Overview: Coin Flip Simulation

Coin Flip is an app that simulates the flipping of a two-sided coin. This app uses App Inventor's random number generator and two images to simulate the coin flip.

Objectives: In this lesson you will learn to:

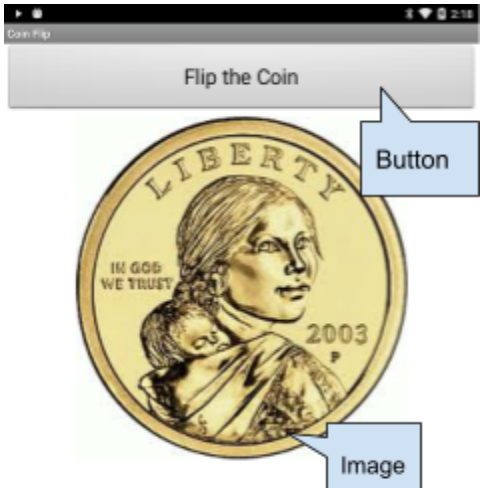
- create an artifact that uses Randomness and simulates a model;
- create a simple model of a coin flipping;
- use random number blocks to generate a random value in a specific range;
- define a global variable and assign it an initial value;
- use a conditional statement, *IF/Else*, to evaluate a variable and follow an algorithm based on the value of a variable;
- use a loop to repeat the coin flip many times and calculate the percentage of heads.

Short Handout

Click to watch Preview Video

Getting Started

Open App Inventor with the Coin Flip Media Only template. This will open a project that contains the images you will need in this lesson. Use the Projects/*Save As* option to rename your project to *CoinFlip.*



Designing the User Interface

The UI for the first version of our Coin Flip app will consist of two *Components:* a *Button* and an Image. The Button is used to flip the coin to either heads or tails. The Image is used to display either heads or tails when the coin is flipped.

Adding the Button

1. Get a Button from the Palette's *User Interface* category
2. Change the text
3. Set the *Width* to Fill Parent

Adding the Image

1. Get an Image component from the Palette's *User Interface* category
2. Change the picture to *heads.jpg* provided in the template
3. To center the image on the screen you can change the *Screen1.AlignHorizontal* property.

Here's a summary of the UI Components and their properties:

UI Component	Name	Properties
Screen	Screen1	Change: Title = Coin Flip AlignHorizontal = Center
Button	Button1	Change: Text = Flip the Coin Width= Fill Parent
Image	Image1	Change: Picture = heads.jpg

Coding the App

The Coin Flip app should simulate the flipping of a two-sided coin. When the user clicks the button, the coin should be flipped and land on either heads or tails. The picture should change to represent the side the coin lands on. A variable *coin* will be used to represent either heads or tails and an If/Else statement will be used to display the correct image.

Handling the Button Click Event

Nearly all of the app's code will be inside the *Button1.Click* event handler. Begin by dragging the *Button1.Click* event handler from the Toolbox onto the Blocks workspace.

Coin Variable

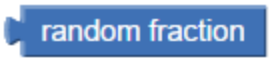
How should we represent the coin that is being flipped? The answer, of course, is we will use a global variable.

Coin will be a global variable that can have one of two values: 1 (for heads) or 2 (for tails). First initialize the variable by getting an *initialize global variable* block from the Toolbox. Name the variable *coin* and set *coin* to heads by giving it an initial value of 1. Now each time Button1 is clicked, *coin* should 'flip' and randomly 'land' on 1 (heads) or 2 (tails). To do this, you will need to get a setter block for *coin* from the Toolbox.

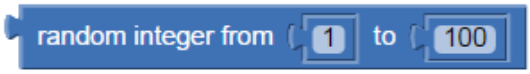
Simulating the Coin Flip

To randomly set the value of *coin* to be either 1 or 2, you will need to use App Inventor's *random integer* block. App Inventor has several blocks for randomness in the Toolbox's *Math* drawer. You may have already seen and used these blocks.

1. **random fraction**: Randomly selects a number (such as 0.532) between 0 and 1 (not including 1):



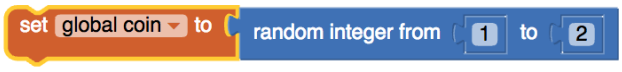
2. **random integer**: Randomly selects a number between two specified whole numbers, inclusive:



Because our *coin* will have a random whole number (1 or 2), we will use the second of these blocks.

In the AP CSP exam, the function RANDOM(1,100) is used to return a random number from 1 to 100 (including both 1 and 100).

Get the *random integer* block from the Toolbox and use it to set the value of the *coin* variable. Then, specify the range to be from 1 to 2 using the number blocks that are provided. This code should go inside the *Button1.Click* handler:

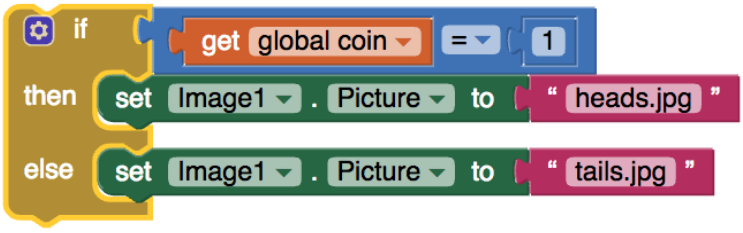


Now, if you were to click the button, *coin* would 'flip' and randomly 'land' on 1 (heads) or 2 (tails) but you would only see the heads.jpg image on your screen. Let's use an If/Else statement to determine when *heads.jpg* should be shown and when *tails.jpg* should be shown.

Displaying the Result: If/Else Algorithm

To display the result of the simulated coin flip, we will either display the *heads.jpg* image or the *tails.jpg* image. For this, we will need an *if/else* block that implements the pseudocode algorithm shown on the left in the following table, with the corresponding App Inventor code shown on the right.

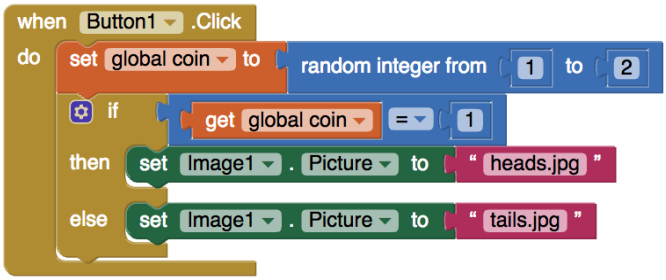
```
IF (coin = 1)
{
    DISPLAY( heads.jpg )
}
ELSE
{
    DISPLAY( tails.jpg )
}
```



Coin Flip Simulation Algorithm

Putting these elements together gives us the following algorithm for simulating the flipping of a two-sided coin, both in College Board-style pseudocode and in App Inventor blocks.

```
WHEN Button1.Click
{
   coin  ←  RANDOM(1,2)
   IF (coin = 1)
   {
       DISPLAY( heads.jpg )
   }
   ELSE
   {
       DISPLAY( tails.jpg )
   }
}
```



In summary, you need to set the following variable and event handler.

Variable	Values
coin	1

Event handler	Algorithms
Button1.click	If coin = 1, then set Image1.picture to the text “heads.jpg” else set Image1.picture to the text “tails.jpg”

Testing the App

Now, you have a fully functioning Coin Flip app that simulates the flipping of a two-sided coin. Test out your app to make sure it works correctly. How accurately can you predict whether the next 'flip' will be heads or tails? If you can't predict any more accurately than flipping a real coin, then we have created a pretty good *computer model* or *computer simulation* of a coin flip.

Inputs	Expected Outputs	Actual Outputs
Click "Flip the Coin"	Coin image changes to heads or tails image randomly.	?

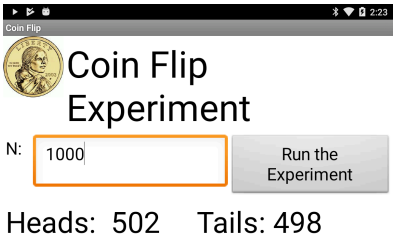
How Does a Computer Model Randomness

App Inventor -- and other computer languages -- use a form of randomness called ***pseudo randomness.*** Pseudo randomness is a model (or simulation) of true randomness. Just like your app models or simulates a coin flip, the App Inventor blocks *random-fraction* and *random-integer,* which we used to generate random numbers, are models or simulations of truly random numbers. In fact, there are algorithms in App Inventor, known as Pseudo Random Number Generators or PRNGs, that simulate the generation of random numbers. We will take a closer look at how PRNGs work in an upcoming lesson.

As we will learn later in the course when talk about encryption, the development of secure networks -- such as the Internet -- depends in crucial ways on the development of good PRNGs. So this is an important area of research in computer science and related fields of mathematics.

Coin Flip Part 2: Repeating the Coin Flip N Times

Now that we know how to model a coin flip, let's create another app that will let us perform the following ***modeling experiment:***



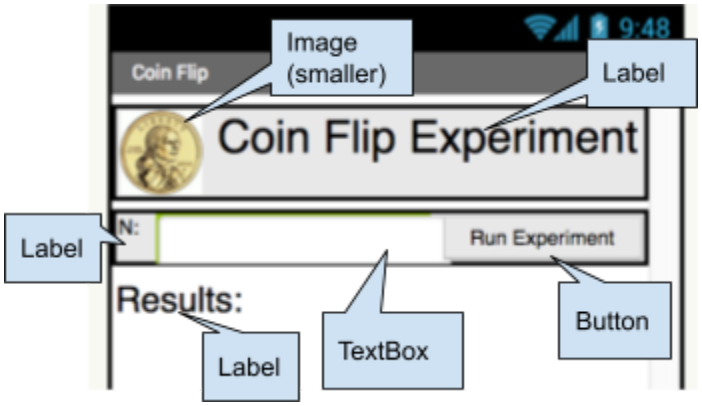
Let the user input a number, *N,* and press a "Run Experiment" button that will cause the app to simulate *N* coin flips and report the number of heads and the number of tails.

If we did this experiment with a real (fair) coin, we would expect to get roughly 50% heads.

If App Inventor's ***random-integer*** function is well designed, we should similarly expect to get roughly 50% heads when we perform this simulation experiment.

Re-Designing the User Interface

Let's revise the UI so it looks like this:



- Add a *HorizontalArrangment* at the very top of the screen. Then add the coin *Image* to the arrangement followed by a Label named *LabelTitle.* Set the image's width and height to 50 pixels and the title's font size to 30. The title label should be labelled "Coin Flip Experiment."
- Add another *HorizontalArrangement* just below the first one and then add a *Label* and a *TextBox* to it. This Label should be named *LabelN* and labeled "N:" and the TextBox should be named *TextBoxN* and have its *NumbersOnly* property set to true (checked).
- Change the name of *Button1* to *ButtonGo* and its label to "Run Experiment" and add it to the right hand side of the HorizontalArrangement.
- Add another label named *LabelResults* below the horizontal arrangements and label it "Results". Set its font to 24. This is where we will display the results of the experiment.

Here's a summary of all the changes:

UI Component	Name	Properties
Horizontal Arrangement	HorizontalArrangement1	Place at the top of the screen. Change: Width = fill parent

Image	Image1	Place in the left side of *HorizontalArrangement1* Change: Width = 50 Height = 50
Label	LabelTitle	Place in the right side of *HorizontalArrangement1* . Change: FontSize = 30 Text = Coin Flip Experiment
Horizontal Arrangement	HorizontalArrangement2	Place under *HorizontalArrangement1*. Change: Width = fill parent
Label	LabelN	Place in the left side of *HorizontalArrangement2*. Change: Text = N:
Text Box	TextBoxN	Place in the middle of *HorizontalArrangement2*. Change: NumbersOnly = checked
Button	Button1	Move to the right side of *HorizontalArrangement2*. Change: Rename = ButtonGo Text = Run the Experiment
Label	LabelResults	Place below *HorizontalArrangement2*. Change: Width = fill parent FontSize = 24

Coding the App

Experiment Variables

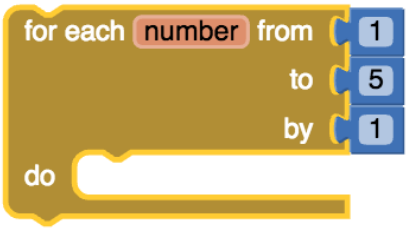
Clearly we're going to need some additional variables to manage the experiment. Add the following variables to the app:

- N - initialize N to 0. This will be the number of coin flips based on the user's input in the TextBox.
- nHeads - initialized to 0. We'll use this variable to count the number of heads in the experiment.

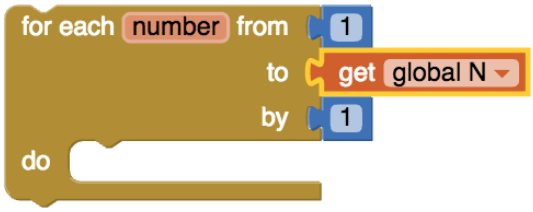
Variables	Values
N	0
nHeads	0
coin	1

Coding the Loop

What kind of problem are we using the loop to solve? Basically, if the user inputs 100 we want to flip the coin 100 times and on each flip we want to test whether it comes up heads or tails. This is a ***counting problem*** -- i.e, we need to count the coin flips, starting at 1 and stopping at 100. More generally, we need to count from 1 to N (whatever the user input). The best loop for this kind of problem is App Inventor's ***For each number from __ to __ by __*** loop, where the __'s are place holders.



In our case, we want to replace the 5 by N, the total number of coin flips to perform:



The loop will repeatedly perform whatever operations we put inside its ***do*** slot.

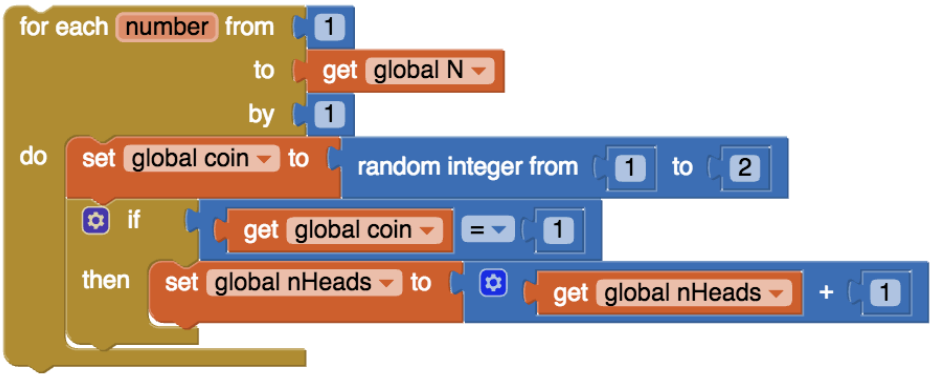
Before the Loop: Initializations

When you're coding a loop, it is necessary to perform some ***initialization steps.*** In this case we need to set the value of ***N,*** which will control when the loop will stop or. This variable will be initialized by taking the value the user inputs into the TextBox. And we need to initialize ***nHeads*** to 0 -- because, as we're going to be counting the number of heads we get, we want to start counting at 0:

The Body of The Loop: N Coin Flips

What needs to be done in the ***body*** of the loop -- i.e., in the ***do*** slot? We need to "flip" the coin and then count whether it came out heads or tails. We already know how to do this using the ***random integer*** block for the coin flip and an ***if/else*** for checking whether it's heads or tails. In

this case, however, rather than displaying a heads or tails image, we're going to add 1 to the ***nHeads*** variable.

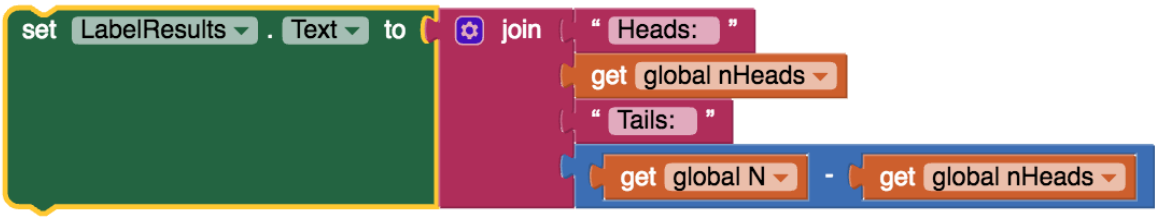


After the Loop: Report Results

When the loop finishes, we will report the results in the *LabelResults.* Here's the format we will use:

Heads: 52 Tails: 48

We can do this by using ***string concatenation*** (the join block) as follows:



Notice here that the number of tails is calculated by simply subtracting the *nHeads* from *N.*

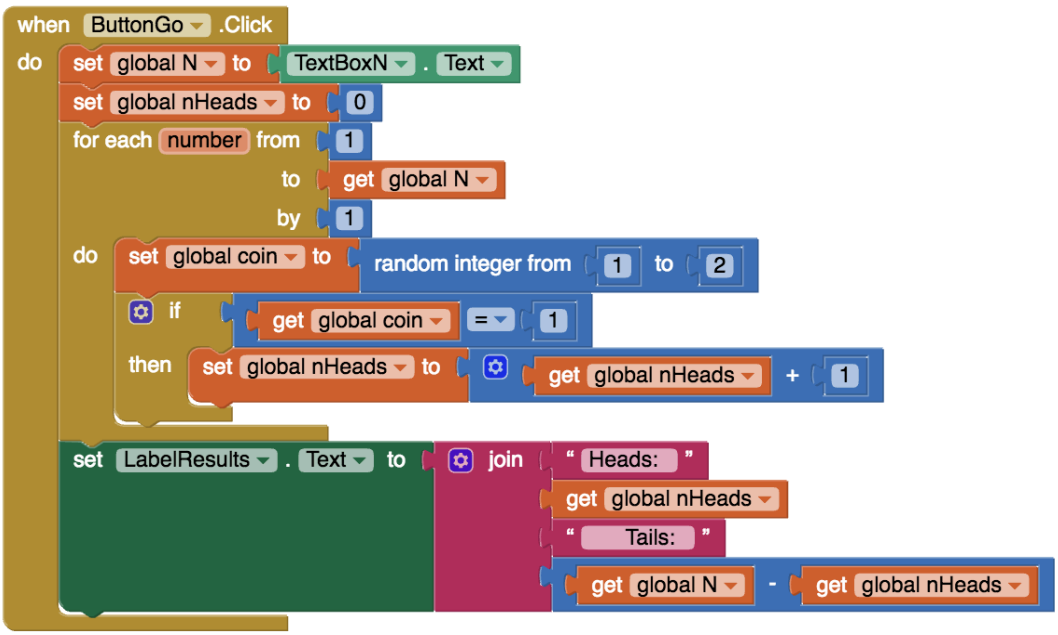
The Whole Algorithm

All of these steps are combined into a single algorithm in the *ButtonGo.Click* event handler, as shown in the following table in both App Inventor and pseudocode. (**NOTE** that College Board style pseudocode does not contain a ***For each number from 1 to N*** loop. Instead we use its ***REPEAT N times*** loop, which is equivalent in this case.)

Pseudocode

```
WHEN ButtonGo.Click
{
   N  ←  INPUT()
   nHeads  ←  0
```

```
    REPEAT N TIMES
    {
        coin  ←  RANDOM(1,2)
        IF (coin = 1)
        {
          nHeads  ←  nHeads + 1
        }
    }
    DISPLAY( "Heads:", nHeads," Tails:", (N - nHeads) )
}
```



In summary, here is the code for the event handler for ButtonGo.Click:

Event handlers	Algorithms
ButtonGo.Click	Set global N to TextBoxN.Text Set global nHeads to 0 For each number 1 to get Global N Set coin to random integer from 1 to 2 If coin = 1, then increment nHeads (set nHeads to get nHeads + 1) Set LabelResults.text join 4 blocks: Text "Heads:", nHeads, Text " Tails:", N - nHeads.

Testing the App

If you have coded the app as shown here, then whenever *ButtonGo* the app will perform one trial of the experiment consisting of *N* simulated coin flips. Try varying the value for N -- you can try 100, or 1000, or 10,000 or any other value. **NOTE: Be careful with very large values of *N* -- that might take a long time causing your device to become unresponsive, which may even generate a runtime error message from Android.**

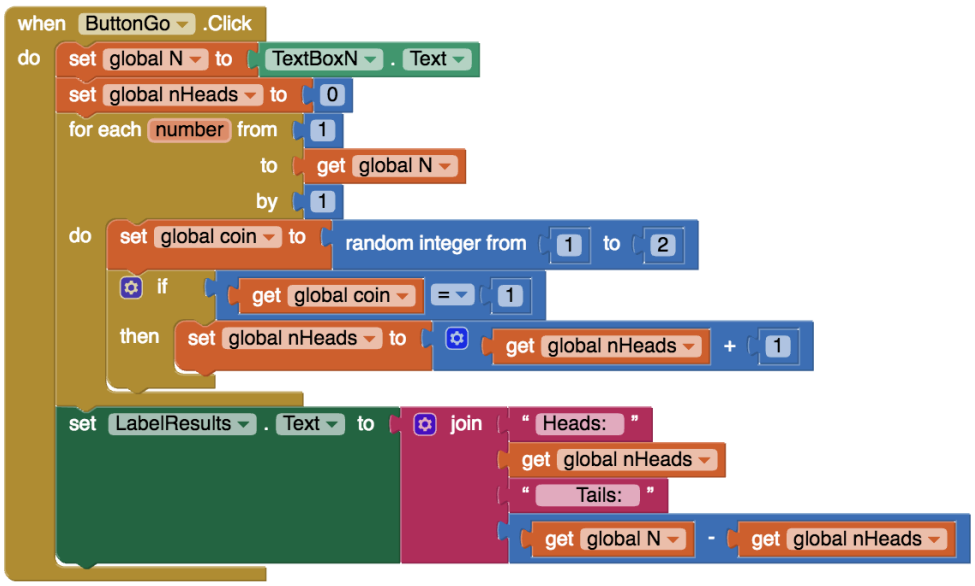
Inputs	Expected Outputs	Actual Outputs
Type in 100 for N and click Run the Experiment.	Results returned should sum up to 100. The results should also be fairly close to 50% heads and 50% tails.	?
Type in 1000 for N and click Run the Experiment.	Results returned should sum up to 1000. The results should be closer to 500 heads and 500 tails.	?

In the next lesson, we're going to use this app to run an experiment designed to test the validity of App Inventor's *random integer* block.

Summary

This app uses a complex algorithm with a loop to model many coin flips.The App Inventor blocks are once again compared to the AP Pseudocode blocks below.

App Inventor Blocks



AP Pseudocode

Pseudocode

```
WHEN ButtonGo.Click
{
   N  ←  INPUT()
   nHeads  ←  0
   REPEAT N TIMES
   {
      coin  ←  RANDOM(1,2)
      IF (coin = 1)
      {
        nHeads  ←  nHeads + 1
      }
   }
   DISPLAY( "Heads:", nHeads," Tails:", (N - nHeads) )
}
```

Vocabulary Review

Review the following new vocabulary in this lesson. If you don't know what these mean, look back through the lesson.

- Random events and numbers
- Pseudo randomness and pseudo random number generators
- Computer model or simulation

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