NoteTaker

NoteTaker is an app that allows you to enter notes and see your previously entered notes. It is an example of an app with dynamic, user-generated data. The data is persistent, meaning if you close the app and reopen it, the notes will still be there. The data is not shared amongst apps or users as the notes are stored on the device, not on the web.
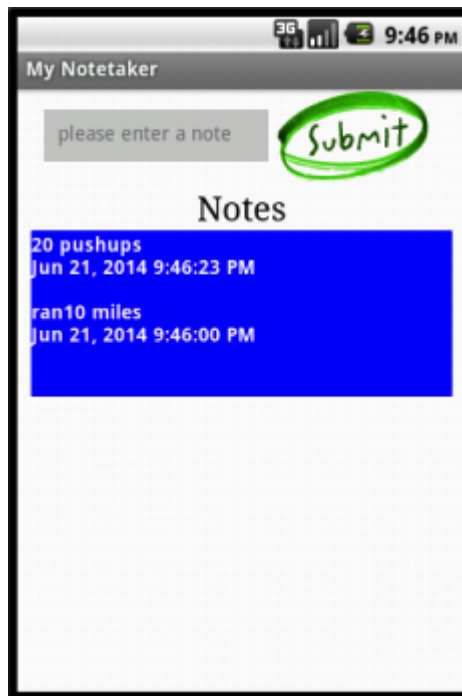
This lesson shows three iterations, each of them slightly more complex:

Iteration 1. The data is not persistent (saved in the database).
Iteration 2. The data is stored in a TinyDB database.
Iteration 3. The notes are timestamped.

Check out the app

The app lets you enter notes in the top text box and then lists them in reverse-chronological order in the blue notes area.

Create the user interface for the NoteTaker app

Display hidden components in Viewer

My Notetaker 🛜📶🔋 9:48

Submit

Notes

EntryTextBox, set Hint to "please enter a note"

SubmitButton with image

NotesLabel, this is where the list will be displayed.

TinyDB is found in the Storage drawer. It is used to store data persistently.

Non-visible components

TinyDB1 Clock1

Clock is used to timestamp notes
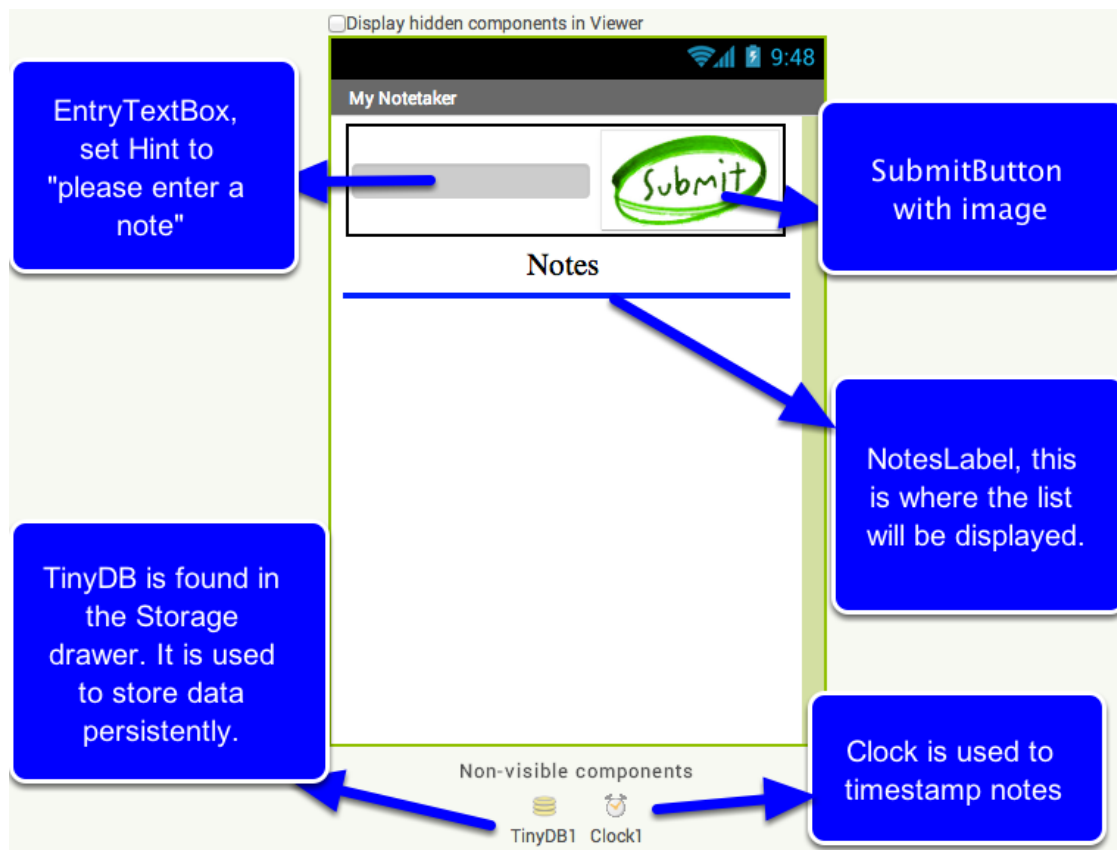
Iteration 1: A NoteTaker with no database

When the app starts, there are no notes, so use 'create empty list'

initialize global `notes` to | ⚙ create empty list

when `SubmitButton` .Click
do
  if (not (is empty (`EntryTextBox` . `Text`)))
  → check to make sure user entered something in text box
  then
    add items to list list (get `global notes`
                        item (`EntryTextBox` . `Text`)
    → Add the entry to the end of the list
    set `NotesLabel` . `Text` to (call `listToText`
                                    list (get `global notes`)
    → Display the new list
    set `EntryTextBox` . `Text` to (" ◯ "
    → Clear the user's entry after submission.

Convert the list of notes into a text string with \n used for new lines

to `listToText` `list`
result (initialize local `listAsText` to (" ◯ "
  in
    do
      for each `item` in list (get `list`
      do
        set `listAsText` to (⚙ join (get `item`
                                      " \n "
                                      get `listAsText`)
      → for each item, add it to the front of the text string being built.
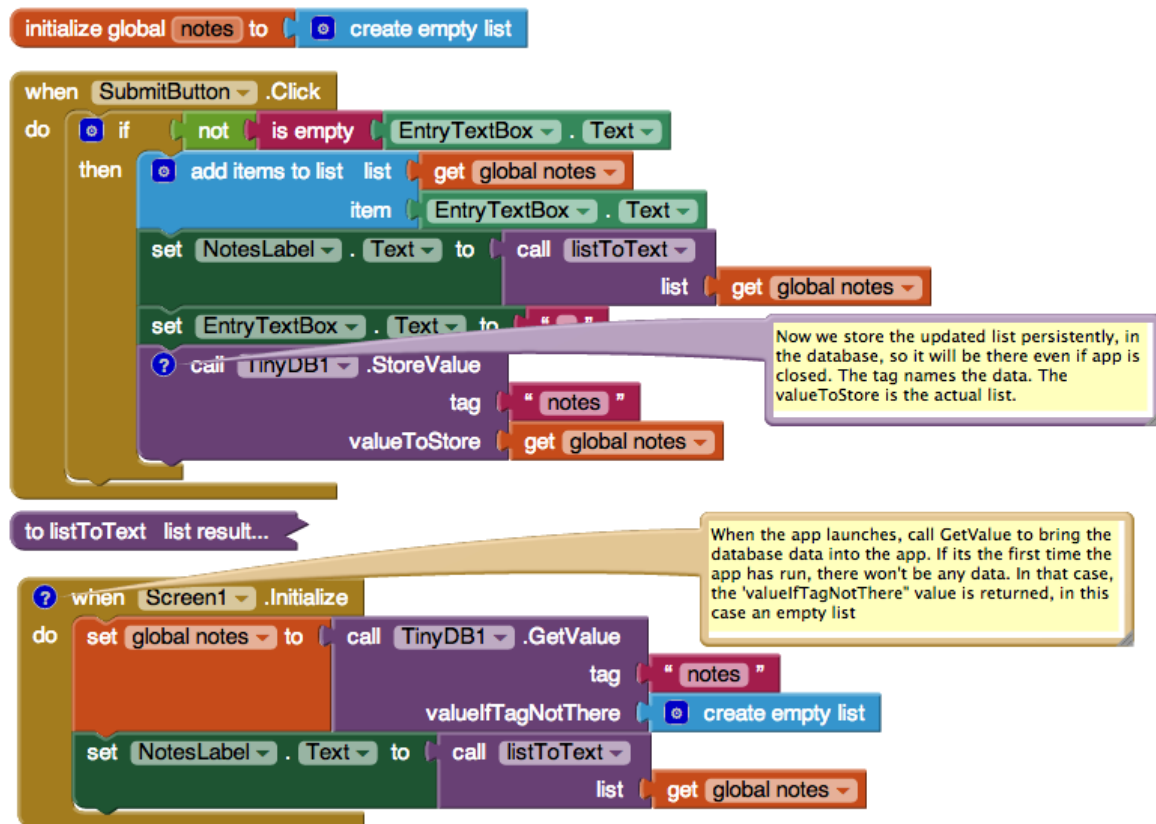   result (get `listAsText`
   → return the resulting text to the 'call listToText' block

Create a global variable 'notes' to store the notes the user enters. Initialize it to 'create empty list' because when the app starts, there aren't any notes yet. Program the SubmitButton.Click so that it adds what the user has entered in the text box to the list. and displayes the updated list. Create a procedure 'listToText' which takes a list as a parameter and converts it into a text object with multiple lines.

Iteration 2: Add TinyDB persistence so the notes are still there when the app is closed and reopened.


```
initialize global notes to [ create empty list ]

when SubmitButton .Click
do   if   not   is empty   EntryTextBox . Text
     then   add items to list   list   get global notes
                                item   EntryTextBox . Text
            set NotesLabel . Text to   call listToText
                                              list   get global notes
            set EntryTextBox . Text to   " "
            call TinyDB1 .StoreValue
                         tag   " notes "
                         valueToStore   get global notes
```

Now we store the updated list persistently, in the database, so it will be there even if app is closed. The tag names the data. The valueToStore is the actual list.

```
to listToText   list result...

when Screen1 .Initialize
do   set global notes to   call TinyDB1 .GetValue
                                       tag   " notes "
                                       valueIfTagNotThere   create empty list
     set NotesLabel . Text to   call listToText
                                       list   get global notes
```

When the app launches, call GetValue to bring the database data into the app. If its first time the app has run, there won't be any data. In that case, the 'valueIfTagNotThere" value is returned, in this case an empty list

At the bottom of SubmitButton.Click, add a call to TinyDB1.StoreValue to store the updated list of notes. When the app is reopened, use TinyDB.GetValue to bring the notes back in from the database into the variable 'notes'.

Iteration 3. Add a timestamp to each note-- the list becomes a list of lists.
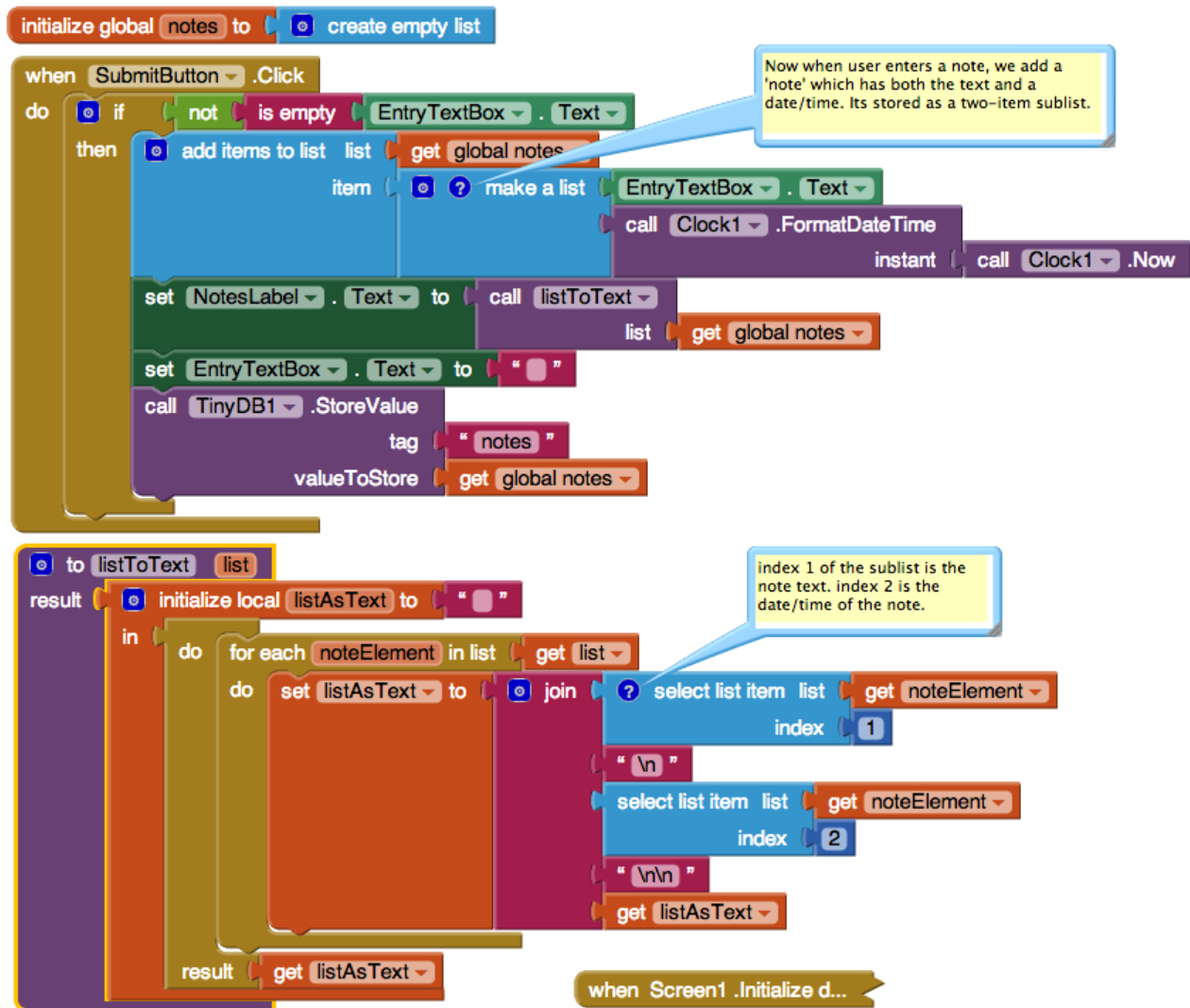
The notes list is now a list of lists. Each 'note' is itself a two-item list with the first item being the text of the note and the second item being the timestamp for the note. To test, you may need to clear out the database of the MIT App Companion to get rid of the data that was formatted in a different say (iteration 2). To do this, on your device, go to Settings | Applications, choose MIT AI2 Companion, and clear data.