

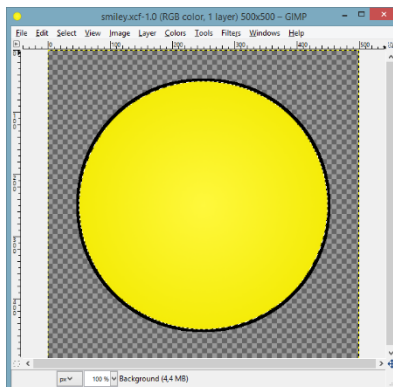


Task 1 | GIMP | "Smiley"


Your task is to create an image looks like this:

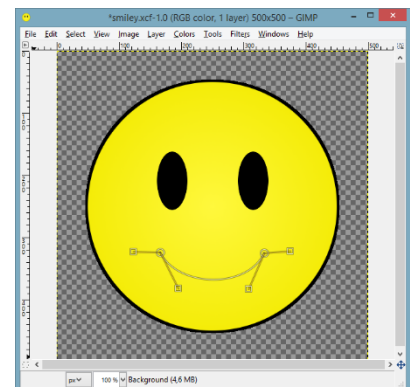


- Create a 500x500 pixel image and fill the background with transparency.
- Use the *Ellipse Select Tool* to make a circular selection and place it in the middle of the image.
- Set the *Foreground color* to black and click on the option *Edit\Stroke Selection*. Set line width to 8 px then click the *Stroke* button on the left.
- Set light yellow to foreground color and a darker yellow to the background color. 
- Select the *Blend Tool*, by clicking on  , set its *Radial Shape* and set *FG to BG* gradient. Now start selection from the middle of the drawn circle and pull it out of the line. Release the button. The result should look like this:



- Select an ellipsoidal area (as one of the eyes) and fill it with black.
- Select the *Move Tool* and choose *Selection* on the left. Move the selection next to the other one (this will be the other eye) and fill it with black, as well.
- Dismiss the selection by hit Shift+Ctrl+A.
- The mouth shape will be created by using the *Paths Tool*. Follow the steps:

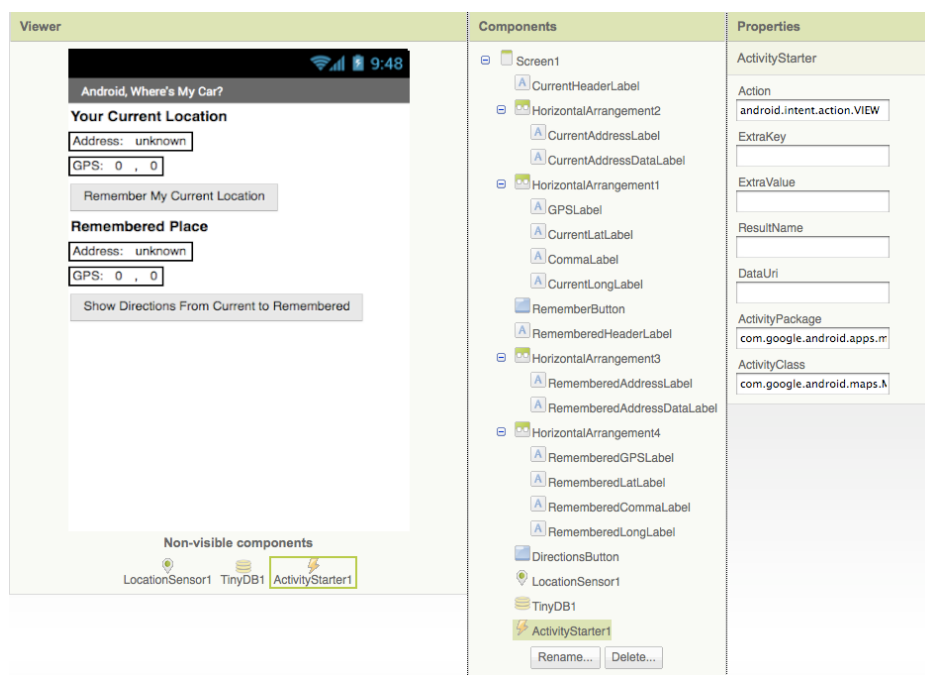
- set painting colour to black
- select paths tool  then click on the two points that will give the start and end points of the mouth
- pull down the middle of the line to get an arc instead of the straight line
- click the stroke path button at *Tool options*. Set line width to 5 px.



- Select an ellipsoidal shape on the top of the head. Select the *Blend Tool*, set the painting color to white and the background color to the well-known light yellow from the previous movements (the easiest tool to do so is the *Color Picker Tool*). Set its shape to *Linear* and set gradient to *FG to BG*. Fill the selected shape from its top to the bottom, then dismiss the selection.
- Save your project as *Smiley* in *.xcf* and also export the image in *.png* format.

Task 2 | MIT App Inventor 2 | “Where’s my car?”

- The app demonstrates how to communicate with the Android location sensor, how to record data in the phone's long-term memory (database), and how you can open the Google Maps app from your app to show directions from one location to another. It makes use of the following App Inventor components:
 - *Location Sensor*
 - *TinyDB* //to store the data
 - *ActivityStarter* //to open a map
- Here are the components for the *Where’s My Car?* app, as shown in the *Component Designer*:



- The *ActivityStarter1* component is used to launch the map when the user asks for directions. Its properties are only partially shown above. Here is how they should be specified:

Property	Value
Action	android.intent.action.VIEW
ActivityClass	com.google.android.maps.MapActivity
ActivityPackage	com.google.android.apps.maps

- Let’s examine the four different event-handlers of the app, starting in the top-left and working around in counter-clockwise order.

- **LocationSensor1.LocationChanged**: This event occurs when the phone's location sensor first gets a reading, or when the phone is moved to produce a new reading. The event-handler just places the readings – latitude, longitude, and current (street) address – into the corresponding 'Current' labels so that they appear on the phone. The *RememberButton* is also enabled in this event-handler. Its enabled setting should be unchecked in the *Component Designer* because there is nothing for the user to remember until the sensor gets a reading.
- **RememberButton.Click**: When the user clicks the *RememberButton*, the location sensor's current readings are put into the 'remember' labels and stored to the database as well. The *DirectionsButton* is enabled as it now makes sense for the user click on it to see a map (though it will make more sense once the user changes location).
- **DirectionsButton.Click**: When the user clicks the *DirectionsButton*, the event-handler builds a URL for a map and calls *ActivityStarter* to launch the Maps application and load the map. `join` is used to build the URL to send to the *Google Maps* application. The resulting URL consists of the *Google Maps* domain along with two crucial parameters, *saddr* and *daddr*, which specify the start and destination for the directions. For this app, the *saddr* is set to the latitude and longitude of the current location, and the *daddr* is set to the latitude and longitude of the location that was 'remembered'.
- **Screen1.Initialize**: This event is always triggered when an app opens. To understand it, you have to envision the user recording the location of the car, then closing the app, then later re-opening the app. When the app re-opens, the user expects that the location remembered earlier should appear on the phone. To facilitate this, the event-handler queries the database (`call TinyDB1.GetValue`). If there is indeed a remembered address stored in the database – the length of the stored address is greater than zero – the remembered latitude, longitude, and street address are placed in the corresponding labels.

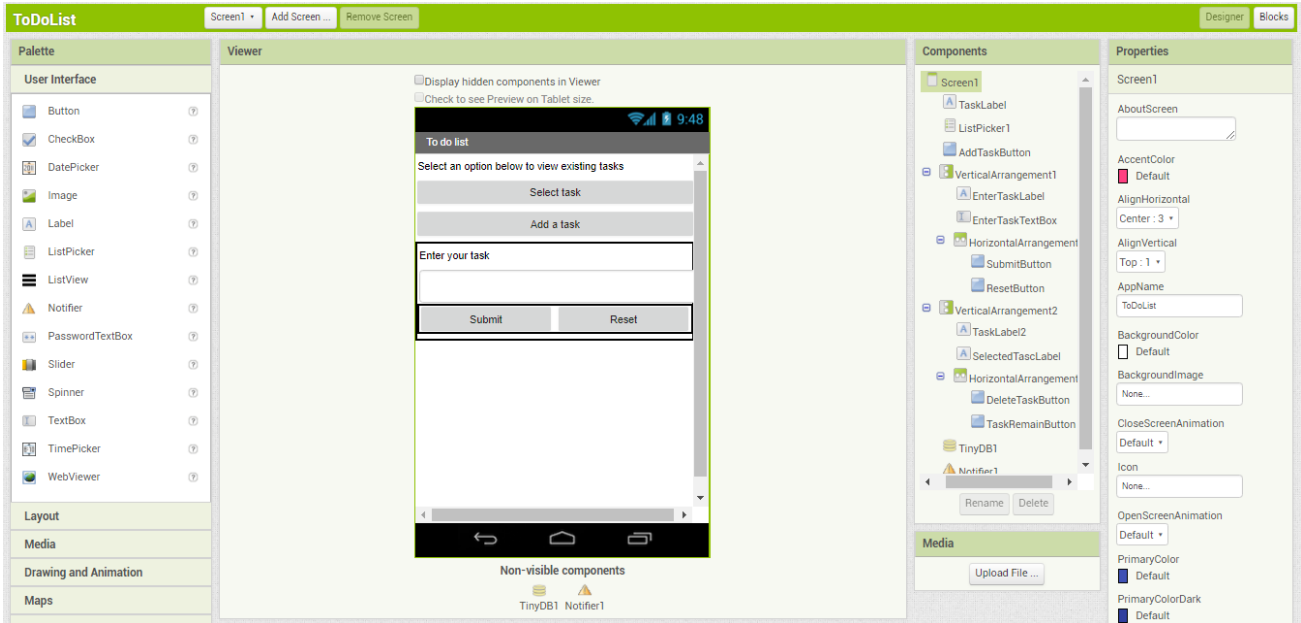
The image displays four screenshots of App Inventor code blocks:

- initialize global tempAddress to**: A block to initialize a global variable `tempAddress` to an empty string.
- when LocationSensor1 . LocationChanged**: A block that sets `CurrentAddressDataLabel` to `LocationSensor1 . CurrentAddress`, `CurrentLatLabel` to `get latitude`, `CurrentLongLabel` to `get longitude`, and `RememberButton . Enabled` to `true`.
- when RememberButton . Click**: A block that sets `RememberedAddressDataLabel` to `LocationSensor1 . CurrentAddress`, `RememberedLatLabel` to `LocationSensor1 . Latitude`, and `RememberedLongLabel` to `LocationSensor1 . Longitude`. It then calls `TinyDB1 . StoreValue` three times with tags `address`, `lat`, and `long`, and finally sets `DirectionsButton . Enabled` to `true`.
- when Screen1 . Initialize**: A block that sets `global tempAddress` to `call TinyDB1 . GetValue` with tag `address`. It then checks `length get global tempAddress > 0`. If true, it sets `RememberedAddressDataLabel` to `call TinyDB1 . GetValue` with tag `address`, `RememberedLatLabel` to `call TinyDB1 . GetValue` with tag `lat`, and `RememberedLongLabel` to `call TinyDB1 . GetValue` with tag `long`. It also sets `DirectionsButton . Enabled` to `true`.
- when DirectionsButton . Click**: A block that sets `ActivityStarter1 . DataUri` to `join` of `http://maps.google.com/maps?saddr=`, `CurrentLatLabel . Text`, `CurrentLongLabel . Text`, and `&daddr=`, `RememberedLatLabel . Text`, `RememberedLongLabel . Text`. It then calls `ActivityStarter1 . StartActivity`.

- Find out more about *Sensors* by clicking on the link below.
<http://www.appinventor.org/bookChapters/chapter23.pdf>

Task 3 | MIT App Inventor 2 | "To do list"

- Your task is to try to create the app on your own according to a not so detailed description shown below.
- Here are the components for the *To do list* app, as shown in the *Component Designer*:



- Here are the blocks for the *To do list* app, as shown in the *Blocks Viewer*:

initialize global tagTask to " task " initialize global listTask to create empty list initialize global numberTask to 0

```

when Screen1.Initialize
do call initData

when ListPicker1.AfterPicking
do set SelectedTaskLabel.Text to ListPicker1.Selection
   set VerticalArrangement1.Visible to false
   set VerticalArrangement2.Visible to true

when AddTaskButton.Click
do set VerticalArrangement1.Visible to true
   set VerticalArrangement2.Visible to false
   set AddTaskButton.Enabled to false
   set EnterTaskTextBox.Hint to "Enter your new task here. Then click to submit b..."

when SubmitButton.Click
do if is empty trim EnterTaskTextBox.Text
   then call Notifier1.ShowDialog
      message "No task has been entered"
      title "Info"
      buttonText "OK"
   else call appendNewTask
      set EnterTaskTextBox.Text to ""
      set VerticalArrangement1.Visible to false
      call Notifier1.ShowAlert
         notice "Tag was added"
      set AddTaskButton.Enabled to true
      set ListPicker1.Enabled to true
  
```

show the selected task

show the input field to add new

if there is no task to add, a popup notification appears

otherwise, the new task is added to the list

```

when ResetButton .Click
do
  set VerticalArrangement1 .Visible to false
  set AddTaskButton .Enabled to true

when TaskRemainButton .Click
do
  set SelectedTascLabel .Text to ""
  set VerticalArrangement2 .Visible to false

when DeleteTaskButton .Click
do
  call deleteTask
  set SelectedTascLabel .Text to ""
  call Notifier1 .ShowAlert
  notice "Task was deleted"
  set VerticalArrangement2 .Visible to false
    
```

close the text input option

keep task

delete task, then notify the user about the modification

- We use procedures in *App Inventor* to create new blocks that we can use repeatedly and take up less space than all of the blocks used in the original procedure. If we are using the same sets of blocks more than once, these blocks are called redundant.

```

to initData
do
  set ListPicker1 .Title to "Task list"
  set VerticalArrangement1 .Visible to false
  set global listTask to call TinyDB1 .GetValue
  tag get global tagTask
  valueIfTagNotThere get global listTask

to appendNewTask
do
  set global listTask to ListPicker1 .Elements
  add items to list list get global listTask
  item EnterTaskTextBox .Text
  call TinyDB1 .StoreValue
  tag get global tagTask
  valueToStore get global listTask

to deleteTask
do
  set global listTask to ListPicker1 .Elements
  remove list item list get global listTask
  index index in list thing ListPicker1 .Selection
  list get global listTask
  call TinyDB1 .StoreValue
  tag get global tagTask
  valueToStore get global listTask
    
```

get existing tasks, stored as a list in *TinyDB1*

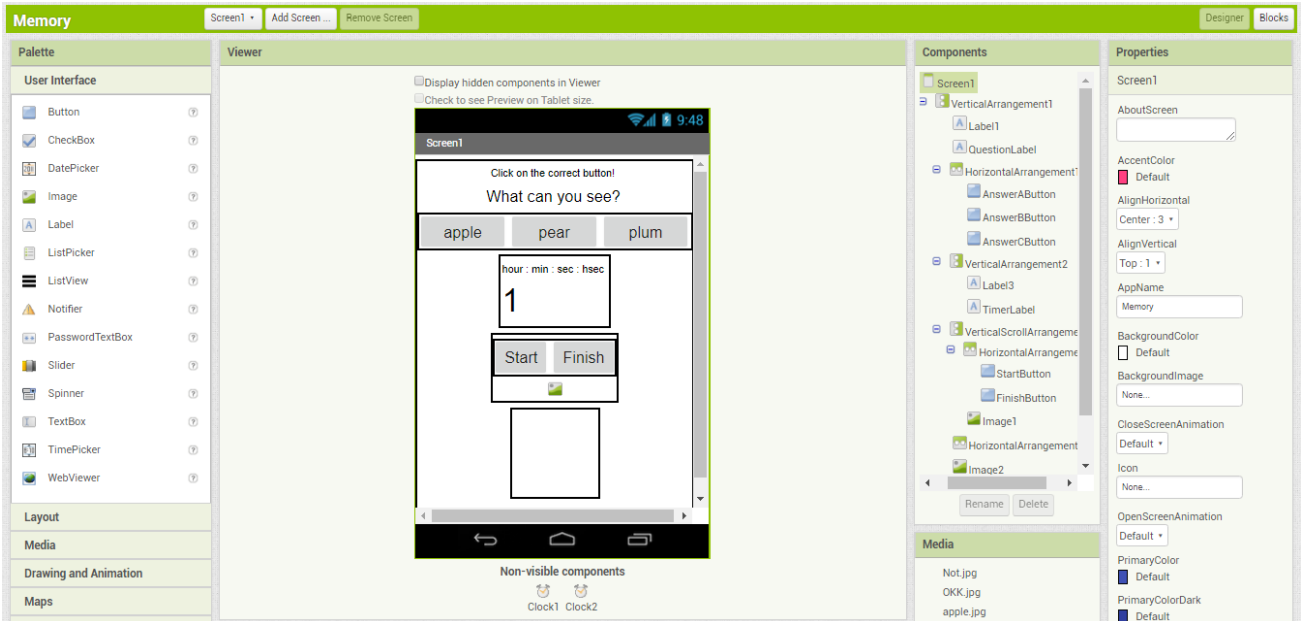
add a new task to the list, then store the list in *TinyDB1*

delete a task from the list, then store the list in *TinyDB1*

- Find out more about *Procedures* by clicking on the links below.
 - <http://appinventor.mit.edu/explore/ai2/support/blocks/procedures.html>
 - <http://www.appinventor.org/Procedures2>

Task 4 | MIT App Inventor 2 | “Memory”

- Your task is to try to create the app on your own according to a not so detailed description shown below.
- Here are the components for the *Memory* app, as shown in the *Component Designer*:



- Set the *TimerInterval* of *Clock 1* and *Clock2* components to 100.
- Here are the blocks for the *Memory* app, as shown in the *Blocks Viewer*:

initialize global `fruitList` to `make a list` ["apple", "pear", "plum"]

initialize global `pictureList` to `make a list` ["apple.jpg", "pear.jpg", "plum.jpg"]

initialize global `index` to `1`

initialize global `ts1` to `0`

initialize global `ts2` to `0`

initialize global `sec2` to `0`

when `Screen1` .Initialize do `call showPicture`

when `StartButton` .Click do `call showPicture`

when `FinishButton` .Click do `close application`

initialize global `min2` to `0`

initialize global `h2` to `0`

to `showPicture` do

- set global `index` to `random integer from 1 to 3` (randomize the order of the images)
- set `Clock1` . `TimerEnabled` to `true`
- set `Image1` . `Visible` to `true`
- set `Image1` . `Picture` to `select list item list` (get global `pictureList`, index: get global `index`) (show the randomly chosen image)
- set `Image2` . `Visible` to `false`
- set `Clock2` . `TimerEnabled` to `true`
- set global `ts2` to `0`
- set global `sec2` to `0`
- set global `min2` to `0`
- set global `h2` to `0`
- set `TimerLabel` . `Text` to `join` (get global `h2`, ":", get global `min2`, ":", get global `sec2`, ":", get global `ts2`) (create time variables, set the time variables to 0 and set the TimerLabel to the value of the variables)

```

when Clock1.Timer
do
  set global ts1 to get global ts1 + 1
  if get global ts1 ≥ 10
  then
    set Image1.Visible to false
    set global ts1 to 0
  
```

by using *Clock1.Timer* you can show each image for exactly 1 second

```

when Clock2.Timer
do
  set global ts2 to get global ts2 + 1
  if get global ts2 ≥ 10
  then
    set global ts2 to 0
    set global sec2 to get global sec2 + 1
  if get global sec2 ≥ 60
  then
    set global sec2 to 0
    set global min2 to get global min2 + 1
  if get global min2 ≥ 60
  then
    set global min2 to 0
    set global h2 to get global h2 + 1
  set TimerLabel.Text to join
    get global h2
    " "
    get global min2
    " "
    get global sec2
    " "
    get global ts2
  
```

by using *Clock2.Timer* you can measure the reaction time

```

when AnswerAButton.Click
do
  set Clock2.TimerEnabled to false
  if select list item list get global fruitList = AnswerAButton.Text
    index get global index
  then
    set Image2.Visible to true
    set Image2.Picture to "OKK.jpg"
  else
    set Image2.Visible to true
    set Image2.Picture to "not.jpg"
  
```

if one of the answer buttons is clicked, the user will be notified if his/her answer is correct or not

```

when AnswerBButton.Click
do
  set Clock2.TimerEnabled to false
  if select list item list get global fruitList = AnswerBButton.Text
    index get global index
  then
    set Image2.Visible to true
    set Image2.Picture to "OKK.jpg"
  else
    set Image2.Visible to true
    set Image2.Picture to "not.jpg"
  
```

```

when AnswerCButton.Click
do
  set Clock2.TimerEnabled to false
  if select list item list get global fruitList = AnswerCButton.Text
    index get global index
  then
    set Image2.Visible to true
    set Image2.Picture to "OKK.jpg"
  else
    set Image2.Visible to true
    set Image2.Picture to "not.jpg"
  
```

- Find out more about the *Clock* component by clicking on the links below.
 - <http://ai2.appinventor.mit.edu/reference/components/sensors.html#Clock3>
 - <https://sites.google.com/site/stevozip/home/AI2/clock>