the user interface displays facts about Mercury.
• Not showing the program running but instead explaining how the program runs. For example, a video that shows a slide show explaining the program.	• High scoring responses show the program running. For example, a video that shows a user entering grades in response to a prompt and then shows the program displaying the GPA based on the grades the user entered.

Program Requirements

- Lack of a student-developed procedure. For example, the response provides program code that includes an event handler instead of a student-developed procedure. Although event handlers collect a sequence of instructions, they are built-in structures and are therefore not student-defined. They also cannot be called throughout the program. For example:

```
when this sprite clicked:
    repeat 25:
        change size by -5
    if message = True:
        say "Goodbye" for 2 seconds
```

- High scoring responses include a student-developed procedure that contains selection and iteration, which is called in the program. For example:

```
def sortChoices(listSearch):
  global choices
  i = len(choices) - 1
  while i >= 0:
    if choices[i] not in listSearch:
      del choices[i]
    i -= 1
```

  which is called here:

```
sortChoices(selectedList)
```

- Lack of use of iteration. For example, the response provides program code that uses a series of conditional statements, but no loop structures. For example:

```
if (position[0] == "prize") {
   answer = 0;
}
else if (position[1] == "prize") {
   answer = 1;
}
else if (position[2] == "prize") {
   answer = 2;
}
else if (position[3] == "prize") {
   answer = 3;
}
else {
   answer = 4;
}
```

- High scoring responses use iteration to accomplish something meaningful in the program. For example:

```
for (var i = 0; i < list1.length;
     i++)
{
  if (formatList[i]== formatString)
{
    catIndex = i;
  }
}
```

This iteration statement uses a loop and a conditional to select a cat that matches the user's input.

- Inclusion of a list that was not used in the operation of the program. For example, the program code creates a list of six elements, and then immediately assigns six variables to the values of the elements in the list. The code then uses the variables and not the list in the program. For example:

```
var colors = ["red", "green",
"blue", "yellow", "cyan",
"magenta"];

var color0 = "red";
var color1 = "green";
var color2 = "blue";
var color3 = "yellow";
var color4 = "cyan";
var color5 = "magenta";

print("Choose from these colors:\n");
print(color0 + "\n");
print(color1 + "\n");
print(color2 + "\n");
print(color3 + "\n");
print(color4 + "\n");
print(color5 + "\n");
```

- High scoring responses define and use a list (or other collection type) in a meaningful way within the program code. For example, a response defines:

```
var lines = [
["eb1", "eb2", "eb3"],
["eb4", "eb5", "eb6"],
["eb7", "eb8", "eb9"],
["eb1", "eb4", "eb7"],
["eb2", "eb5", "eb8"],
["eb3", "eb6", "eb9"],
["eb1", "eb5", "eb9"],
["eb3", "eb5", "eb7"]
];
```

and then uses `lines` as follows:

```
for (var i = 0; i < lines.length;
     i++)
{
   var a = lines[i][0];
   var b = lines[i][1];
   var c = lines[i][2];
   if (getText(a) !== "."
       && getText(a) === getText(b)
       && getText(a) === getText(c))
   {
      if (getText(a) === player)
      {
         showElement("uWin");
      }
      else
      {
         showElement("cWin");
      }
  ...
```

Written Response 1: Program Design, Function, and Purpose	
• Describing the program functionality instead of how the program addresses a concern or interest of users. For example, “The purpose of this program is to create a quiz that will test the user on their knowledge of Taylor Swift songs.”	• High scoring responses clearly identify the user, a user concern, and how the program addresses this need or concern. For example, “My program is a calculator that is used to find the values of velocity and momentum after a specific collision. The targeted group of users would be students or engineers who need to find these values outputted (final velocities and final momentums) after the collision... Examples of this would be a teacher asking you to find the velocity and momentum of a ball after it hits a wall, and an engineer trying to find out how much speed a baseball travels after being hit by a bat.”
• Not clearly identifying a user or user group. For example, “The group of users that are inteded with my program is extremely broad as it is open and avalible to anyone.”	• High scoring responses clearly identify the user or user group. For example, “The expected group of users of my program is teachers. My program addresses the interest of the teachers via making a platform for teachers to sort and add student grades in a list.”
• Identifying concerns of the program developer instead of a concern or interest of a user of the program. For example, “1 concern i do have is the user will get confused ...”	• High scoring responses clearly identify user concerns. For example, “My program is targeted at anyone who seeks to learn how to properly use roman numerals, as well as those who want to test their knowledge. For those looking to learn, there is a mode designed to teach the user with the assistance of a chart that details each number and it’s corresponding numeral, as well as a mode that allows the user to practice converting number to roman numerals, and vice versa. For users who want to test their knowledge, there is a mode where the user has a set number of lives and must solve as many problems as they can correctly before running out of lives.”

Question 2

Task: Written Response 2
Topic: Algorithm development, errors and testing, and data and procedural abstraction

	Max Points:	**Mean Score:**
Written Response 2(a):	1	0.56
Written Response 2(b):	1	0.20
Written Response 2(c):	1	0.25

What were the responses to this question expected to demonstrate?

Responses to this question were expected to demonstrate that the student could:

- explain how a code segment functions, including evaluating expressions that use logic and relational operators and determining the result of a conditional statement. (Written response 2(a): Algorithm Development),
- identify inputs and corresponding expected outputs or behaviors that can be used to check the correctness of an algorithm or program (Written response 2(b): Errors and Testing), and
- explain how the use of procedural abstraction manages complexity in a program (Written response 2(c): Data and Procedural Abstraction).

Written response 2(a) asked students to describe their conditional statement including its Boolean expression and what the procedure does in general when the Boolean expression is false. Students were required to identify a conditional statement (i.e. a selection statement), evaluate its Boolean expression, and explain how it functions in one of two cases.

Written response 2(b) asked students to describe the outcome of the procedure call they identified in the Personalized Project Reference, demonstrating that they could identify the inputs and corresponding outputs of a procedure call. It then asked students to write a new procedure call with at least one different argument that produces the same outcome (or to state why it was not possible to do so). This ability is critical to identifying and correcting errors in code in two ways. First, for students to determine if their procedure is functioning correctly, they must first understand the procedure's expected behavior for a given input so that they can match the expected output to the observed output when they run the procedure. Second, it is important to recognize when two different tests produce the same behavior, or to recognize that no two calls can produce the same behavior, to ensure that all test cases do not test the same functionality of the procedure.

Part (c) asked students to identify the parameters of their procedure and explain how these parameters use abstraction to manage complexity in their program. The use of parameters to generalize the functionality of a procedure is a key aspect of procedural abstraction, allowing a single procedure to be called using a variety of data values to solve multiple instances of the same problem. In this part, students were expected to show that they understood this specific aspect of procedural abstraction, beyond procedural abstraction as a general concept to simply reduce the amount of code written.

How well did the responses address the course content related to this question? How well did the responses integrate the skill(s) required on this question?

Written response 2(a): Algorithm Development - Explain how a code segment functions, including evaluating expressions that use logic and relational operators and determining the result of a conditional statement.

- Most students were able to identify a conditional statement and describe the statement and its Boolean expression.
- Most students were able to determine the result of the conditional statement in the case where the Boolean expression was false. Some responses explicitly identified the path the code would take if the Boolean expression evaluated to false, and then described what that path would do in general. Others described the behavior of the code in the else block, or following the conditional statement in the cases that there was no else block.

Written response 2(b): Errors and Testing - Identify inputs and corresponding expected outputs or behaviors that can be used to check the correctness of an algorithm or program.

- When students provided a procedure call that used specific arguments, they generally were able to identify the outcome their procedure call was intended to produce and provide a separate procedure call that produced the same outcome. In other cases, students provided a call to their procedure that used variables as arguments. In these cases, students generally were able to address the first part of the prompt by describing the output of their procedure in terms of generalized values of the variables (e.g., "shows the desired information corresponding to the user's choice"). However, this often created a challenge in answering the second part of the prompt as there was no specific outcome against which to compare the outcome of their second procedure call. Some students handled this challenge by specifying hypothetical values for their parameters for the call given in the Personalized Project Reference, and then providing a second call with different specific arguments.
- Many students were able to identify and explain situations where it was not possible to write a different call to the procedure that produced the same outcome. Most of the time, this was when the procedure was written to lead to a specific outcome for each individual possible argument (e.g. for a procedure with a parameter named choice, the code `if choice == 1: print "You win"` can only print `"You win"` when `choice` is `1`.)
- This prompt revealed a misunderstanding about arguments and procedure calls for some students. This misunderstanding was demonstrated by two prominent incorrect response patterns. First, many students misinterpreted the prompt to "write a new procedure call with at least one different argument" as having to rewrite their code in one of two ways: (1) to change the procedure call in their code or (2) to rewrite the procedure body to do something different. Second, some responses changed the name of the variable in their given procedure call rather than providing a different argument.

Written response 2(c): Data and Procedural Abstraction - Explain how the use of procedural abstraction manages complexity in a program.

- Most students correctly identified the parameters to their procedure, though some confused arguments with parameters. Many students correctly described how the parameters used abstraction to managed complexity in their program by explaining how the parameters allowed the procedure to be called with different values depending on user input or in various locations in their program. Some responses that did not earn the point included explanations of how procedures in general helped manage complexity in the program by allowing the code to be run multiple times without being copied but failed to link this explanation to the parameters. Other responses confused procedural abstraction with data abstraction. This confusion was particularly likely to happen when the parameter was a list, with responses indicating that the list reduced the number of parameters, rather than focusing on why the parameters themselves allowed for the procedure to be used to solve a broad set of instances of the same problem.

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

Common Misconceptions/Knowledge Gaps	*Responses that Demonstrate Understanding*
Written Response 2(a): Algorithm Development	
• Not describing the conditional statement adequately. For example, "The first part of my code conditional statements would be the movement. In order for my movement to function i had to code in some kind of gravity so that way the character in the game doesn't always float and so with this new part of the code I had to now make my character move. I did this using {If} commands, so if the player had used a specific arrow key on the keyboard that would make my character move a certain distance." This response describes aspects of the procedure's behavior but does not specifically describe the conditional statement.	• High scoring responses describe the conditional statement and Boolean expression clearly. For example, "My conditional statement checks if the user's lunch order is greater than or equal to their cash."

• Misidentifying other control structures such as procedures or loops as conditional statements. For example:
 	
`function findServiceRating(filler)`
`{`
`  while(true){`
`    let serviceRating =`
`    readLine("rate your service");`
`    if (serviceRating == "excellent"`
`        || serviceRating == "good"`
`        ...)`
`    {`
`       return serviceRating;`
`    }`
`    else`
`    {`
`       console.log("Invalid input")`
`    }`
`  }`
`}`
 	
The response states, “My first conditional statement starts my whole process of finding the tip percent they want. When true it prompts the user to enter what you wold rate their service. ... If these conditional statement was to be false the program would not execute and nothing would happen.” This statement describes the `while(true)` structure as the conditional statement.
	• High scoring responses correctly describe if-statements or try/catch blocks as conditional statements. For example, “This conditional statement is a statement within a loop that keeps a timer running for the length of the game and stops the timer when the player’s "health" runs out and the game ends. The Boolean expression is "true" when the health runs out and "false" if the health has not ran out. If the Boolean expression is false, then the timer will keep running and the conditional statement will be executed again since it is looped.”

- Not describing the first conditional statement when there were multiple conditional statements in the provided procedure. For example:

```
def hangman(chosen_word):
  global current_streak
  split_word = list(chosen_word)
  player_progress = []
  strikes = 6
  for i in range(len(split_word)):
    player_progress.append(" _ ")
  for i in range(len(split_word)):
    if str(" ")==str(split_word[i]):
      player_progress[i] = str(" ")

  print((" ").join(player_progress))
  strike_verification_num = 0
  print(" ")
  while True:
    if player_progress == split_word:
      print("Congatulations!")
      print("You guessed the word 
with " + str(strikes) + " strikes 
remaining")
      current_streak=current_streak+1
      break
...
```

The response describes the second if statement stating, “My first conditional statement tests if the player has gotten the word correct. It uses the if statement "if player_progress == split_word:" in order to determine a true or false value.”

- High scoring responses describe the first conditional statement when there are multiple conditional statements included in the procedure. For example:

```
function filterMountains(range) {
  minRange = [];
  if (range == "23,000 - 24,000") {
    for (var i = 0;
      i < mNames.length; i++) {
      if (mHeights[i] >= 23000 &&
          mHeights[i] <= 24000 ) {
        appendItem(minRange,
                   mNames[i]);
      }
    }
  }
...
```

The response describes the first if-statement as, “The first conditional statement in the procedure asks if the parameter, "range," is equal to the string "23,000 - 24,000."”

Written Response 2(b): Errors and Testing	
• Describing how to rewrite the procedure or change the number of parameters to the procedure instead of providing a procedure call with different arguments. For example, “A new procedure which could be called upon to have the same outcome would be to collect the same data from before rather this time inside of the procedure when giving corresponding values to letters, we could add the word "else:" which would tell the computer anything else that cant be assigned a value will be given the value of zero so final_gpa will be set equal to itself plus 0 so we can still know any undefined variables which cant be assigned values will correspong to nothing.”	• High scoring responses describe a second call to the same procedure with different arguments that produces the same outcome. For example, consider a response where the identified procedure takes two parameters, `functionJob` and `mode`, and the body contains an outer if-statement that depends on the value of `functionJob` only. The first procedure call given uses the arguments `"display"` and `"normal"`. The response states, “One procedure call to check/displayPattern that would produce the same outcome would be if I passed in "display," and the string "reverse" instead of "normal." The reason why this procedure call produces the same outcome is because the mode parameter (normal/reverse as arguments) is only utilized when check/displayPattern is passed check, not display, as an argument for the functionJob. When the call including ... "display," and "reverse" arguments is called, the procedure runs the second section of itself which does not use the mode parameter. By passing in "reverse" instead of "normal" as the mode, this call would produce the same outcome as long as the functionJob parameter is still "display."”
• Changing the names of the parameters or the variables that are used as arguments instead of passing different arguments. For example, “I am able to write a new procedure with the same outcome by changing the argument names. Dusk and Dawn are arguments defind by me, I could simply change their name to night and morning which would produce the same output.”	• High scoring responses describe the argument instead of the parameters. For example, consider a response that provides a `checkAnswer(guess)` where `guess` is a character. The procedure `checkAnswer` returns `true` if the guess is in the secret word and `false` otherwise. The response states, “Two procedure calls with different arguments can produce the same outcome as long as the argument is an incorrect letters. For example, if the word was "ORANGE", the procedure will produce the same output if the user guess the letter "S" (checkAnswer("S");) or the letter "C" (checkAnswer("C");) which both result in the user losing a life.”

• Describing why it is not possible to write another procedure call with the same outcome when it is. For example:
 	
`function nbaDivisionInfo(teamName) {`
`  var allTeams = getColumn("NBA Teams", "Team");`
`  var allDivisions = getColumn("NBA Teams", "Division");`
`  for (var i = 0; i < allTeams.length; i++) {`
`    if (allTeams[i] == teamName) {`
`      return (allDivisions[i]);`
`    }`
`  }`
`}`
 	
The response explains, "A new procedure call, in my case, may not be produced, as without the dropdown variable grabbing the strings and the team names inside of the dropdown, the function will not work as intended, leading to errors in the code that will not allow for the program to be run." However, it is possible to write more than one procedure call to produce the same outcome because different NBA Teams are in the same division.
	• High scoring responses correctly recognize when it is not possible for two procedure calls to have the same outcome and describe the reason correctly. For example:
	
	`def checkPriority(priority):`
	`  print("\n \n \n")`
	`  count=0`
	`  for i in range(len(Taskboard)):`
	`    if(priority == userPriorities[Taskboard[i]]):`
	`      count+=1`
	`  return("You have "+str(count)+" task(s) with "+str(priority)+" priority")`
	
	The response states, "If we include a different value for the priority parameter each time, then the output will be different as the return value includes the priority in it." This explanation describes why the outcome will always change depending on the value of the argument.

Written Response 2(c): Data and Procedural Abstraction	
• Confusing data abstraction with procedural abstraction for managing complexity. Many responses describe the use of lists to manage complexity within their procedure. For example, "This parameter uses abstraction to manage complexity in my program as the parameter is a list. So, the parameter stores all of the user's responses in one organized format. If one did not have the "user_responses" list, one would use variables to store each of the user's responses."	• High scoring responses describe procedural abstraction. For example, "These identified parameters use abstraction to manage complexity in my program because they allow me to insert any two lists and a string into the procedure, meaning that instead of writing a new algorithm each time I want to find the value of a list at the same index at which the value of a different list is equal to the value of a string, I can just insert the appropriate arguments into my procedure. I don't need to fully understand how these parameters are used in my function when I call it, because the parameters enable me to enter any two lists and a string I would like to."

• Describing procedural abstraction without relating it to the parameters specifically. For example, "The parameter used in this procedure is "num" which would be the radical of a quadratic equation. This parameter uses abstraction to manage the complexity in my program because instead of doing all this calculations inside of a single statement, I can take it out as a procedure that takes in a parameter to make my code more simple and easier to read. The procedure also helped me break down what I needed to do into steps to sucessfully simplify a radical." This explanation does not relate the benefits of using a procedure (e.g., "make my code simpler and easier to read" or "helped me break down with I needed to do into steps") to the parameter.	• High scoring responses describe procedural abstraction based directly on the parameters. For example, "The parameters used in this procedure are x and y. These parameters, because they are not defined, allow for many different arguments to be plugged when the procedure is called. This helps manage the complexity of my program because it allows me to reuse the same code and plug in different values into the same code. Without these parameters, I would need to create a code segment for each possible set of arguments in order to produce the same outcome."
• Identifying arguments instead of parameters to the procedure. For example: `function dogMatching(temperment, weight, lifespan)` and the procedure call: `dogMatching(selectedT, selectedW, selectedLS)` The response states, "The parameters used in this procedure are (selectedT, selectedW, selectedLS)."	• High scoring responses name the parameter and not the argument that is passed to the procedure call. For example: `printColleges(colleges)` and the procedure call: `printColleges(gpa4)`. The response states, "The parameter used in the procedure for this program was identified as the word, colleges, which was found in parentheses in my procedure call."

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

In order for students to respond to the prompts given on the exam, it is critically important that their program segments include the required elements as described in the Course Exam Description. Use examples from AP Central to highlight well-constructed answers and show students how the program segments include all of the various procedural, algorithmic and list elements so that answering the prompts becomes easier to do.

When creating the videos to demonstrate input, program functionality, and output, the video should be as clear as possible as to where the input is coming from. If the input is coming from a file, from the keyboard, or from a device like a mouse, and there is no text input box or mouse pointer visible, it is recommended that the video include a caption that explains where the input is coming from to make it easier for the Reader to evaluate the video correctly.

When preparing the Personalized Program Reference, remind students that the provided code should be clear to read for the Reader, and the code segments should not include any documentation of any kind.

To help students practice writing for the written response questions (Questions 1, 2(a), 2(b), and 2(c)), provide students with good examples of Personalized Program Reference sheets from examples on AP Central and have them practice writing their own answers to the provided prompts. Also, have students review each other's answers to help them understand why some explanations are better than others. Finally, provide some examples from AP Central where the responses scored low, and ask students to write why the responses are incorrect.

Here are some specific issues to address that tie to the expected learning objectives of this performance task:

- Review data abstraction with students carefully since this topic is difficult to understand for most students when they begin to write programs. Give examples of programs where the amount of data to process can vary (e.g. the number of survey responses from a community of users running a popular phone app) so students see the reason why a list can help simplify the computation and allow an arbitrary amount of data to be processed. Also give examples of programs where storing data in a list makes it easier to use (e.g. the names of songs in a playlist that can be stored and sorted as one collection).
- Review procedural abstraction and the role parameters play in this abstraction. Provide examples to discuss where a program has code repeated several times and show how much easier it is to read and maintain if these copies of code are replaced with a single procedure. Provide examples that show how a procedure can use a parameter to compute the same computation for many different values. For example, discuss a procedure that computes the distance of a projectile above the ground every second as a function of the initial velocity and the angle the projectile is shot into the air. Additionally, an example like this uses iteration and can also use selection if the parameters are out of range (e.g. if the initial velocity is negative). For this example, show how a program can include this procedure and call it with various arguments to solve a set of these problems.
- Review the difference between parameters and arguments with respect to procedures and have students trace the results of a procedure for specific arguments, predicting what the output (or return value) should be. If an argument is a variable, have students consider possible values this variable can have when the procedure is called. Provide exercises where students must determine values for the arguments that will, as a group, cause all procedure instructions to be tested for correctness.

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

- The AP CSP Course and Exam Description and AP Classroom have several written response prompts for students to practice writing throughout the school year. Teachers are encouraged to use these prompts to have students write about their in-class programs and about their Create Performance Task. Once the Create Task is submitted as final to the AP Digital Portfolio, teachers can give feedback to their students on the practice responses before exam day.
- AP Classroom also offers a variety of Daily Videos by high school faculty and University Faculty Lectures. These can be used to supplement existing curriculum as student needs arise. Particularly useful topics for preparing for the Create Performance are the following:
    - Design Process: CRD-2: University Faculty Lecture
    - Input and Output: 1.2: Daily Video 2
    - Selection: 3.6: Daily Video 3
    - Iteration: 3.8: Daily Video 2
    - Procedures, Parameters, Arguments, and Returns: 3.12: Daily Video 1
- AP Classroom also offers Daily Videos specific to preparing for the Create Task. To access these videos, students can navigate to the Course Guide section in the left navigation pane of the AP Classroom homepage, select the Overview page, and then click on the Student Resources header to expand the list of available resources.
    - Create Performance Task: Overview
    - Create Performance Task: Categories and Questions
    - Create Performance Task: Guidelines
    - Create Performance Task: Pacing Your Project
    - Create Performance Task: Scoring Guidelines and Sample Written Response Answers