Monte Carlo Pi Estimation

For full explanation, see http://www.ratracejoe.com/maths/2015/05/25/make-me-a-pi.html

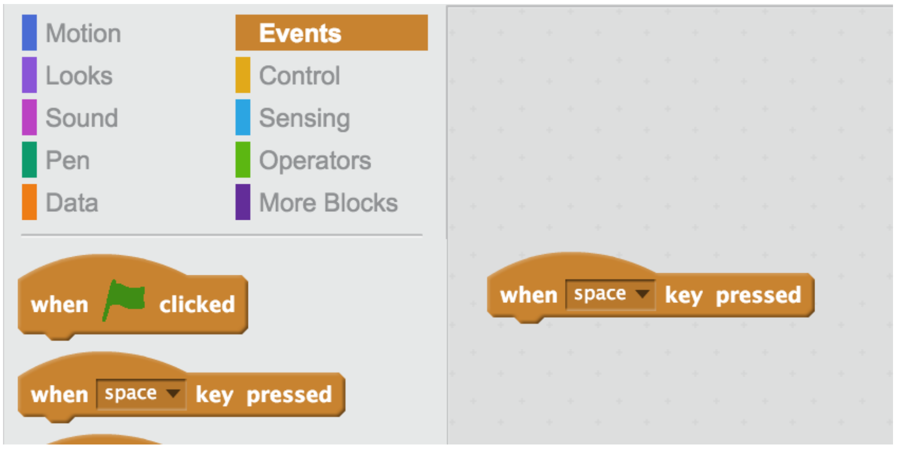
This program will estimate the value of Pi using a statistical method. To start the program, the user will press the <space> bar, the stop the program the user presses <s>.

We will be using the pen tool to draw the generated dots and visualise the estimation process.

Create a new scratch project, which will give a default sprite (the cat). All the code will be attached to the sprite. Although the sprite itself is unimportant.

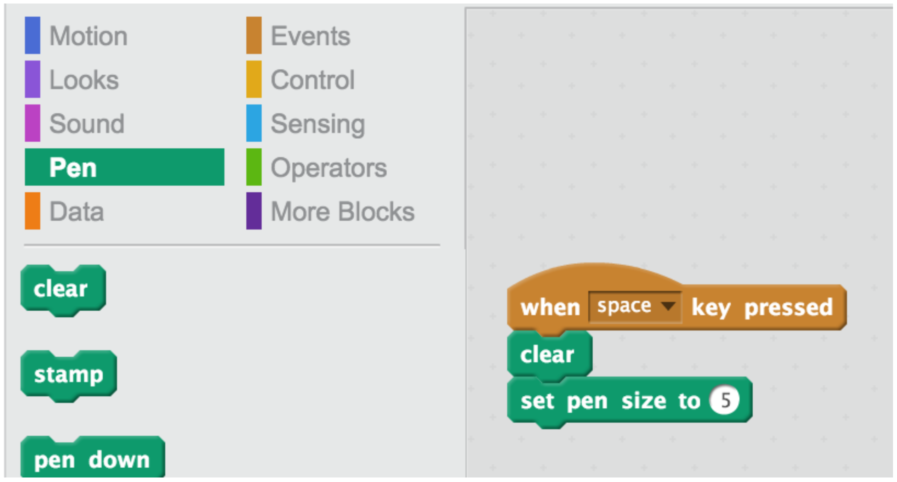
Step 1 - Basic Event Handler, Reset Pen

From **Events**, drag a **When <space> key pressed** block into the code area.



```
when space key pressed
```

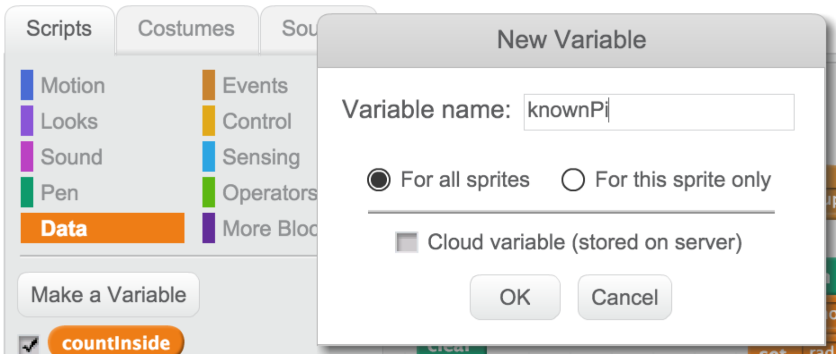
For any fresh run, we should clear any current drawing, so from **Pen**, drag a **Clear** block, and a **Set pen size to <5>** into our event handler.



```
when space key pressed
clear
set pen size to 5
```

Step 2 - Create Variables

Before we can start calculating Pi, we need to create some variables. These will be used to keep track of the estimation process. Under **Data**, click on Make a Variable. Give it the name **knownPi**. This will be used to store the fixed known value for Pi so that we can see how close our estimation is during processing.



Variable name: knownPi

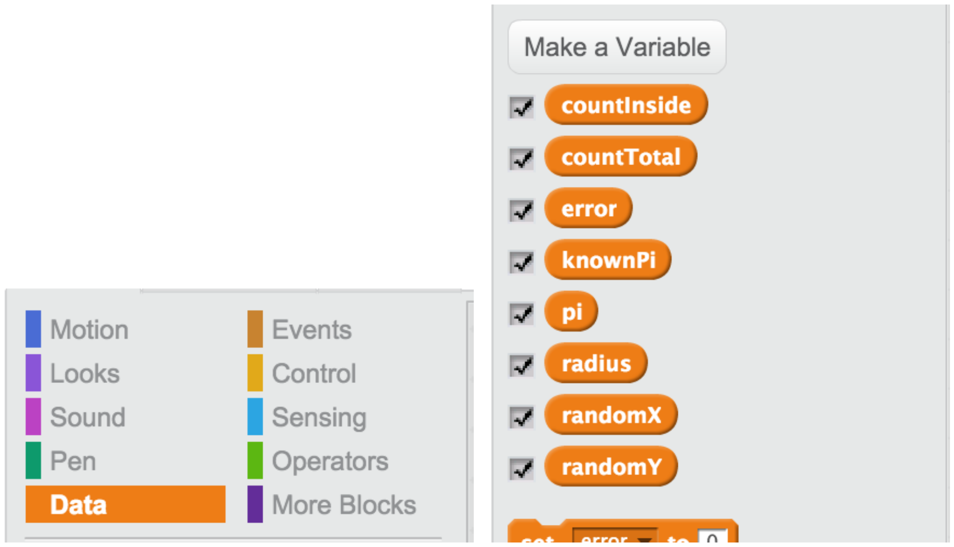
(● For all sprites) ○ For this sprite only

☐ Cloud variable (stored on server)

We will need a few variables, so repeat the variable creation for the following variable names.

- countInside - Will count the number of dots created that are inside the circle
- countTotal - Will count the total number of dots
- error - Will store the percentage error of our estimated value, compared to known Pi
- radius - The radius of the circle (and half the length of the sides of the square)
- pi - The estimated value for Pi, will be updated for every dot created.
- randomX - x-coordinate for next dot
- randomY - y-coordinate for next dot

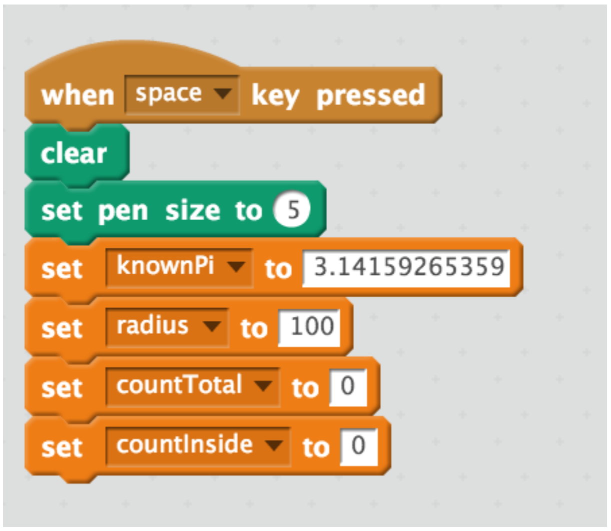
When these are created the **Data** section should look like this.



Step 3 - Initialise Variables

From **Data**, drag a **Set <xxx> to <0>** block into our function 4 times.
Change the variables to set the following

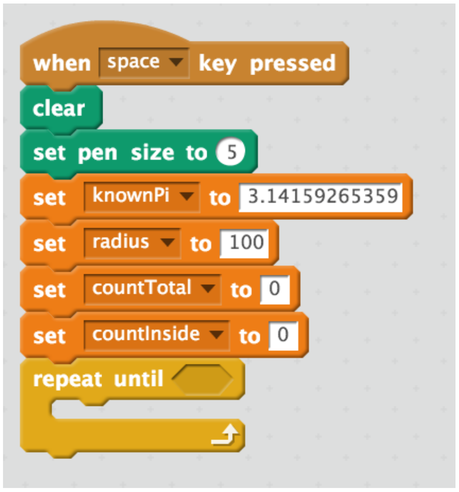
- knownPi = 3.14159265359
- radius = 100
- countTotal = 0
- countInside = 0



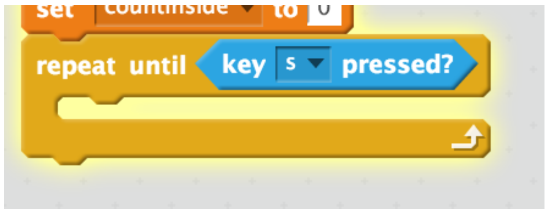
```
when space key pressed
clear
set pen size to 5
set knownPi to 3.14159265359
set radius to 100
set countTotal to 0
set countInside to 0
```

Step 4 - Build Basic Control Loop

From **Control**, drag a **repeat until <>** block into our function.
From **Sensing**, drag a **key <s> is pressed**. Into the repeat condition.



```
when space key pressed
clear
set pen size to 5
set knownPi to 3.14159265359
set radius to 100
set countTotal to 0
set countInside to 0
repeat until <>
```

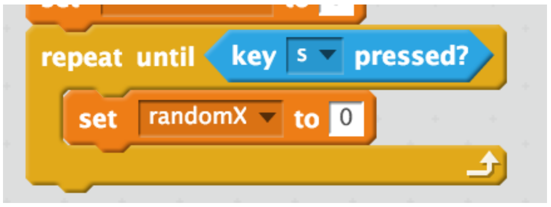


```
set countInside to 0
repeat until <key s pressed?>
```

This gives us a calculation loop that will continue until the user presses <s>.

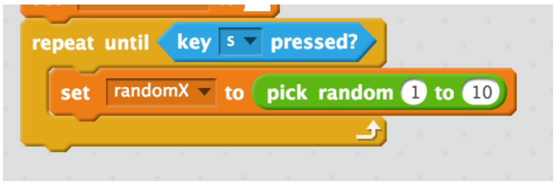
Step 5 - Generate Random Coordinate (x, y)

From **Data**, drag a **Set <randomX> to <0>** into our loop.



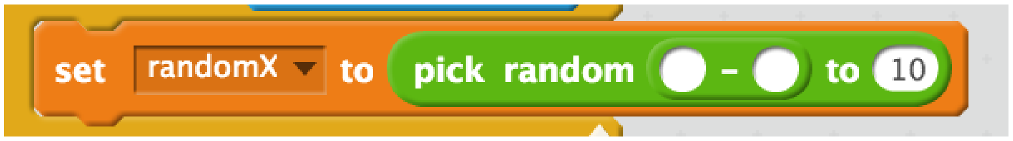
```
repeat until <key [s] pressed?>
    set [randomX] to [0]
end
```

From **Operators**, drag a **pick random <1> to <10>** block into the new setter.



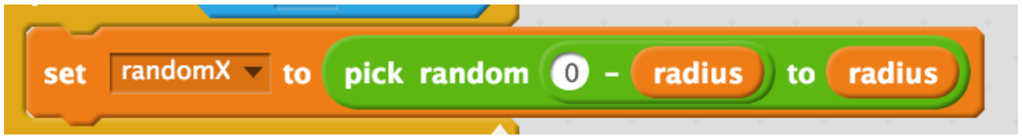
```
repeat until <key [s] pressed?>
    set [randomX] to (pick random (1) to (10))
end
```

The randomiser needs to pick a value between -radius to +radius.
From **Operators**, drag a subtraction block into the first random parameter.



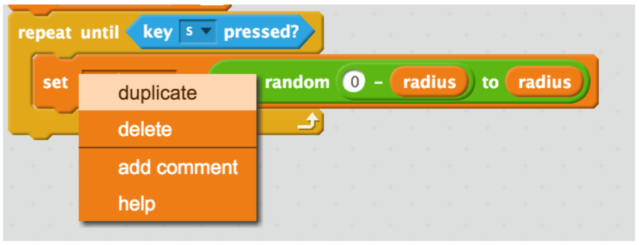
```
set [randomX] to (pick random (() - ()) to (10))
```

Type 0 into the first subtraction parameter, from **Data** drag **radius** into the second subtraction parameter and again into the second random parameter.



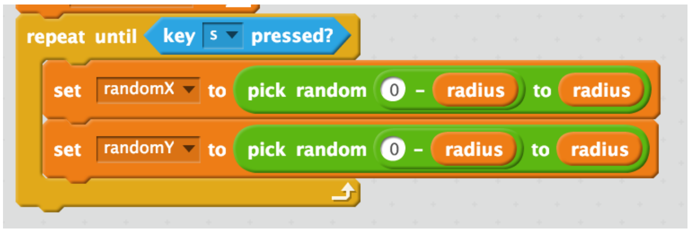
```
set [randomX] to (pick random ((0) - (radius)) to (radius))
```

Random Y will be set in the same way, so we can simply duplicate this new block we have created and change the parameter from randomX to randomY.
Right click on the setter block, and click duplicate.



```
repeat until <key [s] pressed?>
    set ... random ((0) - (radius)) to (radius)
end
```

Drop the duplicate under the first setter, select randomY instead of randomX in the setter.



```
repeat until <key [s] pressed?>
    set [randomX] to (pick random ((0) - (radius)) to (radius))
    set [randomY] to (pick random ((0) - (radius)) to (radius))
end
```

Step 6 - Determine if the new point is inside our circle.

This step uses Pythagoras theorem to determine if the point we have just generated is inside the circle. Pythagoras Theorem gives us the following formula for the length of the longest side of a triangle from the other two sides.

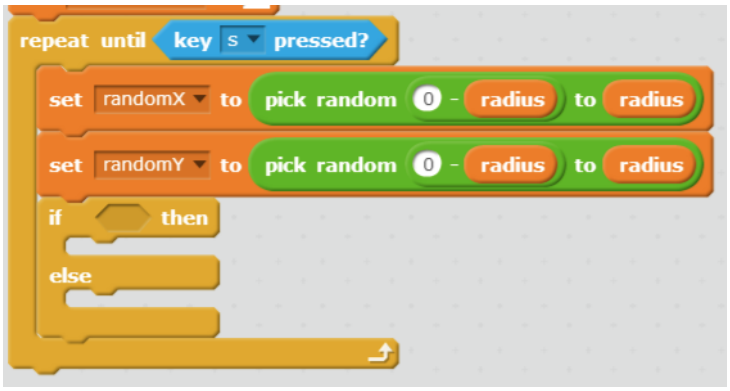
$$h^2 = a^2 + b2$$

Which means

$$h = sqrt\ (a2 + b2)$$

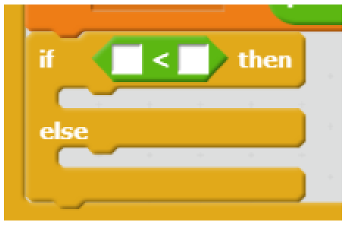
The dot is considered to be *inside the circle* if the h is less than the radius.

First, from **Control** drag an **if <> else** block into our function.



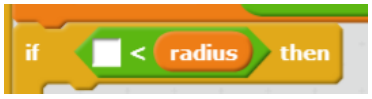
```
repeat until <key s pressed?>
    set randomX to (pick random (0 - radius) to radius)
    set randomY to (pick random (0 - radius) to radius)
    if <> then
    else
```

From **Operators** drag a **less than** block into the if condition.



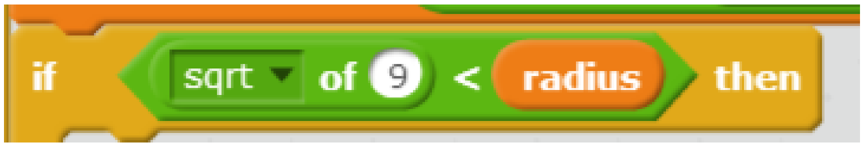
```
if <[] < []> then
else
```

From Data, drag the **radius** variable into the right hand box.



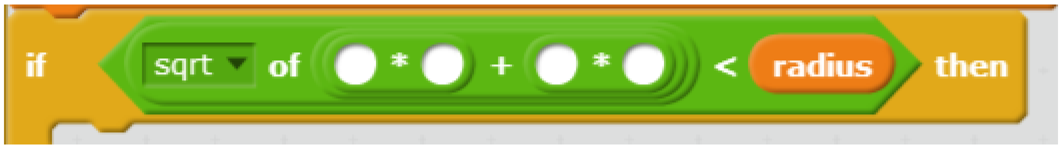
```
if <[] < radius> then
```

The left hand box will contain our Pythagoras calculation, the first thing that is required is a square root. From **Operators** (at the bottom) there is a block that can be used for many different calculations. Drag the **<sqrt> of <9>** block into the left hand box.



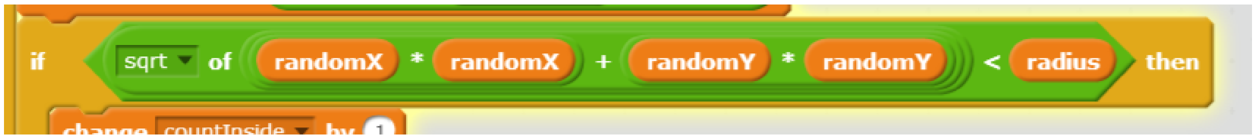
```
if <(sqrt of 9) < radius> then
```

We now need to start building up the expression inside the square root. From **Operators** drag an **addition** block into the sqrt function. Then drag a **multiplication** block into both of the white spaces of the addition block.



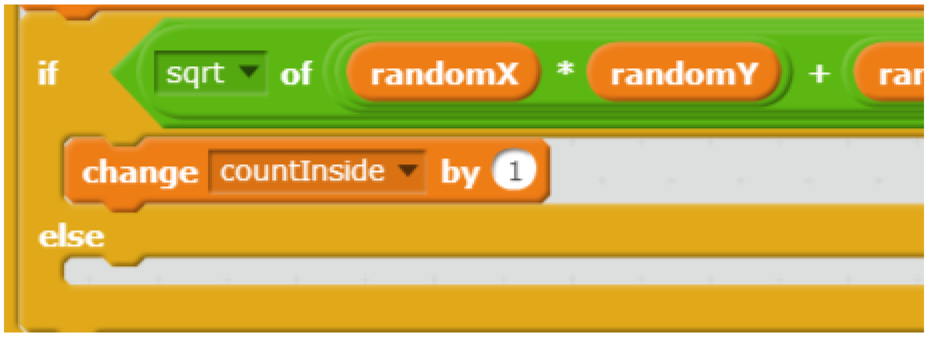
```
if < sqrt of ( ( ) * ( ) + ( ) * ( ) ) < radius > then
```

The left hand multiplication will calculate the square of the x-coordinate. The right hand multiplication will calculate the square of the y-coordinate. From **Data** drag **randomX** into both the white boxes of the left hand multiplication. Then drag **randomY** into both of the white boxes of the right hand multiplication.



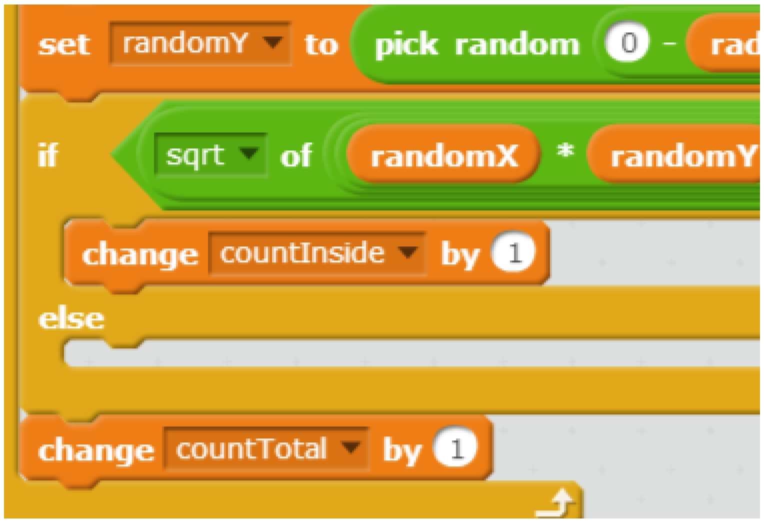
```
if < sqrt of ( randomX * randomX + randomY * randomY ) < radius > then
  change countInside by 1
```

If this condition passes, we need to count the point as being inside the circle. So from **Data** drag a **change <countInside> by <1>** block into the if statement.



```
if < sqrt of ( randomX * randomY + ran
  change countInside by 1
else
```

After this determination has been made, we should count every dot in our **countTotal** variable. So from **Data** drag a **change <countTotal> by <1>** block *after* our if statement.

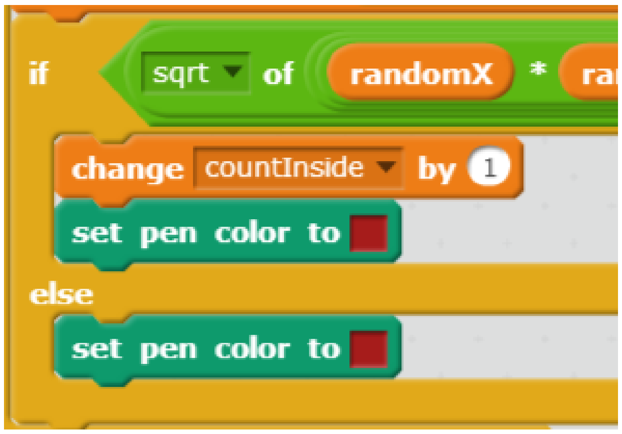


```
set randomY to pick random 0 - rad
if < sqrt of ( randomX * randomY
  change countInside by 1
else
change countTotal by 1
```

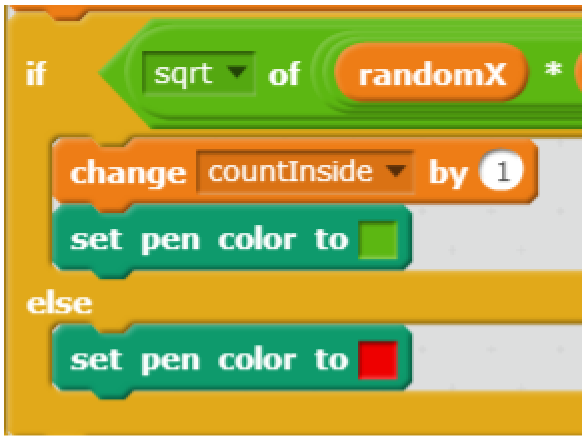
This completes the counting of dots based on being inside the circle. Now we are ready to start drawing!

Step 7 - Draw the point

In order to draw the correct colour, we will use green for inside the circle, and red for outside. From **Pen** drag a **set pen colour to <x>** block into the if statement from the last step, and the else.



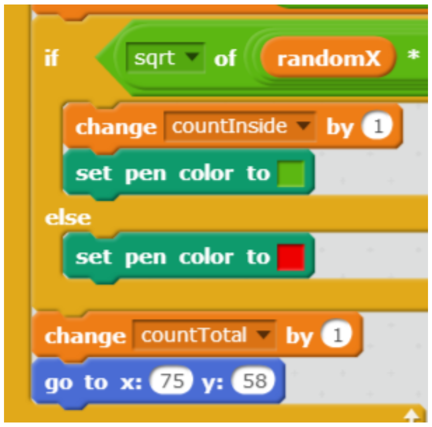
```
if < sqrt of ( randomX * ra
    change countInside by 1
    set pen color to 🟥
else
    set pen color to 🟥
```



```
if < sqrt of ( randomX * 
    change countInside by 1
    set pen color to 🟩
else
    set pen color to 🟥
```

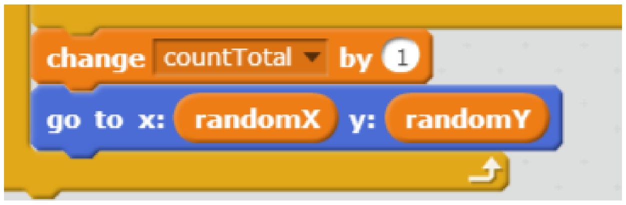
If the dot is inside, change the colour to green, if outside, change to red.

In order to draw the point at the correct place, we must move the sprite to our random coordinates from before. So from **Motion** drag a **go to x: y:** block after we have incremented the **countTotal**.



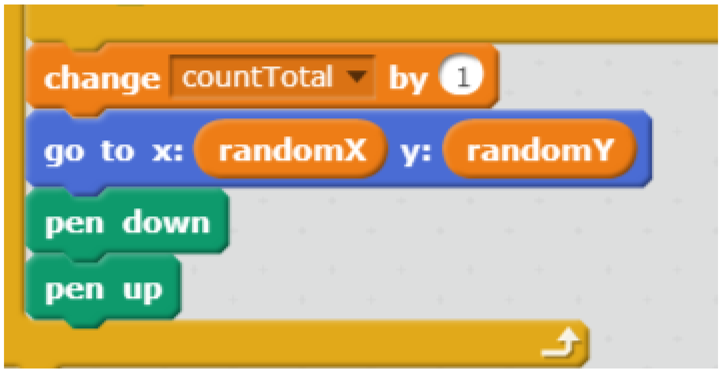
```
if < sqrt of ( randomX *
    change countInside by 1
    set pen color to 🟩
else
    set pen color to 🟥
change countTotal by 1
go to x: 75 y: 58
```

From **Data** drag **randomX** into the x box, drag **randomY** into the y box.



```
change countTotal by 1
go to x: randomX y: randomY
```

In order to draw a single point, we had already set the pen size in an earlier step, we can now just put the pen down, and back up. It works just like a real pen in that it should leave a dot on the page. From **Pen** drag a **Pen Down** and a **Pen Up** block into our function, just after the **goto** block.



```
change countTotal by 1
go to x: randomX y: randomY
pen down
pen up
```

Try running your program now and it should start drawing dots really quickly. If you leave it long enough a green circle should emerge within a red square.

Step 8 - Recalculate Pi and calculate the current error

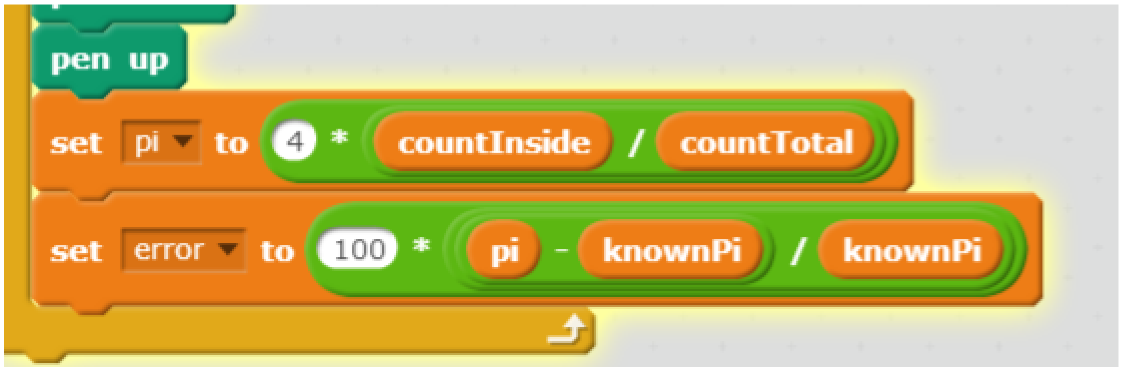
We are nearly there! We just need to calculate Pi, and calculate our error with the known Pi. The full explanation for the maths can be found in the blog linked at the start of these instructions. For now, take it as given that the following is correct.

```
Pi = 4 * (countInside / countTotal)
```

And the error calculation is as follows

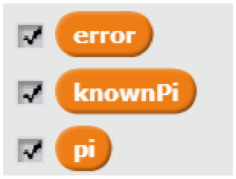
```
% error = 100 * ((pi - knownPi) / pi)
```

We created the pi and knownPi variables in one of the earliest steps. Most of what you need can be found in **Operators** and **Data**. The finished calculations should look like this.



```
pen up
set pi to 4 * (countInside / countTotal)
set error to 100 * ((pi - knownPi) / knownPi)
```

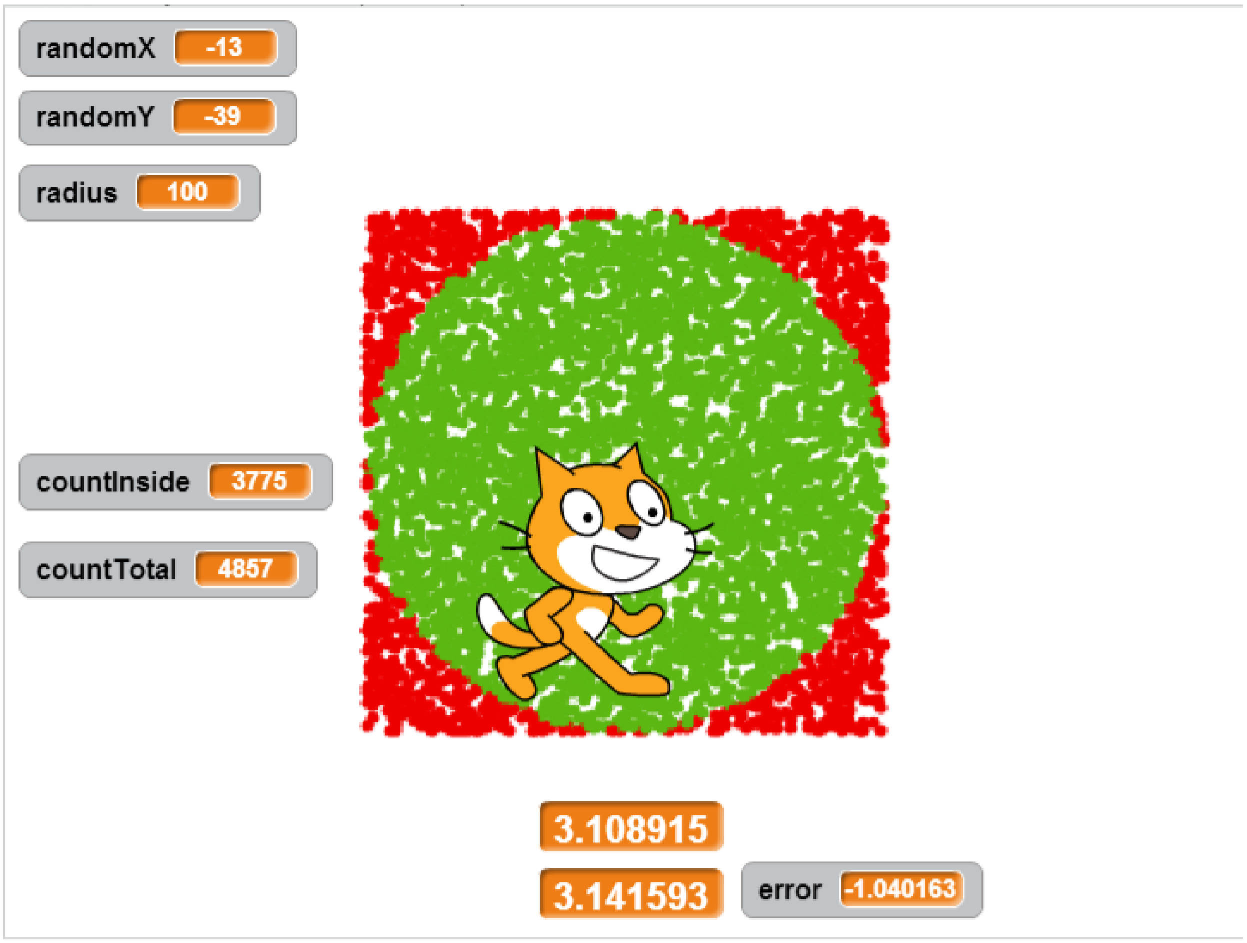
To ensure that pi and error are displayed, under **Data** make sure the tick boxes next to those variables are ticked.



- ✓ error
- ✓ knownPi
- ✓ pi

Step 9 - Finish

We have essentially finished the project now. Try arranging the pi, known pi and error variables in the game area. When the program is running the game area should look something like this.



The cat sprite will jump around painting dots, the estimated pi is shown just below the dots. The real value is shown below that, and the error is next to it.

You can double click on the variables in the game area to change how they are displayed.

CONGRATULATIONS, YOU HAVE FINISHED THE MONTE CARLO PI ESTIMATOR