|  |  |
| --- | --- |
| In PaintPot projects, we added a Camera button to take a photo with the device’s ***Camera*** and use that photo as the Canvas’s background image. In this tutorial, we will learn how to save that photo to a database on the device, so that whenever the app is run, the photo can be retrieved from the database.    By using a database in this way we will turn the photo into an example of ***persistent data*** -- i.e., data that persists between different uses of the app. We will use App Inventor’s ***Tiny DB*** component to allow the app to save the user’s photos on the device.    **Objectives**: In this lesson you will learn to:   * create an app that saves images between sessions; * learn about the concept of persistent data; * learn how to access App Inventor’s simple database component. * learn to use Lists and the ListPicker component. | ***[Click to watch video](https://www.youtube.com/watch?v=k-oD2KLBdD4)*** |

# Introduction: Persistent Data and TinyDb

Up until now, the data in our apps has been stored either in ***global variables*** or as the value of the ***properties*** of the app’s various components. For example, when you store a piece of text in a Label, that data is stored in the computer’s main memory, in its RAM or random access memory. And as we’ve learned, RAM is ***volatile***, meaning that any data stored there will be destroyed when the app is exited.

By contrast, data stored in the computer’s long-term storage -- e.g., on the phone’s flash drive -- will ***persist*** as long as the app is kept on the device. There are various ways to store data permanently on a computer. For example, you could store it in a file, such as a document or image file. Another way to store persistent data is in a ***database.***  App Inventor provides us a very simple, easy-to-use database in its ***TinyDb*** component. Any data that we store in the TinyDb, will not disappear when the app is exited. Instead, it will persist between uses of the app -- even if you turn off the device.

However, before we learn how to incorporate TinyDb into our app, let’s spend a moment to learn about this very important component.

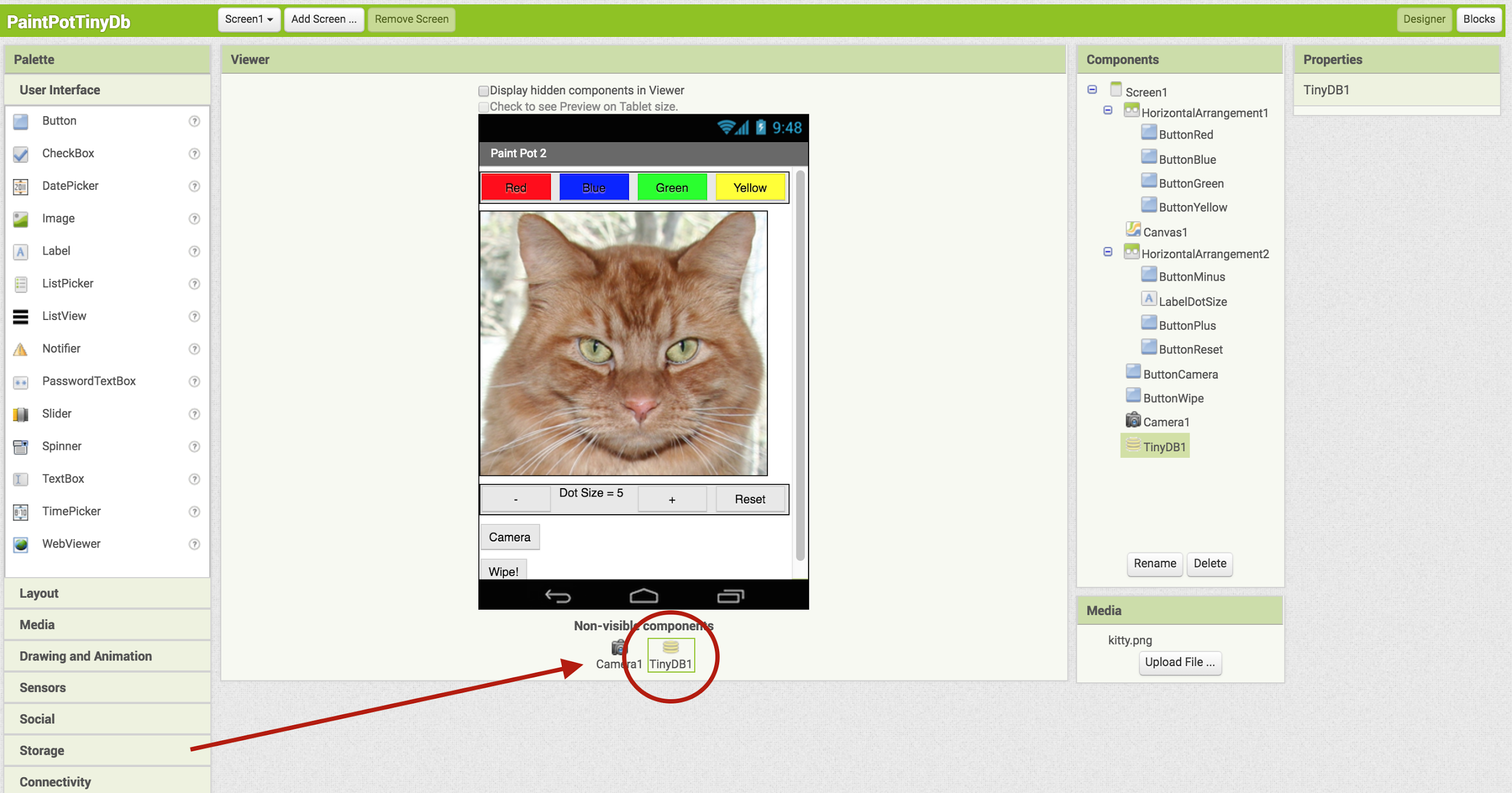
[](https://www.youtube.com/watch?v=VCv59tjUqd0)

(Click the image to start the [video](https://www.youtube.com/watch?v=VCv59tjUqd0).)

# Getting Ready

# Click here to open [App Inventor with the PaintPotTinyDbTemplate](http://ai2.appinventor.mit.edu/?repo=templates.appinventor.mit.edu/trincoll/csp/unit3/templates/PaintPotTinyDbTemplate/PaintPotTinyDbTemplate.asc) in a separate tab and follow along with the video tutorial. Once the template project opens use *Save As* to rename your project ***PaintPotTinyDb****.* Note: If the blocks don’t appear well in the Blocks Editor, right-click on the background and use the *Arrange Vertically* option.

# The User Interface (UI)

The UI for this version of this version of the Paint Pot app will consist of the original UI plus a Camera and Tiny DB components. As shown in the following image, the TinyDb is a non-visible component. It is located in the *Storage* drawer. And it has no properties

## Adding a TinyDB

From the Palette’s *Storage* drawer, drag a [TinyDB](http://ai2.appinventor.mit.edu/reference/components/storage.html#TinyDB) component onto the Viewer. Notice that it drops down to the non-visible components area and does not appear on the screen. Notice that there are no properties for the TinyDB component.

## Adding a Camera

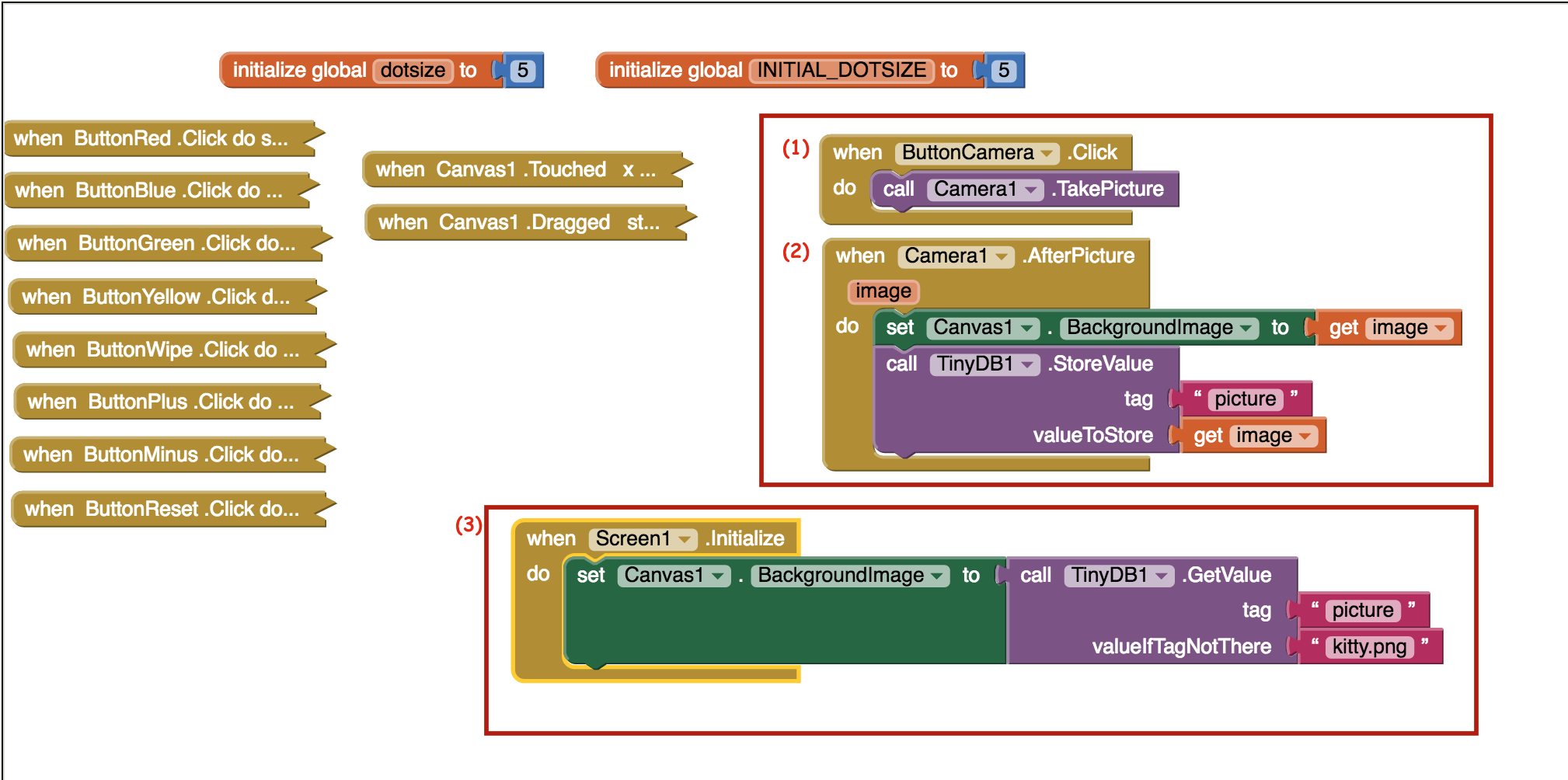
If you haven’t already added a camera component for taking photos, follow these directions. From the Palette’s *Media* drawer, drag a Camera component onto the Viewer. Notice that it drops down into the invisible components area at the bottom of the screen and does not appear on the screen. Thus, you’ll also need a button for the user to click when they want to use the Camera.

Add a Button to the UI and set its *Text* property to "Take A Picture!" or “Camera” and rename the button to *ButtonCamera* so that it can be distinguished from the color and wipe buttons.

When you are done with this step, your app’s UI should like screenshot above.

# Code Walkthrough

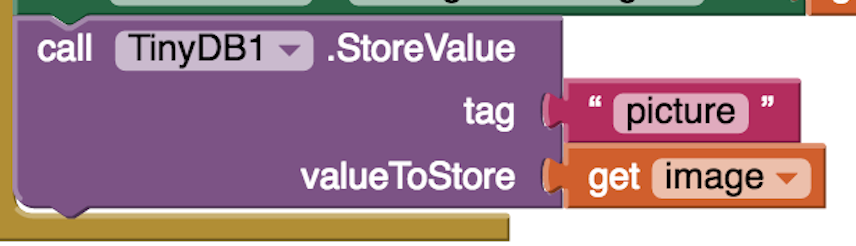
The blocks for this version of the PaintPot app are shown in the following screenshot. The blocks contained in the template have been collapsed. Only the blocks that use the Camera and TinyDb components are shown fully expanded. (NOTE: To collapse or expand a single block you can right-click on it or use CTRL-click on the Macintosh. To collapse or expand all of the blocks in the workspace, right-click or CTRL-click on the background of the workspace.)



Let’s walk through the three numbered blocks.

1. When the *ButtonCamera* is clicked, we call the Camera’s *TakePicture* procedure. This will bring up the devices camera, just as if you had brought it up manually. When you take a picture, the camera will give you the option of keeping the picture (usually a check mark) or deleting it (usually an X). When you keep the picture, the camera will automatically store the picture in an image file on the device. The camera will return a ***file path,*** e.g.: file:///storage/emulated/0/Pictures/app\_inventor\_198249824242.jpg.
2. After you’ve accepted the picture, the camera will close and the *Camera.AfterPicture* event-handler will fire. Its ***image*** property will contain the image’s file path -- i.e., that long string that describes exactly where the image is stored on the device. When you assign that image as the Canvas’s BackgroundImage, App Inventor automatically retrieves the image from the file and displays it on the Canvas. However, if we don’t save the image in the TinyDb -- actually we are saving the file path in the TinyDb not the image itself, which was saved automatically by the camera -- we won’t be able to use it as the Canvas’s BackgroundImage the next time we run the app.

Notice how the TinyDb.Save block works. It associates the image with a tag so that we can later retrieve the image from TinyDb by specifying the tag. The tag here, which must be unique for every piece of data you want to store in TinyDb, is “picture”.



1. The final step in getting this to work is to retrieve the image when the app starts up. Here we are going to use the ***Screen.Initialize*** event-handler. This blocks **fires once** **and only once** when an app first starts up. It is designed to hold any initializations that have to be done when an app starts up. In this case, it is retrieving the image we saved from the TinyDb and setting it as the Canvas’s BackgroundImage. Note that we have to use the “picture” tag in order to retrieve the image.



Notice also how we have set the ***valueIfTagNotThere***value. This is what we call a ***default value.*** The TinyDb.GetValue block will return either the data associated with the “picture” tag or, if there is no such tag in the Db, it will return the default value. In this case, the default value is the name of the file storing the Kitty picture.

When would that default value be needed? Well, one time it will be needed is the very first time the app is started. In that case, we haven’t yet taken a picture and saved it in the TinyDb. So, in that case we want to display the Kitty. Another case where the default would be used is if the programmer gives the wrong tag when trying to retrieve the image. That would be a **bug.**

# Testing the App

When you have completed the changes described here, test your app. It should behave as follows: On the very first running of the app it should display the Kitty as the Canvas background image. After you’ve taken a photo, if you then quit out of the Companion and then reconnect to the Companion, it should display the photo as the Canvas background.

Similarly, if you package the app (using Build instead of Connect), the app should work the same way initially and after you’ve saved a photo. The saved photo will ***persist*** with the app as long as the app is on your device.

## Clearing the TinyDB

When using the Companion to develop an app that uses a TinyDb, it is sometimes necessary to clear the TinyDb in order to properly test that the app is working correctly. To do so you can clear any existing TinyDB data by going to Settings → Applications → MITAICompanion and tapping ‘Clear Data’.

## Lists

In the projects below, you will extend this version of PaintPot to save multiple photos for the canvas background in a list in TinyDB. The simplest data abstraction in programming is a variable, but there are more complex data structures available in all programming languages. App Inventor has a data structure called list which allows the storage of multiple items under one name in memory. The items are indexed which means they are numbered from 1 to the length of the list. The Lists drawer contains all the blocks available for manipulating lists. We first create a global variable to hold a list which can be an empty list or a list of items using make a list:

# 

In the projects below, you will use [List blocks](http://appinventor.mit.edu/explore/ai2/support/blocks/lists.html) such as insert item into list and select random item from list.

In the AP CSP exam pseudode, lists are represented using square brackets [ ] like below. The assignment operator ← can be used to assign a list to a variable. The list items can be numbers or text which are called strings; strings are usually indicated by quotes "" to distinguish them from variables.

# list ← [ "kitty.png", "android.png" ]

# Creative Mini Projects

Now that you've learned the basics of using TinyDb, it's time to add some additional features and enhancements to the Paint Pot app. Working in pairs, implement each of the following enhancements.

1. As we saw in the overview video, one can also store lists of data in TinyDb. So rather than just having a single photo to use as the Canvas background, we could have a selection of photos to choose from. As a first step, initialize a global variable for this list of backgrounds to the create empty list block from the Lists drawer. In the When Camera1.AfterPicture event handler, add the photo that's taken to that list using the add items to list block. Store the variable for the whole list in the TinyDb. Don't forget! You'll need a unique tag to associate with the list.
2. **If/else Algorithm.** What about when the app starts up? This can be a little tricky because now you'll be retrieving a list of photos, rather than a single photo. (What should the default value be when you are retrieving a list from TinyDb?) So you can't assign the list as the background image. You could select a random item (photo) from the list and make that your background. But what if this is the first time the app runs? When the list is empty? This would be a good place for an if/else algorithm controlled by whether or not the list retrieved from the TinyDb is empty or not. To solve this problem, you'll have to look through the Lists drawer in the Blocks Editor for some useful functions to use.
3. Add a ListPicker component to the app's user interface to let the user select the background image. Read more about the [Listpicker component here](http://appinventor.mit.edu/explore/content/basic.html#ListPicker). The ListPicker looks like a button but it displays a list of items to choose from. In its blocks, it has a BeforePicking and an AfterPicking event handler. One of the ListPicker properties is the Elements property which is the list of choices shown to the user. You can set this Elements property to your list of background photos in the BeforePicking event handler. Note that what will appear in the ListPicker are the file paths of the images, not the images themselves. There's no easy way around this. After the user has picked an element from the ListPicker, their choice will be in ListPicker1.Selection and can be put on the Canvas background.

***Nice work! Complete the Self-Check Exercises and Portfolio Reflection Questions as directed by your instructor.***