Create Performance Task

Row 3 - Managing Complexity

Instructions

1. You can only use this version if you can write on a pdf, otherwise please use the ppt version.
2. Read the criteria for this row on slides 3 - 4.
3. For each response *(slides 5 – 14)*:
   a. Underline any code, phrases or sentences that meet any of the criteria.
   b. Tick (✓) any criteria that are satisfied and cross (✗) any criteria that are not satisfied.
   c. Write a **1** in the bottom right corner if the response gets the mark for this row *(all criteria have been satisfied)*, otherwise write a **0**.
4. When finished, save this presentation as a pdf with 2 slides to a page and submit this file.

Scoring Criteria

The written response:

• includes a program code segment that shows a list being used to manage complexity in the program.

• 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.

Most programs can be written without lists, but they would be more complicated without them. Therefore, I strongly advise that you always explain how it could be written differently without a list *(typically by using multiple variables)* **rather than just stating it cannot be written without a list!**

Decision Rules

Responses that do not earn the point in row 2 may still earn the point in this row.Do NOT award a point if any one or more of the following is true:

• 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).

• The written response does not name the selected list (or other collection type).

• The use of the list is irrelevant or not used in the program.

• The explanation does not apply to the selected list.

• The explanation of how the list manages complexity is implausible, inaccurate, or inconsistent with the program.

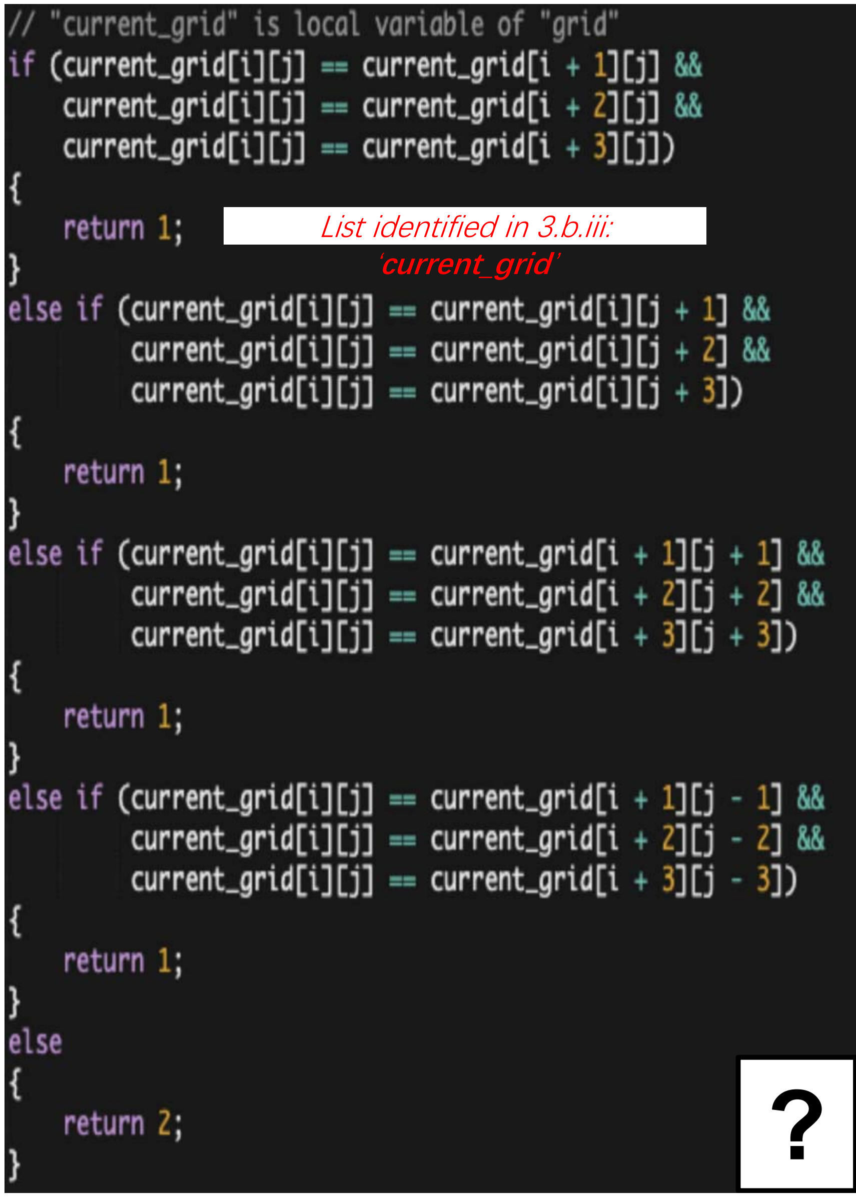
• The solution without the list is implausible, inaccurate, or inconsistent with the program.

• 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.

• 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.

a Without the list, the program would have no way to remember previous player inputs, making gameplay impossible. The list allows for the program to save and update the status of every position on the game board for every turn. In turn, the saved data in the list can then be accessed later, enabling other important functions such as printing the updated game board after every turn (print_grid). The list is also accessed to determine which position is to be filled next in each column. Additionally, after every turn, the list is accessed in the check_overall and check_individual functions to determine whether a player has won.

- Includes a program code segment that shows a list being used to manage complexity in the program.
- 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.

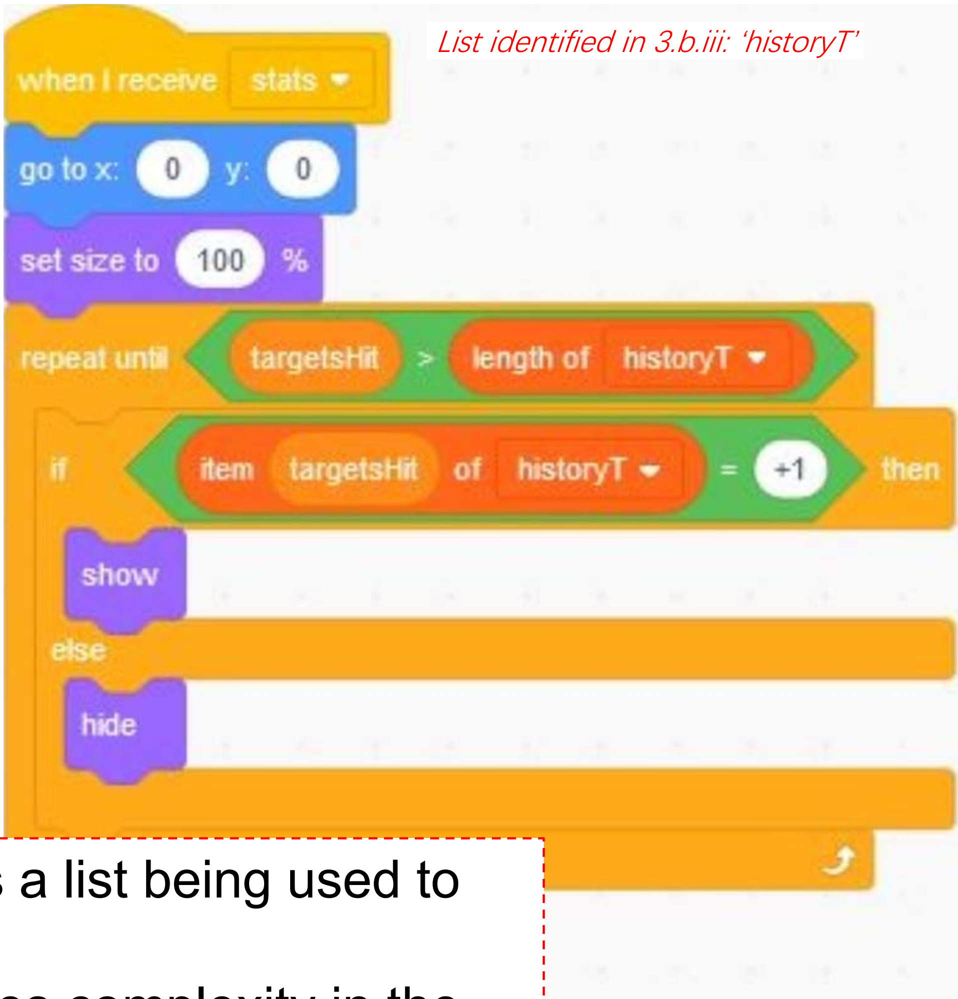


```
// "current_grid" is local variable of "grid"
if (current_grid[i][j] == current_grid[i + 1][j] &&
    current_grid[i][j] == current_grid[i + 2][j] &&
    current_grid[i][j] == current_grid[i + 3][j])
{
    return 1;
}
else if (current_grid[i][j] == current_grid[i][j + 1] &&
         current_grid[i][j] == current_grid[i][j + 2] &&
         current_grid[i][j] == current_grid[i][j + 3])
{
    return 1;
}
else if (current_grid[i][j] == current_grid[i + 1][j + 1] &&
         current_grid[i][j] == current_grid[i + 2][j + 2] &&
         current_grid[i][j] == current_grid[i + 3][j + 3])
{
    return 1;
}
else if (current_grid[i][j] == current_grid[i + 1][j - 1] &&
         current_grid[i][j] == current_grid[i + 2][j - 2] &&
         current_grid[i][j] == current_grid[i + 3][j - 3])
{
    return 1;
}
else
{
    return 2;
}
```

List identified in 3.b.iii:
'current_grid'

?

b Therefore, the lists manage complexity in the program because at the very end, the user can see each target that shows up on the screen to know what targets were hit throughout the ten seconds. This is accomplished using the list's position numbers for the code to track each position's value which correspond for a sprite:target to appear. Without this list, it would be difficult because the variable block would only track the amount but not order and say and ask/answer blocks would require some extra code and may get overwhelmed by the large inputs of multiple target values.



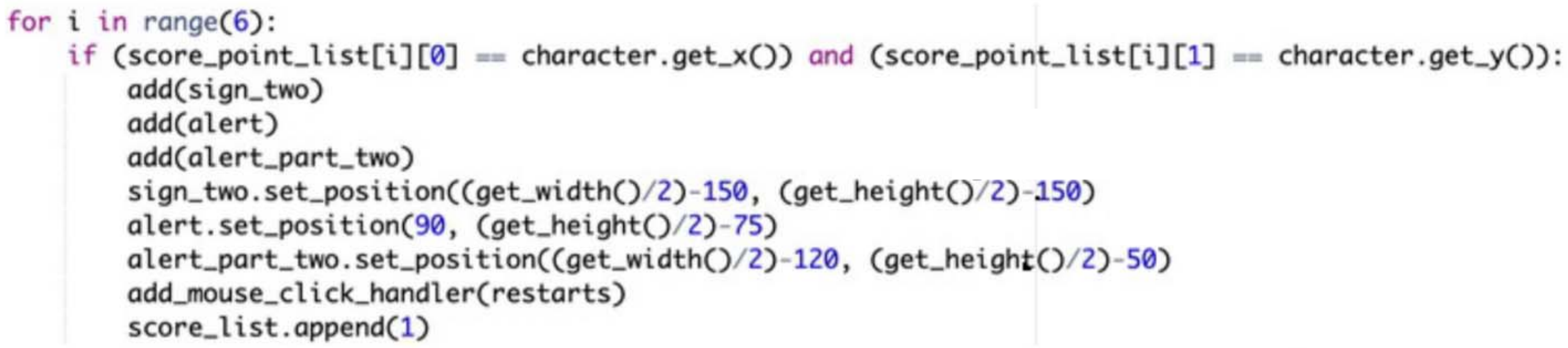
List identified in 3.b.iii: 'historyT'

```
when I receive stats
go to x: 0 y: 0
set size to 100 %
repeat until <(targetsHit) > (length of historyT)>
    if <(item (targetsHit) of historyT) = +1> then
        show
    else
        hide
```

- Includes a program code segment that shows a list being used to manage complexity in the program.
- 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.

?

c



```
for i in range(6):
    if (score_point_list[i][0] == character.get_x()) and (score_point_list[i][1] == character.get_y()):
        add(sign_two)
        add(alert)
        add(alert_part_two)
        sign_two.set_position((get_width()/2)-150, (get_height()/2)-150)
        alert.set_position(90, (get_height()/2)-75)
        alert_part_two.set_position((get_width()/2)-120, (get_height()/2)-50)
        add_mouse_click_handler(restarts)
        score_list.append(1)
```

List identified in 3.b.iii: 'score_point_list'

As shown in the second image, the list manages complexity in the program as it allows it to check if the user moved the red ball onto the stars or the mystery blocks all at once in a much quicker manner, through the use of a for loop and an if statement. If not for the list, I would've had to create several separate if statements for each of the different coordinate points stored in the list, and check if that equals the current location of the red ball. The list allows the for loop to access a different coordinate point each time the loop runs, allowing it to check through the whole list to see if the current coordinates of red ball equals one of the coordinates stored in the list.

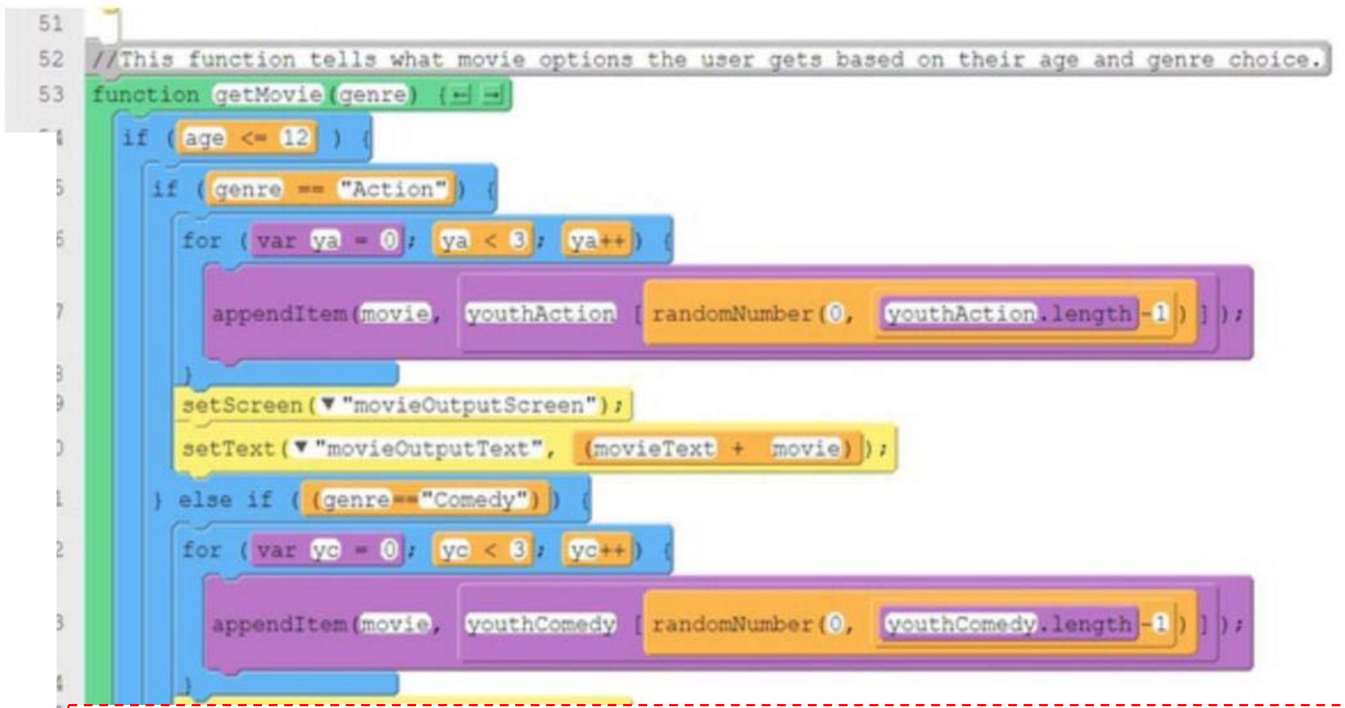
- Includes a program code segment that shows a list being used to manage complexity in the program.
- Explains how the named, selected list manages complexity in the program code by explaining why the program code could not be written, or how it would be written differently, without using this list.

?

d

List identified in 3.b.iii:
*'**youthAction**'*

The lists are extremely helpful when wanting to store a large amount of values/data. Having the ability to use lists prevents one from having to create a bunch of different variables that would all have different names and only store a single piece of data or a single value. Using a bunch of individual variables instead of a list makes your code unorganized, lengthy, and prone to mistakes. A user could possibly reuse the same variable name and delete one of their values altogether. Overall, lists allow for condensed and more organized coding.



```
//This function tells what movie options the user gets based on their age and genre choice.
function getMovie(genre) {
  if (age <= 12) {
    if (genre == "Action") {
      for (var ya = 0; ya < 3; ya++) {
        appendItem(movie, youthAction[randomNumber(0, youthAction.length-1)]);
      }
      setScreen("movieOutputScreen");
      setText("movieOutputText", (movieText + movie));
    } else if ((genre=="Comedy")) {
      for (var yc = 0; yc < 3; yc++) {
        appendItem(movie, youthComedy[randomNumber(0, youthComedy.length-1)]);
      }
```

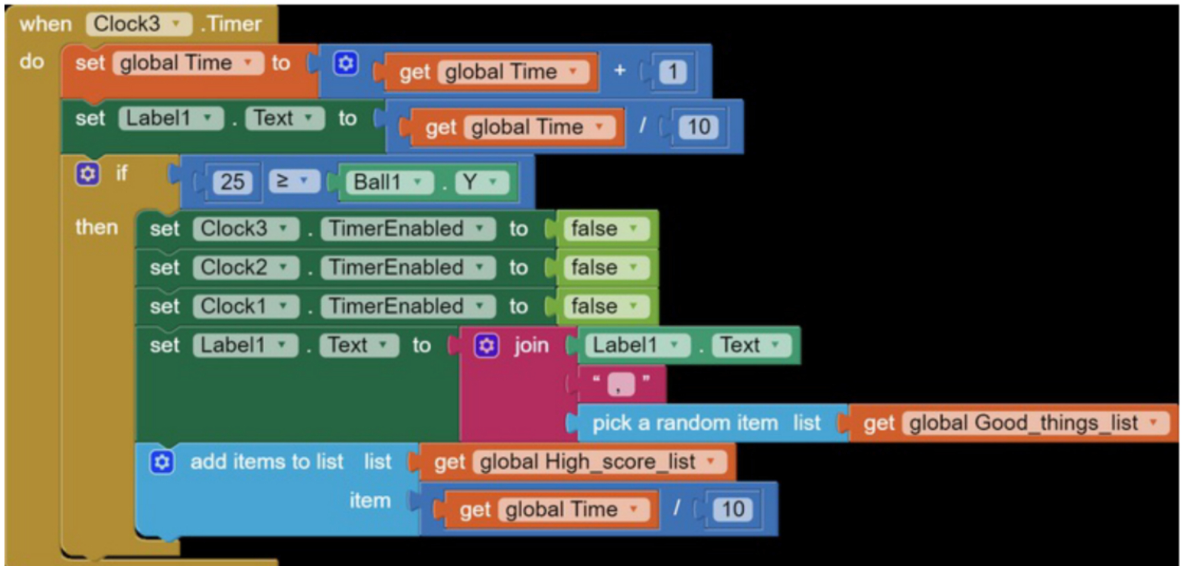
- Includes a program code segment that shows a list being used to manage complexity in the program.
- Explains how the named, selected list manages complexity in the program code by explaining why the program code could not be written, or how it would be written differently, without using this list.

?

e

List identified in 3.b.iii: 'Good_things_list'

If I could not use a list, then this section of my program could not be written because it requires the selection from the list to get different good words to say.

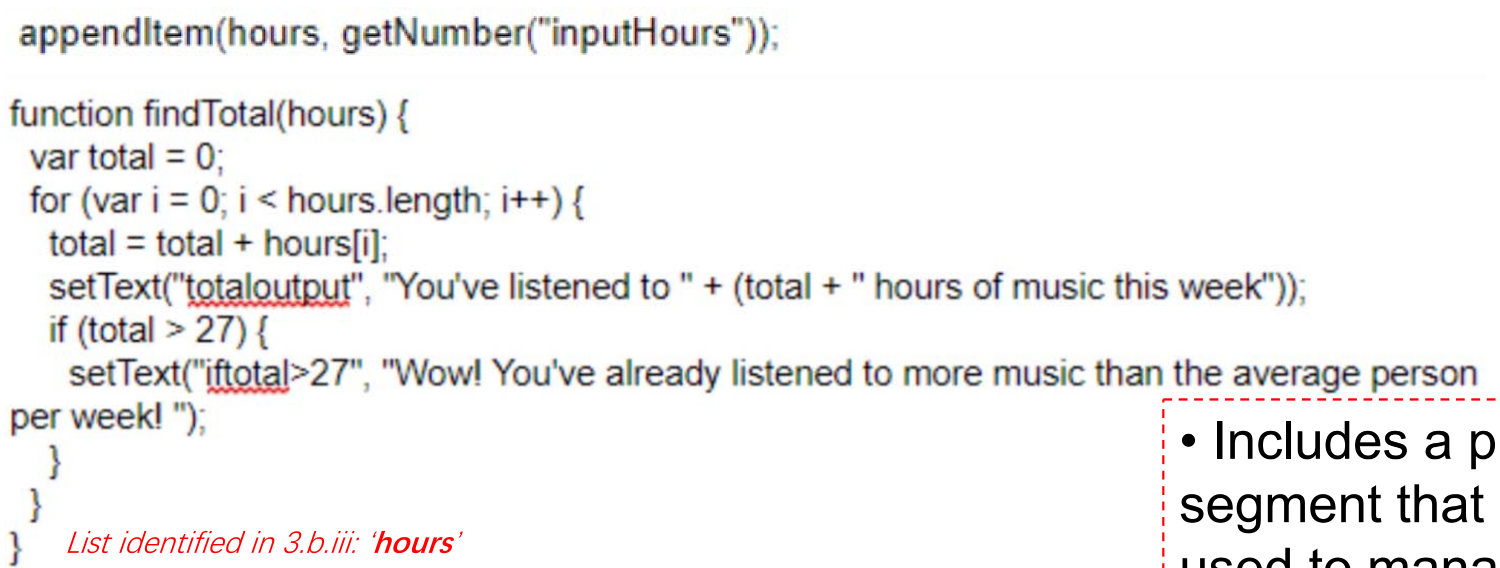


```
when Clock3 .Timer
do  set global Time to ( get global Time + 1 )
    set Label1 . Text to ( get global Time / 10 )
    if 25 ≥ Ball1 . Y
    then set Clock3 . TimerEnabled to false
         set Clock2 . TimerEnabled to false
         set Clock1 . TimerEnabled to false
         set Label1 . Text to join  Label1 . Text
                                    " , "
                                    pick a random item list  get global Good_things_list
    add items to list  list  get global High_score_list
                       item  get global Time / 10
```

• Includes a program code segment that shows a list being used to manage complexity in the program.
• 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.

?

f



```
appendItem(hours, getNumber("inputHours"));

function findTotal(hours) {
  var total = 0;
  for (var i = 0; i < hours.length; i++) {
    total = total + hours[i];
    setText("totaloutput", "You've listened to " + (total + " hours of music this week"));
    if (total > 27) {
      setText("iftotal>27", "Wow! You've already listened to more music than the average person
per week! ");
    }
  }
}
```

*List identified in 3.b.iii: '**hours**'*

The hour list manages complexity because it allows any number of inputted hours to be stored in the list instead of using individual variables for each time the user wants to add to the lists. Without the list, a variable would have to be created for every number possibility the user could type and then an if statement for each of those numbers. This would result in an incredibly large amount of code compared to the only 15 lines of code used for this algorithm

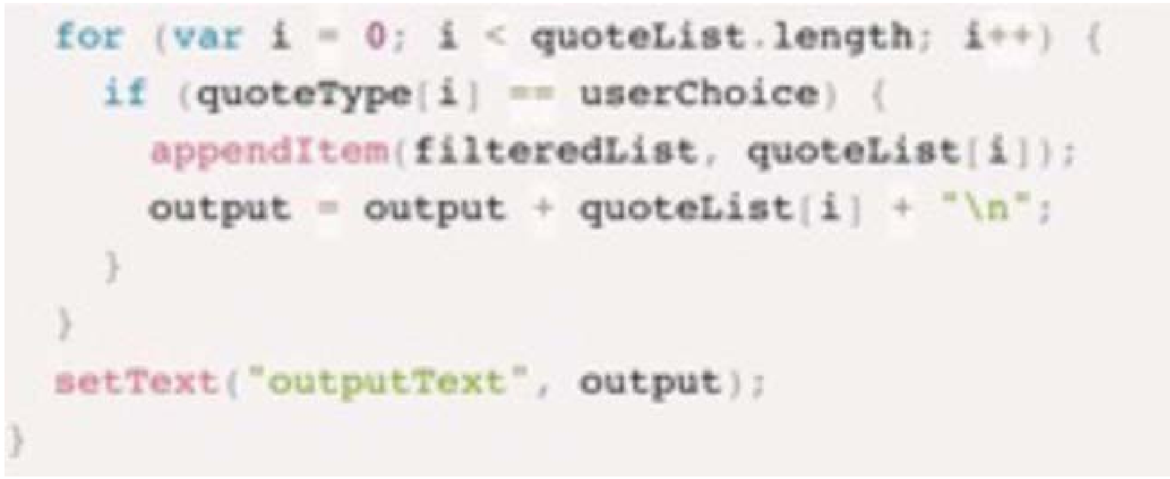
- Includes a program code segment that shows a list being used to manage complexity in the program.
- Explains how the named, selected list manages complexity in the program code by explaining why the program code could not be written, or how it would be written differently, without using this list.

?

g

List identified in 3.b.iii: '**quoteType**'

My list makes it easier to sort the quotes depending on the type of quote. Without my list differentiating the two options, all of the quotes would be printed out together when either of the buttons are pressed. This list also makes it possible to easily add additional movie lines or song lyrics to be printed to the data set without complicating the code at all.



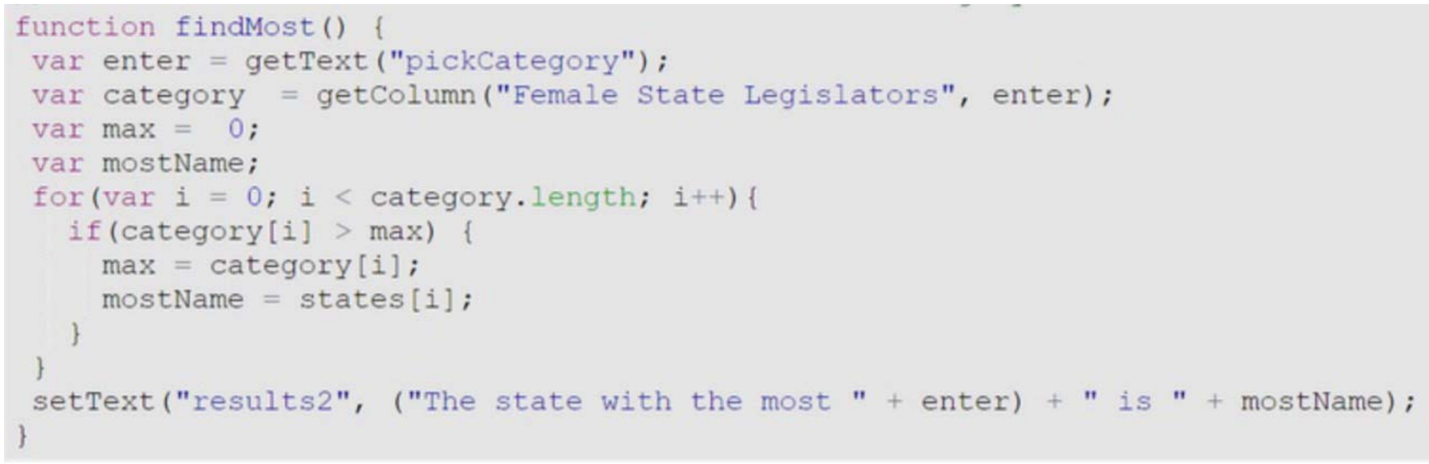
```
  for (var i = 0; i < quoteList.length; i++) {
    if (quoteType[i] == userChoice) {
      appendItem(filteredList, quoteList[i]);
      output = output + quoteList[i] + "\n";
    }
  }
  setText("outputText", output);
}
```

- Includes a program code segment that shows a list being used to manage complexity in the program.
- 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.

?

h

- Includes a program code segment that shows a list being used to manage complexity in the program.
- 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.



```
function findMost() {
 var enter = getText("pickCategory");
 var category  = getColumn("Female State Legislators", enter);
 var max =  0;
 var mostName;
 for(var i = 0; i < category.length; i++){
   if(category[i] > max) {
     max = category[i];
     mostName = states[i];
   }
 }
 setText("results2", ("The state with the most " + enter) + " is " + mostName);
}
```

*List identified in 3.b.iii: '**Female State Legislators***

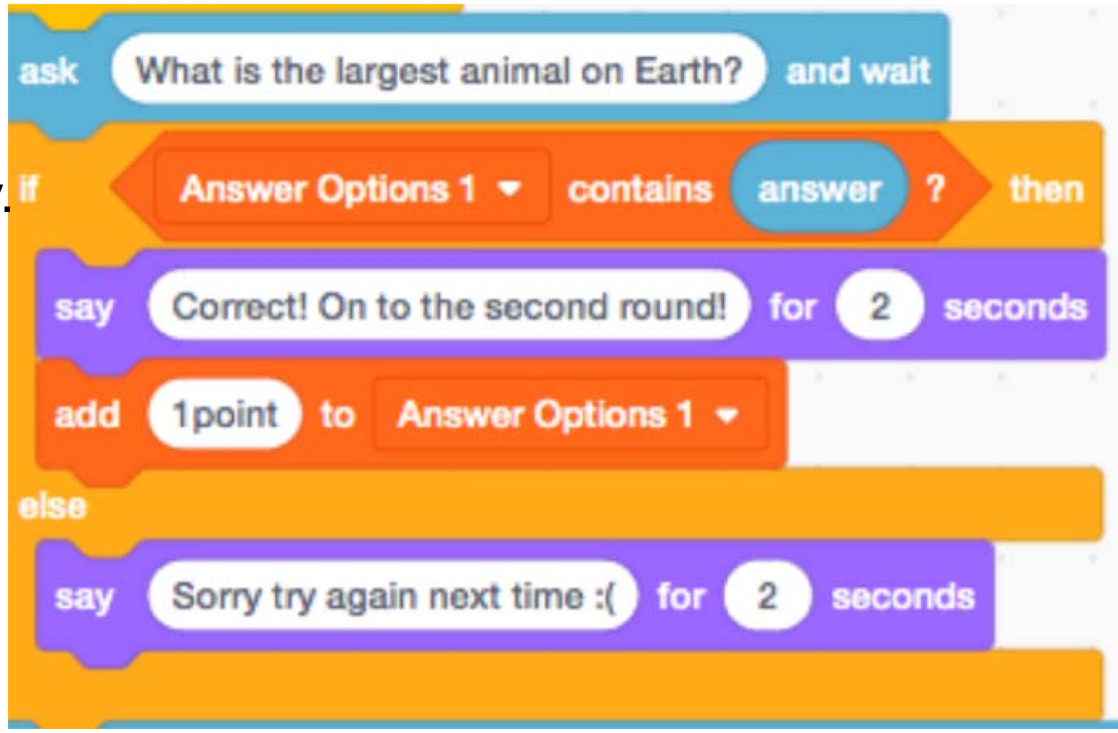
The Female State Legislators list manages complexity because it allows for the user to choose between many different categories and states while also controlling what the user inputs into the app. It also ensures that the user does not choose a category or a state that the program cannot provide an answer for. If the program did not contain the list, then the app would be much more complex because it would have to take into account all of the possibilities the user can input, which can even be a thousand.

?

i

*List identified in 3.b.iii: '**Answer Options 1**'*

Within this code, it was important for me to use the list to reduce further complexity. I used a list in order to take as many answer possibilities as possible while still maintaining a balance of right and wrong. This is inclusive of spaces in between words and capitalization in the program. When using scratch I saw that the only way that multiple answers can be accepted in this manner is if you use the list to assign the answer to multiple variables.



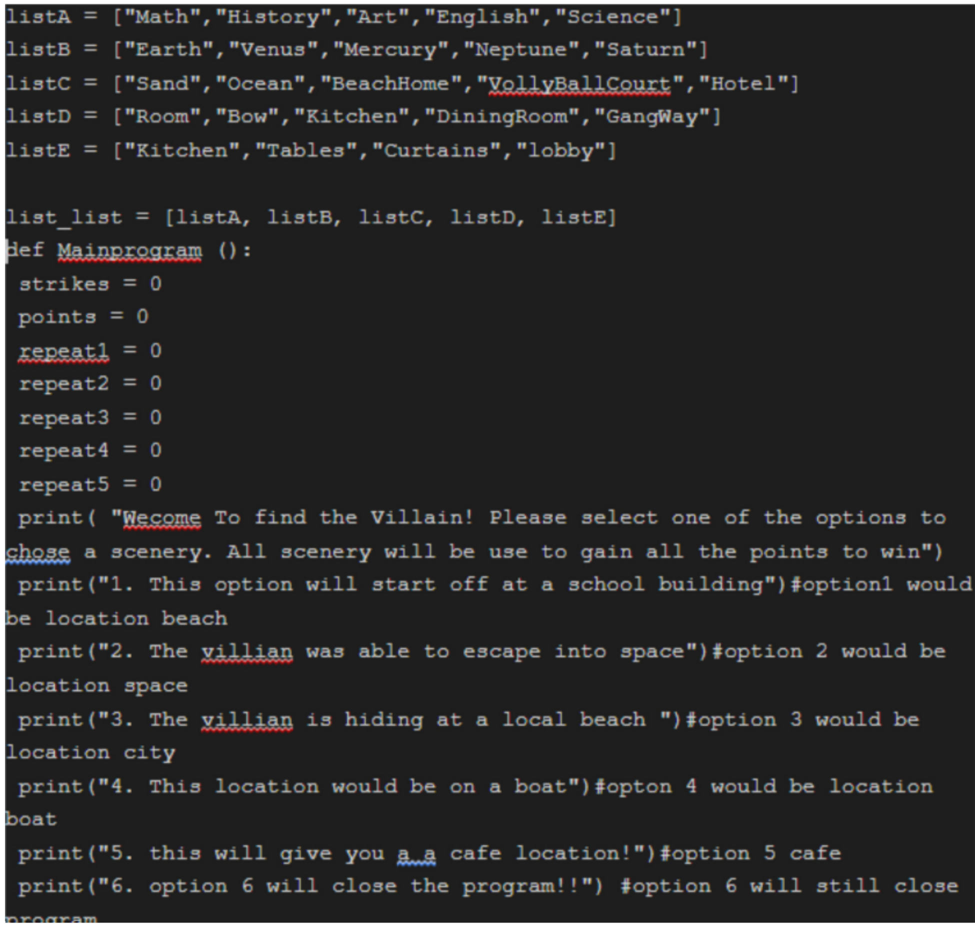
```
ask (What is the largest animal on Earth?) and wait
if <[Answer Options 1 ▾] contains (answer) ?> then
    say (Correct! On to the second round!) for (2) seconds
    add (1point) to [Answer Options 1 ▾]
else
    say (Sorry try again next time :() for (2) seconds
```

• Includes a program code segment that shows a list being used to manage complexity in the program.
• 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.

?

j

- Includes a program code segment that shows a list being used to manage complexity in the program.
- 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.



```
listA = ["Math","History","Art","English","Science"]
listB = ["Earth","Venus","Mercury","Neptune","Saturn"]
listC = ["Sand","Ocean","BeachHome","VollyBallCourt","Hotel"]
listD = ["Room","Bow","Kitchen","DiningRoom","GangWay"]
listE = ["Kitchen","Tables","Curtains","lobby"]

list_list = [listA, listB, listC, listD, listE]
def Mainprogram ():
 strikes = 0
 points = 0
 repeat1 = 0
 repeat2 = 0
 repeat3 = 0
 repeat4 = 0
 repeat5 = 0
 print( "Wecome To find the Villain! Please select one of the options to
chose a scenery. All scenery will be use to gain all the points to win")
 print("1. This option will start off at a school building")#option1 would
be location beach
 print("2. The villian was able to escape into space")#option 2 would be
location space
 print("3. The villian is hiding at a local beach ")#option 3 would be
location city
 print("4. This location would be on a boat")#opton 4 would be location
boat
 print("5. this will give you a a cafe location!")#option 5 cafe
 print("6. option 6 will close the program!!") #option 6 will still close
```

No list was identified in 3.b.iii

The list complexes the program due to all of the elements within the list would need to be used until gaining five points or missing and gaining three points into strikes. If a list was not used the issue would have to be that the elements would separately give you if else statements that would make you code extremely long and would be a tedious task. The list is what the options require for the program would work to its best potential, and without it would make the program extremely unpleasant to use and everything would be unorganized to an extent.

?