



Chief Reader Report on Student Responses: 2025 AP[®] Computer Science Principles Performance Task Set 2

• 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	0.95
Program Requirements:	1	0.85
Written Response 1:	1	0.78
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 of invalid or unexpected input to the program and explain the behavior of the program after it receives the identified input or explain why it is not possible for the program to accept invalid or unexpected input. The video and written response were considered, using the program code if a video was 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 of invalid or unexpected input to the program and explain the behavior of the program after it receives the identified input or explain why it is not possible for the program to accept invalid or unexpected input. For cases where the program could accept invalid or unexpected input, most responses adequately identified an invalid or unexpected input and described the behavior of the program after it receives the identified input. For cases where the program could not accept invalid or unexpected input, most responses correctly explained why it was not possible (e.g. the use of a dropdown menu restricts input to the options provided). Some responses incorrectly described the behavior of the program after receiving the identified input or stated that it was not possible to receive invalid input when the program could in fact receive invalid input. This suggests that the understanding of how input is handled is still developing.

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

<i>Common Misconceptions/Knowledge Gaps</i>	<i>Responses that Demonstrate Understanding</i>
Course Project Video	
<ul style="list-style-type: none"> • Not having input to the program while the video is running. For example, the program shows a tic-tac-board on the screen but never demonstrates input to the program in the video. 	<ul style="list-style-type: none"> • High scoring responses show the program receiving and responding to input such as mouse clicks or text input fields. For example, the video shows the user clicking in squares of a tic-tac-toe board and typing X or O.
<ul style="list-style-type: none"> • Not showing the program running. For example, a video shows the code being displayed, but the program is never shown being run. 	<ul style="list-style-type: none"> • High scoring responses show the program being run. For example, a video shows an outline of a country and a dropdown where users can select the country from a list. The screen would update to show when answers were selected correctly or incorrectly after selecting the submit button.

Program Requirements

- Not including a student-developed procedure. For example, the response provides program code that includes an event handler instead of a student-developed procedure. For example:

```
onEvent("START", "click", function() {
    ms = 0;
    showElement("ms");
    hideElement("START");
    timedLoop(10, function() {
        ms = ms + 10;
        SetText("ms", ms);
    });
});

onEvent("STOP", "click", function() {
    StopTimedLoop();
});
```

- Inclusion of a list that stores data but does not create new data from existing data or access multiple elements in the list. For example, the list is created and appended to, but the data is never accessed:

```
numbers = []

numbers.append(num)
```

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

```
def search(search_name):
    found = False
    for i in range(len(names)):
        if search_name == names[i]:
            found = True
            print("\nName found!")
            print(f"Name: {names[i]}")
            Print(f"Phone: {num[i]}")
    if not found:
        print("Name not found!")

name = input("Enter name: ")
search(name)
```

- High scoring responses define and use a collection in a meaningful way within the program code. For example:

```
colors = ["white", "yellow",
"lightblue", "orange", "red"]
```

The response goes on to use `colors` as follows:

```
for _ in range(100):
    star = trtl.Turtle()
    star.shape("circle")
    star.shapesize(stretch_wid=0.15,
        stretch_len=0.15)
star.speed(0)
star.color(random.choice(colors))
star.penup()
star.goto(random.randint(-400,
400), random.randint(-300, 300))
star.hideturtle()
star.showturtle()
```

<ul style="list-style-type: none"> • Not including iteration that runs until a given condition is met or for a specified number of times. For example: <pre> forever move 15 steps if on edge, bounce </pre>	<ul style="list-style-type: none"> • High scoring responses include iteration that runs until a condition is met. For example: <pre> while True: player_one_num = int(input("Choose a number between 3-30.")) player_two_num = int(input("Choose a number between 3-30.")) if player_one_num > player_two_num: print("Player 1 may go first.") who = 1 break; elif player_two_num > player_one_num: print("Player 2 may go first.") who = 2 break; else: print("Please pick again") </pre>
<ul style="list-style-type: none"> • Including iteration that is trivial. In the following example, the use of iteration is trivial because the loop runs only one time: <pre> let randomName = Game[Math.floor(Math.random() * Game.length)]; for(let i = 0; i < 1; i++) { println("You are looking for a game to play in the" + randomName + "..."); } </pre>	<ul style="list-style-type: none"> • High scoring responses use selection and iteration to accomplish something meaningful in the program. For example: <pre> for ball in balls: if ball.fill == 'blue' ball.radius += boost ball.fill = 'orange' else: if ball.radius > 20: ball.radius -= boost if ball.radius <= 20: ball.fill = 'blue' </pre>
Written Response 1: Program Design, Function, and Purpose	
<ul style="list-style-type: none"> • Stating, without explanation, that the program cannot accept invalid or unexpected input. For example, “It not possible for my program to accept an unexpected or invalid input because of the way I created my code.” 	<ul style="list-style-type: none"> • High scoring responses explain why it is not possible for the program to accept invalid or unexpected input. For example, “My program can not have an unexpected or invalid input. The reason for this is because the user interface is made up of dropdowns that the user can select, so the user can not have an errors since the program already give the appropriate inputs.”

<ul style="list-style-type: none"> • Incorrectly identifying invalid input as changes to the program code. For example, “An invalid input or an unexpected input in my program that can make my program have a sort of behavior is if the user accidentally removes or inputs something in my “if statements” section which can cause problem.” 	<ul style="list-style-type: none"> • High scoring responses identify invalid or unexpected input to the program. For example, “One invalid input a user could provide would be a string unincuded in the list, user_ingredients_opt, which includes all the strings a user could input. For instance, "sauce" is an example of an invalid input, as it is not in user_ingredients_opt.”
<ul style="list-style-type: none"> • Incorrectly identifying an input as invalid when the input is an expected part of the program’s functionality. For example, “For my program, an invalid input that a user could provide to my program is trying to purchase more items when they don't have enough money to purchase any more items. When my programs receives this kind of input, it lets the user know that they have an insufficient amount of funds on screen, and prevents the user from adding said item into their shopping cart.” 	<ul style="list-style-type: none"> • High scoring responses identify invalid input that is not an expected part of the program’s functionality. For example, “One unexpected or invalid input that a user could input would be inputing the grades of each student seperated by a comma in text rather than in numerical form(ex. "ninety-five" instead of "95". Since my program doesn't have the ability to deal with an invalid input like this, it would produce a typeError when it tries to convert the elements of class1Split into integers.”

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	0.45
Written Response 2(b):	1	0.62
Written Response 2(c):	1	0.40
Overall Mean Score:	1.47	

What were the responses to this question expected to demonstrate?

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), and explain how these expressions function (CRD-2.B) by explaining when and why the expression evaluate to `false` (Written response 2(a): Algorithm Development),
- referring to the List section of the Personalized Project Reference, 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
- explain how the use of procedural abstraction manages complexity in a program (AAP-3.B) by describing the functionality of the procedure and how implementing this functionality as a procedure makes the overall program easier to maintain (Written response 2(c): Data and Procedural Abstraction).

Written Response 2(a) asked students to identify a 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 `false`, and explain why the specified value or set of values will cause the expression to evaluate to `false`. Responses need to demonstrate the ability to accurately explain the behavior of this section of the Personalized Project Reference.

Written Response 2(b) asked students to describe a modification another programmer could make in the list section of their Personalized Project Reference that would cause the program to have a logic error and explain why this modification would lead to the error. Responses to 2(b) demonstrated the student's ability to determine how logic errors can occur and understand what the code is doing. Responses to 2(b) required students to understand their code's functionality and modify that functionality.

Written Response 2(c) asked students to describe the functionality of the procedure in the Personalized Project Reference and how implementing this functionality as a procedure results in the program being easier to maintain than if the functionality were not implemented as a procedure. The use of procedures to simplify code is a key concept of procedural abstraction, making code more readable and allowing code to be reused. In this part, students were expected to show they understood the concept of procedural abstraction as a way to make maintaining programs easier.

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 `false`.

- While some responses identified a Boolean expression in a selection statement, other responses 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 `false`. 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 `false`. Some responses did not explain why the values will cause the expression to `false`, stating only that the values would cause it to be `false`.

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 to the code segment in the list portion of their Personalized Project Reference that would result in a logic error by describing a modification that would cause the program to run to completion with a different outcome, thus correctly identifying a potential logic error in their program.
- Some responses described a modification that would cause the program to terminate early, which demonstrated an ability to identify logic errors, since early termination is unexpected behavior.
- Many responses explained why the modification would result in a logic error. This demonstrates an ability to identify the cause of logic errors that occur during program development.
- Some responses 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 an explanation of why the modification would result in a logic error.

Written response 2(c): Data and Procedural Abstraction – Explain how the use of procedural abstraction manages complexity in a program by describing the functionality of the procedure and how implementing this functionality as a procedure makes the overall program easier to maintain.

- Many responses described the functionality provided by their identified procedure. Being able to describe the functionality of a procedure from a high level is a key part of demonstrating understanding of procedural abstraction.
- Some responses described the functionality of the whole program, rather than the functionality of the procedure.
- Many responses explained how implementing the functionality as a procedure results in the program being easier to maintain than if the functionality were not implemented as a procedure. The ability to explain the benefits of implementing a procedure is integral to demonstrating an understanding of procedural abstraction.

- Some responses correctly described the functionality provided by the procedure but were unable to demonstrate their understanding of the benefits of procedural abstraction and instead explained the benefits that the functionality of the procedure provided to the program in general.
- Some responses confused procedural abstraction and data abstraction and provided an explanation of the benefits of using a list within the procedure, rather than the benefits of using the procedure.

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

<i>Common Misconceptions/Knowledge Gaps</i>	<i>Responses that Demonstrate Understanding</i>
Written Response 2(a): Algorithm Development	
<ul style="list-style-type: none"> • Not identifying the Boolean expression, either explicitly or implicitly through description. For example, “One set of values that will result in my Boolean expression evaluating to false would be nothing and nothing. If the user does not input anything in the type and note boxes then nothing will be added to the list, the function will stop running and leave, and the console log will be filled with an error message.” 	<ul style="list-style-type: none"> • High scoring responses identify the Boolean expression explicitly. For example: <pre data-bbox="894 699 1442 1087"> function imageDriver(driver) { var drivers = getColumn("F1 Drivers", "Drive Names"); var images = getColumn("F1 Drivers", "Driver Images"); for (var i = 0; i < drivers.length; i++) { if (drivers[i] == driver) { return images[i]; } } } </pre> <p data-bbox="894 1136 1511 1241">The response states, “The Boolean expression in my selection statement is "drivers[i] == driver".”</p>

<ul style="list-style-type: none"> Not providing a specific value or set of values that would cause the Boolean expression to evaluate to <code>false</code>. For example, “The boolean expression used in my code is that if the user input is correct, then it will display next screen and display the needed information to go on that screen. However some values that can cause this expression to be false is a invalid input from the user, causing the screen to go to home screen.” 	<ul style="list-style-type: none"> High scoring responses provide a specific value or set of values that would cause the identified Boolean expression to evaluate to <code>false</code>. For example: <pre>function decode(string) { var geneList = []; for(var c = 0; c<string.length; c++){ if(string.substring(c, c+1) != " "){ addItem(geneList, string.substring(c, c+1)); } } }</pre> <p>The response states, “If there is a space in the argument the boolean statement would evaluate to false.” The set of values identified is strings with spaces in them.</p>
<ul style="list-style-type: none"> Not explaining why the specified value or set of values will cause the expression to evaluate to <code>false</code>. For example, “if series=="Fruit Series" evaluates to true then that portion of the if statement is executed. However, if series=="Fruit Series" is not true because another series is selected on the dropdown, then it evaluates to false and moves onto the next part of the if statement.” 	<ul style="list-style-type: none"> High scoring responses explain why the specified value or set of values would cause the expression to evaluate to <code>false</code>. For example: <pre>function updateColor(color) { if (color == "white") { setProperty("jerseyColor", "background-color", "grey"); } else { setProperty("jerseyColor", "background-color", "white"); } }</pre> <p>The response states, “The Boolean expression in this selection statement is <code>color == "white"</code>. A value that would cause this expression to evaluate to false would be if the value of the parameter <code>color</code> was <code>"red"</code>. This parameter value of <code>"red"</code> will cause the Boolean expression to evaluate to false because the string <code>"red"</code> (which is value assigned to the parameter <code>color</code>) is not equivalent to the string <code>"white"</code>.</p>

- Identifying the Boolean expression in a later selection statement, rather than the first selection statement. For example:

```
function sum(exercise) {
  var sumtotal = 0;
  for (var i = 0; i <
exercise.length; i++){
    sumtotal=sumtotal +
    Number(exercise[i]);}
    if (sumtotal < 60) {
      sumtotal = sumtotal +
      " minutes";          }
    if (sumtotal >= 60) {
      sumtotal =
      Math.round(10*
      (sumtotal / 60))/10 +
      " hours";
    }
    return sumtotal;
  }
}
```

The response states, “Within the first selection statement in my procedure, the Boolean expression of “if (sumtotal >= 60)””

- 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 false and explain why the expression would evaluate to false. For example:

```
function filterByDifficulty(input) {
  var desiredDifficulty = "";
  if(input ==0) {
    desiredDifficulty = "Least
    Difficult";
  }
  else if(input == 1) {
    DesiredDifficulty =
    "More Difficult";
  }
  else {
    desiredDifficulty =
    "Most Difficult";
  }
  for(var i = 0; i <
trailNames.length; i++){
    if(trailDifficulties[i] ==
desiredDifficulty) {
      FilteredIndexes1.push(i);
    }
  }
}
```

The response states, “The first boolean expression of my procedure's selection statement is “(input == 0)”. This expression evaluates to false when the value of “input” is not equal to 0, such as the value 1, 2, 3, etc. These values will cause the expression to evaluate to false because the “==” operator in JavaScript only evaluates to true when the two quantities it compares are equivalent. And, since 1, 2, 3, and all other values not equal to 0 are by definition not equal to 0, this boolean expression will evaluate to false.”

Written Response 2(b): Errors and Testing

- Not describing a modification to the program code from the list part (ii) section of the PPR. For example:

```
while #leaderstatesFolders > 1 do
  local folderToRemove =
    table.remove(leaderstatesFolders)
    folderToRemove:Destroy()
end
```

The response states, “Changing either `scaleMultiplier` or `GetScaleMultiplier` will result in a logic error because in the code I made local `scaleMultiplier = GetScaleMultiplier(points.value)` and without one the other wouldn't work.” This explanation does not refer to code provided in the list part (ii) section of the PPR.

- Not describing the modification that would result in a logic error. For example, “They could also try to modify the name of the code which would cause the procedure to never happen in the first place and that is bad as this procedure is required for the game to work properly.”

- High scoring responses describe a modification to the code from part (ii) of the List section of the PPR that would result in a logic error. For example:

```
onEvent("pmDropdown", "change",
function() {
  var selectedParent =
    getText("pmDropdown");

  var filteredMountains =
    conversion1(selectedParent);

  setText("infoArea",
    filteredMountains.join(", "));

  parentMountainFilter = [];
});
```

The response states, “If another programmer removed the `parentMountainFilter []`, it would cause a logic error. This is because the whole code segment will still run, but it will produce the wrong output. If this empty list was removed, everytime a different parent mountain is selected from the dropdown, it will keep adding to the previous displays of mountains, since the list will never be emptied.”

- High scoring responses describe a modification that would result in the program crashing after execution and describe the behavior of the program leading up to the crash. For example, “This would result in a logic error because the only possible values for `i` are 0 and 1. By modifying the line to not include 1 but instead only include values greater than one, the code will not end up appending an value to each index value that is equal to one. This, in turn, would cause an error later in the code segment, as the length of the list would not be able to satisfy the required number of elements in the list to display the board.

For example, if there are 12 zeroes and 3 ones, the “display” list would only be 12 elements long. However, the code segment that prints the board requires at least 15 elements in the “display” list, resulting in an error.”

<ul style="list-style-type: none"> • Describing behavior that is implausible, inaccurate, or inconsistent with the program. For example, “If the programmer decides to make it so that the user only has to choose from the dropdown menu and not click the search button it would result in a logic error because if the user chooses two options at the same time by putting their cursor in the middle of both options, lets say if they decide to put it between the golden retriever and the poodle option, the expression would break because it wouldn't know which one to choose.” This behavior is implausible because it does not correctly describe how dropdown menus work. 	<ul style="list-style-type: none"> • High scoring responses describe a modification that would result in a logic error and describe why the modification would result in a logic error. For example, “The programmer could edit the print statement in line 87 to modify the {i+1} part of the f-string and just leave it as {i}...That is because in Python, list indexes begin at 0 and not one, and since this code segment is using the index variable "i" from the enumerate() function on line 86, the value of "i" will start at 0.”
<ul style="list-style-type: none"> • Not explaining why the modification would result in a logic error. For example, “If another programmer were to attempt to modify the code I have developed a potential logic error that may occur could be setting the boolean vairable all_same to false and practically reversing all of the usage of that variable which would result in a logic error as the program wouldnt be able to accurately run through the procedure.” 	<ul style="list-style-type: none"> • High scoring responses explain why the modification would result in a logic error. For example, “An example of a modification made that would result in a logic error is when someone changes the line of code: "var total1 = 0;" to "var total1 = 1;" This would be a logic error because to find a correct total, the counter should start at 0. This is because a counter should only count the number of elements that it is given based on the for loop it is in. An example of how this wouldn't work is if the list being checked is 3 elements long. If the counter starts at 1, and then the amount of elements in the list is checked, and the number of elements in the list are then added, the total1 variable would be 4, because 1+ 3(1) is equal to 4.”

Written Response 2(c): Data and Procedural Abstraction

- Not describing the functionality of the identified procedure from part (i) of the Procedure section of the PPR. For example, the code in the part (i) of the Procedure section of the PPR is:

```
function ChangeColor(Change) {  
  if (Score >= Change) {  
    Color = 0;  
    var Color =  
      getProperty("EvilButton",  
        "background-color");  
    SetProperty("EvilButton",  
      "background-color",  
      "#FF0000");  
  }  
}
```

But the written response states, “In my coding, I used a list to create multiple motivations, it can be said once a certain score is reached it will display it and it would want the user to keep on going. This list makes my coding more clean and less messy, it also makes my game look more like one and make it more enjoyable while also giving it the aesthetic look.”

- Not explaining how implementing this functionality as a procedure results in the program being easier to maintain and instead providing an explanation of how the functionality of the procedure benefits the users of the program. For example, “Adding this function to my code makes it easier for users to know what they are about to see and should help them know what to expect in my app.”

- High scoring responses describe the functionality provided by the identified procedure. For example, “The functionality of this procedure is that it creates new variables to store the the oldest persons year of birth and their name. It then traverses the "birthYears" list to determine if someone is older by seeing if the birth year at a certain index is smaller that the "oldest" variable that holds a certain birth year. Once it has finished traversing the list, it returns the oldest person's name to stop the function.”

- High scoring responses explain how implementing the functionality as a procedure results in the program being easier to maintain than if the functionality were not implemented as a procedure. For example, “This results in my program being easier to be maintained because it takes the cards that the computer players have and runs through the proceedure and at the end just returning a number. And this proceedure is called three times for my program one for each of the computer players. This makes it easier so I dont have to write the same code three times for the same result.”

<ul style="list-style-type: none"> • Providing an explanation of the benefits of using a data abstraction instead of procedural abstraction. For example, “Implementing this functionality as a procedure results in my program being easier to maintain as the program has all the specific details on an assignment categorized in separate lists. So when the user needs to check a certain aspect of their work, the program can easily go back to it instead of having all of the information be stored into one big list.” 	<ul style="list-style-type: none"> • High scoring responses explain how implementing the functionality as a procedure including the use of a parameter resulted in a program that was easier to maintain. For example, “Implementing the function <code>pickpark(state)</code> results in my program being easier to maintain because, without it, I would have to write code for every single state the user could possibly input. Using the function, I can write general code to fit every state and include a parameter that can be changed, so the code can be used for all states. Therefore, the procedure <code>pickpark(state)</code> manages complexity by reducing redundancy, and without it, my code would be repetitive and unorganized, as I would have to write code for every state.”
---	---

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 `false`, and why

the expression evaluates to `false` 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 involving lists that work correctly and change some element of the list code to have students identify what error occurs and what the resulting behavior will be.
- Review procedural abstraction to illustrate why procedures make programs easier to maintain. Procedures with explicit parameter(s) create a section of program code that can be called in a full program to solve a variety of similar problems that depend on the explicit parameter(s). Procedures also simplify testing by isolating functionality so a segment of code can be called and tested independently. Procedures also allow programmers to break a larger programming problem into smaller units, which can be replaced without affecting the rest of the program code. Provide examples of program code that has repeated code sections that can be replaced with a single procedure and an explicit parameter and have students make this change and verify that the program code still works as intended.

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