5COSC023W - Tutorial 5 Exercises

As part of this tutorial for this week, you should complete **ALL** the tasks described in the following specifications: (**make sure that you ask questions to your tutor for anything that you do not understand or if you are stuck at any point**).

Like all other modules, you are expected to study towards you module outside the lecture and dedicated tutorial slots for a number of hours. If you do not finish all of the exercises in the tutorial session, make sure that you finish them on your own time and by the end of the week. This is a normal process and part of your university learning.

For all the tasks you should use Jetpack Compose and NOT Views!

1 The Lost Dog Application

1. Implement the "Lost Dog" application described in the lecture (watch the lecture video if you did not attend the live lecture). The image of the dog used can be downloaded from `https://github.com/udacity/dog-project/blob/master/images/Brittany_02625.jpg`
2. Try to centre the image and the text on the screen by specifying the following as part of the `Column` composable:

```
Column(
    modifier = modifier.fillMaxSize(),
    verticalArrangement = Arrangement.Center,
    horizontalAlignment = Alignment.CenterHorizontally
) {
    // ... rest of code goes here
}
```

   What happens if you do not specify the `modifier = modifier.fillMaxSize()`? Is it still centred, if not, why not?

2 Identify the Dog Breed App

The task is to develop an application that the user will be using to gain knowledge and be able to identify dog breeds.

1. When the application starts it displays to the user 3 different unique random different breed dog images. The images should be clickable. It is not allowed to display the same image more than once, i.e. the 3 images should be unique, as it should also be the breed.

   You can use images of different breeds from the following website: `http://vision.stanford.edu/aditya86/ImageNetDogs/`

The screen should also display the name of a breed corresponding to one of the displayed images.

The user's aim is to click on the dog image corresponding to the displayed breed name, after which (a single attempt is only allowed) the message *CORRECT!* or the message *INCORRECT!* appears, depending on whether the answer given is correct or incorrect respectively.

At the same time, 3 new random images are displayed, giving the user the chance to play again. Every time a new set of different images should be displayed.

Hint: An image can be made clickable by adding a clickable modifier to it as follows:

```
Image(painterResource(id = R.drawable.brittany_02625),
                      contentDescription = null,
                      modifier = Modifier.clickable {
                          // code to execute when clicked
                      }
)
```

2. Extend the application so that the first screen contains a second button with the title `Finish`. As soon as the user clicks on the button, a new activity starts. The new activity displays how many correct guesses and how many incorrect guesses the user has made since the start of the application (no saving required — as soon as the application terminates the score is reset).

   Hint: An intent can carry information (data) in it. The source activity (the activity which starts a new activity) can put data inside the intent using the `putExtra` method, for example:

   ```
   intent.putExtra("Number", 125)
   ```

   The receiving activity can extra the data from its creator intent by:

   ```
   int n = intent.getIntExtra("Number", 0)
   ```

   where 0 is the default value, i.e. if there is no data called `Number` inside the intent.

3. Extend the application so that the first activity also contains a textbox (with the label `Name` which the user can type their name). The second activity displays the score together with the name of the player, i.e. if the user types `James` as their name, the score should appear as:

   ```
   James Score
   Correct guesses: 53
   Incorrect guesses: 10
   ```