PLTW COMPUTER SCIENCE

Activity 1.2.4

Lists: Survey Says

goals

- Manage and manipulate lists
- Learn to use “best so far loops” and “aggregators”
- Modify an existing program to be applied to a new purpose
- Develop an app as part of a pair programming collaboration

description of app

Create an app that allows users to create lists and vote on items in the list. The app will then be modified and applied to a new purpose.

Essential Questions

1. Why are lists considered essential in computer science?
2. Why is sharing code and looking at many examples important to people writing programs?
3. How have you gotten better at collaborating with your partners when pair programming?

Essential Concepts

- Lists and Indexes
- Best So Far Loops
- Aggregators
- Algorithms, Variables, Arguments, Procedures, Operators, Data Types, Logic, Loops, and Strings

Resources

Shell124SurveySays app

Design Overview: Survey Says!

User Story

At the end of a calendar year or academic year, it is not uncommon for people to reflect on the top items for the year. You are going to develop an app that will allow students to vote on their favorite color. After you have a working app, you will modify the category (color) to suit your own interests, such as best movie, best band, favorite song, or favorite ice cream flavor.

App Overview
The focus of this app activity is on programming the lists, so a basic user interface has been designed for you. When you modify the app, modify the user interface, too. The end app will allow a user to select or enter their favorite item in a category.

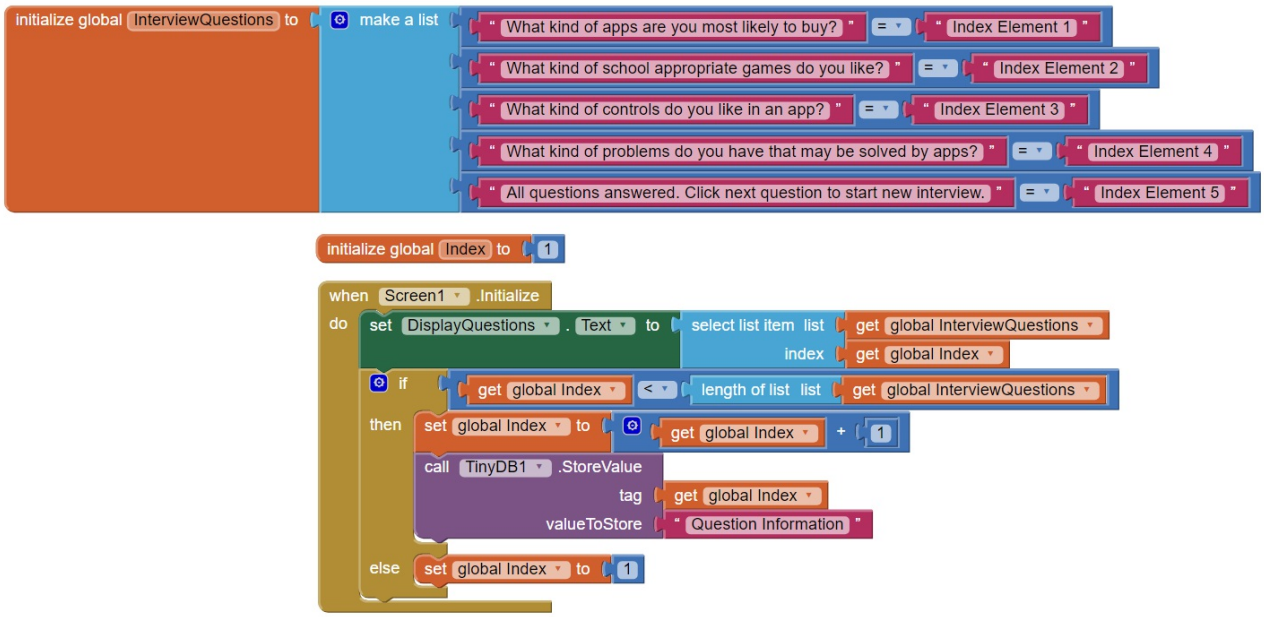
Initial Backlog Breakdown
The user needs:

- ☐ A list to make a custom color background when the screen initializes.
- ☐ A conditional to check whether a user-entered color choice is already on the list, and if not, then add the color. If it is on the list, then it will not add the color.
- ☐ Pick the user's favorite from a list picker.
- ☐ Confirm the user's vote after they pick their choice from the list picker.
- ☐ Replace a word in the list of choices if someone misspelled something.
- ☐ Run through the list and display the number of votes and top pick at the push of a button.
- ☐ Modify the app to rank something other than "favorite color".

Lists

Previous Lists

You have seen lists in other activities, but in this activity, you will manipulate the lists more than you have in the past. To review where you have seen lists before, look at the following images:

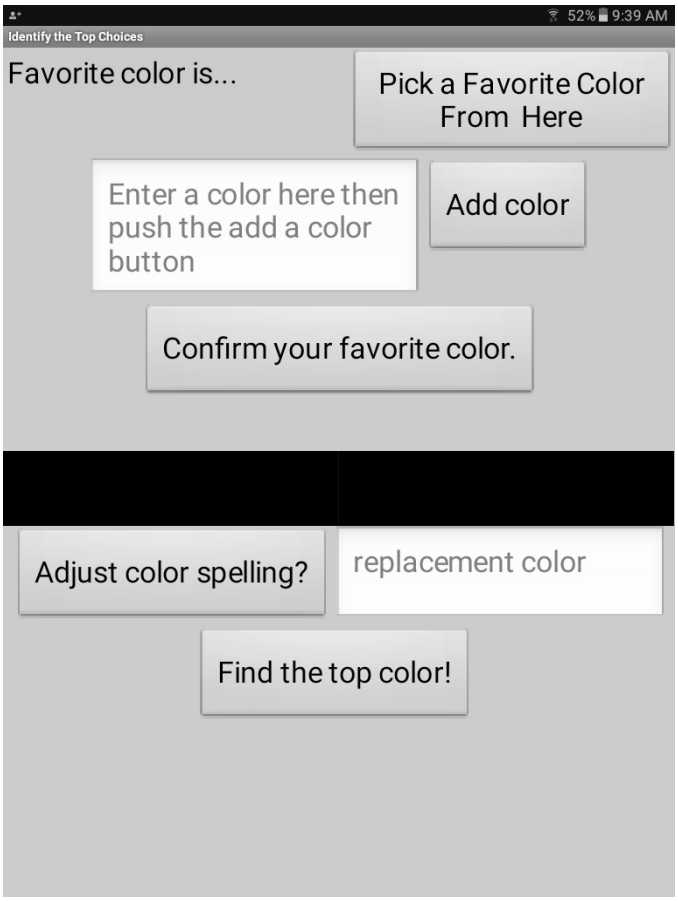


In the blocks above, you established a list of questions, used the index of the question to move through displaying each question in that list, and stored information based on the index number of specific questions. These index numbers are important for lists. They identify specific elements that allow values to be stored. Sometimes, the index element position has specific meaning, such as with custom color creation.

Download the App Shell

1. Navigate to and log in to MIT App Inventor.
2. Download the **Survey Says app**.

   The user interface will look similar to this:



3. Upload the *Shell124SurveySays.aia* file to your MIT App Inventor account.
4. Find your pair programming partner as directed by your teacher.

Creating a Custom Color on the User Interface When Initialized

5. Using the *screen1.Initialization* event handler, set the *screen1* background color block to the *make color* blocks from the built-in *Colors* drawer.

   In App Inventor, the first item in a list has an index of 1, the second has an index of 2, and so on. The index identifies the specific element and position in a list for the computer to access. (Other coding languages, such as *Python*®, start with the first index as 0.)

   Knowing the index number locations or spots helps programmers use and modify lists. The three index values for *make color* block each have a specific purpose to App Inventor. Each index element identifies a color, with index 1 as red, index 2 as green, and index 3 as blue. These three colors in different combinations make up the color of the light you see (or do not see) from computer monitors or tablet screens.

   > Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

6. Adjust the *set color* block numbers to display a custom background color.

You can use color lists to change the colors of the components in the app. You can also combine these with conditionals to have colors change based on what is happening in the app.

Important: In the *Designer* view, toggle the *Screen1* background color to get the screen to refresh with your new color without having to reset your connection to the tablet.

Setting Up the List

Look at the backlog of what the user needs. A central part of the app is taking and storing input from the user. The first app feature you are building is to add colors to vote for, so start by creating a list as a storage space for the user-specified color.

7. Set up a *global variable* called *ColorsList*.
8. Initialize global *ColorsList* to create empty list. The *create empty list* block is in the *List* drawer.

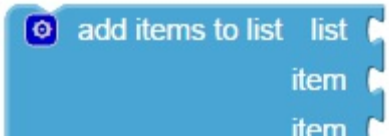
   Important: The *ColorPicker* component was already renamed in the provided app, but it is actually a *ListPicker* component. The component will display a list, allowing a user to pick one element from the list.

You set up a blank color list, but you need to direct the *ColorPicker* to that list when the screen initializes, so that it can show that list to the user when they click on it.

9. In the *Screen1.Initialize* event handler, add *Set ColorPicker.Elements* to *get global ColorsList*.

   After the screen loads, the *ColorPicker* will display all the elements in the *ColorsList* when a user selects the *ColorPicker* button.

10. Add your favorite color to the *ColorsList* that displays when *Screen1* initializes. Doing this also helps avoid errors associated with trying to use an empty list.
    - From the *Lists* drawer, drag out the *add items to list* block.
    - For the *list* input socket, connect a *get global ColorsList* block.
    - Using *strings*, add your favorite color to the *item* socket. Do not add any extra spaces around the color in the string.
    - Test your app.



11. To pick the color you added, select the **ColorPicker** component that shows in the user interface as a button named “Pick a Favorite Color From Here”.

Seeing Picked Elements

Picking a color might seem like a letdown, because nothing happens after you tap on a color. You need to program the *ColorPicker* to return the selected color element to a label after the color is picked.

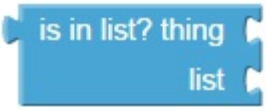
12. Use the *ColorPicker.AfterPicking* event handler to *set ShowColor.text* to concatenate a string to say, “You picked (*ColorPicker.Selection*)”.
13. Test, debug, and adjust.
14. If the app allows you to pick the color and display it to the user in the label, do an iteration save.
15. If you have not already, swap driver and navigator roles with your partner. If you were previously the navigator, talk through the code on the screen with the driver to make sure you both understand what the code is doing.

Users Adding Colors to the List

The app needs to allow for other colors to be added to the list, because a list of one color does not allow for voting. You want other people to be able to generate and add to the list, so that the list is not limited to just what you, the programmer, think are the best colors. You also don’t want to have multiple entries of the same color.

To prevent repeat colors, use a conditional statement to check whether what the user enters matches an item already in the list. Follow the steps below. Compare your block programming to the image at the end of this section.

16. Add user-specified colors. Set up a conditional to check input colors.
    - Add the *AddColorButton.Click* event handler to the viewer.
    - Inside the *AddColorButton.Click* event handler, add an *if then else* conditional block.
    - From the *Lists* drawer, drag out the *is in list? thing*/*list* block to connect to the *If* part of the conditional.

    - Add an *InputColor.Text* to the *thing* socket of the *is in list?* block. This socket will check the *InputColor.Text* against items in the list to see whether the input text matches any text in the list.
    - Add a *get global ColorsList* block to the *list* socket, so that it will check whether the *thing* (*InputColor.Text*) is already part of the *ColorsList*.
17. Set up the rest of the conditional. *If* the color is in the list:
    - *Then*, display a string message in the *ShowColor.Text* label to let the user know to select it from the list.
    - *Else*, use the add items to list block to get the list and add the color to the list. Also, include *ShowColor.Text* to let the user know it was added.

What If a User Enters Something Other Than a Color?

Users can enter text that is not a color name. But you can add conditional statements that display a message to users to try again to enter a color, to prevent people from entering things like numbers or blank strings. Use previous activities and the blocks in the *Logic* and *Math* drawers to help with this.

18. To display a message that says you cannot enter numbers when the user enters integers, expand the *if is in list?* conditional statement with an *else if* block using the *mutator* box.
19. To prevent the user from entering a blank string, further expand the conditional statement with an *else if* block using the *mutator* box.
20. After the conditional, add *Set InputColor.Text* to a blank string, so the next user does not see the color the previous user entered.

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

21. Test, debug, and adjust: If a user enters a color that is already in the list, does it still add the color?

Changing Strings

What if you input the color in all capital letters? If you change the capitals in the same word, the program reads each entry as a different word. Each of the examples below are considered to be different, although we understand them as the same concept.

PLTW
PLTw
PLtw
Pltw
pltw

Unless a word has the exact same capital and lowercase letters, the program interprets the entry as a different word. This is because the capital A is represented differently than the lowercase a in the program at a lower level of abstraction. Although they are the same letter to us at a higher level of abstraction, “a” is different from “A”.

One way to avoid the computer interpreting two words as different when they use different capital or lowercase letters is to convert all letters to one or the other.

22. In the *text* drawer, find the *upcase* block. To make every string with a color uppercase letters, add the *upcase* before each block that evaluates the *InputColor.Text* or that adds colors to the list.
23. Test, debug, and adjust. Reset your connection to start with a fresh list and see whether your app’s conditional is working properly. Test adding each of the following:
    - ☐ 42. You should not be able to add it and should get a message that says no numbers.
    - ☐ A blank string. You should not be able to add it and should get a message that says to add a color.
    - ☐ A new color. It will accept the color.
    - ☐ The same new color again. It will not add the color and will tell you the color is already in the list.

24. If the app is functioning properly, save the app and swap driver and navigator roles with your partner. If you were previously the navigator, talk through the code that is on the screen with the driver to make sure you both understand what the code is doing.

Increment Voting Across Multiple Lists

Users need to be able to cast their vote and have that vote counted, so that the

“favorite color" picked by students in your class can be identified. To do this, it is necessary to start a second list, *VoteList*, that will store the vote totals for each element in the first list.

Every time a color is picked, 1 additional vote should be added to the *VoteList* in the same index spot as the selected color. The colors and the votes need to have the same index spot; otherwise, there is no way to know how many votes each color got.

Matching Vote and Color Indexes When New Color Is Added

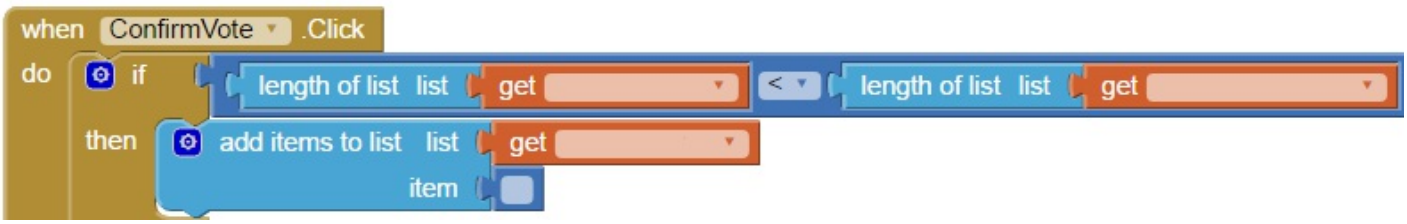
ColorList		VoteList	
ColorList Index	ColorList Element	VoteList Index	VoteList Element
1	Blue	1	7
2	Red	2	3
3	Green	3	1

For each *ColorList* element, there should be a spot in the *VoteList* to store the votes for that color. As a new color is added to the color list, a new vote with a value of 1 must be added to the vote list. By adding them both at the same time, they will have the same index number, but in different lists. In the example above, votes for Blue and Red *ColorList* elements increase the values of the corresponding *VoteList* elements. When someone adds Green for the first time, a value of 1 is populated in the *VoteList* element automatically.

One way to decide whether a new *VoteList* index is needed is to create a conditional statement that checks whether the *VoteList* is shorter than the *ColorsList.*

- When the user clicks the **ConfirmVote** button, a color is added to the *ColorsList*.
- If the *VoteList* is shorter than the *ColorsList*, then a color does not have a place to store the votes. The program needs to add another index element location to *VoteList* to store the number of votes the color gets.
- As colors are added to the list by vote, the *VoteList* element for that color should be set to 1 to show that the new item picked now has 1 vote.
- After this, the lists will be the same length, and every color will have a designated spot in the vote list to store the number of votes it has.

The example below illustrates how to add a vote to the vote list each time a new color is added to the color list. The names of the variables are not shown.



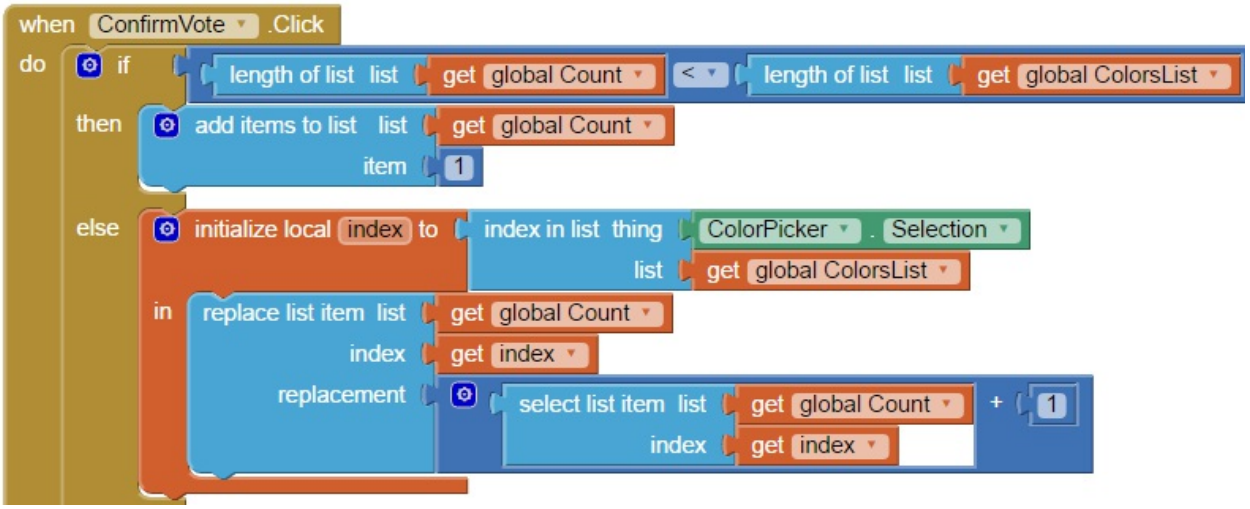
Adding a Vote (Incrementing) for a Color Already in the List

- The *else* part of the conditional handles when the two lists are the same length.
- When the lists are the same length, each color has a vote of 1.
- Each time the color is picked again, the *VoteList* needs to increment the number of votes at the same index spot.

  Example: If a student places a second vote for the color Red, the value at index spot 2 in the *VoteList* must be incremented by 1.

ColorList Index	ColorList Element	ColorList Index	VoteList Element
1	Blue	1	1
2	**Red**	**2**	**1 (Increment +1)**
3	Green	3	1

- Although the color selected and the number of votes are in separate lists, they have the same index number.
- A *local variable* can be used to track the index number. In this example, it is named “index”.
- After the index location has been set based on the color that was selected from the list, the program will replace the current vote for that color with the new value, which has been incremented by 1.
- The example below illustrates how to:
  - Initialize a new local variable called “index” and set the variable to the index number of the color in the *ColorList*.
  - Replace the value in the *VoteList* at the same index number as the *ColorList.*
  - Set the index number in the *VoteList* to the current value plus 1 (increment).



Set Up the Vote Counting System

Before you code any of the functionality described above, you need to initialize both lists.

25. Initialize a *global VoteList* variable to an empty list just like you did in the code for the *ColorList*.
26. In the *Screen1.Initalize* event handler, add another *add items to list* block.
27. Connect the *get global Count variable* to the list socket and set the item socket to 0.

> Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

Setting Up the Conditional to Vote

28. Drag out the *ConfirmVote.click* event handler.
29. Add an *if then else* conditional to the event handler.
30. Set up the *if then* conditional:
    - *If length of list,* the *global VoteList,* is *less than,* the *length of the list,* the *global ColorsList*
    - *Then* add items (1) to *global VoteList*

Data Types

As described in the overview for this activity, *the length of list* blocks in your program will check to see whether the lists are the same length. If the lists are not the same length (a new color was added to the color list), a new *VoteList* item with 1 vote will be added for the new color.

Using the number block with a “1” in it instead of a string with "one" is important. The number block tells the list that an integer will be stored there, so the program can later increment that integer. If it was a string of "one", then it would not be able to have “1” added to it later.

31. Set up the *ELSE*. The *Else* portion of the conditional will use a local variable to identify and store the index of the item selected from the *ColorPicker*.

    Important: Inside the local variable, using the index value, the local variable will replace that same index value in the *global VoteList* by taking what is already there and adding 1.

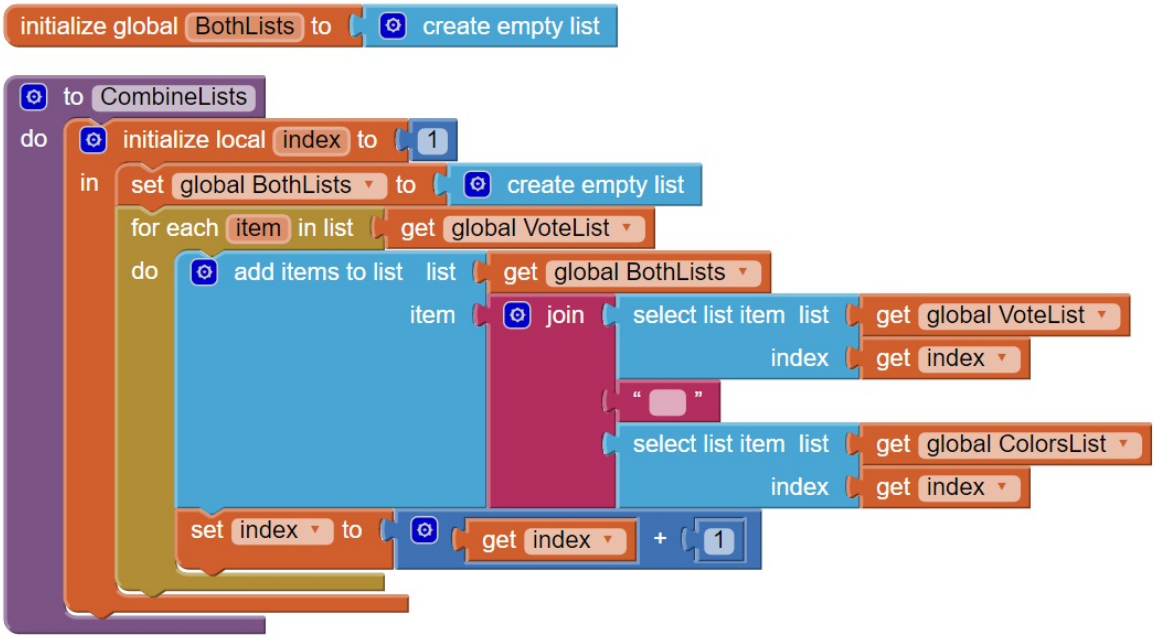
> Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

32. Before you continue, do a **Save As** with a new name to show the new iteration, and switch roles as navigator and driver.

Combining Both Lists to Display to Users

Now you have two separate lists. Each list is connected to the other by what is defined in the same index spots (Example: Color Blue and votes for Blue have the same index spots in different lists . However, displaying two separate lists to users is cumbersome, when they only want to know the vote counts. To help fix this, you will create a third list to combine the two lists and display *BothLists* in the *ListView1* component.

33. Define a procedure to *CombineLists*. While you set up the procedure, talk through what the procedure is doing to combine the two lists.
34. Review this procedural example to combine lists:
    - Initialize a new *global variable BothLists*.
    - Initialize a local index to 1.
    - Set a new list *BothLists* to an *empty list*.
    - For each item in the vote list:
        - Add to the *BothLists* list.
        - Concatenate the values for the colors and the number of votes with the same index number.
        - Increment the index number so it adds all combined colors and votes.



35. Build the procedure as described above.

36. Call the *CombineLists* procedure at the end of the *ConfirmVote.Click* event handler.

37. Update the *ListView1* in the *ConfirmVote.Click* event handler to *get global BothLists*.

38. Test, debug, and adjust.

    - ☐ Limit the colors added to avoid repetition, blank strings, and numbers.
    - ☐ When a color is picked and confirmed, a number and color show up in the list viewer.
    - ☐ As you confirm a color, the count increments by 1.

Adding Features - Replacing Misspelled Words

If a user enters a color that is not spelled correctly, and another user notices the misspelled word in the list, the second user should be able to select that word in the *ListView* and change the spelling. The user would select the misspelled word and then replace the string saved in that index location with a new string that has the color correctly spelled.

39. Drag out the *ReplaceWord.Click* event handler.
    - Add to it the *replace list* item from the *Lists* block drawer.
    - Connect the *get global ColorsList* variable.
    - Connect the *ListView1.SelectionIndex* to the *index* socket of the *replace list* item block.
    - Connect an *upcase* block between the *replacement* socket and a *ReplacementWord.Text* block.
    - Call the *CombineLists* procedure.
    - Set *ListView1.Elements* to get *Global BothLists*.
    - Set *ReplacementWord.Text* to a blank string.

> Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

40. Test, debug, and adjust.
    - ☐ Add a color that is misspelled. Because there is no spellcheck feature, the app will allow you to add the color.
    - ☐ Select the color in the *ListView*, and change the word to the correct spelling. This may also allow a user to remove words that are not colors, by replacing them with a new color name.

Best-So-Far Loop

Finding the Winner

To help make it easier to find the winner, you are going to program a procedure to *FindTop* entry whenever a new vote is placed. The procedure will go through the list, identify, and display the top color and the number of votes it has in the *ShowTop* label, already in the design of the app.

A **Best-So-Far Loop** will go through the list, always storing the item that is the best—in this case—has the highest number of votes. A *best-so-far* loop has a variable that stores the top value while it moves through the list comparing the rest of the items to that top value. If the loop finds an element that is better (higher), then it replaces that variable value with the new value until it reaches the end of the list.

41. Set up a procedure to *FindTop*:
    - Inside the procedure, set up a *local* variable block.
    - Add a *Top* variable initialized to 0. When you set up a *Best-So-Far-Loop*, make sure that

the number you start with can be beaten. For example, starting with 0 guarantees that the first number with at least 1 vote will be higher.
    - Add a second variable, *index*, initialized to 1. This way, you can direct the program to look at the first index spot and increment the index value being looked at, so the *Best-So-Far-Loop* will look at each index spot to compare with the record holder in the *Top Variable*.
    - Inside the local variable, add the control block, *for each item in list*.
    - In the socket of the *For Loop* list control block, get global *VoteList*.
    - Add an *IF THEN* conditional inside the *For Loop*.
42. Set up the *IF* part of the conditional. Compare whether the item variable (from the *for* control block, which identifies each element in the list one by one) is greater than the top local variable. Whatever is greater than 1 will replace the stored, initialized variable of 0. Then, the next number that is higher will replace the current highest, and so on until the end of the list.
43. Set up the *THEN* part of the conditional:
    - If the current element is greater than the last record holder, the *then* part will set *top local* variable to *get item* variable.
    - The *index* local variable also needs to be set to *index* of the item from the *for* loop, to know which index spot has the greatest number.
44. Show the results of the loop to the user. After the *IF THEN* conditional, and still inside the local loop, add a *set ShowTop.Text* to concatenate the *get Top* local variable with a *select item from list* that pulls the *Index* local variable from the *get global ColorsList*.

> Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

45. Call the procedure in any event handler that you want to update the top item display. Consider, which event handler would be called on the most?
46. Test, debug, and adjust. At this point, when you test the app, you should be able to do and see:
    - ☐ To avoid repetition, blank strings, and numbers, the colors added are limited.
    - ☐ When a color is picked and confirmed, a number and color show up in the *ListViewer.*
    - ☐ As you confirm a color, the count should increment by 1.
    - ☐ When you click the event handler that the *call FindTop* procedure is in, the highest count and color show.
    - ☐ After more votes are added to a different color, the new color with the most votes is automatically updated and displayed.
47. If the app is functioning in this way, do an iteration save, and swap driver and navigator roles with your partner. If you were previously the navigator, talk through the code that is on the screen with the driver to make sure you both understand what the code is doing.

Total Count Aggregator

Sometimes it is helpful to know how many votes there are in total—for example, to make sure that results are valid by not having more votes than people in the class nor a unanimous class color with only 1 vote. Setting up an **aggregator variable**, a variable that will add together all the votes as they currently are, will allow the user to check the validity of the results.

48. To set up an Aggregator procedure, use the image and steps below, and then call that procedure in the appropriate event handlers.
    - Initialize a *global AggregatorValue*.
    - Set up an Aggregator procedure with blocks that will do steps and not just return a result.
    - Because it always needs to reset to 0 to start the count again, set the *global aggregator* to 0. Otherwise, you are adding to what you added previously.
    - Set up a *for* loop that will get the *VoteList*.
    - In the *VoteList* loop, set *global AggregatorValue* to add the item value to the *global aggregator*.
    - Use concatenation to create a message telling the user what the total vote count is. Set the *AggregatorLabel* to concatenate the string and explain to the user what the *global AggregatorValue* is.

> Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

49. Call the procedure in any event handler that you want to update the top item display. Consider, which event handler would be called on the most?

> Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

50. Test, adjust, and debug.
    - ☐ All previous app features still work.
    - ☐ When a color is selected, it shows the total votes and the most picked color.
51. Share your app with your teacher.
52. Modify the program so that you can collect information on a topic other than colors. What parts do you need to update to give the user a different experience? Does it need to be all the code and user interface, or just one of them?

PLTW DEVELOPER'S JOURNAL

1. Look at your code and write a paragraph, using complete sentences, explaining what the

AddColorButton or *ConfirmVote* event handler is doing and how the code supports that.
2. Looking at your code, explain the importance of knowing and using the index values in a list.
3. Describe how the *For Loop* and the *best-so-far loops* work together through the data by detailing what the start value is, what it looks at, and what it does with each iteration.
4. Explain why procedures were helpful in this app.

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

Conclusion

1. Describe the difference between incrementing a count and an aggregator.

2. How did you interpret and respond to the **essential questions**? Capture your thoughts for future conversations.