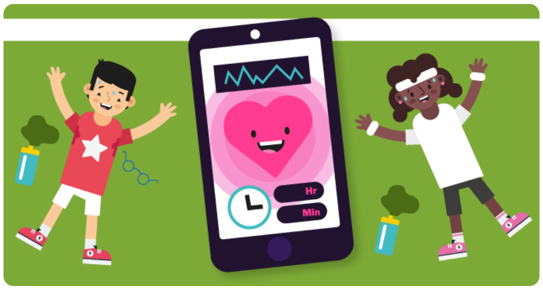
Exercise tracker

Make an app that tracks how much exercise you do

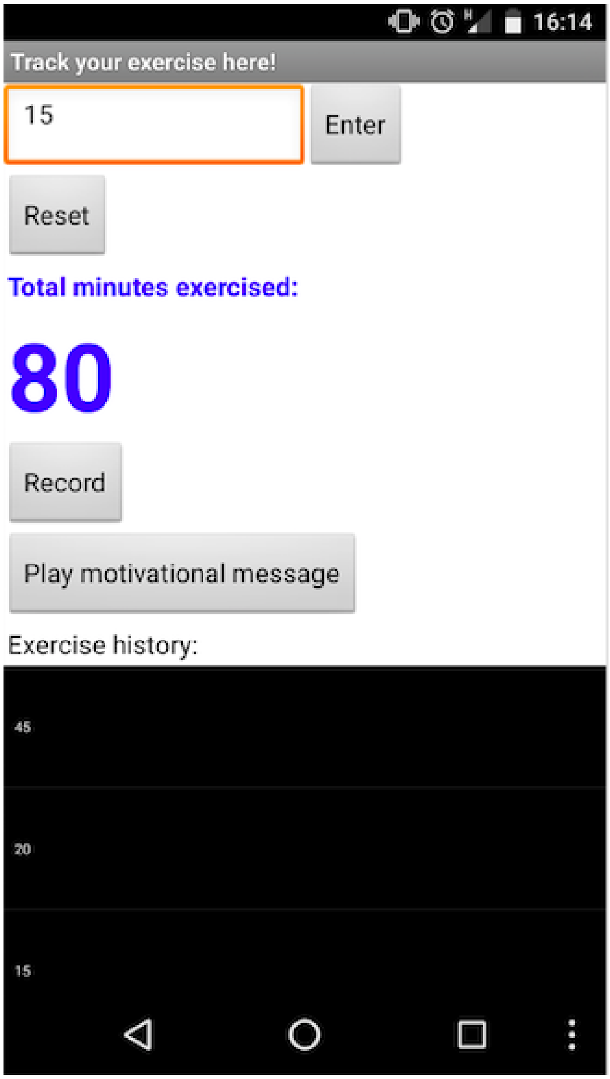


Step 1 Introduction

These cards are going to show you how to use App Inventor to make an app that tracks how much exercise you've done.

What you will make

- You'll end up with something that looks like this:



What you will learn

- Taking text input from the user and display it
- Storing information in a list
- Using a loop to read the elements of a list
- Saving information to a file on the phone
- Loading and displaying information from a file
- Making your own procedures
- Using the phone’s sound recorder and play back a sound you’ve recorded

What you will need

Hardware

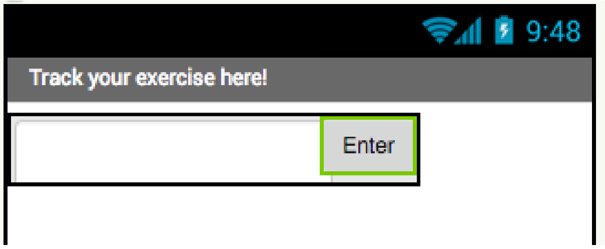
- A computer capable of accessing App Inventor
- An internet connection

Optional:

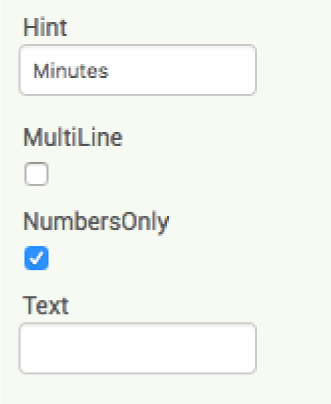
- An Android phone or tablet

Step 2 Recording exercise

- Create a new project and give it a name, for example `GetFit`.
- Click on Screen1 under Components and change the Title property to `Track your exercise here!`.
- Find the TextBox component in the Palette (under User Interface) and add it to your app, along with a Button.
- Change the Text property of the button to `Enter`.
- To arrange the components side by side, drag a HorizontalArrangement onto the screen (you'll find it in Layout) and drag the TextBox and Button into it.



- Find the Hint property for the TextBox and type `Minutes`. This will appear faintly in the textbox if the user hasn't typed anything in yet, so they know what to type.
- Check the box that says 'NumbersOnly' so that only a number can be entered in the TextBox.



Hint

Minutes

MultiLine

□

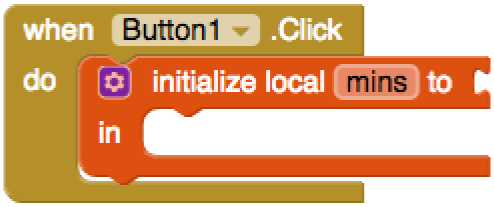
NumbersOnly

☑

Text

Great! The user can type in the number of minutes they exercised for. Now you want to save that information when they press the button.

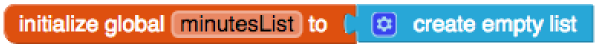
- Switch to the Blocks and take out a `when Button.Click` block.
- The first thing you'll need is to create a local variable to store the TextBox value in. Grab the `initialise local name to` block from Variables, and slot it into the `when Button Click` block.
- Then click where it says `name` and instead type `mins` to name your local variable.



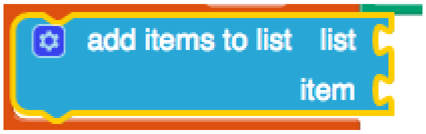
- Take out a `Textbox.Text` block and attach it onto the `initialise local mins` block to store what's been typed into the TextBox.

Now that you've retrieved this information, you're going to create a list to put it in. After all, you want to be able to record lots of exercise sessions!

- At the top of your code, add an `initialize global name to` block, and name it `minutesList`. Then find the `create empty list` block from Lists and use it to initialise your list.

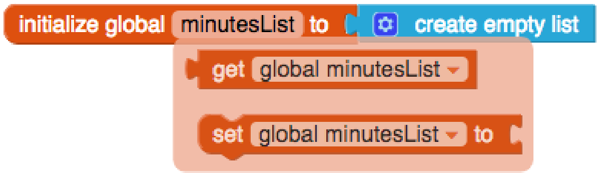


- From Lists, take the block `add items to list` and put it into your local variable block.

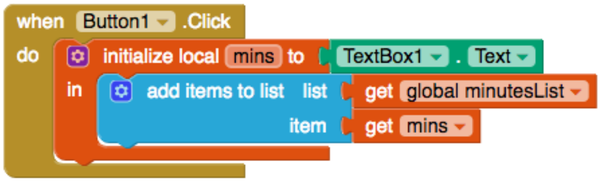


You need to attach two things to this block: the list you want to add something to, and the 'something' you want to add, meaning the item.

- Hover over the name of your global list variable and grab the `get global minutesList` block that appears. Connect this to the `list` attachment of the `add items to list` block.



- Then do the same with the local variable, `mins`, to attach a `get mins` block to the item part of `add items to list`.

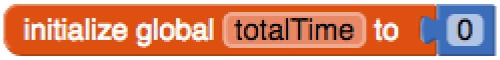


On the next card, you'll add all the list items together to work out the total amount of exercise you've done!

Step 3 Calculating the total

- Create another global variable called `totalTime`.

- Attach the `0` block from Math to initialise the variable to `0`.

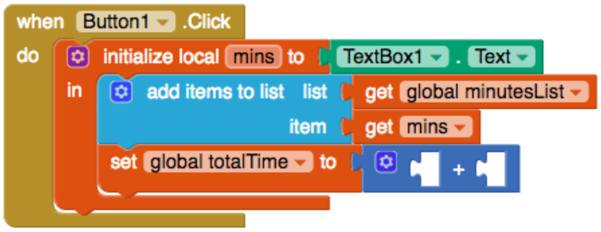


```
initialize global totalTime to 0
```

Every time you save a new time, you are going to add it to the value of `totalTime`.

- Hover over the `totalTime` variable and grab a `set global totalTime to` block. Attach it below the `add items to list` block.

- From Math, take the `+` block and attach it to `set global totalTime to`.



```
when Button1 .Click
do  initialize local mins to TextBox1 . Text
    in  add items to list  list  get global minutesList
                           item  get mins
        set global totalTime to  ( + )
```

- On the left side of the `+`, plug in a `get global totalTime` block. On the right side, plug in `get mins`.

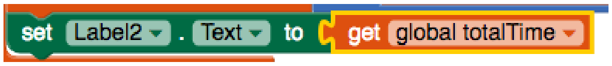
Now, display the total so the user can see it!

- Go back to the Designer view and add two more labels to your app. Set the Text property of the first one to `Total minutes exercised:`

- Change the Text property of the second label so that it's blank, and make a note of this label's name (for example, Label2) so that you can set it to the total in your code!

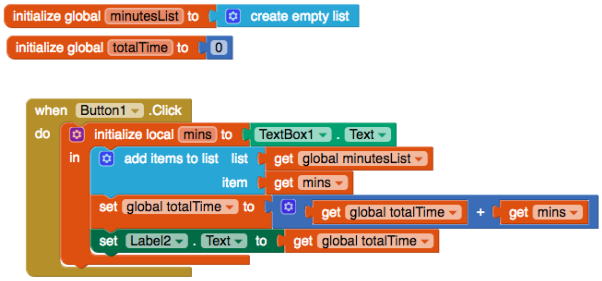
- If you want to, change the size and colour of the labels. I've made mine blue and checked FontBold to make them bold, and I changed the FontSize of the second label to `50`!

- Switch back to Blocks and add a `set Label.Text to` block to your code, together with a `get global totalTime` block (choose the label name you made a note of above!).



```
set Label2 . Text to  get global totalTime
```

Here's what your code should look like:

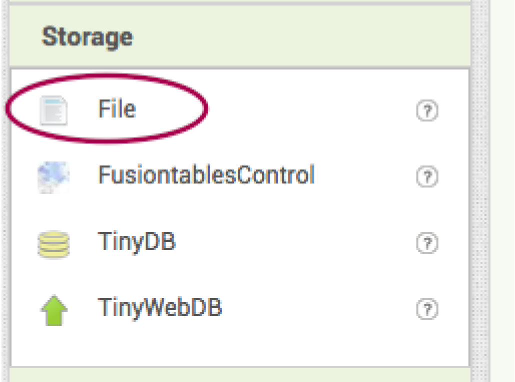


```
initialize global minutesList to  create empty list

initialize global totalTime to  0

when Button1 .Click
do  initialize local mins to  TextBox1 . Text
    in  add items to list  list  get global minutesList
                           item  get mins
        set global totalTime to  get global totalTime + get mins
        set Label2 . Text to  get global totalTime
```

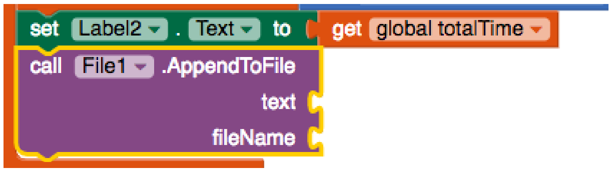
Step 4 Saving the information

Right now, your app only saves information so long as the app is running. It would be much more useful for it to remember your exercise times even after you close the app and restart it, right? To do this, you will store the information in a file on the phone or tablet, and read from this file every time the app starts.

- In the Designer view, add a File component to your app. You'll find it in `Storage`. This is an invisible component, so you won't see it on the screen.

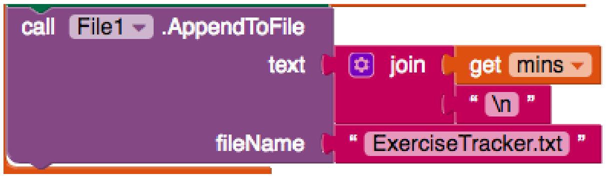


- Now go to Blocks and click on File1 to get the block `call File1.AppendToFile`. Add this to your code after the `set Label.Text to` block.



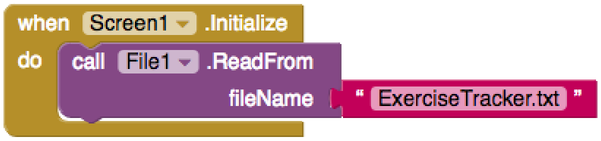
- For the `fileName` parameter, attach a `""` block from Text, and type in `ExerciseTracker.txt`.

- For the `text` parameter, attach a `join` block, a `get mins` block, and another blank `""` Text block. Type `\n` into the blank text block (make sure you use a backslash `\` and not a forward slash `/`).



Now you've saved data to the file, you need to read that data whenever the app loads!

- Take a `when Screen1.initialise` block, and add in `call File1.readFrom`, attaching a Text block with the file name `ExerciseTracker.txt` typed in as before.

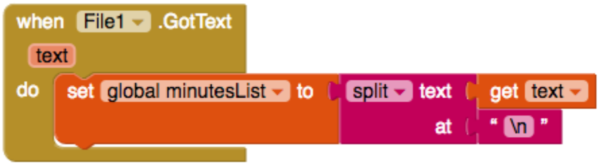


This call is asynchronous, meaning it will go and read the file and then tell you when it's done.

- From File1, take out a `when File1.GotText` block.

The `text` variable contains all the text from the file. You will use this to fill the list variable you created to collect the minutes. But first, you need to split it up to separate each line.

- Add the following blocks inside the `GotText`:

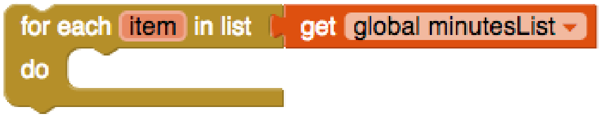


```
when File1 .GotText
  text
do  set global minutesList to  split text  get text
                                     at  " \n "
```

Now you will sum up all the minutes you just loaded from the file and display the total.

- Under the `set global minutesList`, add code to set the global variable `totalTime` to `0`:

- In the Control blocks, find the block `for each item in list`, and attach a `get global minutesList` to it.

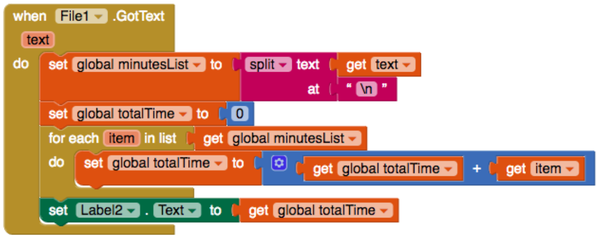


```
for each item in list  get global minutesList
do
```

- Inside this, add a `set global totalTime to` block, and then a `+` block with `get global totalTime` on the left. Remember, you did this before to add something to the total. The only difference this time around is that the variable you put on the right of the `+`: the current `item` of the list.

- Finally, add `set Label.Text to` the and `get global totalTime` block as before.

- Here's what your `GotText` block should look like now. Test out your app to make sure it all works!

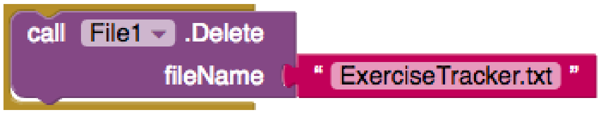


```
when File1 .GotText
  text
do  set global minutesList to  split text  get text
                                     at  " \n "
    set global totalTime to  0
    for each item in list  get global minutesList
    do  set global totalTime to  get global totalTime + get item
    set Label2 . Text to  get global totalTime
```

Step 5 Resetting

If you're anything like me, you might want a way of deleting all the previously entered data and starting the tracking again at zero. Let's add a button to let you do that!

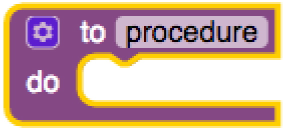
- In the Designer view, add a Button. Change its label to `Reset`.
- Go to Blocks and add a `when Button Click` block for the new button.
- Into this block, add `call File1.Delete` with a Text block giving the file name `ExerciseTracker.txt`.



```
call File1 .Delete
        fileName " ExerciseTracker.txt "
```

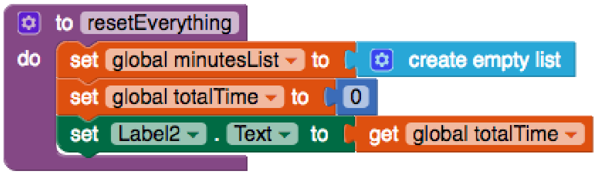
Now you will create a brand-new block of your own!

- Click on Procedures in the Built-in blocks and drag out a `to procedure do` block.



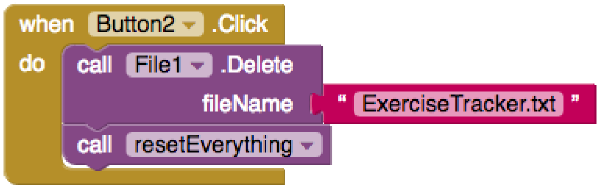
```
to procedure
do
```

- Click the `procedure` label and change it to `resetEverything`.
- Inside the `resetEverything` block, add blocks to set the global `minutesList` to a new empty list, set the global `totalTime` to `0`, and display the new total:



```
to resetEverything
do  set global minutesList to create empty list
    set global totalTime to 0
    set Label2 . Text to get global totalTime
```

- Finally, in the `when Button Click` block of the button you've just made, add the block `call resetEverything` from Procedures.



```
when Button2 .Click
do  call File1 .Delete
            fileName " ExerciseTracker.txt "
    call resetEverything
```

Now you should be able to clear all the recorded exercise by clicking the new button!

Step 6 Record a message!

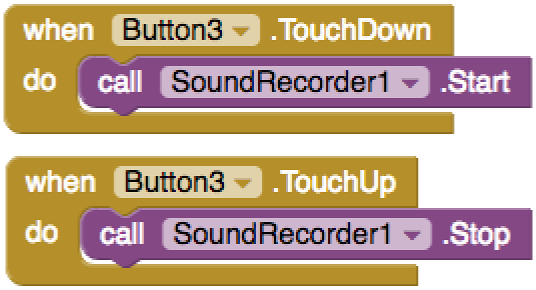
Getting fit isn't easy: sometimes it can be hard to motivate yourself to go and exercise. How about letting the user record a message that they can play whenever they need some extra motivation?

- Go to the Designer view and add two more Buttons to your app. Set their labels to `Play motivational message` and `Record`, or something similar.

- Then, from Media, add a Sound and a SoundRecorder component. Just like the File component, these won't be visible on the screen.

- In Blocks, add a `when Button.TouchDown` block and a `when Button.TouchUp` block for the `Record` button. This time, you're not going to detect the usual click of the button. Instead, you're going to start recording when the user presses and holds the button, and you'll stop recording when they stop pressing.

- Add `call SoundRecorder.Start` to the `TouchDown` block, and `call SoundRecorder.Stop` to the `TouchUp` block, like this:



```
when Button3 .TouchDown
do   call SoundRecorder1 .Start

when Button3 .TouchUp
do   call SoundRecorder1 .Stop
```

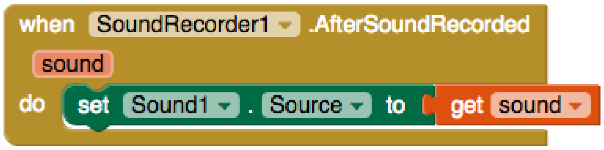
Now you can record sound, you need to set up the Sound component to play it!

- Drag out the `when SoundRecorder.AfterSoundRecorded` block.

- In the Sound component, find the `set Sound1.Source to` block and put that inside the block you just took out.

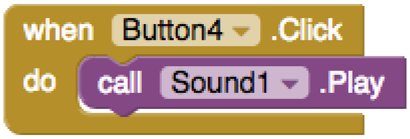
The `AfterSoundRecorded` block has a variable called `sound`. This is the where you tell the block where for find the sound file you've recorded.

- Hover over the `sound` variable and take the `get sound` block to attach on as the source for the Sound component:



```
when SoundRecorder1 .AfterSoundRecorded
  sound
do   set Sound1 . Source to   get sound
```

- Finally, take out a `Button.Click` block for the `Play` button. In it, put a `call Sound1.Play` from the Sound component.



```
when Button4 .Click
do   call Sound1 .Play
```

- Test out the app and have some fun recording and playing back your own motivational messages!

Challenge: save the sound

- See if you can use a File component to make the app remember the location of the sound file to play.
  - Use another File component and a separate file called something else, for example `MotivationalMessage.txt`.
  - Use a `SaveFile` block instead of `AppendToFile`, so that you always overwrite the previous file with the new recording.

Step 7 Displaying the exercise history

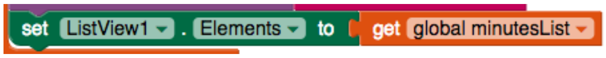
At the moment your app only displays the total minutes exercised, but since you've got the list of all the individual sessions, why not show that as well?

- Go to the Designer view, and add a ListView from User Interface.
- If you want, you can also add a label above the list that says something like `Exercise history:`.

As you might have guessed, a ListView displays a list of things. Similar to how you set the Text property of a Label to some text, you set the Elements property of a ListView to a list. You will do this in two places in your code.

First, you need to update the ListView whenever the user enters a new exercise time.

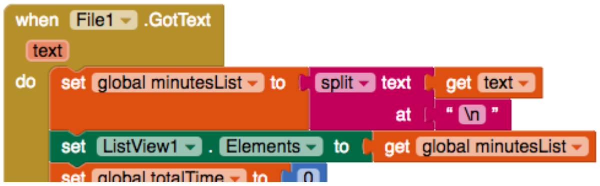
- In the `Button.Click` for the `Enter` button, add a `set ListView.Elements to` block and a `get global minutesList` block below `AppendToFile`.



```
set ListView1 . Elements to get global minutesList
```

Secondly, you need to update the ListView whenever you load the list file.

- Find your `File1.GotFile` code, and add `set ListView.Elements to` and `get global minutesList` (the same code as above) right below the `set global minutesList to` block.



```
when File1 .GotText
 text
do set global minutesList to split text get text
                                 at " \n "
   set ListView1 . Elements to get global minutesList
   set global totalTime to 0
```

And your app is complete!

Challenge: track the type of exercise

- How about adding another TextBox that lets the user also record what kind of exercise they did? You'll have to think about what extra code you'll need, such as lists and loops, and how to store the new information in a file.
- You can either use the same file (with some extra `join` and `split` code), or a separate one.

You can see an example of this app on App Inventor at dojo.soy/intermedapp (http://dojo.soy/intermedapp).