# Module 1 Unit Test: Create Task Written Responses

## 1. Program Code

Please see the final pages of this PDF file for complete program code.

How to generate a PDF of your program code is described in this 3-minute video.

## 2. Video

The video demonstration my program can be found here.

## 3. Written Responses

a)

i. The program's purpose is to find and save "icebreaker" jokes along with notes about when such humour might be useful in a particular social situation.

ii. From 0:00 to 00:30 the user reviews three jokes: the setup, then the punchline. The user adds notes for each joke before saving them for future reference. From 00:30 to 00:43 the user searches through saved jokes to find those that contain only certain words.

iii. Input occurs when buttons are clicked to obtain jokes, show punchlines, or save jokes. Input occurs when the user types notes about a joke and when providing terms to filter the saved jokes list. When jokes are read from a file in the app bundle, that is also input. Output occurs when jokes are shown, when jokes are saved for future reference, and when a list is filtered based on a search term.

**3. WRITTEN RESPONSES** (CREATED INDEPENDENTLY)

Submit your responses to prompts 3a – 3d, which are described below. Your response to all prompts combined must not exceed 750 words (program code is not included in the word count). Collaboration is **not** allowed on the written responses. Instructions for submitting your written responses are available on the **AP Computer Science Principles Exam Page** on AP Central.

**3 a.** Provide a written response that does all three of the following:

*Approx. 150 words (for all subparts of 3a combined)*

i. Describes the overall purpose of the program

ii. Describes what functionality of the program is demonstrated in the video

iii. Describes the input and output of the program demonstrated in the video

**General Scoring Notes**

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

| Reporting Category | Scoring Criteria | Decision Rules |
|---|---|---|
| **Row 1 Program Purpose and Function**<br><br>**(0-1 points)**<br>**4.A** | The video demonstrates the running of the program including:<br>• *input*<br>• *program functionality*<br>• *output*<br>**AND**<br>The written response:<br>• describes the overall *purpose* of the program.<br>• describes what functionality of the program is demonstrated in the video.<br>• describes the input and output of the program demonstrated in the video. | **Consider ONLY the video and written response 3a when scoring this point.**<br><br>**Do NOT award a point if the following is true:**<br>• The video does not show a demonstration of the program running (screenshots or storyboards are not acceptable and would not be credited). |

In your program, you must include student-developed program code that contains the following:

☐ Instructions for input from one of the following:
- the user (including user actions that trigger events)
- a device
- an online data stream
- a file

b)

i.

```
293     func saveJoke() {
294
295         let jokeToSave = SavedJoke(
```

```
296             joke: currentJoke,
297             comments: providedComments
298         )
299
300         history.append(jokeToSave)
301
302         providedComments = ""
303
304     }
305 }
```

**3b.** Capture and paste two program code segments you developed during the administration of this task that contain a list (or other collection type) being used to manage complexity in your program.

***Approx. 200 words (for all subparts of 3b combined, exclusive of program code)***

- **i.** The first program code segment must show how data have been stored in the list.
- **ii.** The second program code segment must show the data in the same list being used, such as creating new data from the existing data or accessing multiple elements in the list, as part of fulfilling the program's purpose.

Then, provide a written response that does all three of the following:

- **iii.** Identifies the name of the list being used in this response
- **iv.** Describes what the data contained in the list represent in your program
- **v.** Explains how the selected list manages complexity in your program code by explaining why your program code could not be written, or how it would be written differently, if you did not use the list

DEFINITION:

***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 may be referred to by different names, such as **arrays**, depending on the programming language.

DEFINITION:

***Collection Type***

A **collection type** is a type that aggregates elements in a single structure. Some examples include lists, databases, hash tables, dictionaries, and sets.

IMPORTANT:

The data abstraction must make the program easier to develop (alternatives would be more complex) or easier to maintain (future changes to the size of the list would otherwise require significant modifications to the program code).

ii.

```
267             List(
268
269                 filtering(
270                     listOfSavedJokes: history,
271                     on: searchText
272                 )
273
274             ) { savedJoke in
275
276                 ListItemView(toShow: savedJoke)
277
278             }
279             .searchable(text: $searchText)
```

iii. The name of the list is '*history*'.

iv. The data in the list represent jokes saved by the user for future reference.

v. The '*history*' list (array) manages complexity because it is inconvenient to use the SwiftUI framework to build a scrollable list in the user interface (lines 267 to 279) with individual variables. While it is possible to use individual variables to build a scrollable list, with the '*history*' array, the program simply iterates over all elements in '*history*' to show them in the user interface.

Additionally, by using the '*history*' array, new elements can be added dynamically. The app can store many jokes – limited only by the available memory of the device running the app. With individual variables the user could only save a fixed number of jokes. To save more jokes would require program code changes.

| | | |
|---|---|---|
| **Row 2**<br>**Data Abstraction**<br><br>**(0-1 points)**<br>**3.B** | The written response:<br>• includes two *program code segments*:<br>- one that shows how *data has been stored in this list* (or other *collection type*).<br>- one that shows the data in this same *list being used* as part of fulfilling the program's purpose.<br>• identifies the name of the variable representing the list being used in this response.<br>• describes what the data contained in this list is representing in the program. | **Consider ONLY written response 3b when scoring this point.**<br><br>**Requirements for program code segments:**<br>• The written response must include two clearly distinguishable program code segments, but these segments may be disjointed code segments or two parts of a contiguous code segment.<br>• If the written response includes more than two code segments, use the first two code segments to determine whether or not the point is earned.<br><br>**Do NOT award a point if any one or more of the following is true:**<br>• The list is a one-element list.<br>• The use of the list does not assist in fulfilling the program's purpose. |
| **Reporting Category** | **Scoring Criteria** | **Decision Rules** |
| **Row 3**<br>**Managing Complexity**<br><br>**(0-1 points)**<br>**3.C** | The written response:<br>• includes a program code segment that shows a list being used to manage complexity in the program.<br>• explains how the named, selected list manages complexity in the program code by explaining why the program code could not be written, or how it would be written differently, without using this list. | **Consider ONLY written response 3b when scoring this point.**<br><br>**Responses that do not earn the point in row 2 may still earn the point in this row.**<br><br>**Do NOT award a point if any one or more of the following is true:**<br>• The code segments containing the lists are not separately included in the written response section (not included at all, or the entire program is selected without explicitly identifying the code segments containing the list).<br>• The written response does not name the selected list (or other collection type).<br>• The use of the list is irrelevant or not used in the program.<br>• The explanation does not apply to the selected list.<br>• The explanation of how the list manages complexity is implausible, inaccurate, or inconsistent with the program.<br>• The solution without the list is implausible, inaccurate, or inconsistent with the program.<br>• The use of the list does not result in a program that is easier to develop, meaning alternatives presented are equally complex or potentially easier.<br>• The use of the list does not result in a program that is easier to maintain, meaning that future changes to the size of the list would cause significant modifications to the code. |

c)

```
25 func filtering(
26     listOfSavedJokes: [SavedJoke],
27     on searchText: String
28 ) -> [SavedJoke] {
29
30     if searchText.isEmpty {
31         return listOfSavedJokes
32     }
33
34     var filteredListOfJokes: [SavedJoke] = []
35
36     for savedJoke in listOfSavedJokes {
37
38         if savedJoke.comments.lowercased().contains(searchText.lowercased()) ||
39             savedJoke.joke.setup.lowercased().contains(searchText.lowercased()) ||
40             savedJoke.joke.punchline.lowercased().contains(searchText.lowercased())
41         {
42
43             filteredListOfJokes.append(savedJoke)
44
45         }
46
47     }
48
49     return filteredListOfJokes
50
51 }
```

**3c.** Capture and paste two program code segments you developed during the administration of this task that contain a student-developed procedure that implements an algorithm used in your program and a call to that procedure.

***Approx. 200 words (for all subparts of 3c combined, exclusive of program code)***

IMPORTANT:
Built-in or existing procedures and language structures, such as event handlers and main methods, are not considered student-developed.

**i.** The first program code segment must be a student-developed procedure that:

- ☐ Defines the procedure's name and return type (if necessary)
- ☐ Contains and uses one or more parameters that have an effect on the functionality of the procedure
- ☐ Implements an algorithm that includes sequencing, selection, and iteration

**ii.** The second program code segment must show where your student-developed procedure is being called in your program.

Then, provide a written response that does both of the following:

**iii.** Describes in general what the identified procedure does and how it contributes to the overall functionality of the program

**iv.** Explains in detailed steps how the algorithm implemented in the identified procedure works. Your explanation must be detailed enough for someone else to recreate it.

ii.

```
267             List(
268
269                 filtering(
270                     listOfSavedJokes: history,
271                     on: searchText
272                 )
273
274             ) { savedJoke in
275
276                 ListItemView(toShow: savedJoke)
277
278             }
279             .searchable(text: $searchText)
```

iii. The '*filtering*' function limits the list of saved jokes; jokes are shown based on text the user provides, demonstrated from 00:30 to 00:43 in the video.

iv. A selection statement (line 30) checks whether the '*searchText*' parameter holds an empty string; when true, no search text has been provided, so the unfiltered original list containing all jokes is returned (line 31).

When '*searchText*' does not contain an empty string results are filtered as follows. An empty list, '*filteredListOfJokes*', is created to hold jokes that will be returned (line 34). The function then iterates over the original list of saved jokes (line 36). Within the loop, for each saved joke, the comments the user made about that joke, the joke setup, and the joke punchline, are all examined to see if they contain whatever text is in the '*searchText*' parameter. Strings on each side of these comparisons are converted to lower case, so that matching is case-insensitive. Whenever the search text is found within a saved joke's comment, setup, or punchline, the saved joke is added to the list to be returned from the function (line 43). After the loop completes the filtered list of jokes is returned (line 49).

| | | |
|---|---|---|
| **Row 4**<br>**Procedural Abstraction**<br><br>**(0-1 points)**<br>3.B | The written response:<br>• includes two program code segments:<br>- one showing a *student-developed procedure* with at least one *parameter* that has an effect on the functionality of the procedure.<br>- one showing where the student-developed procedure is being called.<br>• describes what the identified procedure does and how it contributes to the overall functionality of the program. | **Consider ONLY written response 3c when scoring this point.**<br><br>**Requirements for program code segments:**<br>• The procedure must be student developed, but could be developed collaboratively with a partner.<br>• If multiple procedures are included, use the first procedure to determine whether the point is earned.<br>• The parameter(s) used in the procedure must be explicit. Explicit parameters are defined in the header of the procedure.<br><br>**Do NOT award a point if any one or more of the following is true:**<br>• The code segment consisting of the procedure is not included in the written responses section.<br>• The procedure is a built-in or existing procedure or language structure, such as an event handler or main method, where the student only implements the body of the procedure rather than defining the name, return type (if applicable) and parameters.<br>• The written response describes what the procedure does independently without relating it to the overall function of the program. |

| Reporting Category | Scoring Criteria | Decision Rules |
|---|---|---|
| **Row 5**<br>**Algorithm Implementation**<br><br>**(0-1 points)**<br>2.B | The written response:<br>• includes a program code segment of a *student-developed algorithm* that includes<br>- *sequencing*<br>- *selection*<br>- *iteration*<br>• explains in detailed steps how the identified algorithm works in enough detail that someone else could recreate it. | **Consider ONLY written response 3c when scoring this point.**<br><br>**Responses that do not earn the point in row 4 may still earn the point in this row.**<br><br>**Requirements for program code segments:**<br>• The algorithm being described can utilize existing language functionality or library calls.<br>• An algorithm that contains selection and iteration, also contains sequencing.<br>• An algorithm containing sequencing, selection, and iteration that is not contained in a procedure can earn this point.<br>• Use the first code segment, as well as any included code for procedures called within this first code segment, to determine whether the point is earned.<br>• If this code segment calls other student-developed procedures, the procedures called from within the identified procedure can be considered when evaluating whether the elements of sequencing, selection, and iteration are present as long as the code for the called procedures is included.<br><br>**Do NOT award a point if any one or more of the following is true:**<br>• The response only describes what the selected algorithm does without explaining how it does it.<br>• The description of the algorithm does not match the included program code.<br>• The code segment consisting of the selected algorithm is not included in the written response.<br>• The algorithm is not explicitly identified (i.e., the entire program is selected as an algorithm without explicitly identifying the code segment containing the algorithm).<br>• The use of either the selection or the iteration is trivial and does not affect the outcome of the program. |

d)

i. The first call to '*filtered*' occurs on line 269 and at 00:33 in the video. The argument for the '*listOfSavedJokes*' parameter is '*history*' which contains the complete list of saved jokes. The argument for the '*on*' parameter is the local variable '*searchText*' whose value is set when the user types in the search field.

Another call to '*filtered*' also occurs on line 269 but at 00:43 in the video. The argument for the '*originalList*' parameter is '*history*' which contains the complete list of saved jokes. The argument for the '*on*' parameter is the local variable '*searchText*' whose value is set to an empty string by the user at this point in time.

ii. On line 30 the condition is whether '*searchText*' is an empty string. In this first call, '*searchText*' contains 'factory', so the condition in the selection statement does not evaluate to 'true', and line 31 is skipped. The algorithm continues on lines 34 to 49.

In the second call, on line 30, the condition is again whether '*searchText*' contains an empty string. In this second call, '*searchText*' does contain an empty string so the condition in the selection statement does evaluate to true. As a result, line 31 is run.

iii. The result of the first call is that a filtered list of jokes is returned.

The result of the second call is that all jokes are returned.

**3 d.** Provide a written response that does all three of the following:

***Approx. 200 words (for all subparts of 3d combined)***

**i.** Describes two calls to the procedure identified in written response 3c. Each call must pass a different argument(s) that causes a different segment of code in the algorithm to execute.

First call:

Second call:

**ii.** Describes what condition(s) is being tested by each call to the procedure

Condition(s) tested by the first call:

Condition(s) tested by the second call:

**iii.** Identifies the result of each call

Result of the first call:

Result of the second call:

| Reporting Category | Scoring Criteria | Decision Rules |
|---|---|---|
| **Row 6**<br>**Testing**<br><br>**(0-1 points)**<br>**4.C** | The written response:<br>• describes two calls to the selected procedure identified in written response 3c. Each call must pass a different *argument(s)* that causes a different segment of code in the algorithm to execute.<br>• describes the condition(s) being tested by each call to the procedure.<br>• identifies the result of each call. | **Consider ONLY the written response for 3d and the selected procedure identified in written response 3c.**<br><br>**Responses that do not earn the point in row 4 may still earn the point in this row.**<br><br>**Requirements for program code segments:**<br>• Consider implicit or explicit parameters used by the selected procedure when determining whether this point is earned. 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.<br>• A condition that uses the procedure's parameter(s) to execute two different code segments can earn this point.<br>• A condition that uses the procedure's parameter(s) to execute or bypass a code segment can earn this point.<br><br>**Do NOT award a point if any one or more of the following is true:**<br>• A procedure is not identified in written response 3c.<br>• The written response for 3d does not apply to the procedure in 3c.<br>• The two calls cause the same segment of code in the algorithm to execute even if the result is different.<br>• The response describes conditions being tested that are implausible, inaccurate, or inconsistent with the program.<br>• The identified results of either call are implausible, inaccurate, or inconsistent with the program. |