Chief Reader Report on Student Responses:

2025 AP® Computer Science Principles Performance Task

Set 1

- Number of Students Scored 175,935
- Number of Readers 679
- Score Distribution

Exam Score	N	%At
5	18,847	10.7
4	35,015	19.9
3	54,777	31.1
2	37,604	21.4
1	29,692	16.9

- Global Mean 2.86

The following comments on the 2025 free-response questions for AP® Computer Science Principles were written by the Chief Reader, Thomas Cortina (Carnegie Mellon University). They give an overview of each free-response question and of how students performed on the question, 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, all 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. Their Personalized Project Reference includes a student-developed procedure definition with at least one explicit parameter, demonstrating use of sequencing, selection and iteration, and a call to the included procedure, as well as program code for a list or collection that shows how data is stored in the list and how the data in the list or collection are used in the program. Their Personalized Project Reference must not include any comments in the program code or other course content.

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:

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. 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	0.84

Overall Mean Score: 2.57

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 that includes sequencing, selection, and iteration, calls the student-developed procedure, and includes the creation and use of at least one list or collection (Course Project: Program Requirements), and
- identify at least one valid output produced by the program (CRD-2.D) and how the output shows an aspect of the program's functionality (CRD-2.B) (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 it is not clear from the video.

Additionally, the students were asked to provide code from their program that demonstrated a student-developed procedure, demonstrating sequencing, selection and iteration, a call to the student-developed procedure, a list (or other collection type), and a use of the list. All of these features must contribute to the program's functionality. Designing a program that includes these core features is critical to understanding basic programming in any language.

In Written Response 1 of this form, students were asked to identify at least one example output produced by the program and explain how the output shows an aspect of the program's functionality. The video and written response were considered, using the program code if a video is unavailable.

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?

- Almost all responses demonstrated a working program that demonstrated input, output, and functionality in the video response. Most responses demonstrated user-generated input from a mouse click or from text entered into a prompt box, which could be seen in the video. In some cases, the full program code had to be referred to when the input was a keystroke, data source, or other non-traditional input.
- Most responses included a student-developed procedure that included iteration and selection, and the creation and use of a list or another collection type. In most of these responses, these elements were essential to the program's functionality and not trivial.

- Most responses included a student-developed procedure and a call to that procedure. Some students used event handlers in lieu of student-developed procedures, though this was mostly limited to block languages.
- Responses demonstrated a variety of collection types, with most being list-style collections and dictionaries. Some responses used a list to store information retrieved from a user, and some used lists to store information to filter through large data sets. Some responses accessed specific elements in the list with hard-coded statements where iteration would have been more appropriate. This suggests that these students' understanding of when and how to use lists and collections is still developing.
- Students were asked to identify at least one example output of the program and explain how the identified output shows an aspect of the program's functionality. Most responses adequately explained their identified output and how it relates to the functionality. On the other hand, some responses explained how the entire program functioned rather than focusing on output.

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 showing the program running. For example, a video shows the code being displayed, but the program is never shown being run.	• High scoring responses show the program being run. For example, a video shows a user entering their guess for a word guessing game and displays whether any of the letters guessed were correct in the right position, correct in the wrong position, or incorrect.
• Not showing input to the program while the video is running. For example, response shows rabbits moving across the screen for 17 seconds, but no input is observable during the run or in the program code.	• High scoring responses show the program receiving and responding to graphical input such as mouse clicks or text input fields. For example, the video shows the user typing in a stock symbol and price and clicking the add stock button, after which the program adds the stock to a list that displays the current stocks.

Program Requirements	

• Not including a student-developed procedure. For example, the response includes event handlers containing code but not named procedures, preventing the reuse of this code, as shown here:
 	
`onEvent("Roll", "click",`
`  function() {`
`  for var i=0; i<10; i++) {`
`  roll = randomNumber(1,6);`
`  if (roll % 2) {`
`  appendItem(odd, roll);`
`  } else {`
`  appendItem(even, roll);`
`  }`
`  });`
	• High scoring responses include a student-developed procedure that contains selection and iteration, which is called from within the program. For example:
	
	`function alreadyinlist(food)`
	`{`
	`  for (var i=0; i<foods.length; i++)`
	`  {`
	`  if (foods[i] == food)`
	`  {`
	`  return true;`
	`  }`
	`  }`
	`  return false;`
	`}`
	
	`alreadyinList(food);`

• Not including iteration that runs until a given condition is met or for a specified number of times. For example:
 	
`forever`
`  turn left <rotation degree> deg`
	• High scoring responses include iteration that runs until a condition is met. For example:
	
	`int mode = scan.nextInt();`
	`while(mode < 1 || mode > 4) {`
	`  println("please Enter an integer that is between 1 and 4");`
	`  mode scan.nextInt();`
	`}`

• Including iteration that is trivial. In the following example, the use of iteration is trivial because the loop always runs once:
 	
`output = output + "I recommend you watch a ";`
`for (var i = 0; i < 1; i++) {`
`  if (mood == "Happy") {`
`  output = output + list[0];`
`  } else if (mood == "Sad") {`
`  output = output + list[1];`
`  }`
`}`
	• High scoring responses use selection and iteration to accomplish something meaningful in the program. For example:
	
	`for q in quiz_questions:`
	`  if ask_question(q):`
	`  print("Correct!\n")`
	`  score += 1`
	`  else:`
	`  print("Wrong!\n")`
	
	This iteration statement is a loop that runs once for each question in a list of quiz questions.

• Including a list that was not used in the program to create new data from existing data or to access multiple elements in the list. This example defines the list `number`:
 	
`number = [1,2,3,4,5,6,7,8,9,10, 11,12,13,14,15,16,17,18,19,20]`
 	
But the response then uses a random number generator to select a number, rather than using the list.
 	
`secret_num = random.randint(1,20)`
`guess = int(input("Enter your guess:"))`
`if guess < secret_num:`
`print("too low")`
`if guess > secret_num:`
`print("too high")`
	• High scoring responses define and use a collection in a meaningful way within the program code. This example defines the list `special_chars`:
	
	`special_chars = ["!", "@", "#", "$", "%", "^", "&", "*", "_"]`
	
	and then uses `special_chars` as follows:
	
	`for i in range(len(list_pwd)):`
	`if list_pwd[i] in special_chars:`
	`strength += 1`
	`break`

Written Response 1: Program Design, Function, and Purpose	
• Identifying example output that is inconsistent with the program. For example, “The output of my program gives recommendations of different workouts the user would like to do,” but the program video does not show workout recommendations, and the program code does not include code that would display recommendations.	• High scoring responses identify one example output of the program that is shown in the video. For example, “an output of either “Correct!” or “Wrong” is displayed.”
• Not explaining how the identified output shows an aspect of the program’s functionality, but instead just stating that the program runs. For example, “This shows the aspect of program’s functionality by showing the product of all the rest of the code on the screen.”	• High scoring responses explain how the identified output(s) show an aspect of the program’s functionality. For example, "‘Correct, The answer was milkshake’" This output is an aspect of my program's functionality because this output can only be acheived through guessing the right answer in 20 questions. The program is supposed to be similar to the game 20 questions where you have 20 yes or no questions to guess an item. Therefore, if an output of my program shows that the user guessed the item it is an aspect of my program's functionality.”

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	0.43
Written Response 2(b):	1	0.57
Written Response 2(c):	1	0.57

Overall Mean Score: 1.57

What were the responses to this question expected to demonstrate?

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

- evaluate Boolean expressions that use relational and/or logic operators (AAP-2.E.b, AAP-2.F.b), and explain how these expressions function (CRD-2.B) by explaining when and why the expression evaluates to true (Written response 2(a): Algorithm Development),
- identify errors in a program (CRD-2.I.a) by describing a modification another programmer could make to cause a logic error in their program and the program behavior resulting from this change (Written response 2(b): Errors and Testing), and
- develop a data abstraction using lists to store multiple elements (AAP-1.D.a) by asking the student to explain how the code that uses this data abstraction would need to change if another programmer added several new elements to the list or collection, or explain why no change is needed (Written response 2(c): Data and Procedural Abstraction).

Written Response 2(a) asked students to identify the Boolean expression in the first selection statement in the procedure section of their Personalized Project Reference, identify a specific value or set of values that will cause the Boolean expression of the selection statement to evaluate to `true`, and explain why the specified value or set of values will cause the expression to evaluate to `true`. Responses needed to demonstrate the ability to accurately explain the behavior of this subset of the code the student had written.

Written Response 2(b) asked students to describe a modification another programmer could make that would cause the procedure on their Personalized Project Reference to have a logic error and describe how the behavior of the procedure would change because of this introduced error. Responses to 2(b) demonstrated the student's ability to recognize logic errors and understand what code is doing without running the program. Responses to 2(b) required students to modify their code without running it and to understand its functionality.

Written Response 2(c) asked students to explain how the code segment that accesses or manipulates data stored in a collection on their Personalized Project Reference would need to change if another programmer added several new elements to the end of the list or explain why no changes to the code segment would be necessary. In this part, students were expected to refer to the data abstraction they had created and to show that they understood concepts of data abstraction by explaining how code may need to change if the data abstraction changes internally.

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 – Evaluate Boolean expressions that use relational and/or logic operators and explain how these expressions function by explaining when and why the expression evaluates to `true`.

- While some responses identified a Boolean expression in a selection statement, other responses incorrectly identified the entire selection statement as the Boolean expression.
- Most responses identified a specific value or set of values that will cause the Boolean expression of the selection statement to evaluate to `true`. However, some responses listed a generic set of values instead of a specific set of values, thus not demonstrating an ability to evaluate a specific Boolean expression.
- Most responses explained why the specified value or set of values will cause the expression to evaluate to `true`. Some responses only stated that the values would evaluate to `true` without explaining why.

Written response 2(b): Errors and Testing – Identify errors in a program by describing a modification another programmer could make to cause a logic error in their program and the program behavior resulting from this change.

- Many responses described a modification that would result in a logic error by describing a modification that would cause the program to run to completion with an incorrect outcome, thus correctly identifying a potential logic error in their program.
- Some responses described a modification to the procedure that would cause the program to terminate early, which demonstrated an ability to identify logic errors, which are errors that cause an incorrect output or unexpected behavior.
- Most responses described how the behavior of the procedure would change as a result of the introduced error.
- Some responses incorrectly identified syntax errors as logic errors. These responses introduced a syntax error and explained that the program would not run. A program's inability to run does not allow a programmer to observe or evaluate unexpected behavior, as the program produces no behavior at all.
- Some responses described a modification that would result in a logic error, but did not provide a description of how the behavior of the procedure would change as a result. Some of these descriptions included the behavior of the entire program but did not relate this to the behavior of the procedure.

Written response 2(c): Data and Procedural Abstraction – Develop a data abstraction using lists to store multiple elements by asking the student to explain how the code that uses this data abstraction would need to change if another programmer added several new elements to the list or collection, or explain why no change is needed.

- Most responses correctly identified whether code changes would or would not be necessary if another programmer added several new elements to the list, demonstrating understanding of the included data abstraction.
- Many responses identified the code that needed to be changed due to added items to the list and correctly explained the modification that would need to be changed. Often these responses detailed how an iteration through a list that was hard coded for the size of the list would need to be updated

for the new items in the list. On the other hand, some responses did not explain the changes that would need to be made to the list manipulation but explained how to add new items to the list.

- Most responses that identified that no changes would be necessary correctly explained that no changes would need to be made because their code could handle any number of items in a list, due to iterating through the list using the length property, demonstrating an understanding of the data abstraction. On the other hand, some responses stated that no changes would be necessary, but did not explain that the new items added to the list would not be processed or did not explain the reason why their code worked without changes.

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

<table>
<tr><th>Common Misconceptions/Knowledge Gaps</th><th>Responses that Demonstrate Understanding</th></tr>
<tr><td colspan="2">Written Response 2(a): Algorithm Development</td></tr>
<tr><td>

- Not identifying the Boolean expression, either explicitly or implicitly through description. For example:

```
forever
  set rotation style(left-right)
  if key.up arrow.pressed? then
    change y by 5
  if key.down arrow.presseed? then
    change y by -5
  if key.right arrow.pressed? then
    change x by 5
    point in direction 90
  if key.left arrow.pressed? then
    change x by -5
    point in direction -90
```

The response incorrectly states, “The Boolean expression in this selection is the entire procedure.” This response does not identify a Boolean expression in the first selection statement in the procedure section of the PPR.

</td><td>

- High scoring responses identify the Boolean expression explicitly. For example:

```
Add1(idex#)
for each item in yourlist
  if index of item in yourlist
  = idex#
    set count to item2 of item
    Replace item 2 of item with
    count+1
```

The response states, “The boolean expression in my Procedure is (INDEX OF item IN yourlist = idex#)”.

</td></tr>
</table>

- Not providing a specific value or set of values that would cause the Boolean expression to evaluate to `true`. For example:

```
OnEvent("getPet", "click",
    function() {
  geneder =
      getText("genderDropdown");
  name = getText("nameInput");
  age = getText("ageInput");
  if (age<=30 and gender=="Male"){
    pet = petList[0]
  UpdateScreen();
});
```

The response states, “To evaluate to being true you must be in between certain ages as well as being a specific gender.” These are not specific values or set of values that will cause the Boolean expression `age<=30` and `gender=="Male"` to evaluate to `true`.

- High scoring responses provide a specific value or set of values that would cause the identified Boolean expression to evaluate to `true`. For example:

```
function
setDisplay(tastePickedDisplay,
sizePickedDisplay,
texturePickedDisplay)
{
  for(var i=0; i < name.length; i++)
  {
    if(taste[i] ==
       tastePickedDisplay
       && size[i] ==
       sizePickedDisplay
       && texture[i] ==
       texturePickedDisplay)
      {
         appendItem(pickedFruits,
         " " + name[i]);
    }
  }
  if(pickedFruits.length == 0)
  {
    setText("fruitsPicked",
    "There are no fruits that meet
    your criteria, try again with
    new combinations!");
  }
  else
  {
    setText("fruitsPicked", "You can
    pick: " + pickedFruits);
  }
}
```

The response states that the Boolean expression in this selection statement is "taste[i] == tastePickedDisplay && size[i] == sizePickedDisplay && texture[i] == texturePickedDisplay". A way this Boolean expression will be true is when tastePickedDisplay(the taste the user picked) is "sweet" and taste[i](the taste of the fruit in index i) is also "sweet", when sizePickedDisplay(the size the user picked) is "medium" and size[i](the size of the same fruit in index i) is also "medium", and when texturePickedDisplay(the texture the user picked) is "crunchy" and texture[i](the texture of the same fruit in index i) is also "crunchy".

- Not explaining why the specified value or set of values will cause the expression to evaluate to `true`. For example:

```
define analyzePassword
set i to 1
repeat length of password
 set char to letter i of password
 if abcdefghijklmnopqrstuvwxyz
 contains char ? then
   set lower to 1
 if ABCDEFGHIJKLMNOPQRSTUVWXYZ
 contains char ? then
   set upper to 1
```

The response states, “The boolean expression would be the variable char that would look to see if the any condition meets the char. If it does, it would set char variable true and would increase the variable statement condition by 1. Set of values that will cause this expression to evaluate to true woud be the length of password given by the user. The input that the user has given. The reason it will set true is that, the user would add a specific amount of alphabets, numbers, and/or special characters that would continue the procedure on the condition being true and would not continue for that condition when it is false.” This explains that the “length of password given by user” would cause the expression to evaluate to `true` but does not explain why.

- High scoring responses explain why the specified value or set of values will cause the expression to evaluate to `true`. For example:

```
def want_continue(improper_format):
  improper_format = True
  while improper_format == True:
    answer = input("Do you want
     another prompt? (Yes or No): ")
    if answer == "Yes" or answer ==
         "yes":
      improper_format = False
      return True
    elif answer == "No" or answer ==
        "no":
      improper_format = False
      return False
    else:
       print("Sorry we didn't get
          that, please type either
          Yes or No.")
```

The response states, “The Boolean expression within the first selection statement is *answer == "Yes" or answer == "yes"*. The set of values that the user can input that will evaluate the expression to true is "Yes" or "yes". This is due to the fact that the condition is looking to see if the user is following the proper formatting for it to recognize what the user wants. It does this by seeing if "answer" is equal to "Yes" or "yes". The way it does this is case sensitive, so the user must be precise with what they input. The value of "answer" is determined by the user's input, and "answer" can only be equal to "Yes" or "yes" if the user inputs "Yes" or "yes". Thus, when the user inputs "Yes" or "yes" the Boolean expression evaluates to true.”

- Not identifying the Boolean expression in the first selection statement. For example:

```
def choose_stock():
  if risk=="Safe" or risk=="Moderate"
or risk=="Risky":
    show_stocks_by_risk(risk)
    symbol = input("Enter the symbol
        of the stock you want to
        invest in: ")
    for stock in stocks:
      if stock[0] == symbol and
          stock[2] == risk:
        stock_found = True
        name = stock[1]
        rate = stock[3]
        break;
     if stock_found:
       print("You selected " + name +
             " (" + symbol + ")")
```

The response states, “A value or set of values that will cause this part of mu expression to evaluate false would be stock_found.” While `stock_found` is a Boolean expression in this code segment, the Boolean expression in the first selection statement is `risk=="Safe" or risk=="Moderate" or risk=="Risky"`.

- High scoring responses identify the Boolean expression in the first selection statement, give a value or set of values that would cause the expression to evaluate to `true` and explain why the expression would evaluate to `true`. For example:

```
function calculate(amount, atom) {
  Newatomicw= [];
  for (var i = 0; i < Index.length;
       i++) {
    if (Index[i] == atom) {
      appendItem(Newatomicw,
                 atomicw[i]);
    }
  }
  var weight = Newatomicw[0];
  console.log(weight);
  console.log(amount);
  var mass = weight * amount;
  return mass;
}
```

The response states, “The boolean expression, Index [i] == atom, will evaluate to true if the text in Index [i] is the same as the text the user selects, which is stored in atom. This evaluates to true because the boolean expression, == , checks to see if the values of the variables on either side will match each other. For example, if Index[1] is hydrogen and the user also selcts hydrogen then the expression will evaluate to true.”

<table>
<tr><th colspan="2">Written Response 2(b): Errors and Testing</th></tr>
<tr>
<td>

- Describing a general modification to the program code rather than one specific to the procedure in part (i) of the Procedure section of the PPR. For example: “A modification the other programmar could make that can cause this procedure to have a logic error is the overcomplication and addition of new procedures to a code that already functions properly and is easy to understand. The behavior of more complicated procedures would change because of the error by not properly giving you the desired output for your code.”

</td>
<td>

- High scoring responses describe a modification specific to the procedure from part (i) of the Procedure section of the PPR that would result in a logic error. For example:

```
function clearFirstTile(oldX,oldY) {
  if(mineGrid.get(oldX,oldY)==1) {
    var newX = oldX;
    var newY = oldY;
    while(mineGrid.get(newX,newY)
          ==1) {
    newX = Randomizer.nextInt(0,
           numTiles-1);
    newY = Randomizer.nextInt(0,
           numTiles-1);
    }
    mineGrid.set(oldX,oldY,0);
    mineGrid.set(newX,newY,1);
  }
  isFirstTile = false;
}
```

The response states, “if the programmer replaced the 1 in the while loop condition with a 0, it would result in a logic error because the procedure would try to place a mine only where there already is one.”

</td>
</tr>
<tr>
<td>

- Describing a modification that would result in a syntax error instead of a logic error. For example, “For their to be a logic error their must be a problem with the language of the code or program. This could happen if someone where to modify the code by eliminating a bracket.”

</td>
<td>

- High scoring responses describe a modification that would result in the program crashing and describes the behavior of the program leading up to the crash. For example, “A modification another programmer could make that would cause a logic error in the procedure would be to change the range which the code uses to assign a random integer to the variable 'number'. This change would result in the possibility of trying to find a question that does not exist in the list. The program would run normally until or unless the variable 'number' is assigned the number which represents the whole length of the list. This would result in the program providing a blank question with no correct answer or a refusal to run.”

</td>
</tr>
</table>

• Not describing the behavior of the program as a result of the logic error. For example, “A modification the other programmer could make that would cause the procedure to have a logic error is to change the first line of this procedure:
for i in range(len(moods)):
if moods[i] == user_mood:
songs_for_mood = songs[i]
to:
for i in range(len(songs)):
because instead of checking the length of the moods list to find the matching songs in the songs list, it would evaluate the length of the songs list to check for corresponding songs. This would not make sense; hence, why it would be a logical error.” This response does not describe the change in behavior due to the introduced error. The program may not examine all moods or it may go off the end of the list and crash.
	• High scoring responses describe a modification that would result in a logic error and describe how the behavior of the procedure would change because of this error. For example, “A modification that another programmer could make that would cause this procedure to have a logic error is changing userInput == yearList[i] to userInput > yearList[i]. If this happens the behavior of the procedure will change. An example is if the user inputs 2001. If the user inputs this value then the code will first compare it to the first elements of yearList, which is also 2001. This will result the code into evaluatin it to false which then outputs that information for the selected year is not available, even though 2001 is a year in yearList with its respective winning team, runner up team and stadium.”

• Describing behavior that is implausible, inaccurate, or inconsistent with the program. For example, “They could possibly remove the else if statements which will cause it to output multiple return statements and it goes down the sequence. For example if the ouptut was 30 degrees the code would output all return statements from 90 to 20. This would not fill the requirement of the user, and will cause multiple errors within.” This description is inaccurate because in this program, the procedure can only return one value.	• High scoring responses describe a modification that would result in a logic error and describe how the behavior of the procedure would change because of this error. For example, “Changing the portion of code currentSignup.get(1).equals(day) to currentSignup.get(0).equals(time) would be a logic error because since the user's data is stored in an array of arrayList<String>, the user's data is stored in the following way List<String> user1 = ["Joe", "Monday", "Morning]. In the List user1, the index user1[0] equals "Joe" and the index user1[1] equals "Monday." The change above would be a logic error and cause the program to behave differently because we are trying to check if a slot is full and we need the day and time of the current user to compare with other users to find out if a slot is full. Changing the index that we are checking from 1 to 0 would cause the user's name to be checked with the occupied days instead of the user's day which would always result in a return value of false causing the day to be overbooked unless the user's name is a weekday like "monday."”

Written Response 2(c): Data and Procedural Abstraction	

• Providing an explanation of how the code would need to change if another programmer added several new elements to the end of the list that is inconsistent with the program. For example:
 	
`def print_even_numbers(numbers):`
`  for idx, num in enumerate(numbers):`
`  if num % 2 == 0:`
`  print(f"Even Number")`
`  else:`
`  print(f"Odd Number")`
 	
The response states, “Considering the list included in part one and if it were to be added onto by another programmer the code segment in part two would need to be modified. The code could be modifed by adding "else:" into there the way it originally was. Another way it could be modified would be by deleting the next line and making a new function.” This response incorrectly explains necessary changes when no changes would be necessary because the enumerate function will access all items in the list no matter how many items are in the list.
	• High scoring responses explain how the code segment that accesses or manipulates data stored in the list would need to change if several new elements were added to the list. For example:
	
	`def onMousePress(mouseX, mouseY):`
	`  app.index+=1`
	`  if (app.index == 6):`
	`  app.index = 0`
	
	The response states, “Depending on the amount of vaules added to my list, for an example 2 more colores were added, then the 6 that was orgnally there would change to 8, to acomadate the two new colors. The reason for these changes is my procedure only takes in the fact that there are 6 (since in phyon, list start at 0) vaules in the list its referning to”.

• Stating no changes are necessary without explaining why no changes to the code segment would be necessary. For example:
 	
`for student_name, grades in student_list.items():`
`  if len(grades) > 0:`
`  average = sum(grades) / len(grades)`
`  mastery = get_mastery_level(average)`
`  print(f"{student_name}'s average grade is {average:2f} - Mastery Level: {mastery}")`
`  else:`
`  print(f"{student_name} has no grades yet.")`
 	
The response states, "No changes would need to be made. This is because there is a if statement at the bottom of the list section part i that makes it so if the students name is not in the student list, it will be added to it:" This is referring to how items are added to the list, rather than handling extra items in the list.
	• High scoring responses explain why no changes to the code segment would be necessary. For example:
	
	`function filterBooks(userInput) {`
	`  titlesOutput = [];`
	`  for (var i=0; i<bookGenre.length; i++) {`
	`  if (userInput == bookGenre[i]) {`
	`  AppendItem(titlesOutput, bookTitles[i]);`
	`  }`
	`  }`
	`}`
	
	The response states, "This is becasue the for loop ensures that the variable(i) is less than the length of the list, ensuring that the loop will run correclty no matter how long the list gets."

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 the various procedural, algorithmic, and list elements.

When creating 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 the difference between Boolean expressions and selection statements. Boolean expressions are evaluated and result in a value of `true` or `false` during program execution. These expressions are used within selection statements to determine whether subsequent instructions are executed or not. Provide examples of selection statements to students and ask them to identify the Boolean expression, what specific value(s) would make the expression evaluate to `true`, and why the expression evaluates to `true` for these value(s). Include examples of Boolean expressions that include logical operators (e.g. "and" vs. "or") and similar relational operators (e.g. "<" vs. "<=") to distinguish when the expression will be `true` or `false`.
- Review the different types of errors that can occur during the programming process and when these errors can occur. Syntax errors typically occur before a program is run when there is code that does not follow the rules of the programming language. Logic errors occur while the program is running and will cause the program to exhibit unexpected behavior such as outputting or returning the wrong result or terminating early without computing the final results. Provide program examples that work correctly and change some element of the program to have students identify what error occurs and what the resulting behavior will be.
- Review data abstraction through the use of lists in a program. Lists are generally used to collect data of a similar type together as a group for processing (e.g. a list of daily temperature measurements of a pool for each day of the summer). Use of lists in this manner is useful because all elements can be accessed easily using the same variable identifier along with an index that can be controlled by a loop, simplifying program code to make it easier to debug and adapt to new uses. In uses like this, using the length of the list as a control for the iteration allows the list to be used regardless of its length. Provide examples of program code that could use a list (but do not use a list) and have students rewrite the code to define and use a list to perform the same action. Be sure that students use the length of list if appropriate and ask why this makes the code more flexible for the programmer.

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