PLTW COMPUTER SCIENCE

Activity 1.2.1

Problem Solving: Interview Database

goals

- Learn a problem solving process
- Practice communication skills to interview end users of an app
- Generate ideas for possible problem solving in the future
- Learn how databases and lists allow data to be collected, persist, and be retrieved
- Develop an app as part of a pair programming collaboration

description of app

Create an app that will generate a list of questions to be asked and capture responses from someone who is surveyed. These interview questions will help identify potential problems to solve.

Essential Questions

1. What problems are really worth the effort to try to solve?
2. What does it mean for data to persist?
3. What are the similarities and differences between developing an app as creative expression and trying to solve a problem with an app?

Essential Concepts

- Databases, Lists, and Indexes
- Problem Solving
- Professionalism and Engaging Stakeholders in User Centered Design
- Pair Programming

- Algorithms, Variables, Arguments, Procedures, Operators, Data Types, Logic, Loops, and Strings

Resources

Interview Database app icon

Design Overview: Interview Database

App Overview
In Project 1.1.6, you had creative freedom and minimal guidance to create an app. Future problems will have even less guidance; you will get to choose what you want to create, as long as you can convince your teacher the app will have value to others (and that it doesn't already exist). To gather information and ideas on future app development, you will begin the process of conducting interviews.

In this activity, the app you create will be a powerful tool for getting started in problem solving. You will create an Interview Database app to help record interview information that you will be able to use for later course problems.

Initial Backlog Breakdown
The user needs:

- ☐ A list of questions to ask in an interview to find out what kind of app might benefit a person.
- ☐ A list of questions to ask a user to get feedback about an app. What do they like? What do they dislike?
- ☐ A system that allows a user to record interview information, such as the date, interviewee's name, and answers to specific questions.
- ☐ A system where as soon as the app starts, the first question is displayed, so that an interviewee may independently complete the interview or an interviewer can use the questions as a script for discussion points.

  ☐ A feature that allows a person to speak and the app automatically puts that speech into a written form in the app. You might need a button to start the feature.

- ☐ As an answer is recorded, the app automatically displays the next question and notifies the user when all questions have been asked.

- ☐ After the last question notification appears, the app refreshes to the first question, so the user does not need to close and reopen the app between each interview.
- ☐ A way to store interview information between each app use, so that a user can pull up that information whenever they need it.
- ☐ A way to view and clear the stored information.

Development Process

Whether you are creating for fun, creating with entrepreneurial intent, or trying to solve a problem, the process you will use is generally the same:

1. Part A: Find an Idea Worth Pursuing
2. Part B: Document Your Development Milestones
3. Part C: Prepare, Investigate, and Plan
4. Part D: Design, Create, and Test
5. Part E: Evaluate and Reflect
6. Part F: Present

The Interview Database app is designed to help you with the first part in the process, Part A: Find an Idea Worth Pursuing.

Using Questions to Uncover Solvable Problems

The Interview Database app will help you get started by providing an outline and a space to record answers to questions that help uncover problems you might want to pursue. There are many ways to find a problem that needs solving.

The more interviews you do, the more ideas you will have for a problem you want to solve with an original app. Having a way to record the ongoing interviews and potential ideas is important, therefore, you will create an app that can store information on an Android™ device so you can close and reopen the app without losing data. After you build the interview app, you will use the app to practice recording interview information.

What Is Reasonable to Ask Someone?

You should design your app so that you can capture information about the people being interviewed, such as names, dates, and contact information. However, it is important to remember that not everyone wants to share their personal information. You must communicate professionally and explain why you are documenting the responses and what you intend to do with the information you collect.

1. Form pairs as directed by your teacher.

Each time you pair program or collaborate, you should take a moment to introduce yourself to your new partner. Make clear how you will handle file management and make sure you both understand the roles and rules of pair programming. Each new partnership is a chance to get better at collaboration, which is a critical skill for successful people.

2. Navigate to MIT App Inventor and log in.

PLTW DEVELOPER'S JOURNAL Turn to your partner and discuss what information should be off limits for this app and why. What are some of the reasons you would want to be careful about what you capture from the people you interview? Summarize some guidelines for performing interviews that you and your partner discuss.

Connection to Computer Science

- User-defined criteria are always at the heart of an iterative development process.
- A database is an essential concept that is used often in programs.
- The Interview App will give you a head start on later problems by showing you strategies to identify problems that will have value to others.
- The activity provides opportunities for your team to model professional communication as you interview real people with your app, capturing the data and responses.

Find an Idea Worth Pursuing

Consider the following categories and questions that might help trigger ideas:

- Personal Experiences or Interests
    - What do you like to do?
- School Culture
    - What do you see every day here at school that could be done better?
- Education
    - How can you make learning more fun, easier to understand, or just better?
    - What can help students learn for future years?
- Technical Problems
    - What do people complain about a lot?
- Health, Wellness, and Safety Issues
    - What causes injuries or health issues?
- Community Service and Volunteering
    - What is a problem in your community?
        - Pet populations?
        - Nutrition?
- Environment
    - What environmental challenges exist where you live?

    - Water?
    - Pollution?
    - Recycling?

- Emergency Preparedness
  - What should people do in emergencies?
- Legal Issues
  - What problems cause people to take each other to court?
- Global Challenges
  - What are global challenges no one has solved that you care about?

PLTW DEVELOPER'S JOURNAL

Begin to brainstorm which category you are the most interested in. After you have picked a category, brainstorm about people who might have more knowledge about that problem and what kind of app you might develop to make a solution for that problem.

Brainstorm at least five questions you can ask these experts to gain insight into the problem and into your app idea. You do not have to stay with whatever category you select now. In fact, the sooner you start doing interviews, the more time you have to change if you discover a different problem you want to help solve.

Databases for Persistent Data

The Interview Database app requires use of a local database, a storage space for the app on the user's device. The storage space will allow the app to store interview information, such as the interviewee's name, the date of the interview, and their responses to the questions you ask. You will program the database to store, display, and reset information.

Set Up the Designer View

Review the components you need for this app and set up a *Designer* view and user interface that is easy for a user to navigate and use. Consider what layout arrangements and component properties will help a user navigate your app with ease, such as hint options for *textboxes* and user-viewable text descriptions.

Note: Use the

beside each component in the drawer to learn more about the components before you add them.

3. From the *Storage* drawer, drag and add the *TinyDB* component to the Viewer*.* This creates a small storage place for the app on the phone that it can record and retrieve information from. To learn more about the component, click the *?* beside the *TinyDB* in the *Storage* drawer.
4. From the *Media* drawer, drag and add the *SpeechRecognizer* component to allow a user to speak and have their answers recorded.
5. From the *User Interface* drawer, drag and add the following components to the *viewer*:
    - *DatePicker* to allow the user to select the specific date the interview was done
    - *ListView* to display items from a list in one place
    - *Notifier* to display a pop-up window to the user
    - *TextBox* to record the *IntervieweesName*
    - *TextBox* to record the *IntervieweesAnswers*
    - *Button* to record what a person says in text form, labeled *Record*
    - *Button* to record the input and to *GetNextQuestion*
    - *Button* to *RetrieveDBList*
    - *Button* to clear the *ResetDBList*
    - *Label* to *DisplayQuestions*
    - *Label* to test the *DatePickerTester*

PLTW DEVELOPER'S JOURNAL While you add these components, consider making a list to remind you about what certain components can offer to the features of an app. Soon, you will be thinking of the features you want in your own app and will need to find the components to make those features happen.

Making Your Apps Unique

As part of this process, you will want to get feedback from others on what they like (or do not like) about your design and how your app works. Your app design may look different than the example provided. What is important is that a new user can navigate it easily.

You will want the app to function in two different ways:

- An interviewer may read the questions and record information that they discuss with the interviewee.
- A user may use the app to scroll through the questions and record the answers themselves and pass the device on.

6. Work with your partner to determine how you want your app to look and function.
7. Build your own custom layout that meets the look you and your partner want.

   Below are examples of what the app could look like. Make yours look better.

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

8. Swap tablets with another pair to receive their feedback as to whether they can clearly tell how to use your app.
9. Make adjustments so that your app is user friendly to others beyond your partner and you.

Now that your user interface is complete, think about the types of information you will want to store from the interviews.

Setting Up the Variables and Lists

You need two local variables:

- The question number (called an index in a list)
- The question text

List components store questions in a set order, with each question having its own unique identifying number called an **index**.

Note: In a future activity, we will discuss lists in more detail; for now, the ordering of the list is what matters.

Example Question List

1. What do you like to do?
2. What are your hobbies?
3. What do you see every day at school that could be done better?
4. What do people complain about a lot?
5. What causes injuries or health issues?
6. What is a problem in your community? For example, pet populations? nutrition?
7. What challenges are there where you live?
8. What problems have caused people to take each other to court?
9. What should people do in emergencies?
10. What are global challenges no one has solved but that you care about?

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

10. Create a *global variable* that will hold the **index** value, the numerical value for the list item.
11. Initialize the index variable to 1, so it will access the first item in the list. In MIT App Inventor, the first item in a list is index 1. (However, when you get into less abstracted programming languages like *Python*®, the first index is 0.)
12. Create a *global variable* to hold the interview questions.
    - From the *Lists* drawer in the *Blocks* view, drag out the *make a list* block to connect to the *global variable* that will store the interview questions.
    - To add additional item sockets to the *make a list* block, use the blue *mutator* button on the block. You will need at least five sockets in the *make a list* block.
    - Connect a *string* to each open socket of the *make a list* block. In the last *string* block, add a statement to let the user know that all the questions have been asked and that clicking the button again will restart the interview.
13. In the remaining strings, ask the user what they think should be changed about the app they are using right now. Do they like the look of the user interface? Is it easy to navigate? These questions should be the same ones you ask when you iterate on future apps.
14. Add a final string that shows all the questions have been asked.

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

Initializing the Screen

Sometimes you want events to occur as soon as the app starts without waiting for the user. For these situations, use the *when Screen.Initialize* block to program what you want to happen. In this app, you will use the *Screen.Initialize* block to display the first question to the user as soon as the app starts.

15. Open the *Screen1* blocks drawer and find the *Screen.Initialize* event handler block to drag into the view.
16. Inside the event handler:
    - Drag out and add the set *DisplayQuestion.text* label block.
    - From the *Lists* drawer, select and drag out the *select list item* block.
    - Set the *list* socket to get *global InterviewQuestions* variable, letting the program know what list to look at.
    - Set the *index* socket to the *get global Index variable*, so the index knows what question from the list to display.

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

17. Test your app. When the app starts, you should see the first question from your list.

Important: As you go through adding and testing individual parts, you may need to periodically reset the connection between MIT App Inventor and the Android™ device to test new app features. The next features address how to progress through all the questions in your list.

Using the Index Variable to Loop Through Questions

Before you start actual interviews, set up the list so that all questions will be asked and recorded, and after the last response is recorded, the user is taken back to the first question.

When the user taps the button that r**ecords the answer and then get's the next question**, the event handler records the following in the tiny database (*TinyDB*):

- Name of the person interviewed
- Date (via the date picker)
- User response to the question

The app will also display the next question and clear only the *IntervieweeAnswers* textbox.

In this way, a user can go through each question recording the information to the database. Once the user has gone through all the questions, the loop needs to restart to display the first question again. A control structure in this loop will determine whether the app should progress to the next question or restart the loop because all the questions were displayed.

18. Select and drag out the *GetNextQuestion.Click* event handler.
19. Inside the event handler, add an *If, Then, Else,* control structure.
    - *If* the *global index* is less than the length of the *global InterviewQuestions* variable
    - *Then* (Not all the questions have been asked)
        - *set IntervieweeAnswers.text to* blank
        - *set global index to* add 1 to increment a counted loop
    - *Else* (All the questions have been asked)
        - *set IntervieweeAnswers text* to blank
        - *set IntervieweeName text to* blank
        - *set global index* to 1

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

Each time the program runs through the conditional, the index number changes for the question that should display. However, changing the index does not change what is displayed to the user. After the conditional has completed, regardless of whether the *then* or *else* part is executed, the user needs to see the next question based on the index number.

20. Add a *set DisplayQuestions.Text* at the end, outside the control structure. You may copy and paste the blocks from inside the *initialize.Screen* event handler or recreate it.
21. Set the *DisplayQuestions.text* to select the *global InterviewQuestions* list and the global index value for the index. Add these blocks after the conditional statement, but inside the *GetNextQuestion.Click* event handler.

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

22. Test, debug, and adjust.
    - ☐ You see the first question from your list when the app starts.
    - ☐ Click the **GetNextQuestion** button to cycle through the questions.
    - ☐ Click the button for the *DatePicker* and pick the date (nothing happens though—much like you may type in the user *textboxes*, but nothing happens).

You now have a way to record the date a question is answered as well as the response to each question.

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

Concatenating into One String for the DatePickerTest

The user needs to be able to select the date of the interview. You are going to use the date to help store to the database the interviewee's name, question index number, and their response to a question. This information will be the identifier for each question, so information retrieved from the database has more context about the interview conducted.

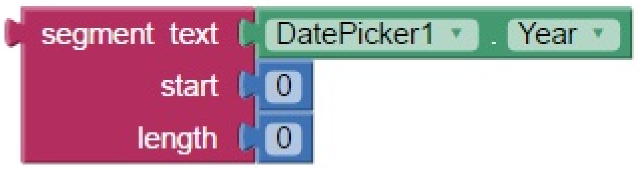
To help you understand the *date picker* component, it is important to know how it provides information about the date to the rest of the app.

23. Use the *DatePickerTest* to practice seeing what the string will look like before you store information to the database.
24. To see the date that the date picker provides, set the output of the date picker component to the *DatePickerTest* label. Once you know what that looks like, adjust the string that the component returns. For example, make the date format look like the ones below (month-day-year):
    - 7-4-17
    - 11-23-17

25. In the *Screen1* blocks drawer, select the **DatePicker** drawer and drag out the *DatePicker1.AfterDateSet* event handler.
26. Add a set *DatePickerTest.Text* component inside the *DatePicker.AfterDateSet* event handler.
27. Use blocks to concatenate the month, day, and year from the *DatePicker* drawer.

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

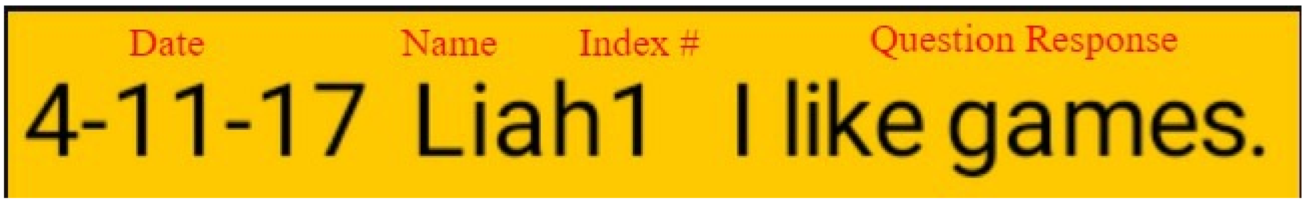
28. Test and learn the abstracted code in your app. Select a date and see how the *DatePicker* formats the information. It is not the easiest thing to read. Make it easier to read:
    - Concatenate additional strings and the *segment text* block from the *Text* drawer, to format what the user sees.
    - Use the *segment text* block to divide the text into just the part you want to keep.
29. Adjust the *start* and *length* values to pull out only the part of the string you need.



The end display to the user should look like “11-23-17”.

When your text has the same formatting as above, you are ready to use the *DatePicker* information in other parts of the app.

30. Modify your code in the *DatePicker.AfterDateSet* event handler to make your output easy to identify, such as the text in black below:



Storing Known Output in a Database

The app is set up to loop through the questions and record what the user inputs. However, with each question, the previous one is lost as the label is reset to the new question. To keep data persistent, you are going to program to store the information in the database. Every time a user clicks to get the next question, the previous answer will be stored with the concatenated text that you practiced in the *DatePicker* text label.

31. From the *TinyDB1 drawer,* pull out the *TinyDB.StoreValue* procedure.
32. Add the procedure block to the *then* part of your conditional that gets the next question.

The tag in the procedure will mark the value to be stored with a specific identifier so that the database can pull out just that one record. The tag helps identify different pieces of information. In this app, the person's name and the question number identify the date of the interview and their response to a particular question.

33. For the *tag* socket, concatenate the interviewee's name from the input text and the index value. That way, you know who answered which question.
34. For the *value* socket, copy the previous concatenation that showed the date, name, index, and question response from the set *DatePickerTest.Text* to plug into the value socket. Now, that whole string will be saved to the database, exactly as you saw it in the *DatePickerTest* label.

    Important: Sometimes in programming, you will add ways to check what the program is doing as you develop it, but once the final component is working, you can take out the individual testing parts. In this case, being able to see the whole date as reported by the *DatePicker* means you can delete the event handler.

35. Because you no longer need it to test the strings you will store, you may delete or hide the *DatePicker.AfterDateSet* event handler.

36. Test, debug, and adjust:

    - ☐ You see the first question from your list when the app starts.
    - ☐ Cycle through the questions when you click the **GetNextQuestion** button.
    - ☐ Store the date, name, and question response to the database as you cycle through the questions.

      Note: You *hope* they are stored as they disappear, but it would be nice to have a way to check the database. The information may be getting stored, but you have no way to check that function until you can retrieve the information from the database.

ListView Setup

The *ListView* component is designed to show entries from a database by using the tag to retrieve information. You are going to use the *RetrieveDBList* button to run through a loop that will pull all the items from the database and display them in the list.

Important: As a programmer, you now know the program populates each item on the list one at a time as the loop executes. To a user, it happens so quickly that it looks instantaneous.

37. Drag out the *RetrieveDBList.Click* event handler to the *Blocks* view.
38. Inside that event handler, set up a *local variable* block that initializes three local variables:
    - *AllTags* – A variable for each piece of information previously stored with the app;

initialized to call *TinyDB.GetTags*.
  - *StoreList* – A variable to store what the list gets; initialized to an empty list. (As the tags are retrieved from the database, it puts them into a single list, much like the interview questions.)
  - *Count* – A variable to count and track what list element to get next (incrementing the item being retrieved from the database), initialized to 0.

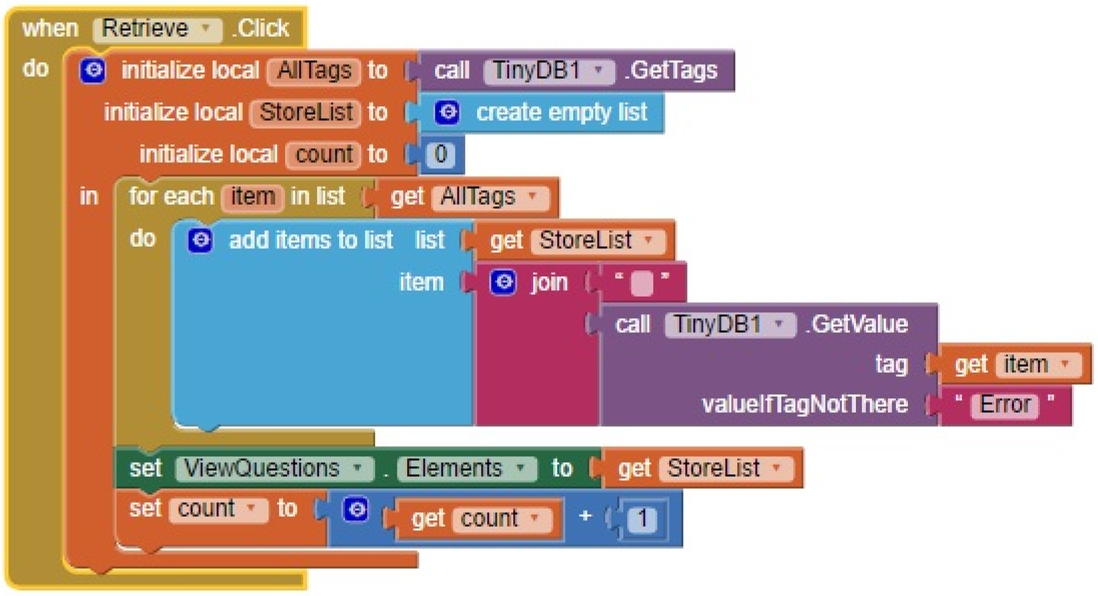
> Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

39. Inside the local variable block, you will use a *for each item in list* loop from the *Control* drawer.
40. Set the *list* socket value to be equal to the get *AllTags* local variable.

This loop will look at each individual tag (piece of stored data) and then complete a specific action. The action is that the retrieved information is put into a list that the user can see in the list viewer.

Important: Inside the *for each* loop, the loop needs to look at each tag and then add it to the *StoreList*, so that the *StoreList* data may then be shown in the *ListViewer*.

41. Duplicate these blocks to retrieve information from a database:



```
when Retrieve .Click
do  initialize local AllTags to  call TinyDB1 .GetTags
    initialize local StoreList to  create empty list
    initialize local count to  0
    in  for each item in list  get AllTags
        do  add items to list  list  get StoreList
                               item  join  " "
                                           call TinyDB1 .GetValue
                                                tag  get item
                                                valueIfTagNotThere  " Error "
        set ViewQuestions . Elements to  get StoreList
        set count to  get count + 1
```

Important: You can copy the basic structure of this event handler into the backpack and then into other apps to work the same way. To make these blocks work in a different app, you would just need to change the list picker and *TinyDB* name to match what is in the other app.

42. Talk with your partner about what is happening in the *RetrieveDBList* event handler.
43. Test, debug, and adjust:
  - ☐ **Tap** the **button that enters the the date of the interview** and pick the date (but

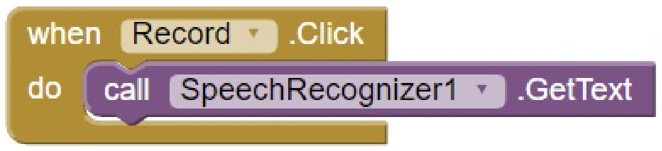
nothing happens yet).

- ☐ You see the first question from your list when the app starts.
- ☐ **Tap** the **button that retrieves the interviewee information** to see all the entries in the database. (You may need to adjust the properties for width and height in the *Designer* view in the *Properties* panel.)
- ☐ **Tap** the **button that records answers and then gets the next question.** Clicking the button cycles through the questions and records previous answers to the database. (Enter information, click **Record answer, then get the next question.**, then click **Retrieve Interview Info** to make sure it works each time.)

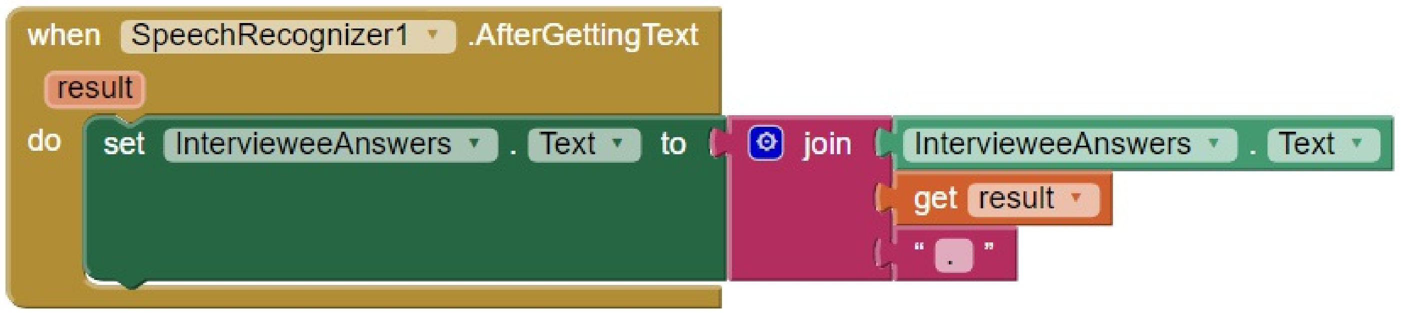
Speech to Text

Adding answers to questions may be a little time consuming on a tablet. Making an app to meet a need should focus on simplifying life, not making it more complex and taking more time than other non-app alternatives. For this reason, you are going to add a *speech-to-text* feature. This feature will allow a user to read a question and record their spoken answer.

Below are the blocks required to use the *SpeechRecognizer* component.



```
when Record .Click
do  call SpeechRecognizer1 .GetText
```



```
when SpeechRecognizer1 .AfterGettingText
  result
do  set IntervieweeAnswers . Text to  join  IntervieweeAnswers . Text
                                            get result
                                            " . "
```

PLTW DEVELOPER'S JOURNAL With your partner, talk through and record what these blocks do:

1. Procedure call
2. Result variable
3. Concatenation
4. Final string that adds ". " at the end of the concatenation

44. Test to make sure the app records from speech to text and stores information in the database.

Clearing a Database

Now that the program is set up and working, you probably have many test entries that the end user will not need. You want to clear the entries from the database, but you need to make sure that when a user empties the entries, it is intentional. To do this, add a display to notify the user that the data is about to be cleared.

45. From the *ResetDBList* blocks drawer, drag out the *ResetDBList.Click* event handler.
46. Add the *call TinyDB.ClearAll* procedure from the *TinyDB* blocks drawer.
47. Touch the **Reset** button on the Android device. Did the database clear? Could a user accidentally select Reset?
48. Working with your partner using what you have done in previous apps, add a notifier that requires the user to select **Yes** before it deletes the information stored in the database.
49. Show your completed app to your teacher.

User Testing

Now that your app is complete, you will give your app to another team as directed by your teacher. Testing others' apps and having your peers test your app allow your team to accomplish multiple tasks:

- You get a final round of feedback related to the database interview app.
  - What do they like about it?
  - What could be improved?

- By having others record real information in the app's database, you can identity problems to pursue in the development of future apps.
- You have a chance to practice your professional communication skills.

50. Based on user feedback, update your app to be more user friendly for all potential users.

51. Can you think of other questions that might help you identify problems to solve? Update and expand your app question list.

PLTW DEVELOPER'S JOURNAL Take note of the key questions you will ask others in interviews.

Future Problem Developments

52. In Problem 1.3.1, you will identify a problem to solve. As you prepare to tackle bigger problems, you will likely need to interview people who have knowledge of the problem but whom you might not know. Start working with your teacher now to think about who you might be able to interview.

PLTW DEVELOPER'S JOURNAL Brainstorm about some people you might be able to interview. What steps do you need to take to set up a time to interview them?

Conclusion

1. Why is it so important to engage others in problem identification?
2. Why is it so important to get feedback from end users on what you are developing as you are developing it?
3. What are three ideas you came across through exploring problem identification and interviewing others that you might pursue later?
4. How did you interpret and respond to the **essential questions**? Capture your thoughts for future conversations.