

---

# AP<sup>®</sup> Computer Science Principles

## Sample Student Responses and Scoring Commentary Set 1

### **Inside:**

#### **Written Response 2**

- ☒ **Scoring Guidelines**
- ☒ **Student Samples**
- ☒ **Scoring Commentary**

**Digital Portfolio Components provided separately**

Written Response 2

3 points

- General Scoring Notes**
- Written responses should be evaluated solely on the rationale provided.
  - Written responses must demonstrate all scoring criteria, including those within bulleted lists, in each reporting category to earn the point for that category.
  - Terms and phrases defined in the terminology list are italicized when they first appear.

Reporting Category	Scoring Criteria	Decision Rules
Written Response 2(a): Algorithm Development  (0–1 points)	The written response: <ul style="list-style-type: none"><li>• describes what is being accomplished by the code in the body of the <i>iteration</i> statement.</li></ul>	<p><b>Consider the Personalized Project Reference and Written Response 2(a) when scoring this point.</b></p> <ul style="list-style-type: none"><li>• If multiple iteration statements are included in the Procedure section of the Personalized Project Reference, use the first iteration statement to determine whether the point is earned.</li><li>• The first iteration statement can be found in either part (i) or part (ii) of the Procedure section of the Personalized Project Reference.</li><li>• The iteration statement does not need to be contained in a procedure to earn this point.</li><li>• If a procedure is identified, it does not need to contain a parameter to earn this point.</li><li>• The response may describe a summary of what the iteration does in the context of the program or describe the purpose of each statement in the body of the iteration.</li></ul> <p><b>Do NOT award a point if any one or more of the following is true:</b></p> <ul style="list-style-type: none"><li>• The Procedure section of the Personalized Project Reference does not contain an iteration statement.</li><li>• The description of what is being accomplished by the code does not match the code in the body of the first iteration statement.</li><li>• The response only restates the lines of code in the body of the iteration statement.</li><li>• The response describes a trivial use of iteration.</li><li>• The response describes an iteration statement or behavior that is implausible, inaccurate, or inconsistent with the program.</li></ul>

Reporting Category	Scoring Criteria	Decision Rules
<p><b>Written Response 2(b): Errors and Testing</b></p> <p><b>(0–1 points)</b></p>	<p>The written response:</p> <ul style="list-style-type: none"> <li>includes two calls to the <i>procedure</i>. Each call must cause a different program <i>code segment</i> in the procedure to execute.</li> <li>describes the expected behavior of each call.</li> </ul> <p>OR</p> <ul style="list-style-type: none"> <li>explains why it is not possible for two calls to the procedure to cause different code segments to execute.</li> </ul>	<p><b>Consider the Personalized Project Reference and Written Response 2(b) when scoring this point.</b></p> <ul style="list-style-type: none"> <li>If multiple procedures are included in part (i) of the Procedure section of the Personalized Project Reference:             <ul style="list-style-type: none"> <li>Use the procedure identified in the written response to determine whether the point is earned.</li> <li>If no procedure is identified in the written response, then use the first procedure to determine whether the point is earned.</li> </ul> </li> <li>The parameter(s) used in the procedure must be explicit. Explicit parameters are defined in the header of the procedure.</li> <li>A procedure that uses its parameter(s) to execute two different code segments can earn this point.</li> <li>A procedure that uses its parameter(s) to execute or bypass a code segment can earn this point.</li> <li>The syntax of the procedure calls does not need to be correct as long as the correct arguments are identified.</li> <li>A description of each call rather than program code is acceptable.</li> <li>A general description of argument value(s) is considered acceptable.</li> </ul> <p><b>Do NOT award a point if any one or more of the following is true:</b></p> <ul style="list-style-type: none"> <li>A procedure is not identified in part (i) of the Procedure section of the Personalized Project Reference.</li> <li>The response does not apply to the procedure in part (i) of the Procedure section of the Personalized Project Reference.</li> <li>The procedure identified in part (i) of the Procedure section of the Personalized Project Reference does not include at least one explicit parameter.</li> <li>The use of the explicit parameter is irrelevant (e.g., does not affect the code segment of the procedure that is executed or is reassigned before being used).</li> <li>The two calls are to two different procedures.</li> <li>The response describes expected behavior that is implausible, inaccurate, or inconsistent with the program.</li> </ul>

Reporting Category	Scoring Criteria	Decision Rules
<b>Written Response 2(c): Data and Procedural Abstraction</b>  <b>(0–1 points)</b>	<p>The written response:</p> <ul style="list-style-type: none"> <li>explains in detailed steps an <i>algorithm</i> that uses <code>checkValidity</code> to check whether all elements in the <i>list</i> are considered valid.</li> </ul>	<p><b>Consider the Personalized Project Reference and Written Response 2(c) when scoring this point.</b></p> <ul style="list-style-type: none"> <li>If multiple lists are included in the List section of the Personalized Project Reference, use the list identified in the written response to determine whether the point is earned.</li> <li>The algorithm can be described in code, pseudocode, as a sequence of steps in English, or as a paragraph in English.</li> <li>The algorithm must describe how each element of the identified list is passed into <code>checkValidity</code> at least up to the first invalid element, if applicable.</li> </ul> <p><b>Do NOT award a point if any one or more of the following is true:</b></p> <ul style="list-style-type: none"> <li>A list is not identified in the List section of the Personalized Project Reference.</li> <li>If the algorithm described assumes the list contains a single element.</li> <li>The list identified in the Personalized Project Reference is not referenced in the response.</li> <li>The response implements <code>checkValidity</code> rather than describing its use.</li> <li>The response is too vague to allow another programmer to recreate the algorithm.</li> </ul>

## AP Computer Science Principles Create Performance Task Terminology

**Algorithm:** An algorithm is a finite set of instructions that accomplish a specific task. Every algorithm can be constructed using combinations of sequencing, selection, and iteration.

**Arguments:** The values of the parameters when a procedure is called.

**Code segment:** A code segment refers to a collection of program statements that are part of a program. For text-based, the collection of program statements should be continuous and within the same procedure. For block-based, the collection of program statements should be contained in the same starter block or what is referred to as a “Hat” block.

**Collection type:** Aggregates elements in a single structure. Some examples include: databases, hash tables, dictionaries, sets, or any other type that aggregates elements in a single structure.

**Data stored in a list:** Input into the list can be through an initialization or through some computation on other variables or list elements.

**Input:** Program input is data that are sent to a computer for processing by a program. Input can come in a variety of forms, such as tactile (through touch), audible, visual, or text. An event is associated with an action and supplies input data to a program.

**Iteration:** Iteration is a repetitive portion of an algorithm. Iteration repeats until a given condition is met or for a specified number of times. The use of recursion is a form of iteration.

**List:** A list is an ordered sequence of elements. The use of lists allows multiple related items to be represented using a single variable. Lists are referred to by different terms, such as arrays or ArrayLists, depending on the programming language.

**List being used:** Using a list means the program is creating new data from existing data or accessing multiple elements in the list.

**Output:** Program output is any data that are sent from a program to a device. Program output can come in a variety of forms, such as tactile, audible, visual, movement, or text.

**Parameter:** A parameter is an input variable of a procedure. Explicit parameters are defined in the procedure header. Implicit parameters are those that are assigned in anticipation of a call to the procedure. For example, an implicit parameter can be set through interaction with a graphical user interface.

**Procedure:** A procedure is a named group of programming instructions that may have parameters and return values. Procedures are referred to by different names, such as method, function, or constructor, depending on the programming language. A procedure is executed through the use of a procedure call.

**Program functionality:** The behavior of a program during execution, often described by how a user interacts with it.

**Purpose:** The problem being solved or creative interest being pursued through the program.

**Selection / conditional statement:** A selection / conditional statement affects the sequential flow of control by executing different statements based on a condition being true or false. The use of if-statements and try / exception statements are examples of selection / conditional statements.

**Sequencing:** The application of each step of an algorithm in the order in which the code statements are given.

**Student-developed procedure / algorithm:** Program code that is student-developed has been written (individually or collaboratively) by the student who submitted the response. Calls to existing program code or libraries can be included but are not considered student-developed. Event handlers are built-in abstractions in some languages and will therefore not be considered student-developed. In some block-based programming languages, event handlers begin with “when.”

## QUESTION 2

Begin your responses to QUESTION 2 parts (a), (b), and (c) on this page.

- (a) The code in the iteration (which begins on line 19) perpetually runs through the if/else statement within it, until the condition in the if/else statement is met, at which point it is able to run line 28 of the code and break (stop iterating the code). Specifically, the code will perpetually run and ask the user to give inputs until ~~the input~~ the user gives an input that is a part of the usernames-list list. This allows the program to keep asking the user for a username until it receives a valid, previously created, username.
- (b) No matter what string the parameter 'message' in this procedure is changed to, all code segments will always run. This is because the 'message' parameter impacts what text is displayed to the user before they give the program an input, but it does not effect whether or not that line of code, with any other code segment, to be executed, as the line will always execute, just with a different line of text during each execution. There are no other parameters that could change which code is executed in separate calls to the procedure. Furthermore, whether or not the code segments in the if/else statement is executed or not is dependant on user input, not how the procedure is called, thus two different calls to the procedure will not always result in a code segment always being executed or always not being executed.

Page 3

Use a pencil or a pen with black or dark blue ink. Do NOT write your name. Do NOT write outside the box.

0062498



## QUESTION 2

Continue your responses to QUESTION 2 parts (a), (b), and (c) on this page.

(c) First, set the variable 'valid' to 0. Iterate through every Key and Value in the south-america dictionary. (note: south-america is a dictionary, which is not a list, however it is another collection type that has a similar functionality to a list). Use selection, so that if when the 'value' parameter in the procedure checkValidity is the 'key' in the dictionary, it will increase the variable 'valid' by 1, if checkValidity returns true. Then, if the 'value' parameter in the checkValidity procedure is set to the value in the dictionary, it will increase the variable 'valid' by 1 if checkValidity returns true. Finally, to display these results, you may print/display "All elements are valid" if the 'valid' variable is equal to 6 (the valid variable must equal 6, since there are 3 keys and 3 values in the dictionary, for a total of 6 elements, that the dictionary, all of which needed to be true and increase the value of valid to result in the conclusion that all elements were valid).

Here is an example of this code in Python code:

```
valid = 0
for key, value in south-america.items():
```

```
    if checkValidity(key) == True:
```

```
        valid = valid + 1
```

```
    if checkValidity(value) == True:
```

```
        valid = valid + 1
```

```
if valid == 6:
```

```
    print("All elements are valid")
```

```
else:
```

```
    print("Not all elements
```

Page 4

Use a pencil or a pen with black or dark blue ink. Do NOT write your name. Do NOT write outside the box.

## QUESTION 2

Begin your responses to QUESTION 2 parts (a), (b), and (c) on this page.

- a. In the iteration statement, the code ~~creates~~ takes a selected list, and for each value of the list the code runs through the rest of ~~the~~ list to check if there are duplicates, and if so, removes them. The code assigns the selected list item to the variable "selected". Then there is a loop that checks if the preceding items in the list = the "selected" variable. If the statement is evaluated to be "True" the tested item is removed from the list, so this is repeated for every item in the list, effectively removing duplicates, and returning a list of the remaining values.

Later, my code runs a list of the types of all the drinks through this loop, returning only ~~one~~ a list of the ~~types~~ possible types that can be found, with no repetition.

- b. The function "gettypes" ~~fun~~ takes a list and returns a new list of all the possible values for that list.

One call: var x = gettypes([0, 1, 2, 3, 4]) ~~I~~ ints

In this call, the function runs through the list [0, 1, 2, 3, 4], and checks if one value in the list is equivalent to any other in the list, however this if statement always evaluates to false, thus the ~~an~~ following code never activates.

call two: var y = gettypes([0, 11, 11, 13, 13, 14]) ~~ints~~ ints

In this call however, the if statement does evaluate to true when comparing items 11, and 11. The following code activates in contrast, removing the latter "11" from the code.

The first call doesn't run the code to remove the duplicate, while the second call does.

Page 3

Use a pencil or a pen with black or dark blue ink. Do NOT write your name. Do NOT write outside the box.

0113468

■ ■ ■ ■ ■ ■ ■ ■ ■ ■



## QUESTION 2

Continue your responses to QUESTION 2 parts (a), (b), and (c) on this page.

c. To check if all elements in the list "drinktype" is valid an algorithm would need to run through all items in drinktype, and return false if one value evaluates to "false" using checkValidity, and to return true if no values are considered invalid. First set a variable "validity" to true, then

A for loop should be run for every item in the list, with  $i$ . An if statement should be run inside the loop to check if the item  $i$  is valid

if (checkValidity( $i$ ) = ~~for~~ false) { ... should ~~run~~ checkValidity return false

Set "validity" = false and end the loop for loop. Should no value be evaluated to false, all elements in drinktype would be considered valid, with validity = true. Should any one item be "invalid" validity would = false and not every item in drinktype is valid.

Use a pencil or a pen with black or dark blue ink. Do NOT write your name. Do NOT write outside the box.

## QUESTION 2

Begin your responses to QUESTION 2 parts (a), (b), and (c) on this page.

- a. In the body of the first iteration statement in the Procedure, the code looks at every item in "equationList", & if an item is equal to " $\wedge$ ", it takes the previous item & puts it to the power of the next item. It then stores this value in the variable "calcAns". The code then removes the operands & operator (" $\wedge$ ") from "equationList", & inserts the value of "calcAns" into its place. The code then repeats until there are no more " $\wedge$ " in "equationList".
- b. If the "calculate" procedure was called with the parameters  $[ [2^+3], 1, 2 )$  it would run the code segment that adds 2 to 3, removes " $+$ " & both 1's, stores the value of  $2+3$  in "calcAns" & adds the value back into "equationList". However if the parameters were to be  $[ [2, "x", 3], 1, 2 )$  it would run a similar code segment that multiplies instead of adds & would return the value 6 instead of 5.
- c. An algorithm that uses "checkValidity" would have to run the program with valid inputs first since "eqnList" doesn't hold any elements before running. It would then have to determine if each string in "eqnList" is equal to its integer. For example it would have to check if  $[ "3" ]$  is equal to  $[ 3 ]$  after it is changed from a string to an integer. If it did, it would return "true", if not it would return "false".

Page 3

Use a pencil or a pen with black or dark blue ink. Do NOT write your name. Do NOT write outside the box.

0043565

## QUESTION 2

Begin your responses to QUESTION 2 parts (a), (b), and (c) on this page.

2a) In my procedure, the iteration being used is a for i loop to iterate through the 'sorting list'. It starts with the first element, then keeps going until it reaches the end. Based on the user's answers, what the list holds will vary. If one of the elements is an 'a', then the 'counter a' will increase by 1. This goes the same for 'b', 'c', and 'd'.

2b) In my procedure, the argument being passed is 'sorting list'. Based on the user's answers, the elements in the list can vary. If the list [a,a,a,a,a] was passed to the procedure it would cause the "if item i of input = a" segment to occur 5 times, making the value of 'counter a' to be 5. The else if statements would not occur. If the list [a,b,c,d,d] was passed to the procedure, the first if statement would occur, making the 'counter a' be worth 1. Then, the first else if statement would occur, making 'counter b' be worth 1. Then, the second else if statement would occur, making 'counter c' be worth 1. Finally, the third else if statement would occur twice (for both occurrences of 'd'). The final values of the counters would have counter a = 1, counter b = 1, counter c = 1, and counter d = 2.

Page 3

Use a pencil or a pen with black or dark blue ink. Do NOT write your name. Do NOT write outside the box.

0038765





## QUESTION 2

Continue your responses to QUESTION 2 parts (a), (b), and (c) on this page.

2c) The list that I created is called 'sorting list'. Using the procedure checkValidity(value), it is possible to create an algorithm to check the validity of a value. To create the algorithm, it is best to use a for i loop to iterate each spot in the list. Using an example list [a, b, c, d], the procedure can start off with saying if the first element is an 'a', return true, else, return false. Then, if the second element is a 'b' return true, else, return false. Next, we can say that if the third element is a 'c', return true, else, return false. Then, if the fourth element is a 'd', return true, else, return false. Finally, if the last element is a 'd', return true, else, return false. If all are returned as true, the list is considered valid.

Use a pencil or a pen with black or dark blue ink. Do NOT write your name. Do NOT write outside the box.



## QUESTION 2

Begin your responses to QUESTION 2 parts (a), (b), and (c) on this page.

a) the card logic function is used by the computer to determine what card of 14's hand needs to be changed by comparing each card to each other and finding the match, then sends back what card needs to be changed.)

X<sup>9</sup> the cardlogic function is only called once and it's inside the computerPlay function and it's comparing all of the computer's cards and saying which one to change and storing it inside the variable results

b) cardlogic(2, K, J, 3) and cardlogic(3, 3, 4, K), the expected behavior of the first call is that it will see that the first 2 cards don't match so it will return Hand[1] indicating that the 2nd card needs to be changed.

the expected output of the second call is that the computer sees that the first 2 cards match and checks them to the 3rd card and sees it doesn't match and will send back Hand[2] indicating the 3rd card needs changed

Page 3

Use a pencil or a pen with black or dark blue ink. Do NOT write your name. Do NOT write outside the box.

0092913

## QUESTION 2

Continue your responses to QUESTION 2 parts (a), (b), and (c) on this page.

c) the check validity function could be run at the start of the code to count if there are 52 items in the values list inside the deck function and if theres 52 items then return True else if theres less than 52 items then return false, the number 52 indicates 52 cards inside the deck.

Page 4

Use a pencil or a pen with black or dark blue ink. Do NOT write your name. Do NOT write outside the box.



## QUESTION 2

Begin your responses to QUESTION 2 parts (a), (b), and (c) on this page.

a) The first iteration statement is, `for i in range(len(Playlist-1))`. This statement runs code that searches for a specific value, given to the program by the user, and removes the part of the list that is equal to this value. ~~Then~~ ~~every value that is not equal to the specified value will get appended to a second playlist, playlist\_2, to ensure the playlist is an accurate representation of what the user wanted to listen to.~~ Every value that is not equal to the specified value will get appended to a second playlist, playlist\_2, to ensure the playlist is an accurate representation of what the user wanted to listen to.

b) In this procedure, ~~there~~ there is only one possible call, because no other call could trigger different parts of my code. An example of my call would be, `remove_song("Sicko Mode")`. The procedure can only run one segment of code, because only one code segment is ~~needed~~ necessary to fulfill the task of my program.

c) The procedure `checkValidity(value)` is used to check to see if the values are valid, meaning it could iterate through my lists and check to see if any values in the list are equal to a value given by the user. ~~checkValidity goes through Playlist 1 and see if any of the values are equal to the value given. If the value is equal to the value given, it will return true. If the value is not equal to the value given, it will return false. This is to ensure the validity of the list for the user to enjoy a playlist that is fully functional with no flaws.~~ To further clarify, I would have the program, `checkValidity`, iterate through the list to check for duplicate songs. If there are 2 versions of the same value it would flag the recurrence of the value as false, while the rest are true. This is to ensure the validity of the list for the user to enjoy a playlist that is fully functional with no flaws.

Use a pencil or a pen with black or dark blue ink. Do NOT write your name. Do NOT write outside the box.

0135481

## Question 2

**Note:** Student samples are quoted verbatim and may contain spelling and grammatical errors.

### Overview

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 the iteration statement 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 students to determine if their procedure is functioning correctly, they must first understand the procedure's expected behavior for a given input so that they can match the expected output to the observed output when they run the procedure. Second, it is important for tests to cover many different cases, including those that cause different segments of the code to execute. Because the students have flexibility in their procedures, it could have not been possible to have two different calls to their procedure that caused different segments of their 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.

## Question 2 (continued)

**Sample: A**

**Score:**

**Question 2(a): 1**

**Question 2(b): 1**

**Question 2(c): 1**

**Question 2(a):**

The response earned this point:

- The response describes what is being accomplished by the code in the body of the iteration statement: “Specifically the code will perpetually run and ask the user to give inputs until the user gives an input that is a part of the `usernames_list` list. This allows the program to keep asking the user for a username until it receives a valid, previously created, username.”

**Question 2(b):**

The response earned this point.

- The response explains why it is not possible for two calls to the procedure to cause different code segments to execute: “No matter what string the parameter ‘message’ in this procedure is changed to, all code segments will always run. This is because the ‘message’ parameter impacts what text is displayed to the user before they give the program an input, but it does not effect whether or not that line of code, or any other code segment, to be executed....”

**Question 2(c):**

The response earned this point:

- The response explains in detailed steps an algorithm that uses `checkValidity` to check whether all elements in the dictionary are considered valid by giving the algorithm both in text and in pseudocode. The given algorithm uses a loop and a conditional statement to count the number of elements in the dictionary for which `checkValidity` returns `true`. After the loop, it prints a statement if all elements are valid. This algorithm correctly determines whether all elements in the list are valid, and it is explained in sufficient detail that another programmer could implement it.

## Question 2 (continued)

**Sample: B**

**Score:**

**Question 2(a): 1**

**Question 2(b): 1**

**Question 2(c): 1**

**Question 2(a):**

The response earned this point:

- The response describes what is being accomplished by the code in the body of the iteration statement: “the code takes a selected list, and for each value of the list the code runs through the rest of the list to check if there are duplicates, and if so, removes them.”

**Question 2(b):**

The response earned this point, demonstrating both criteria:

- The response includes two calls to the procedure that cause a different program code segment in the procedure to execute: "getTypes([0,1,2,3,4])" and "getTypes([0,11,11,13,14])". The first call will not cause the body of the if statement to execute, while the second will.
- The response describes the expected behavior of each call. The response states that there are no duplicates in the first call, “thus the lines of code never activates.” In the second call, “when comparing items 11, and 11. The following code activates in contrast, removing the latter “11” from the code.” The response also describes the expected behavior of each call: "The first call doesn't run the code to remove the duplicate, while the second call does."

**Question 2(c):**

The response earned this point:

- The response explains in detailed steps an algorithm that uses `checkValidity` to check whether all elements in the list are considered valid by establishing a variable “validity” and setting it to `true`. The response states, “A for loop should be ran for every item in the list, i is valid if (`checkValidity(i) = false`).” The response goes on to describe that “validity” should be set to false and the for loop should terminate, when `checkValidity` returns `false` and if no elements return `false` then “validity” will still be `true`. The response concludes, “should any one item be “invalid” validity would = false and not every item in drinktype is valid.”

## Question 2 (continued)

**Sample: C**

**Score:**

**Question 2(a): 1**

**Question 2(b): 1**

**Question 2(c): 0**

**Question 2(a):**

The response earned this point:

- The response describes what is being accomplished by the code in the body of the iteration statement: “the code looks at every item in “equationList” & if an item is equal to “^”, it takes the previous item and puts it to the power of the next item. It then stores the value in the variable “calcAns”....the code then repeats until there are no more “^” in “equationList”.

**Question 2(b):**

The response earned this point, demonstrating both criteria:

- The response includes two calls to the procedure: “if the “calculate” procedure was called with the parameters ([2 “+” 3] , 1, 2) it would run the code segment that add 2 to 3...” and “...if the parameters were ([2 “x” 3],1,2) it would run a similar code segment that multiplies instead of adds.”
- The response describes the expected behavior of the first call: “stores the value of 2 + 3 in “calcAns” & adds the value back into “equationList” and that of the second call: “...multiplies instead of adds and would return the value 6 instead of 5.”

**Question 2(c):**

The response did not earn this point:

- The response does not explain in detailed steps an algorithm that uses `checkValidity` to check whether all elements in the list are considered valid. The response describes the implementation of `checkValidity` rather than an algorithm that passes values as arguments into `checkValidity`.

## Question 2 (continued)

**Sample: D**

**Score:**

**Question 2(a): 1**

**Question 2(b): 1**

**Question 2(c): 0**

**Question 2(a):**

The response earned this point:

- The response describes what is being accomplished by the code in the body of the iteration statement by stating, “based on the user’s answers, what the list holds will vary. If one of the elements is an ‘a’ then the ‘counter a’ will increase by 1” and so on.

**Question 2(b):**

The response earned this point, demonstrating both criteria:

- The response includes two calls to the procedure by providing the list “[a,a,a,a]” and the list “[a,b,c,d,d]” as arguments in each.
- The response describes the expected behavior for the first call by stating, “it would cause the “if item i of input =a” segment to occur 5 times, making the value of ‘counter a’ to be 5.” The response describes the second call by explaining how the if-else statements would run and then stating, “The final values of the counters would have counter a=1, counter b=1, counter c=1 and counter d=2.”

**Question 2(c):**

The response did not earn this point:

- The response does not explain in detailed steps an algorithm that uses `checkValidity` to check whether all elements in the list are considered valid. The response describes an algorithm that iterates through the list, but the algorithm does not use `checkValidity` to check each element in the list. Instead, the algorithm checks elements against a particular character: “the procedure can start off...if the first element is an ‘a’, return true...if the second element is a ‘b’, return true...if the third element is ‘c’” and so on. Additionally, the algorithm will not check each element because it returns either `true` or `false` after it checks the first element. Therefore, it will only check the first element.



## Question 2 (continued)

**Sample: E**

**Score:**

**Question 2(a): 0**

**Question 2(b): 1**

**Question 2(c): 0**

**Question 2(a):**

The response did not earn this point:

- The response describes what is being accomplished by the code in the body of the procedure. However, the body of the procedure does not contain iteration, nor is iteration present in the additional procedure “computerPlay” identified in part (ii).

**Question 2(b):**

The response earned this point, demonstrating both criteria:

- The response includes two calls to the procedure: "cardlogic(2,K,J,3)" and "cardlogic(3,3,4,K)."
- The response describes the expected behavior of the first call: “it will see that the first 2 cards dont match so it will return cHand[1].” It also describes the expected behavior of the second call: “the computer sees that the first 2 cards match and checks them to the 3rd card and sees it doesnt match and will send back cHand[2].”

**Question 2(c):**

The response did not earn this point:

- The response does not explain in detailed steps an algorithm that uses `checkValidity` to check whether all elements in the list are considered valid. The algorithm in the response checks if there are 52 items in the values list but does not describe how `checkValidity` could be used to check whether each individual item in the list is valid.

## Question 2 (continued)

**Sample: F**

**Score:**

**Question 2(a): 1**

**Question 2(b): 0**

**Question 2(c): 0**

**Question 2(a):**

The response earned this point:

- The response describes what is being accomplished by the code in the body of the iteration statement by stating, "This statement runs the code that searches for a specific value, given to the program by the user, and removes the part of the list that is equal to this value." It also explains, "Every value that is not equal to the specified value will get appended to a second playlist."

**Question 2(b):**

The response did not earn this point, demonstrating none of the criteria:

- The response explains why it is not possible for two calls to the procedure to cause different code segments to execute. The response states, "there is only one possible call, because no other call could trigger different parts of my code." However, this explanation is not correct. A call to the procedure with the name of a song that is in the list (`playlist_1`) will cause the body of the if statement (`if playlist_1[i] == name:...`) to execute, while a call to the procedure with a name of a song that is not in the list will cause the `else` statement to execute. In addition, the parameter `name` is overwritten before it is used.

**Question 2(c):**

The response did not earn this point:

- The response does not explain in detailed steps an algorithm that uses `checkValidity` to check whether all elements in the list are considered valid. Instead, the response explains an algorithm designed to check if there are duplicate songs in the list. It states that `checkValidity` will "iterate through the list to check for duplicate songs. If there are 2 versions of the same value it would flag the reoccurrence of the values as false, while the rest are true." The algorithm described is both too vague to allow another programmer to recreate the algorithm and does not check whether all elements in the list are considered valid.