Chief Reader Report on Student Responses:

2024 AP® Computer Science Principles Performance Task
Set 1

- 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	0.92
Program Requirements:	1	0.75
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 explain at least one valid input into their program and how the input was used in the program's functionality. The video and program code could be considered, in addition to the student's written response. The response was expected to describe program functionality when discussing the effect of the input and not merely the resulting output.

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 explain their program input and how their program used that input to accomplish its functionality. Most students were able to adequately explain their program function in this way. Some students provided a thorough and detailed explanation of the code's processes upon receiving input, describing specific code segments that were activated. Most students provided concise responses limited to the immediate, direct result of the input or a high-level description of the program functionality based on that input.

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 (list1[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

- Incorrectly identifying the input to the program. For example, “The input of my program is my user defined function that calls for the conditional statement of the buttons in my program.”
- Not explaining what the program does with the input. For example, “One input is that the user types a number 1-5 and once the user hits submit the program will then run.”

- High scoring responses describe an input and what the program does with the input. For example, “The input is in the form of a dropdown box where users are able to select any of the 50 states. Once the user selects a state the program then takes that input and filters through the list of target addresses to find the one located in said state. Then the program displays all the options in a text box for a user to scroll through.”

• Confusing the command or click that starts the program with input to the program once it is running. For example, “One valid input in my procedure is when the green flag is clicked ... It runs an action called "players" which clears all of my Lists of any items and adds new ones to each list.”	• High scoring responses describe the impact of user input after the program starts running. For example, “When a player inputs a difficulty level, the variable is set to the number of seconds related to the difficulty.” This response describes the external user input in a block program.

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	0.54
Written Response 2(b):	1	0.32
Written Response 2(c):	1	0.22

What were the responses to this question expected to demonstrate?

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

- explain how program code functions, including identifying and determining the result of an iteration 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
- develop an algorithm that uses iteration statements to traverse a list (Written response 2(c): Data and Procedural Abstraction).

Written response 2(a) asked students to describe what was being accomplished in the body of the iteration statement they had identified from their program code. Responses needed to demonstrate the ability to accurately explain the behavior of this subset of the code the student had written either at a high level, or with a more detailed line-by-line description.

Written response 2(b) asked students to identify two different calls to the procedure that caused a different segment in the procedure to execute, along with the associated output of each call. This ability is critical to identifying and correcting errors in code in two ways. First, for a student to determine if their procedure is functioning correctly, the student 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 for tests to cover many different cases, including those that cause different segments of the code to execute. Because the students had flexibility in their procedures, it could have not been possible to have two different calls to their procedure that caused different segments of the code to execute. Responses to 2(b) required students to recognize this situation if it applied to their code.

Written response 2(c) asked students to write an algorithm that iterated over the list from their program and apply a pre-existing procedure to each element in the list without knowing how the procedure works. The presence of the list in their program code also demonstrated their ability to develop data abstraction by using a list to store multiple elements that can be processed.

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 program code functions, including identifying and determining the result of an iteration statement.

- Most students were able to explain what the body of their iteration statement accomplished and provided their descriptions at a high level rather than explaining each of the specific lines of the code.

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.

- Many responses provided two calls to the procedure that executed different lines of code and produced different outcomes. This ability demonstrates students' ability to identify appropriate test cases for finding and correcting errors in their programs.
- Many responses correctly explained the expected behavior of their identified procedure calls. Observable behaviors described in the responses commonly included print statements, return values, changes to the user interface, and whether the program continued or exited. Any of these observable behaviors can be used for ensuring the correctness of the procedure's functionality.
- Some responses included a procedure where it was impossible for the procedure to run different sections of code by passing different arguments. This occurred most often when the selection statement was contained inside a loop and the Boolean expression would always be true for at least one iteration. In some cases, students were able to explain this correctly. However, in other cases responses stated only that it was not possible without explaining why in reference to how the code worked.
- Some responses to this part indicated a confusion between parameters and arguments. Sometimes these responses used a variable as an argument in a procedure call and did not consider a potential value for that variable in determining the expected output.

Written response 2(c): Data and Procedural Abstraction - Develop an algorithm that uses iteration statements to traverse a list.

- Many responses expressed an algorithm that correctly iterated through the list and used `checkValidity` to test the validity of each element in this list. These algorithms generally included a loop to iterate through the elements in the list and a call to `checkValidity` on each element. Some responses also reported the validity for each element in the list by printing a statement reporting the result of `checkValidity` for each element. The strongest responses went further to determine (not just check) if all elements were considered valid by exiting the loop when finding an invalid element or by creating a Boolean variable which was initially set to `true` and then changed to `false` when `checkValidity` returned `false`. A common algorithmic mistake was to try to return the result from `checkValidity` for every list element, but this prematurely exits the loop and the procedure after the first element is examined and the result is returned.
- Many responses provided the algorithm in detailed steps, but some also provided pseudocode.
- In some cases, students misinterpreted the prompt as meaning that they should write the code for the `checkValidity` function rather than use it, or that they needed to define what the notion of validity meant for their list.
- Finally, some students struggled to provide sufficiently precise language in describing their algorithm. Often these responses had the right ideas (iteration, selection, and applying the `checkValidity` function), but their algorithm did not clearly link the pieces together.

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 distinguishing between the procedure and the body of the iteration. Some responses explain what the whole procedure is accomplishing, not specifically the body of one iteration statement. For example, "In the first iteration statement segment the code is responsible for updating the health values and text boxes associated with said values." This statement refers to the two health values—`robotHealth` and `playerHealth`. But the first iteration statement pertains only to the `robotHealth`; there is a separate iteration statement to update the player's health:
 	
`robotHealth = 0;`
`for (i = 0; i < 5; i++)`
`{`
`  robotHealth += robotSkill[i];`
`}`
`playerHealth = 0;`
`for (j = 0; j < 5; j++)`
`{`
`  playerHealth += playerSkill[j];`
`}`
	• High scoring responses describe what is accomplished by the body of the iteration statement. For example:
	
	`for (i = 0; i < planet.length; i++)`
	`{`
	`  if (planet[i][0] ==`
	`  userChoice1.selected())`
	`  {`
	`  index1 = i;`
	`  }`
	`}`
	
	The response explains, "The first iteration statement finds the index of the correct planet that the user chooses ... the for loop iterates through the planet list to find the item that matches the user input. Then the index of the item is stored for future use in the program." This describes what the body of the iteration is accomplishing.

• Misidentifying other programming constructs (e.g., conditional statements or procedures) as iteration. For example, "In the iterative statement, the goal was to congratulate the player on reaching a specific amount of attempts through the utilization of an if else statement."	• High scoring responses correctly identify iteration as a looping structure. For example, "My iteration statement uses a for loop with the variable i."

Written Response 2(b): Errors and Testing

- Not recognizing that additional procedure calls are possible beyond what is written in the program. Some responses state that only one call to the procedure was possible because the program code only made one call to the procedure. For example, "it is not possible for two calls to be made because the information outputted by the code only required one call and no further information needed to be displayed in order to get the code to execute."

- High scoring responses recognize that a procedure call that takes a variable value in their program will result in multiple procedure calls with different arguments either in a single run or in different runs of the program. For example, when the call to the procedure is `dealCards(game)` where `game` is a variable, the response provides the calls `dealCards(blackjack)` and `dealCards(go_fish)` where `blackjack` and `go_fish` are (intended to be) specific strings.

- Providing two calls to the procedure that produce different outcomes, but that execute the same lines of code. For example:

```
def num_var( x, y, label):
   c.hideturtle()
   c.speed(0)
   c.penup()
   c.goto(x, y)
   c.pendown()
   for i in range(2):
      c.forward(62.5)
      c.right(90)
      c.forward(40)
      c.right(90)
   c.penup()
   c.goto(x + 20, y - 45)
   c.write(label, font=('Courier',
           30, 'bold'))
```

  The provided calls `num_var(-120, -80, "1")` and `num_var(-120, -130, "0")` will result in different behaviors but execute the same lines of code. In this example, the response should have explained why it is not possible to have the code run two different sets of statements for different arguments.

- High scoring responses provided two procedure calls to the same procedure that cause different segments of the code to execute. For example:

```
def dndroll(rolls, type_dice):
  while True:
    if type_dice.lower() == "d6":
      for i in range(rolls):
        print(random.choice(d6))
      break
    elif type_dice.lower() == "d8":
      for i in range(rolls):
        print(random.choice(d8))
      break
    elif type_dice.lower() == "d20":
      for i in range(rolls):
        print(random.choice(d20))
      break
    else:
      print('not a valid die')
      type_dice = input('Pick a
                  valid dice to roll
                  [D6, D8, D20]: ')
```

  The provided calls `dndroll(5,"d6")` and `dndroll(5,"d20")` will cause different branches of the if/else statement to execute.

- Providing calls to two different procedures instead of two different calls to the same procedure. For example, “The first call, `call dude`, allows the function dude to run... The second call, `call afterDeath2`, is the outcome that happens when the moveable sprite Ivan gets striked by any of the car sprites more than three times”.

- High scoring responses provided two different calls to the same procedure using specific arguments, at least one of which is different between calls. For example:

```
var moneyCountry = ["affordable",
                    "median"]
var countryCount = [0, 0]

money(moneyCountry,"A",countryCount)
money(moneyCountry,"B",countryCount)
```

- Confusing arguments with global variables or user input. Some responses provided calls that rely on the value of a global variable or user input to execute different code segments. In these cases, it was not the calls themselves that caused the different code segments to execute. For example:

```
def add_ons(num_toppings):
  print("For the toppings you would
        like to add")
  for a in range(num_toppings):
    selection = input("Type here: ")
    if selection == "cookie dough":
      ic.append(selection)
    elif selection == "crushed
                      nuts":
      ic.append(selection)
   ...
```

The if/else branch taken depends on the user’s input and not on the value of the `num_toppings` parameter. Additionally, the selection used in this procedure is trivial because all branches execute the same line of code.

- In high scoring responses, the provided arguments were what cause the different code segment to execute, even when global variables or user input were used in the procedure. For example:

```
function ExpenseTracker(sign,
                        expense)
{
  if (sign == "-") {
    if ((expenses - expense) < 0) {
      setText(...)
    } else {
      expenses -= expense
    }
  }
  if (sign == "+") {
    if ((expenses + expense) >
         income)
    {
      setText(...)
    }
    else
    {
      expenses += expense;
    }
  }
...
```

Although this procedure uses a global variable (`expenses`), which if branch of the conditional statement executes depends on the values of the parameters `sign` and `expense`, not the value of `expenses`.

- In cases where it was not possible to provide two procedure calls that cause different code segments to execute, incorrectly or insufficiently explaining why it was not possible. For example:

```
function filterDog(breed)
{
  for (var i = 0;
      i < breedList.length - 1; i++)
  {
    if (breedList[i] == breed)
    {
       appendItem(filteredName,
                  nameList[i]);
       appendItem(filteredBreed,
                  breedList[i]);
       appendItem(filteredMinHeight,
                  minHeightList[i]);
       appendItem(filteredMaxHeight,
                  maxHeightList[i]);
       appendItem(filteredMinWeight,
                  minWeightList[i]);
       appendItem(filteredMaxWeight,
                  maxWeightList[i]);
       appendItem(filteredImage,
                  imageList[i]);
       findDog(breed);
    }
  }
}
```

It is not possible for two procedure calls to `filterDog` to cause different code segments to execute because `breedList` is always non-empty and breed always matches at least one element in `breedList`. However, the response explains, "It is not possible for two calls to my procedure to cause different code segments to execute because the code only runs dependent on only one parameter. The code only functions one way because of only one parameter is being inputted." This explanation is not correct.

- High scoring responses correctly recognized that it was not possible to provide two procedure calls that cause different code segments to execute and explained why. For example:

```
function releaseYear(date) {
  var dateList = getColumn("Netflix
      Content", "Release Year");
  var titleList = getColumn("Netflix
      Content", "Title");
  var filterTitle = [];
  for (var i = 0;
       i < dateList.length; i++) {
    if (date == dateList[i]) {
      appendItem(filterTitle,
      titleList[i]);
    }
  }
  return filterTitle;
}
```

The response explains, "It is not possible for 2 calls for my procedure to cause different code segments to run. Because my procedure takes in a number parameter and this number exists in my lists, every part of my procedure will run at least once. There is at least one value that is the same as the date inputted, triggering the if statement, because that is how I chose which values to put in my dropdown."

Written Response 2(c): Data and Procedural Abstraction	
• Defining the procedure `checkValidity` instead of using that procedure to check the elements of the list. For example, "the procedure checkValidity(value) have an "if" statement starting "if value < 0 false ; else true". "Value" being the value obtained through "text list"."	• High scoring responses give a detailed algorithm that uses `checkValidity` to check the validity of list elements. For example, "I could initialize a count variable as a tracker for this. I would use iteration through the list ... in this format "var i = 0; i < expenseslist.length; i++". Inside this iteration... I would also use selection and if checkValidity(expenseslist[i])=true, I would increase count by 1. This would be repeated for the length of the expsenseslist. After the iteration, there will be selection. It will ask if count = length of expenseslist, and if true, it will return True... and if false it will return False."

• Not recognizing that `checkValidity` can and should be called to check the validity of the list items. Some responses implement an algorithm that used their own definition of what is considered valid rather than using `checkValidity` to determine whether the elements in their list were valid. For example, "If a movie from my list appeared in the dictionary for the same genre, it would be considered valid and true. If the movie did not appear in the dictionary then it would be considered invalid and false."	• High scoring responses provide an algorithm that calls `checkValidity` with each item in the list. For example:
	
	`function checkListValidity(list)`
	`{`
	`  var validity = true;`
	`  for each element in list`
	`  {`
	`  if ((checkValidity(current element)) == false)`
	`  {`
	`  validity = false;`
	`  }`
	`  }`
	`  DISPLAY[validity]`
	`}`
	This response also explains that this algorithm would be run on the list `cardDeck`, which is the list identified in the List section of the Personalized Project Reference.

• Not providing a correct or detailed enough algorithm. For example, “First I would use a for-loop to make sure that every element in my list is compared and I would use the variable "value" to represent the element currently being compared... In this for-loop I would have the procedure "checkValidity". If the other programmer finds an item valid, then the procedure will return true, and if the programmer finds it to be non-valid, then the procedure will return false.” This algorithm does not include how checkValidity is called., It is also inaccurate because it will exit the loop after the first element is checked due to the return.	• High scoring responses give a written algorithm or pseudocode that is sufficiently detailed that another programmer could implement the algorithm. For example, “use a while loop to filter through each element of my list (centerList) through checkValidity, until checkValidity returns false, or until the index being checked is greater than the last index of centerList. If checkValidity returns false, display a message saying that not all values are valid. If checkValidity never returns false, display a message saying that all values in the list are valid.”

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 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