Project: Board for Snakes and Ladders

Difficulty level: Medium/Hard

Description

In this project you will create the ***board*** for the classic game snakes and ladders.

You can try out the complete version of the project here:

https://goo.gl/TeQW7T (https://goo.gl/TeQW7T)

Use the up, left, right and left arrows to move the cursor. Press the middle button (ENTER) to place your piece.

Project Manual

This project guide will tell you step-by-step the main things you have to do in order to create a *snakes and ladders board*. For some of the steps, you'll have to use your own creativity to proceed, good luck!

Introducing the project

The first thing you should do is open the *skeleton code* for the project.
In programming, skeleton code means code that only has the basic elements of a program. It is up to you to fill in the rest!

You can find the skeleton code on Trinket, here:
https://goo.gl/rXr3vQ (https://goo.gl/rXr3vQ)

If you can, you should also create and account and log in to Trinket. This will allow you to save the Trinket projects. Otherwise you have to copy the code on to your computer to save it.

On Trinket, you'll be able to test your code on a *virtual* Sense HAT, before you try your code on the real thing.

As you might see, the skeleton code is split up into sections, divided by the headlines.
For example:

```
#### 2. Code section
```

This guide will go through the various sections (not necessarily in order), and help you write your code. **Very important note:** *You should add the code in the specified section in your skeleton code as you follow this guide.*
The next part of this guide will explain the stuff that's already in the skeleton code when you first open it.

Explanation of the skeleton code

Before we get on to the coding, it's worth looking over the *skeleton code* and make sure you are familiar with it.

The first few lines in the script are:

```
#### 1.1 Import libraries

import sys
sys.path.insert(1,'/home/pi/Go4Code/g4cSense/graphics')
sys.path.insert(1,'/home/pi/Go4Code/g4cSense/skeleton')
from senseGraphics import *
from snakes_ladders_lib import *
from sense_hat import SenseHat
```

Without going into detail, these lines are called *import statements*. They are used to *import* code from other Python files into your own file. This is useful because you can use other people's code to simplify your own.

The next part of the code (Sec. 1.2) creates some important *Objects* (don't worry if you're not sure what that means) that we'll use later on.

Sec. 1.3 is where the initial variables of the project are set up.

Sec. 2 is where you will set up the board by specifying the board colour, adding snakes, ladders, a player and a cursor.

Sec. 3 is where we'll write code that controls the *cursor*, that places piece on the screen.

Sec. 4 is where we'll write code that changes the checks if the piece is on a snake or ladder.

Understanding the Problem

First let's break down the problem by focussing on all the features that the board should have.

Think about these and list them out below as this will help you plan out your code.

If you are unsure ask a supervisor.

If you have understood the problem and know what the board should do, let's program it!

Writing the code

(Sec. 1.3) Initialise the game

This section is already complete so you don't have to code anything here. The line of code below initialises the Sense Hat and the snakes and ladders game.

```
sense = SenseHat() #initialises the SenseHat
game = snakesLadders(sense) #Initialises the game
```

(Sec. 2.1) Setting the board colour

The colour in which the numbers will be displayed needs to be defined here in *RGB* format.As you might remember, in programming a color is usually represented as *three* numbers *[r, g, b]*. The numbers in the square brackets specify how much red (*r*), green (*g*) and blue (*b*) .By mixing different amounts of red, green and blue light, we can make almost any colour. The amount of red, green or blue you can put in to make a colour goes from 0 to 255. For example, orange is made by mixing 255 red, 127 green and 80 blue. In code this would look like this:

```
#This will set the colour variable to a combination of RGB that gives orange
colour = [255,127,80]
```

Set the variable *board_colour* to the colour you want the board to be. The next line will set the colour of the board to whatever you pick.

Hint: The snakes in the game are set to a red colour and the ladders are a blueish colour. To make sure you can tell all the things on the board clearly, pick a colour that is different from red or blue. Yellow seems to work well. Look up the RGB value for yellow and experiment to find whatever works for you. If you are unsure, ask a supervisor.

(Sec. 2.2) Adding Snakes

In this section we add snakes to the board using the *addSnake()* function on the *game* object. To add a snake onto the board, we must specify the position of the snakes head and the position of the snakes tail. But how do we specify position on the Sense Hat?

Specifying Position (Reminder)

If you know how position and coordinates work on the SenseHat, you might want to skip this section

In programming, we specify positions using coordinates. That might sound complicated but it is actually really easy and useful. The image below illustrates coordinates on the SenseHat.

coordinates on the SenseHat

The *x* coordinate specifies how many squares along horizontally we mean and the *y* coordinate specifies how many squares vertically down we mean. We can use these coordinates to set individual pixels to a particular colour.

For example, if I wanted to set the third pixel along and 4th pixel down to green, I would use the following code:

```
green = [0,255,0] #Defining the colour green in RGB format

###Remember that in programming we count from zero!
sense.set_pixel(2,3,green) #Set the third pixel along and 4th pixel down to
green
```

Try it in the Trinket below:

Trinket Emulator

In the *set_pixel()* function, we first specify how many pixels along we mean and then how many pixels down we mean. In coordinate form this would be written like this: *(2,3)* where the first number, *2*, is the x position and the second number *3*, is the y position.

Back to snakes

To add a snake we use the *addSnake()* function. This function takes 4 arguments (arguments are the values we give the function), the first is the x position of the head of the snake, the second is the y position of the head, the third is the x position of the tail and the fourth is the y position of the tail.

If we wanted to add a snake with it's head at the position (4,1) and tail at (3,3) like in the image below:

We would write:

```
game.addSnake(4,1,3,3)
```

In the *skeleton code* add at a few snakes on to your board. Don't forget to make sure the head of the snake is above the tail!

(Sec. 2.2) Adding ladders

Adding ladders on to the board is also very simple. Just like the *addSnake()* function we have a function to add a ladder on to the board. The following code will add a ladder with foot at position (2,7) and top at position (4,4):

```
game.addLadder(4,4,2,7)
```

(Sec. 2.2) Creating Players

The next lines of code create a player and add it to the game. But this code is incomplete. You need to define the colour of the player's piece.

Set the variable pieceColour to the colour you want the player to be on the board. Remember, colour is defined in RGB format.

```
pieceColour = [ ] #Define the piece's colour
```

(Sec. 2.3) Creating a cursor

The cursor starts in the bottom left corner of the Sense Hat's screen. The two lines below set this position as the cursors starting position. Later in the code you will change these variables to make the cursor move up, down, left and right.

```
cursorPositionX = 0
cursorPositionY = 7
```

In the next line specify the colour of the cursor.

Setting up the Game

The next few lines of code will set up the variables before we get down to the nitty gritty coding in the *main-loop*.

```
game.setBoard() ## Setting the board on to the pixels
image = sense.get_pixels() # Save the screen as it is for now
```

The line above takes all the objects we added into the game (snakes, ladders and players) and displays them on the screen. The second line takes an image of the screen as it is after we set it.

The first line in the code-block below takes gets all the snakes in the game and puts them in a *list* called *snakes*. The second line gets a *list* of ladders. These variables will be used in checking whether the player lands on a snake or ladder by *looping* through all the snakes in the game and *looping* through all the ladders.

```
snakes = game.getSnakes() # Getting list of snakes in the game
ladders = game.getLadders() # Getting a list of ladders in the game
```

(Sec. 3) Main program code

Before we start coding, we'll explain a bit what's going on in Sec. 2.

In this part of the code, we'll do all the main programming.
If we look at the skeleton code, we see that the section is within a *while*-loop.
Remember that code inside a while loop runs again and again, until we tell it to stop. The reason why we want our program to be in a loop is because we want to continually run until the user turns it off.

Because the code is in the *while*-loop, remember that you have to add a *Tab* at every line, to make it indented. Like this:

```
while True:
    # write your code like this,
    # with a tab at the start of the line.
```

Structure

The structure of the *main-loop* might look a bit complicated, but it is split up into three clear sections.

The first section is where the user plays their turn by moving the cursor using the joystick and placing their piece by pressing the middle button.

The second section is where we program the game to check if the player has landed on a snake's head. If the player does land on a snake's head the players piece is moved back to the position of the snakes tail.

The third section is where we code the game to check if the player has landed on the foot of a ladder. If they do, the piece will be moved to the top of the ladder!

(Sec 3.0) Moving the cursor

This part is a bit complicated, so pay attention! Don't be afraid to ask a supervisor if you don't understand something.

This section allows the player to place his piece on the board. It will run in a *while-loop* until the player places the piece with the middle button. The skeleton code should look like this:

```
turnPlayed = False #Defines whether the player has played their turn
while not turnPlayed:

    for event in sense.stick.get_events():

        ##Shows the cursor
        sense.set_pixels(image)
        sense.set_pixel(cursorPositionX, cursorPositionY, cursorColour)

        ## Your code goes in the sections below

        if event.action == "pressed":
            if event.direction == "up":
            ## 3.1 Make cursor go up one square

            elif event.direction == "down":
            ## 3.2 Make cursor go down one square

            elif event.direction == "left":
            ## 3.3 Make cursor go left one square

            elif event.direction == "right":
            ## 3.4 Make cursor go right one square

            elif event.direction == "middle":
                ## 3.5 Set player position to cursor position

                ## Refresh the board
                game.setBoard()
                image = sense.get_pixels()

                turnPlayed = True
```

This code will check if the user has pressed either the *up*, *down*,*left*, *right* or the *middle* button on the joystick. Never mind the complicated structure, try to look at the code and figure out how to use it.

Inside the code you find the comments with hints. Your code should go underneath these comments.

If the user presses *up*, *down*,*left* or *right* on the joystick, the variables *cursorx* and *cursory* should be changed. We'll give you a hint to start off with: If the user presses the up button, cursory should *decrease* by 1.

Bonus: One thing that you might want to add is that if the user pushes the brush off the screen, it should pop back up on the other side of the screen. You can do this using further if-statements.

Finally, if the user presses the *middle* button, the cursor should set the players position to the position of the cursor. To set the players position we use the function setPlayerPosition(x,y). Where x and y are the position we want to put the player at. For example if we were to set player1 to the position (3,1) we would write:

```
player1.setPlayerPosition(3,1)
```

This will set the player position to the pixel that is 3 along horizontally and 1 down vertically.

(Sec 4.0) Check if the piece is on a snake

To see if the piece is landed on a snakes head we must check if the position of the piece is the same as any of the snakes in the board. To do this, we loop through the *list* of snakes and check all of the snakes in the game!

```
for snake in snakes:
    snakeHead = snake.getHead() ## The snakes head
    snakeTail = snake.getTail() ## The snakes tail

     ## 4.1 Insert conditions that checks if player is on a snake
    if *put condition here * and * put second condition here *:

        ## 4.2 Insert what happens if the conditions are met


        ## Refresh board
        game.setBoard()
        image = sense.get_pixels()

        checkingSnakes = True
        break # breaks out of the loop
    else:
        checkingSnakes = False
```

The variable snakeHead is a coordinate like this [x,y]. Where x is how far along horizontally the snake's head is and y is how far down it is.

We can get the x position from the snakeHead variable like this:

```
snakeHead = snake.getHead()
headX = snakeHead[0] ## The x position of the head
```

Similarly, we can get the y position from the snakeHead variable like this:

```
snakeHead = snake.getHead()
headY = snakeHead[1] ## The y position of the head
```

The snakeTail works exactly the same way!

```
snakeTail= snake.getTail()
tailY = snakeHead[1] ## The y position of the tail
```

Optional: The snakeHead variable is actually a list. To get the first element in a list we use square brackets. The number inside the square brackets is the position of the element in the list. The first element is at position 0, the second at 1 and so on.

In the ***if*** statement add your conditions to check if the player has landed on the snake's head. You should replace the lines that say *put condition here*.

Try to think how you would test if two things are at the same position.

(Sec 4.2) Bitten by a snake!

If the player lands on a snake's head, where is the piece meant to go?

Remember, you can set the player's position using the setPlayerPosition() function on the player *object*.

(Sec 5.0) Check if the piece is on a ladder

This section is very similar to the one above. The code does very similar things to the one in the previous section.

Getting the x and y position for the ladderTop and ladderBottom is done using square brackets just like the snake heads.

Try to use the code that you completed above and figure out how to complete this section.

Finished!

If it's all done, correctly, the game should now work! Don't worry if it doesn't, things often go wrong in programming. Errors in code are usually called *bugs*. If you have a bug in your code, you'll have to *debug* it!

If it works, congratulations! You can either move on to another project or try to come up with new things to add to the current project. Use your creativity! You can discuss any ideas you have with a supervisor.

Bonus: Make the game multiplayer

Currently, you only have 1 player in the game. Try to think about how you would change the code to add some more players to the game.

To get a list of players added to the game you can use the following code:

```
players = game.getPlayers()
```

Hint: The solution lies in a loop!

Author: Ishan Khurana

Date: August 15, 2017

Project: Countdown clock

Difficulty level: Hard

Description

In this project you will be creating a simple countdown clock. The user should be able to set a number from 0 to 99, and the clock will tick from that number down to 0.

You can try out the complete version of the project here:

Use the up- and down-keys to select the number, and press the middle joystick button (ENTER), to start counting.

Project Manual

This project guide will tell you step-by-step the main things you have to do in order to create a *countdown clock*. For some of the steps, you'll have to use your own creativity to proceed, good luck!

Introducing the project

The first thing you should do is open the *skeleton code* for the project.
In programming, skeleton code means code that only has the basic elements of a program. It is up to you to fill in the rest!

You can find the skeleton code on Trinket, here:
https://goo.gl/iPQjfU (https://goo.gl/iPQjfU)

If you can, you should also create and account and log in to Trinket. This will allow you to save the Trinket projects. Otherwise you have to copy the code on to your computer to save it.

On Trinket, you'll be able to test out your code on a *virtual* Sense HAT, before you try out your code on the real thing.

As you might see, the skeleton code is split up into sections, divided by the headlines. For example:

```
#### 2. Code section
```

This guide will go through the various sections (not necessarily in order), and help you write your code. **Very important note:** *You should add the code in the specified section in your skeleton code as you follow this guide.*

The next part of this guide will explain the stuff that's already in the skeleton code when you first open it.

Explanation of the skeleton code

Before we get on to the coding, it's worth looking over the *skeleton code* and make sure you are familiar with it.

The first few lines in the script are:

```
#### 1.1 Import libraries

import sys
sys.path.insert(1,'/home/pi/Go4Code/g4cSense/skeleton')

from sense_hat import SenseHat
from senselib import *
```

Without going into detail, these lines are called *import statements*. They are used to *import* code from other Python files into your own file. This is useful because you can use other people's code to simplify your own.

The next part of the code (Sec. 1.2) creates some important *Objects* (don't worry if you're not sure what that means) that we'll use in the later on.

Sec. 1.3 is where the initial variables of the project are set up.
You'll need to edit these values later on in order to customize the it.

Sec. 2 is where you'll let the user set how many seconds the timer should count.

Sec. 3 is where the actual counting happens.

Sec. 4 is where you let the user know that the countdown has finished.

Writing the code

(Sec. 1.3) Set up the variables

In this section we'll be defining some variables that we will use in the project.
A lot of programming is just about knowing what information to store, and where to store it. For example, some very important information to store is the position how many seconds we should count.

In this project we have three variables: *countdown*, *number_colour* and *countdownMessage*. The names are pretty self-explanatory. *countdown* is how many seconds the clock should count down for, *number_colour* is the colour of the numbers on the clock (right now it's black, so you should probably change it!) and *countdownMessage* is the message that should be shown when the clock has finished counting down.

You should change these variables to what you think is appropriate!

(Sec. 3.1) Create a *for*-loop

We're going to skip Section 2 for now. In this section (Sec. 3) we'll create the count down code.

In order to do this, we'll use a *for*-loop. As you heard before, a for-loop can be used for counting things (which is exactly what we want to do!). The following code counts from 10 to 20, in steps of 2

```
number1 = 10
number2 = 20
step = 2

for number in range(number1, number2, step):
    print(number)
```

This should print out:

```
10
12
14
16
18
```

Now, in our case, we want to create a for-loop that counts *down* (rather than up) from the number stored in the *countdown* variable (see Sec. 1.3) to 0.
Using the information I just gave you above, see if you figure out how to

do that.

(Sec. 3.2) Draw the countdown number

Now that you have created a for-loop, you want to write code inside of it. Remember that to write code *inside* of a for-loop, you should press the *Tab* button on your keyboard to *indent* the text. Like this:

```
for ....

  # [Here's some indented code]
```

Note: Just to be clear, this also applies to other types of statements like if-statements, while-loops, and so on.

In this part of the code, you want to draw the number that the clock is currently on. Since you are coding in a loop, you have to remember that this code will run *multiple* times. So the first time it runs the code might draw the number 99, and the next time it'll draw 98, and so on. So in-between all the times you draw the number, you have to remember to *clear the screen* (otherwise all the numbers will be drawn on top of each other.)

To find out how to clear the screen, and to draw the numbers, check the *Function Reference*. You'll have to read through the different functions, and see which ones would let you do these things.

(Sec 3.3) Wait a second

It wouldn't be much of a clock if it didn't actually count *seconds*! Check the function reference again, and look for a function that let you halt the program for a certain amount of time.

After you've completed this, you should be able to run the code! Try it and see if it counts down properly. If it doesn't, there's probably an error somewhere.

(Sec. 2.1) Create a *while*-loop

In this part of the code we're going to skip back to section 2 and write some code that lets the user choose a number between 0 and 99, to count down from. To do this, we'll have to use a *while*-loop.

As you might remember, a while-loop is similar to a for-loop, as it repeats the same code several times. The difference is that a while-loop will repeat again-and-again until *you tell it to stop*.

Go back to the link in the beginning of this document, and check out the example solution on Trinket. See how pressing the middle-button lets the user choose which number to count down from.

It might seem complicated, but this means that we want the while-loop to end when the user presses the middle button (after the while-loop ends, the code will move on to the stuff we wrote in Sec. 3 earlier).

So to complete this section of the code, write the following into (Sec. 2.1).

```
selecting = True

while selecting:
    #rest of code inside ...
```

Later on, to make the program leave the *while*-loop, we would set the variable *selecting* to *False*, like this:

```
  # Somewhere inside the while-loop
  selecting = False
```

(Sec. 2.2) Let the user choose the countdown time

Inside the while-loop, you should write the following:

```
for event in sense.stick.get_events():
    if event.action == "pressed":
        if event.direction == "up":
            # Fill in with your own code

        elif event.direction == "down":
            # Fill in with your own code

        elif event.direction == "middle":
            # Fill in with your own code
```

This code will check if the user has pressed either the *up*, *down* or the *middle* button on the joystick. Never mind the complicated structure, try to look at the code and figure out how to use it.

Inside the code you find the comments *# Fill in with your own code*. Replace these with the code that should be in there. Remember what should happen when the user presses up, down or middle.

(Sec. 2.3) Draw the current countdown time

This is pretty much the same as (Sec. 3.2). Draw the countdown number that the user is currently on.

(Sec. 4) Clock has finished

This is the very last part of the code. Here you should display a message to the user that the clock has run out. Remember that the message is stored in the *countdownMessage* variable in (Sec. 1.3).

See the *Function Reference* to find out how to display a message to the user.

Finished!

If it's all done, correctly, the game should now work! Don't worry if it doesn't, things often go wrong in programming. Errors in code are usually called *bugs*. If you have a bug in your code, you'll have to *debug* it!

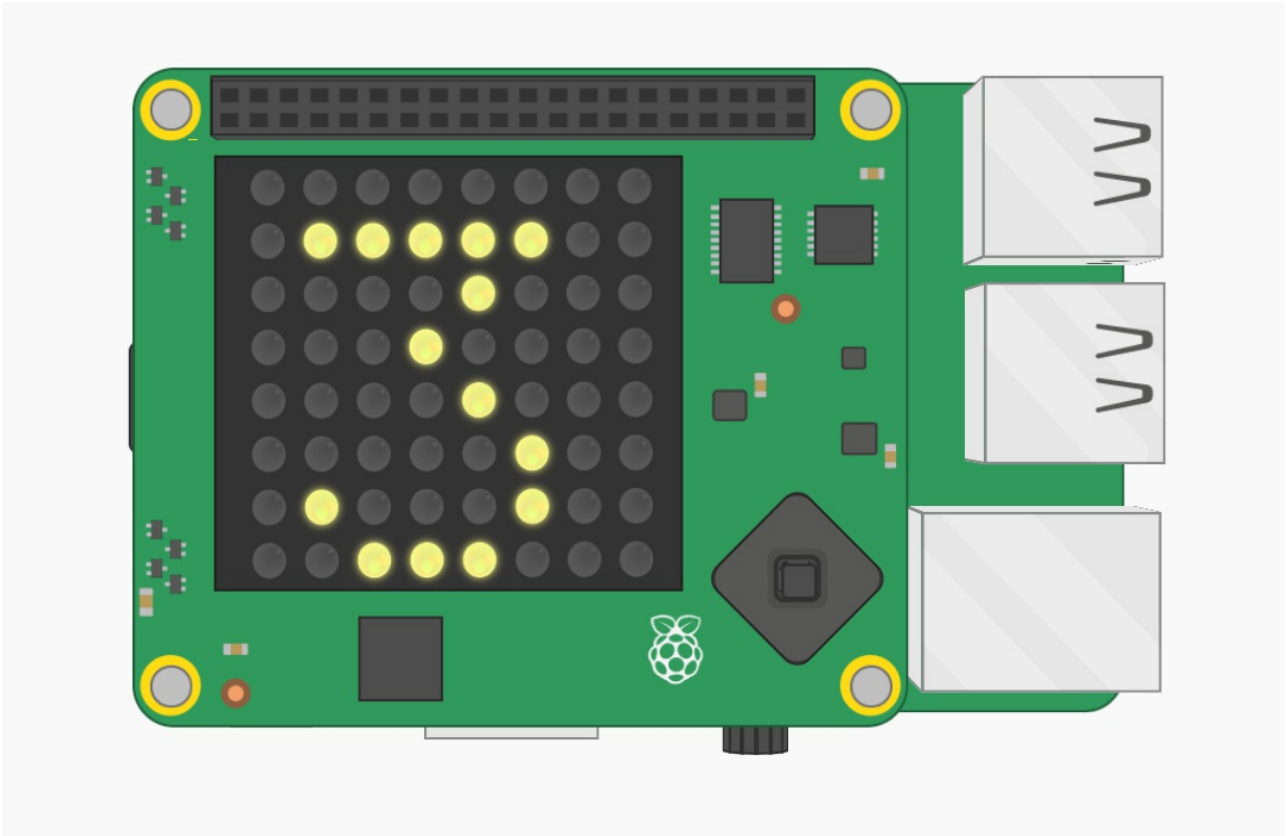
If it works, congratulations! You can either move on to another project or try to come up with new things to add to the current project. Use your creativity! You can discuss any ideas you have with a supervisor.

Author: Lukas Kikuchi

Date: August 09, 2017

Project: Dice

Difficulty level: Easy



Description

In this project you'll be creating dice for the classic board game *Snakes and Ladders* on the Sense HAT.

The aim is to shake the raspberry pi get a random number between 1 and 6 displayed on the screen.

Project Manual

This project guide will tell you step-by-step the main things you have to do in order to create the *Dice*. For some of the steps, you'll have to figure out how to proceed yourself, good luck!

Introducing the project

The first thing you have to do is open the file you'll be programming in. In the *Dice* project folder, open the file called *dice.py*. The code in this file is not enough to actually run Dice, you'll have to fill in the blanks! In programming, we call this *skeleton code*.

As you might see, the skeleton code is split up into sections, divided by the headlines. For example:

```
#### 2. Code section
```

When you set about your task you should let these headlines guide you on what to do in each part of the code. The next part of this guide will explain the stuff that's already in the skeleton code when you first open it.

Note: The text starting with a '#' is called a comment. These comments explain what the code does
to programmers and people reading the code, but is ignored by the computer.

Coding in Trinket

You should write and develop your code on the website https://trinket.io/ (https://trinket.io/). There, you'll be able to test your code on a *virtual* Sense HAT, before you try your code on the real thing. Go on to the website, and create an account.

You should copy all of the code in *dice.py*, as well as the other files in the project folder, on to a new Trinket project.

Explanation of the skeleton code

The first few lines in the script are:

```
#### 1.1 Import libraries

import sys
sys.path.insert(1,'/home/pi/Go4Code/g4cSense/skeleton')
sys.path.insert(1,'/home/pi/Go4Code/g4cSense/graphics')

from sense_hat import SenseHat
from senseGraphics import *
from senselib import *
import random
import time
```

Without going into detail, these lines are called *import statements*. They are used to *import* code from other Python files into your own file. This is useful because you can use other people's code to simplify your own.

The next part of the code (Sec. 1.2) creates some important *Objects* (don't worry if you're not sure what that means) that we'll use in the later on.

Sec. 1.3 is where the initial variables of the project are set up.
You'll need to fill in the lists to customise your project.

Sec. 2 and 3 are where all the main coding will take place. This is the*Main Loop*

Sec. 2 is where we check if the raspberry pi has been shaken.

Sec. 3 is where we display random numbers on the screen.

Writing the code

(Sec. 1.3) Set up the variables

In this section we have some definitions of variables that we'll use in the code later on.

The colour in which the numbers will be displayed needs to be defined here in *RGB* format.As you might remember, in programming a color is usually represented as *three* numbers *[r, g, b]*. The numbers in the square brackets specify how much red (*r*), green (*g*) and blue (*b*) .By mixing different amounts of red, green and blue light, we can make almost any colour. The amount of red, green or blue you can put in to make a colour goes from 0 to 255. For example, orange is made by mixing 255 red, 127 green and 80 blue. In code this would look like this:

```
#This will set the colour variable to a combination of RGB that gives orange
colour = [255,127,80]
```

The next variable is *numbers*. This variable simply contains a list of numbers that the dice can possibly give out. If we had a special dice with 5 faces that could give numbers between 4 and 9, the possible numbers would be given like this:

```
numbers_on_dice = ['5','6','7','8','9']
```

(Sec 1.4) Display a Message

To tell the user that the Raspberry Pi is ready to show some dice, we should display a message.

Add a line of code that will display the words "Dice!" in any colour you want.

(Sec. 2) Main program code

Before we start coding, we'll explain a bit what's going on in Sec. 2.

In this part of the code, we'll do all the main programming.
If we look at the skeleton code, we see that the section is within a *while*-loop.
Remember that code inside a while loop runs again and again, until we tell it to stop. The reason why we want our program to be in a loop is because we want to continually check if the user has shaken the device.

Because the code is in the *while*-loop, remember that you have to add a *Tab* at every line, to make it indented. The indentation tells the computer what is inside the*while*-loop. Like this:

```
while True:
    # write your code like this,
    # with a tab at the start of the line.
```

(Sec 2.1) Calculate and check if the Sense Hat is been shaken.

This part is a bit complicated, so pay attention! Don't be afraid to ask a supervisor if you don't understand something.

To see if the user has shaken the Sense HAT, we'll have to use the *accelerometer* inside the device. Acceleration is just the rate at which the speed of something changes. The accelerometer basically tells us how fast the device has been accelerating.

To get the information from the accelerometer, use the following code:

```
ac = sense.get_accelerometer_raw()
x = ac["x"]
y = ac["y"]
z = ac["z"]
```

Now, in the *x*, *y* and *z* variables, we have information recorded by the accelerometer. This is all a bit complicated, but the basic gist of it is this, calculate this:

```
shake = x*x + y*y + z*z
```

We have stored *x times x plus y times y plus z times z* into the variable *shake*. Now, the larger the variable *shake* is, the more has the Sense HAT been shaken.
We're going to say that if *shake* is *larger* than *5*, the Sense HAT has been shaken. To do this, you're going to have to use an *if*-statement.

If you have detected that the Sense HAT has been shaken (in other words, if *shake* is larger than 5) you want to start displaying the numbers. This will be done in another *while* loop as explained in the next section.

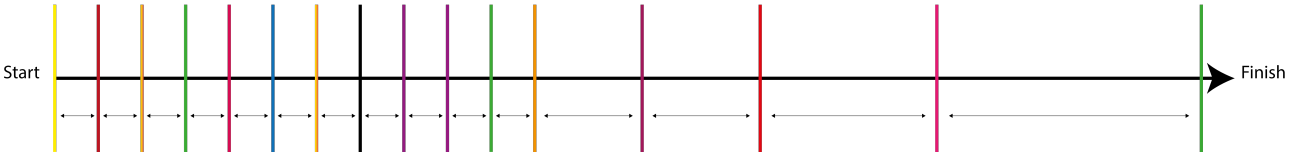
(Sec 3) Displaying numbers on the screen.

Programming has two parts to it. First understanding the problem and finding a step by step way to solving it ***(The Algorithm)*** and then telling the computer that solution in the language it can understand ***(The code)***

Understanding the Algorithm

In the dice we are trying to make, the Sense Hat changes the number on the screen after a time interval for 5 seconds. As the we get closer the 5 seconds the time interval starts getting bigger. If we were to have a timeline of the events happening on the SenseHat it would look like this:

The coloured vertical lines represent each time the Sense Hat changes the number. As we get closer to the finish the spacing between these events increases.



The coloured vertical lines represent an event i.e. the number on the display screen changing. Near the start, these changes happen very quickly! But as we get closer to the finish, the time step starts increasing and the numbers start slowing down on the SenseHat.

- At each vertical line, the Raspberry Pi randomly selects a number from the choice of possible numbers we specified earlier.
- Then displays this number on the screen in whatever colour we give.
- After that it waits until it is time to change the colour again.
- It then increases the time between events.

This continues in a *loop* until the time between events is greater than 1 second. After this we get the final number.

The Code

Now that we hopefully understand how to solve the problem, having broken it down a bit. Let's fill in the sections and code the solution!

We have first *initialised* some variables that are used to control the dice.

```
  #### Set Dice Variables
  time_between_changes = 0.01 #This is the minimum time between events.
  start = time.time() #This line makes the Raspberry Pi note the start of the
dice roll.
```

The next variable we must define is the *run time* which is the amount of time (in seconds) that the Sense Hat will show random numbers before stopping at a final one.

The next line starts the*while* loop.

(Sec 3.1) Randomly selecting a number

In*Python,* random selection from a list is actually really easy. The code below demonstrates how we can randomly select from a list.

```
  list = ['a','b','c',d]
  random_letter = random.choice(list)
```

If you were to run that code and print the *random_letter* variable, you would get a letter randomly selected from the list given above.

Now figure out how to randomly select a number and show it on the screen. To show characters on the screen use the function *sense.show_letter()*. Have a look at code below:

```
yellow = [255,255,0] ## Yellow is made of 255 red and 255 green.
blue = [0,0,255]
sense.show_letter('8', text_colour = yellow, back_colour = blue)
```

The first two lines define the colours yellow and blue in *[r,g,b]* format. The next line shows the character '8' on the Sense Hat's screen with the text colour in yellow and the back colour in blue.

(Sec 3.2) Waiting to change the number

Here we make the Pi wait until it is time to select a new number. This is the time between events.

To a program wait for a certain amount of time, we simply tell it to sleep! The code below tells the program to sleep for a given number of seconds:

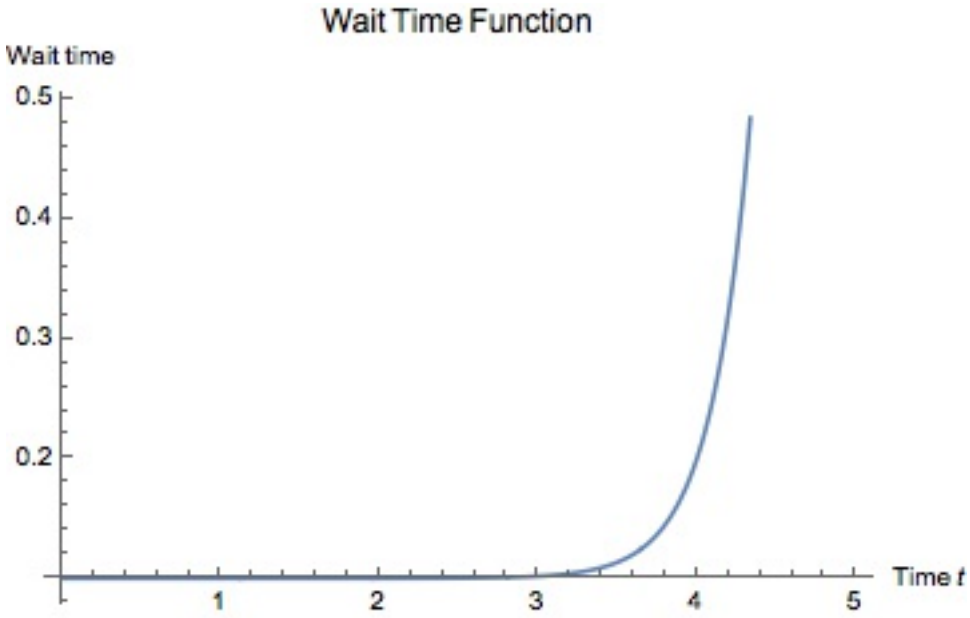
```
sleepTime = 2
time.sleep(sleepTime)
```

This will make the program wait 2 seconds before moving on to the next line of code. Use this to implement the time interval between the changing of numbers.

Increasing the wait time

Reading this is optional

The time step between the changing of numbers increases very slowly right until we get close to the end of the run time. This is done by increasing the time step by almost nothing and then suddenly increasing it nearer to the end time. This means that as we get closer to the end time the rate at which the numbers change starts going down because the wait time increases. Right up until it is bigger than a second, which is when the loop stops! This is what the time step looks like in graphical form.



We have implemented this in the background using some clever mathematics!

(Sec 3.3) Showing the final number

Pick a text and back colour and show the user the final number!

After this Raspberry Pi will go back back to the top of the *Main*-loop and start checking for a shake again.

If everything went well, you will have made dice! If you want, try implementing some of the features suggested below.

Additional features

These are optional

In this section we suggest some additional features you could implement on your project. We encourage you to come up with your own features that you can implement with the help of the supervisors.

- Display the numbers in random colours. *Hint:* Use the randomColour() function.
- Make the amount of time the dice runs for increase with how much you shake it. *Hint:* Make sure not to let it run forever!
- Make the final number blink. *Hint* You will need another loop.

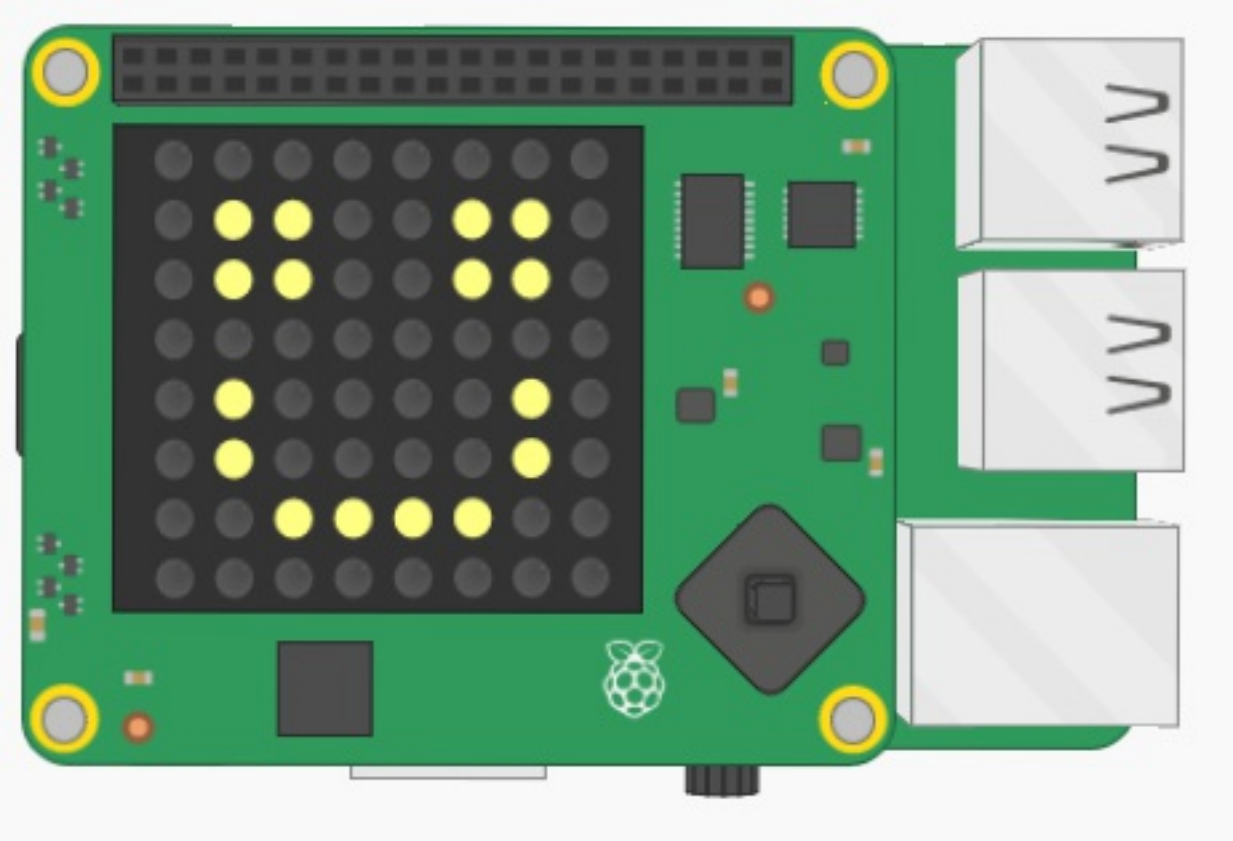
Ask your group mentor for help if you have an idea!

End

Author: Ishan Khurana
Date: August 09, 2017

Project: Draw a picture

Difficulty level: Easy



Description

In this project you'll learn how to draw a picture on the Sense HAT.

You can try out the complete version of the project here:

Press the middle joystick button (ENTER), to make the smiley face wink. As a bonus, you can challenge yourself by adding a similar feature to your code.

Project Manual

This project guide will tell you step-by-step the main things you have to do in order to create a draw a picture on the Sense HAT. For some of the steps, you'll have to use your own creativity to proceed, good luck!

Introducing the project

The first thing you should do is open the *skeleton code* for the project.
In programming, skeleton code means code that only has the basic elements of a program. It is up to you to fill in the rest!

You can find the skeleton code on Trinket, here:
https://goo.gl/mBxSCg (https://goo.gl/mBxSCg)

If you can, you should also create and account and log in to Trinket. This will allow you to save the Trinket projects. Otherwise you have to copy the code on to your computer to save it.

On Trinket, you'll be able to test your code on a *virtual* Sense HAT, before you try your code on the real thing.

As you might see, the skeleton code is split up into sections, divided by the headlines. For example:

```
#### 2. Code section
```

This guide will go through the various sections (not necessarily in order), and help you write your code. **Very important note:** *You should add the code in the specified section in your skeleton code as you follow this guide.*

The next part of this guide will explain the stuff that's already in the skeleton code when you first open it.

Explanation of the skeleton code

Before we get on to the coding, it's worth looking over the *skeleton code* and make sure you are familiar with it.

The first few lines in the script are:

```
    #### 1.1 Import libraries

    import sys
    sys.path.insert(1,'/home/pi/Go4Code/g4cSense/skeleton')

    from sense_hat import SenseHat
    from senselib import *
```

Without going into detail, these lines are called *import statements*. They are used to *import* code from other Python files into your own file. This is useful because you can use other people's code to simplify your own.

The next part of the code (Sec. 1.2) creates some important *Objects* (don't worry if you're not sure what that means) that we'll use in the later on.

Section 2 is where you'll actually be coding.

Writing the code

(Sec. 2) Draw a picture

Use your own imagination to draw a picture on the Sense HAT screen.

The screen is composed of 8 *rows*, where each row has 8 pixels each. So in total we have 8*8=64 pixels. We often refer to which *row* a pixel is at as its *y-coordinate*, and which column is at as its *x-coordinate*.

To draw a pixel, you should use the *sense.set_pixel* function. To see how to use this function, check out the *Function Reference*.

As a simple example, the following code draws a red box:

```
    sense.set_pixel(0, 0, 255, 0, 0)
    sense.set_pixel(0, 1, 255, 0, 0)
    sense.set_pixel(1, 0, 255, 0, 0)
    sense.set_pixel(1, 1, 255, 0, 0)
```

Try copying the code, and then play around with it by changing the numbers in the function.

The last three numbers in the function call decides which colour the pixel should be drawn in. They go in the order of *red*, *blue*, *green*. Where 255 is the maximum red/green/blueness, and 0 is the lowest.

If this sounds confusing try playing around with the colour values and try to get a hang of it. By mixing the red, blue and green colours you can get any of the other colours (for example, purple is blue mixed with red).

```
# Draws a purple pixel
sense.set_pixel(2, 2, 255, 0, 255)
```

After you're done, run the code, and your picture should appear on the screen!

Finished!

If it's all done, correctly, the game should now work! Don't worry if it doesn't, things often go wrong in programming. Errors in code are usually called *bugs*. If you have a bug in your code, you'll have to *debug* it!

If it works, congratulations! You can either move on to another project or try to come up with new things to add to the current project. Use your creativity! You can discuss any ideas you have with a supervisor.

Bonus: Add a "blinking"-feature (Very hard)

Note: This part is completely optional, as it's very hard.
If this is the first project you have attempted, you should come back to this after you've
had your go at some other projects first.

In the example solution at the top of the document, you might've noticed that the smiley-face would *blink* if you hold in the middle joystick button.

Although it's fairly difficult, you can try adding this feature to your project. We'll give you the ingredients you'll need in order to make this happen, but the rest is for you to figure out.

If you drew something other than a smiley-face, you might want to get your picture to go through some other change. It's entirely up to you!

Main loop:

The first thing you'll need is to make all the code be in a main loop, using a while loop.

```
while True:
    # more code
```

Check if the user has pressed a button:

Read the *Checking the joystick* part of the Function Reference to see how to see if the user has pressed the middle joystick-button.

If the user has pressed the button, you should change the picture. In the smiley-face case, we make the program do this change by setting a *boolean* value to True, like this:

```
isBlinking = True
```

Then, in the drawing part of the program, we have to use an *if*-statement to check if the program should draw the winking smiley-face, or the normal smiley-face.

Clearing the screen:

You have to remember to use *sense.clear()* to clear the screen at the end of each loop, otherwise the drawings will overlap!

Author: Lukas Kikuchi

Date: August 03, 2017

Function reference
==========

The following are some functions that you can use in your programming.
You can use this document as a reference during your work.

drawHorizontalLine(position, size, color)

Draw a horizontal line

Draws a horizontal line on the screen. You have to specify the position, size and color of the line. For example here's some code that draws a horizontal line that is three pixels wide, lies at the position (2, 3), and is purple:

```
drawHorizontalLine( (2, 3), 3, (255, 0, 255) )
```

Another way to do it would be

```
pos = (2, 3)
size = 3
color = (255, 0, 255)
drawHorizontalLine(pos, size, color)
```

drawVerticalLine(position, size, color)

Draw a vertical line

Draws a vertical line on the screen. You have to specify the position, size and color of the line. For example here's some code that draws a horizontal line that is three pixels long, lies at the position (2, 3), and is purple:

```
drawVerticalLine( (2, 3), 3, (255, 0, 255) )
```

Another way to do it would be

```
pos = [2, 3]
size = 3
color = [255, 0, 255]
drawVerticalLine(pos, size, color)
```

wait(seconds)

Halt the program for some time

Will halt the program for the amount of seconds that you specify. For example to wait for 5 seconds:

```
wait(5)
```

draw_two_digit_number(number, red, green, blue)

Draws any one-digit or two-digit number

Draws any number from 0 to 99 on the screen. You can also specify the color of the number. For example, to draw 45 in the colour red:

```
draw_two_digit_number(45, 255, 0, 0)
```

sense.clear()

Removes all drawings on the screen

If you want to clear all drawings on the screen (reset all pixels to black), you use this function.

```
sense.clear()
```

sense.set_pixel(positionx, positiony, red, green, blue)

Sets a pixel to a color

This function can be used to draw things pixel-by-pixel on the screen. *positionx* and *positiony* is the x and y position of the pixel to draw. If you don't know what that means, the x-position is how many squares the ball is away from the left side of the screen, and the y-position is how many squares the ball is away from the top side of the screen. The following code will draw a yellow pixel at position (5,1).

```
pos = [5, 1]
color = [0, 255, 255]
sense.set_pixel(5, 1, color[0], color[1], color[2])
```

sense.show_message(message, time)

Displays a text message on the screen

You can use this function to display a message on the screen. You can also decide how fast the message will scroll past the screen. The following displays the message "Hello World!" on the screen over 2 seconds.

```
sense.show_message("Hello world!", 2)
```

Checking the joystick

You can let the user control the program using the joystick. The way to do this can seem a bit complicated, but read this section to get the hang of it. If you don't understand something, ask a supervisor!

It's easiest to learn through an example. The following displays "You pressed up!" if the user pressed up on the joystick.

```
for event in sense.stick.get_events():
  if event.action == "pressed":
    if event.direction == "up":
      sense.show_message("You pressed up!")
```

That's a lot of code, and you probably don't know what it all means. Don't worry about it! By looking at the code you can modify it for your own purposes. To change it so that it checks whether a button is released, rather than pressed, change the second line in the code snippet to

```
if event.action == "released":
```

Quite simple right? Similarly, if you want to check the down-button rather than the up-button, change the third line to

```
if event.direction == "down":
```

And you can change it to "left" or "right" to check the left or right buttons.

As a final example, here's some code that checks all the 5 different buttons (up, down, left, right, middle). Feel free to use this code directly in your program

```
for event in sense.stick.get_events():
  if event.action == "pressed":

    if event.direction == "up":
      sense.show_message("You pressed up!")

    if event.direction == "down":
      sense.show_message("You pressed down!")

    if event.direction == "left":
      sense.show_message("You pressed left!")

    if event.direction == "right":
      sense.show_message("You pressed right!")
```

Note that these code snippets should be in the *main loop* of your program.

random.randint(minNumber, maxNumber)

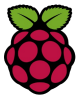
Gives a random number between two specified number

To get random number between 1 and 10:

```
random.randint(1, 10)



---
```

Author: Lukas Kikuchi

Date: August 09, 2017



Graphics on the Sense HAT

Go4Code

Project: Glitter Lights

Difficulty: Easy

In this project we will be making use of the Raspberry Pi's display panel to make some glittery lights. Ask your supervisor to show you what they should look like!

Introducing the project

The first thing you should do is open the *skeleton code* for the project.
In programming, skeleton code means code that only has the basic elements of a program. It is up to you to fill in the rest!

You can find the skeleton code on Trinket, here:
Glitter Project (https://trinket.io/python/77b6a432af)

If you can, you should also create an account and log in to Trinket. This will allow you to save the Trinket projects. Otherwise you have to copy the code on to your computer to save it.

On Trinket, you'll be able to test your code on a *virtual* Sense HAT, before you try your code on the real thing.

As you might see, the skeleton code is split up into sections, divided by the headlines.
For example:

```
#### 2. Main Loop
```

This guide will go through the various sections (not necessarily in order), and help you write your code. **Very important note:** *You should add the code in the specified section in your skeleton code as you follow this guide.*

The next part of this guide will explain the stuff that's already in the skeleton code when you first open it.

Explanation of the skeleton code

Before we get on to the coding, it's worth looking over the *skeleton code* and make sure you are familiar with it.

The first few lines in the script are:

```
#### 1.1 Import libraries

import sys
sys.path.insert(1,'/home/pi/Go4Code/g4cSense/skeleton')

from sense_hat import SenseHat
from senselib import *
```

Without going into detail, these lines are called *import statements*. They are used to *import* code from other Python files into your own file. This is useful because you can use other people's code to simplify your own.

The next part of the code (Sec. 1.2) creates some important *Objects* (don't worry if you're not sure what that means) that we'll use in the later on.

Section 2 is where you'll actually be coding.

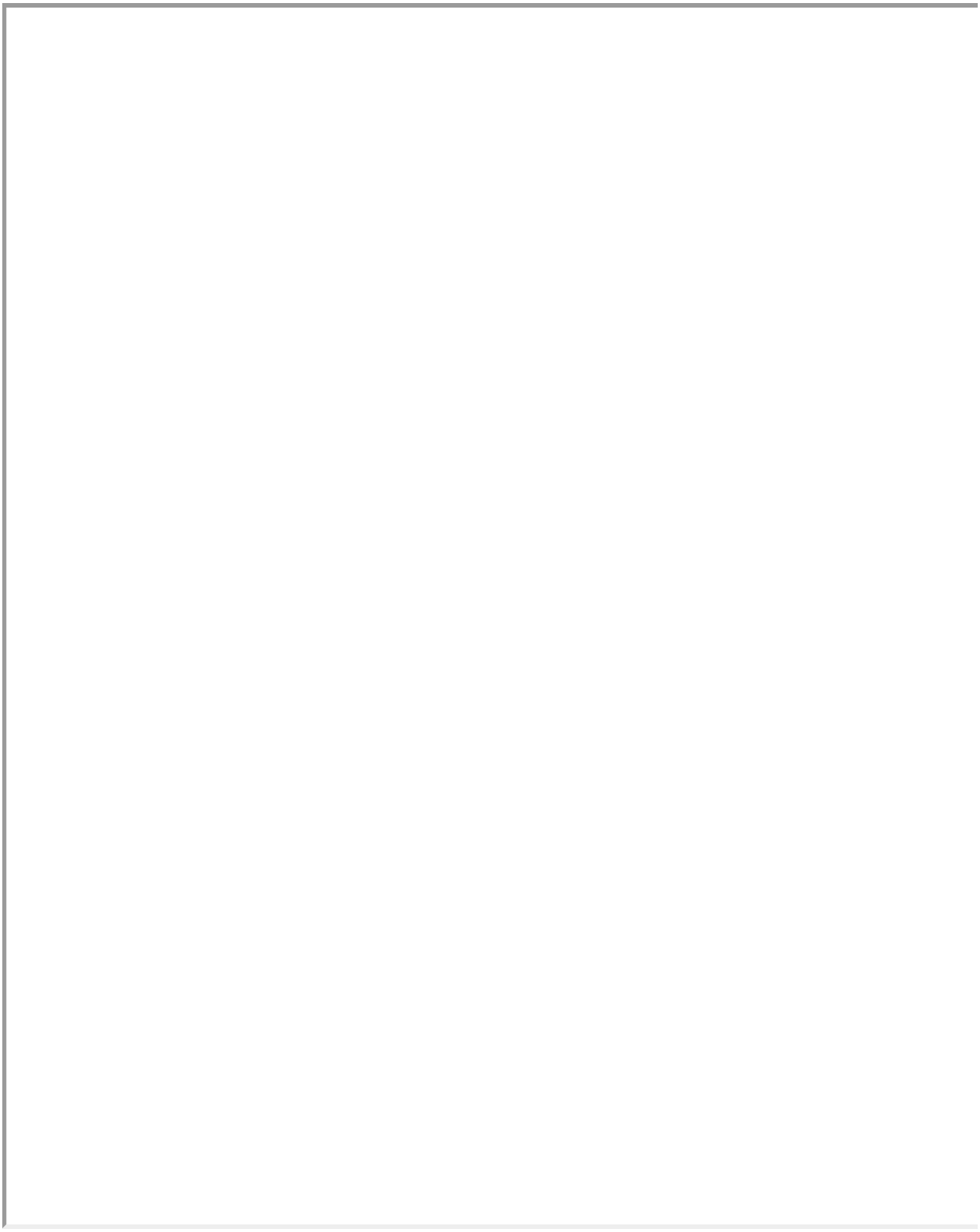
Setting Pixels on the Sense HAT (Recap)

This section is meant to remind you how to set all the pixels on the Sense HAT using a list. If you already know how to do this from a previous project, you may skip this section.

The Sense HAT allows us to use the *set_pixels* function to set all the pixels to certain colours as defined by in a *list*. This is better understood with an example. The code below defines a *list* with 64 values. Each value is a colour. The *list* tells the Raspberry Pi what colour each pixel will be set to.

```
r = [255,0,0]

image = [
  r,r,r,r,r,r,r,r, #This is the first row of pixels
  r,r,r,r,r,r,r,r, #The second
  r,r,r,r,r,r,r,r, #The third
  r,r,r,r,r,r,r,r, #And so on..
  r,r,r,r,r,r,r,r,
  r,r,r,r,r,r,r,r,
  r,r,r,r,r,r,r,r,
  r,r,r,r,r,r,r,r,
]
```

Let's try the code above in the Trinket. ***Read the comments for explanations!***



Trinket Emulator

Making a picture

The *set_pixels* function is actually really useful as it let's us make images really easily. The *list* essentially acts as a drawing board that resembles the LED matrix on the Sense HAT. For example if I take the 3rd element along in the first row and set that to yellow in the *list* it will set the 3rd pixel along in the first row to orange.

Remember: In programming, we count from zero not one!

```
r = [255,0,0]
y = [255,255,0]
image = [
  r,r,r,y,r,r,r,r, #Setting the third pixel along to yellow
  r,r,r,r,r,r,r,r, #The second
  r,r,r,r,r,r,r,r, #The third
  r,r,r,r,r,r,r,r, #And so on..
  r,r,r,r,r,r,r,r,
  r,r,r,r,r,r,r,r,
  r,r,r,r,r,r,r,r,
  r,r,r,r,r,r,r,r,
]
```

Let's try it out in the Trinket below:

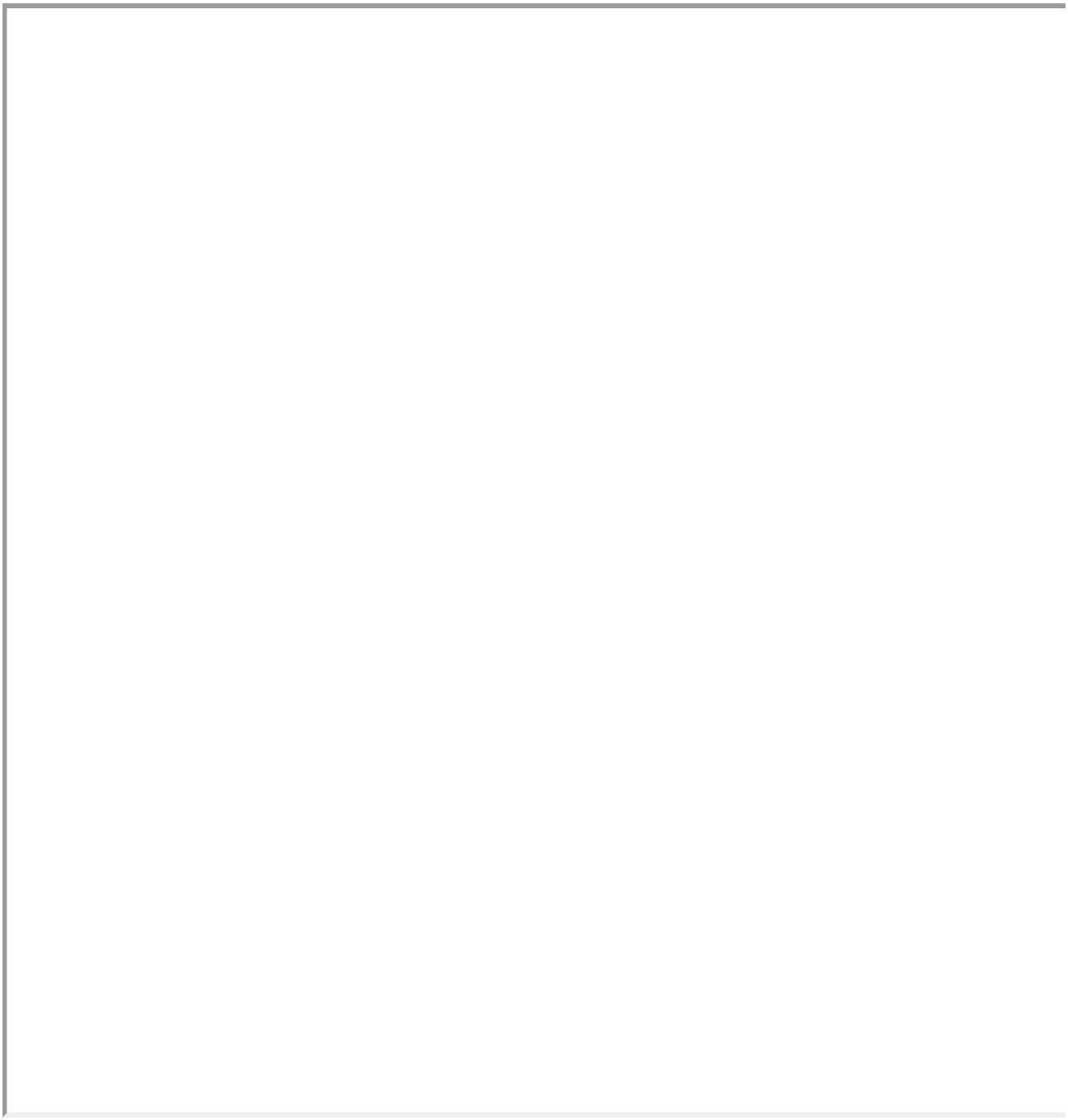
Trinket Emulator

Great! So now that we can see how to use *lists* and the *set_pixels* function to set all the pixels at the same time. We can make some images!

If you are unsure about how the code above works ask your supervisor to explain it to you

Making a plus sign

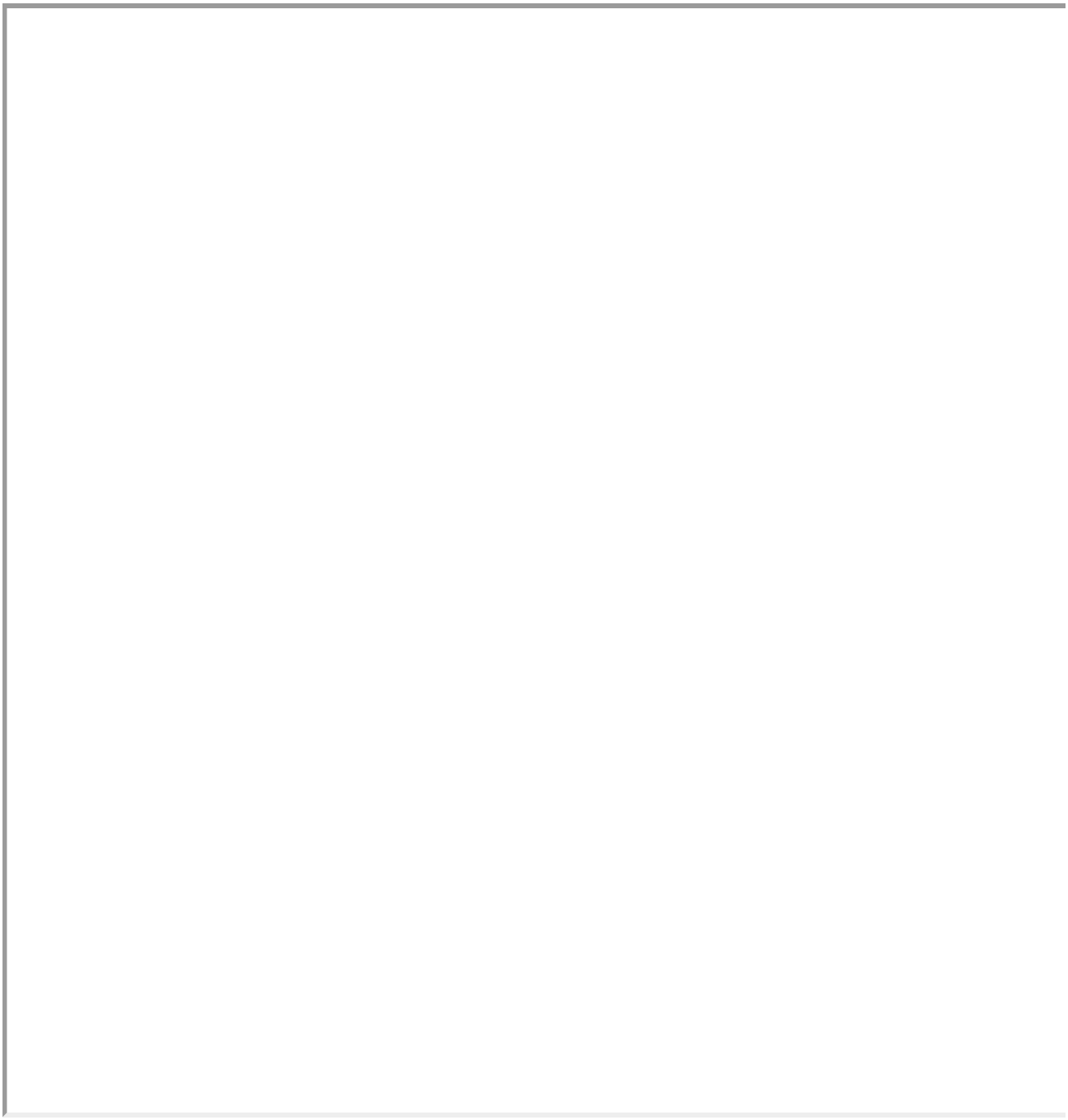
Let's try making a simple image first. We're going to make a plus sign as shown in the Trinket below:



Trinket Emulator

To make that plus sign, we used the *set_pixels* function and a *list* called image as a drawing board to make the drawing.

Exercise 1: Change the values of the *list* in the Trinket below to make a plus sign:



Trinket Emulator

The Raspberry Pi Logo

We can add any colour to our *list*. We can also add no colour to a pixel, this would mean that pixel would be off!

By setting one of the elements to *e = [0,0,0]* we can turn the corresponding pixel off. Using this and what you have just learned, we made a *list* that contains the image for the Raspberry Pi Logo.

```
r = [255,0,0] ## Defining the colour red
g =[0,255,0] ## Defining the colour green
e = [0,0,0] ## This variable set's the pixel to off

logo = [
    e, g, g, e, e, g, g, e,
    e, e, g, g, g, g, e, e,
    e, e, r, r, r, r, e, e,
    e, r, r, r, r, r, r, e,
    r, r, r, r, r, r, r, r,
    r, r, r, r, r, r, r, r,
    o, r, r, r, r, r, r, o,
    o, o, r, r, r, r, o, o,
    ]
```

Exercise 2: Using the *list* given, try changing your code to set the pixels to display the Raspberry Pi Logo in the Trinket emulator.

Back to Glitter

Creating glitter on the screen is actually easily done by creating a *list* with random shades of a certain colour. To get a random shade of a certain colour we use the *randomShade()* function. For example the code below will set all the pixels to some random shade of green using a *list* and the *set_pixels* function.

```
g = randomShade('green')

image = [
  g,g,g,g,g,g,g,g, #Setting the third pixel along to yellow
  g,g,g,g,g,g,g,g, #The second
  g,g,g,g,g,g,g,g, #The third
  g,g,g,g,g,g,g,g, #And so on..
  g,g,g,g,g,g,g,g,
  g,g,g,g,g,g,g,g,
  g,g,g,g,g,g,g,g,
  g,g,g,g,g,g,g,g,
]

sense.set_pixels(image)
```

Try it in the Trinket Emulator below. Each time you run the code the shade of green will change.

Filling lists using Loops

Typing out *lists* can get quite annoying and it would be easier if we can automate this. We can add elements to a *list* using the *.append* function on a *list*. Let's demonstrate what this means.

First we create an empty *list*.

```
my_list = []
```

If we were to *print* the *list* defined above, we would get an empty *list* because we haven't actually put anything in it. The code bellow will add the colour variable *g* and the variable *r* to the list.

```
g = [0,255,0]
r = [255,0,0]
my_list.append(g)
my_list.append(r)
```

Now if we were to *print* the *list* defined above we would get the following output:

```
[[0,255,0],[255,0,0]]
```

The *list* given above has 2 elements. Both of the elements are also *lists* but that is not important here, as we can make a list of any type of object in python. To create a image for the Sense HAT's screen we need a *list* with 64 elements (all the elements will be also be lists that hold the RGB values to set the colour).

This means we need to *append* to an empty *list* 64 times. But what can we use to repeat code?

Loops!

The *for-loop* below will run 64 times and add the colour green (variable *g*) to a *list*.

```
image = []
g = [0,255,0]

for i in range(64):
    image.append(g) ## Adding the variable g to the list
```

If we were to use the *set_pixels* function with the *image* list after the *for-loop*, we would set all the pixels to green.

Writing Code

If you understood the concepts explained above, you are ready to make glitter! **If you are unsure about anything, ask your supervisor to explain it to you.**

* Sec 2.1 Picking a colour

Define a variable with the colour of your glitter. This variable will be used in the *randomShade* function so it must be a *string*. The possible colours are given in the list *colours*.

* Sec 2.2

Create an empty list that you will add the random shade of your chosen colour to.

* Sec 2.3

Get a random shade of the colour you chose using the *randomShade* function.

* Sec 2.4

Add the random shade to the list.

* Sec 2.5

Finally set the pixels on the Sense HAT to the list

* Sec 2.6

There is nothing to add here for now. The *time.sleep(0.1)* function makes the Raspberry Pi wait for 0.1 seconds before changing the pixels to a new list. Every 0.1 seconds the pixels are set to a random shade of the chosen colour, this is what makes the glittery effect.

Finished!

If it's all done, correctly, your screen will show a glittery pattern! Don't worry if it doesn't, things often go wrong in programming. Errors in code are usually called *bugs*. If you have a bug in your code, you'll have to *debug* it!

Transfer it to your Raspberry Pi to try it out!

If it works, congratulations! You can either move on to another project, add the suggested **bonus features** or try to come up with new things to add to the current project. Use your creativity! You can discuss any ideas you have with a supervisor.

Bonus Features

- Change the Glitter Colour by Shaking

Note: The Trinket Emulator doesn't let you shake the Raspberry Pi, ask your supervisor to help you test your code on your Raspberry Pi

The Sense HAT comes with an accelerometer. This means that you can detect if the Raspberry Pi has been moved! The code below will measure acceleration of a *shake* and if it is above a certain value, all the pixels will be set to the Raspberry Pi Logo

```
r = [255,0,0] ## Defining the colour red
g =[0,255,0] ## Defining the colour green
e = [0,0,0] ## This variable set's the pixel to off

logo = [
    e, g, g, e, e, g, g, e,
    e, e, g, g, g, g, e, e,
    e, e, r, r, r, r, e, e,
    e, r, r, r, r, r, r, e,
    r, r, r, r, r, r, r, r,
    r, r, r, r, r, r, r, r,
    o, r, r, r, r, r, r, o,
    o, o, r, r, r, r, o, o,
    ]

ac = sense.get_accelerometer_raw()
  x = ac["x"]
  y = ac["y"]
  z = ac["z"]
  shake = x*x + y*y + z*z

  if shake > 5.0:
    sense.set_pixels(logo)
```

Think about how you can use the code above to randomly change the colour of the glitter when the Raspberry Pi is shaken.

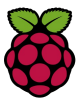
You will find the following function useful:

To randomly select something from a list, you can use the *random.choice* function. The code below will randomly choose a number from the list *numbers* and *print* it to the console.

```
numbers = [1,2,3,4,5,6]

print(random.choice(numbers))
```

Author: Ishan Khurana

Date: August 15, 2017



Graphics on the Sense HAT

Go4Code

Project: Glitter Lights

Difficulty: Easy

In this project we will be making use of the Raspberry Pi's display panel to make some glittery lights. Ask your supervisor to show you what they should look like!

Introducing the project

The first thing you should do is open the *skeleton code* for the project.
In programming, skeleton code means code that only has the basic elements of a program. It is up to you to fill in the rest!

You can find the skeleton code on Trinket, here:
Glitter Project (https://trinket.io/python/77b6a432af)

If you can, you should also create an account and log in to Trinket. This will allow you to save the Trinket projects. Otherwise you have to copy the code on to your computer to save it.

On Trinket, you'll be able to test your code on a *virtual* Sense HAT, before you try your code on the real thing.

As you might see, the skeleton code is split up into sections, divided by the headlines.
For example:

```
#### 2. Main Loop
```

This guide will go through the various sections (not necessarily in order), and help you write your code. **Very important note:** *You should add the code in the specified section in your skeleton code as you follow this guide.*

The next part of this guide will explain the stuff that's already in the skeleton code when you first open it.

Explanation of the skeleton code

Before we get on to the coding, it's worth looking over the *skeleton code* and make sure you are familiar with it.

The first few lines in the script are:

```
#### 1.1 Import libraries

import sys
sys.path.insert(1,'/home/pi/Go4Code/g4cSense/skeleton')

from sense_hat import SenseHat
from senselib import *
```

Without going into detail, these lines are called *import statements*. They are used to *import* code from other Python files into your own file. This is useful because you can use other people's code to simplify your own.

The next part of the code (Sec. 1.2) creates some important *Objects* (don't worry if you're not sure what that means) that we'll use in the later on.

Section 2 is where you'll actually be coding.

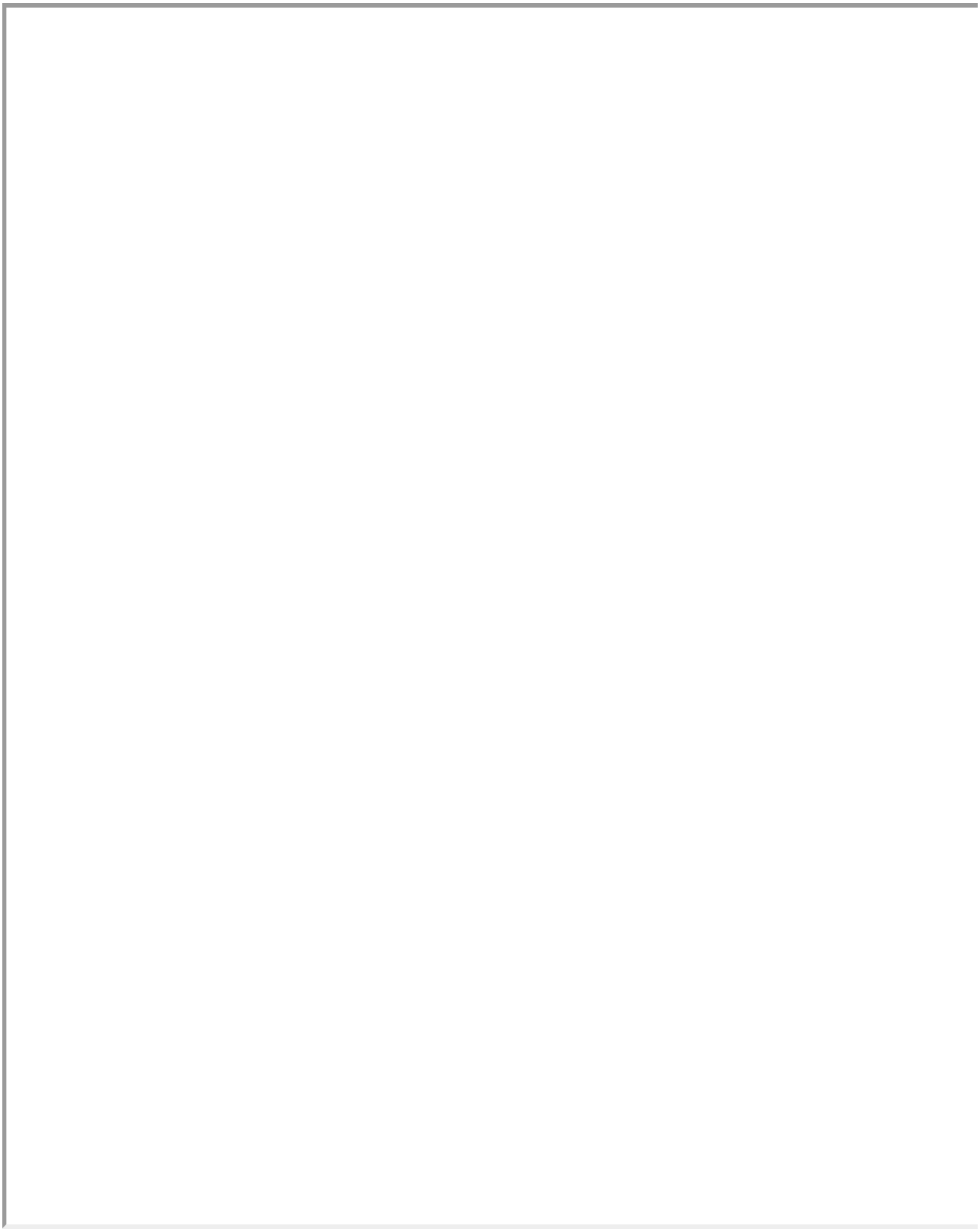
Setting Pixels on the Sense HAT (Recap)

This section is meant to remind you how to set all the pixels on the Sense HAT using a list. If you already know how to do this from a previous project, you may skip this section.

The Sense HAT allows us to use the *set_pixels* function to set all the pixels to certain colours as defined by in a *list*. This is better understood with an example. The code below defines a *list* with 64 values. Each value is a colour. The *list* tells the Raspberry Pi what colour each pixel will be set to.

```
r = [255,0,0]

image = [
  r,r,r,r,r,r,r,r, #This is the first row of pixels
  r,r,r,r,r,r,r,r, #The second
  r,r,r,r,r,r,r,r, #The third
  r,r,r,r,r,r,r,r, #And so on..
  r,r,r,r,r,r,r,r,
  r,r,r,r,r,r,r,r,
  r,r,r,r,r,r,r,r,
  r,r,r,r,r,r,r,r,
]
```

Let's try the code above in the Trinket. ***Read the comments for explanations!***



Trinket Emulator

Making a picture

The *set_pixels* function is actually really useful as it let's us make images really easily. The *list* essentially acts as a drawing board that resembles the LED matrix on the Sense HAT. For example if I take the 3rd element along in the first row and set that to yellow in the *list* it will set the 3rd pixel along in the first row to orange.

Remember: In programming, we count from zero not one!

```
r = [255,0,0]
y = [255,255,0]
image = [
  r,r,r,y,r,r,r,r, #Setting the third pixel along to yellow
  r,r,r,r,r,r,r,r, #The second
  r,r,r,r,r,r,r,r, #The third
  r,r,r,r,r,r,r,r, #And so on..
  r,r,r,r,r,r,r,r,
  r,r,r,r,r,r,r,r,
  r,r,r,r,r,r,r,r,
  r,r,r,r,r,r,r,r,
]
```

Let's try it out in the Trinket below:

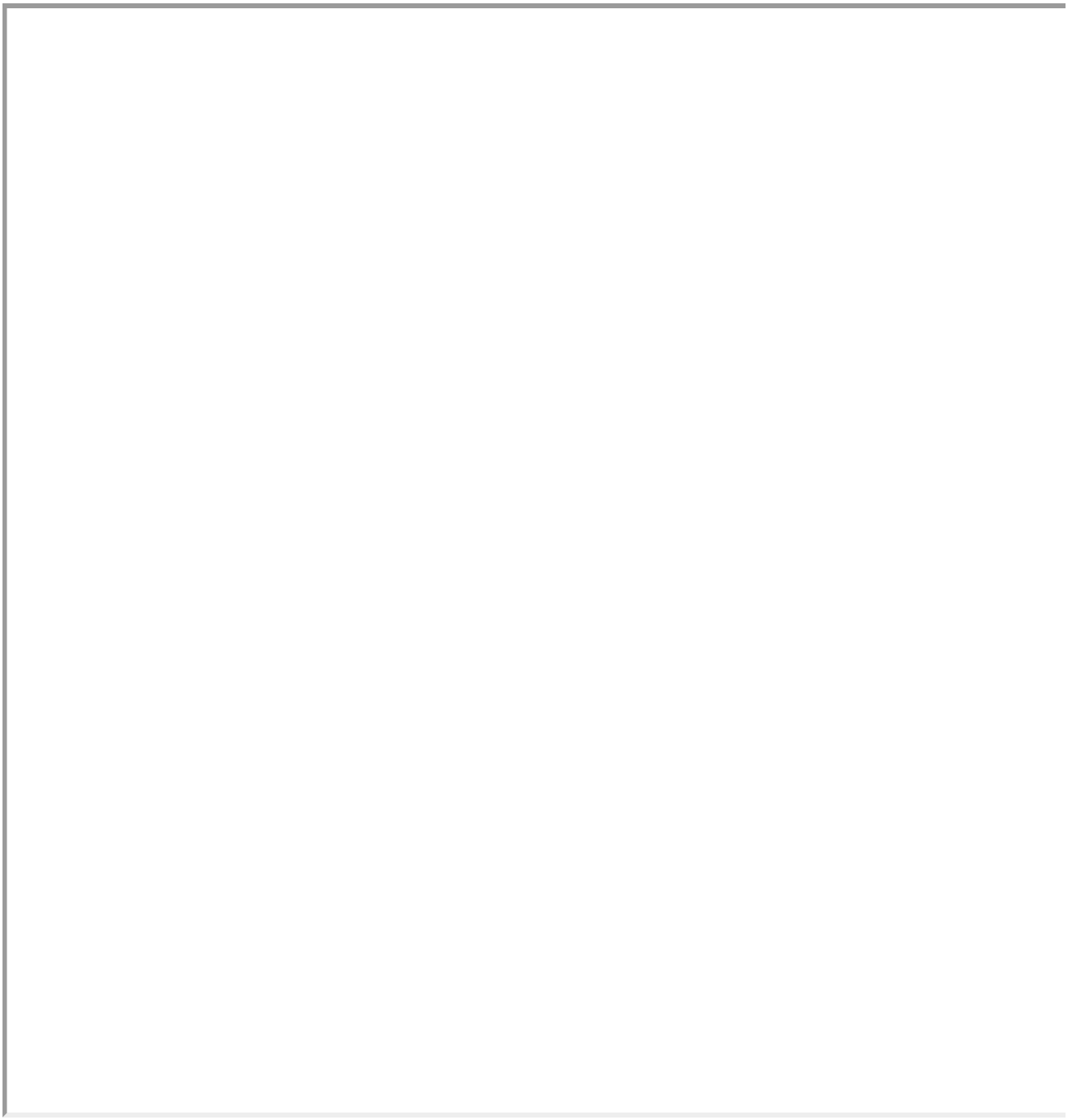
Trinket Emulator

Great! So now that we can see how to use *lists* and the *set_pixels* function to set all the pixels at the same time. We can make some images!

If you are unsure about how the code above works ask your supervisor to explain it to you

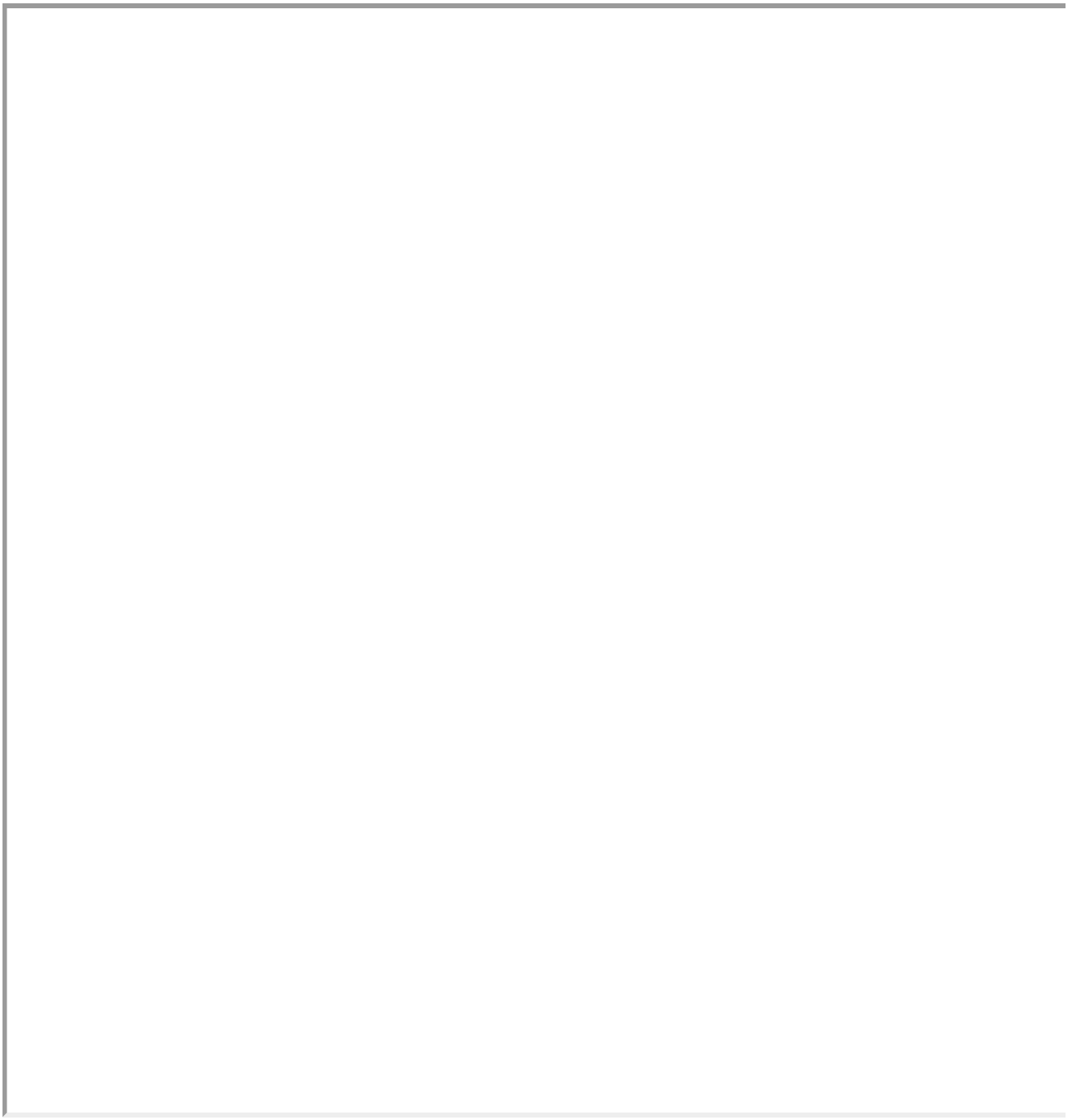
Making a plus sign

Let's try making a simple image first. We're going to make a plus sign as shown in the Trinket below:



Trinket Emulator

To make that plus sign , we used the *set_pixels* function and a *list* called image as a drawing board to make the drawing.
Exercise 1: Change the values of the *list* in the Trinket below to make a plus sign:



Trinket Emulator

The Raspberry Pi Logo

We can add any colour to our *list*. We can also add no colour to a pixel, this would mean that pixel would be off!

By setting one of the elements to *e = [0,0,0]* we can turn the corresponding pixel off. Using this and what you have just learnt, we made a *list* that contains the image for the Raspberry Pi Logo.

```
r = [255,0,0] ## Defining the colour red
g =[0,255,0] ## Defining the colour green
e = [0,0,0] ## This variable set's the pixel to off

logo = [
    e, g, g, e, e, g, g, e,
    e, e, g, g, g, g, e, e,
    e, e, r, r, r, r, e, e,
    e, r, r, r, r, r, r, e,
    r, r, r, r, r, r, r, r,
    r, r, r, r, r, r, r, r,
    o, r, r, r, r, r, r, o,
    o, o, r, r, r, r, o, o,
    ]
```

Exercise 2: Using the *list* given, try changing your code to set the pixels to display the Raspberry Pi Logo in the Trinket emulator.

Back to Glitter

Creating glitter on the screen is actually easily done by creating a *list* with random shades of a certain colour. To get a random shade of a certain colour we use the *randomShade()* function. For example the code below will set all the pixels to some random shade of green using a *list* and the *set_pixels* function.

```
g = randomShade('green')

image = [
  g,g,g,g,g,g,g,g, #Setting the third pixel along to yellow
  g,g,g,g,g,g,g,g, #The second
  g,g,g,g,g,g,g,g, #The third
  g,g,g,g,g,g,g,g, #And so on..
  g,g,g,g,g,g,g,g,
  g,g,g,g,g,g,g,g,
  g,g,g,g,g,g,g,g,
  g,g,g,g,g,g,g,g,
]

sense.set_pixels(image)
```

Try it in the Trinket Emulator below. Each time you run the code the shade of green will change.

Filling lists using Loops

Typing out *lists* can get quite annoying and it would be easier if we can automate this. We can add elements to a *list* using the *.append* function on a *list*. Let's demonstrate what this means.

First we create an empty *list*.

```
my_list = []
```

If we were to *print* the *list* defined above, we would get an empty *list* because we haven't actually put anything in it. The code bellow will add the colour variable *g* and the variable *r* to the list.

```
g = [0,255,0]
r = [255,0,0]
my_list.append(g)
my_list.append(r)
```

Now if we were to *print* the *list* defined above we would get the following output:

```
[[0,255,0],[255,0,0]]
```

The *list* given above has 2 elements. Both of the elements are also *lists* but that is not important here, as we can make a list of any type of object in python. To create a image for the Sense HAT's screen we need a *list* with 64 elements (all the elements will be also be lists that hold the RGB values to set the colour).

This means we need to *append* to an empty *list* 64 times. But what can we use to repeat code?

Loops!

The *for-loop* below will run 64 times and add the colour green (variable *g*) to a *list*.

```
image = []
g = [0,255,0]

for i in range(64):
    image.append(g) ## Adding the variable g to the list
```

If we were to use the *set_pixels* function with the *image* list after the *for-loop*, we would set all the pixels to green.

Writing Code

If you understood the concepts explained above, you are ready to make glitter! **If you are unsure about anything, ask your supervisor to explain it to you.**

* Sec 2.1 Picking a colour

Define a variable with the colour of your glitter. This variable will be used in the *randomShade* function so it must be a *string*. The possible colours are given in the list *colours*.

* Sec 2.2

Create an empty list that you will add the random shade of your chosen colour to.

* Sec 2.3

Get a random shade of the colour you chose using the *randomShade* function.

* Sec 2.4

Add the random shade to the list.

* Sec 2.5

Finally set the pixels on the Sense HAT to the list

* Sec 2.6

There is nothing to add here for now. The *time.sleep(0.1)* function makes the Raspberry Pi wait for 0.1 seconds before changing the pixels to a new list. Every 0.1 seconds the pixels are set to a random shade of the chosen colour, this is what makes the glittery effect.

Finished!

If it's all done, correctly, your screen will show a glittery pattern! Don't worry if it doesn't, things often go wrong in programming. Errors in code are usually called *bugs*. If you have a bug in your code, you'll have to *debug* it!

Transfer it to your Raspberry Pi to try it out!

If it works, congratulations! You can either move on to another project, add the suggested **bonus features** or try to come up with new things to add to the current project. Use your creativity! You can discuss any ideas you have with a supervisor.

Bonus Features

- Change the Glitter Colour by Shaking

Note: The Trinket Emulator doesn't let you shake the Raspberry Pi, ask your supervisor to help you test your code on your Raspberry Pi

The Sense HAT comes with an accelerometer. This means that you can detect if the Raspberry Pi has been moved! The code below will measure acceleration of a *shake* and if it is above a certain value, all the pixels will be set to the Raspberry Pi Logo

```
r = [255,0,0] ## Defining the colour red
g =[0,255,0] ## Defining the colour green
e = [0,0,0] ## This variable set's the pixel to off

logo = [
    e, g, g, e, e, g, g, e,
    e, e, g, g, g, g, e, e,
    e, e, r, r, r, r, e, e,
    e, r, r, r, r, r, r, e,
    r, r, r, r, r, r, r, r,
    r, r, r, r, r, r, r, r,
    o, r, r, r, r, r, r, o,
    o, o, r, r, r, r, o, o,
    ]

ac = sense.get_accelerometer_raw()
  x = ac["x"]
  y = ac["y"]
  z = ac["z"]
  shake = x*x + y*y + z*z

  if shake > 5.0:
    sense.set_pixels(logo)
```

Think about how you can use the code above to randomly change the colour of the glitter when the Raspberry Pi is shaken.

You will find the following function useful:

To randomly select something from a list, you can use the *random.choice* function. The code below will randomly choose a number from the list *numbers* and *print* it to the console.

```
numbers = [1,2,3,4,5,6]

print(random.choice(numbers))
```

Author: Ishan Khurana

Date: August 15, 2017

UCL Coding Summer School - Introduction to Python and the Sense HAT

Welcome!

Hello! This guide will take you through the basics of Python programming.

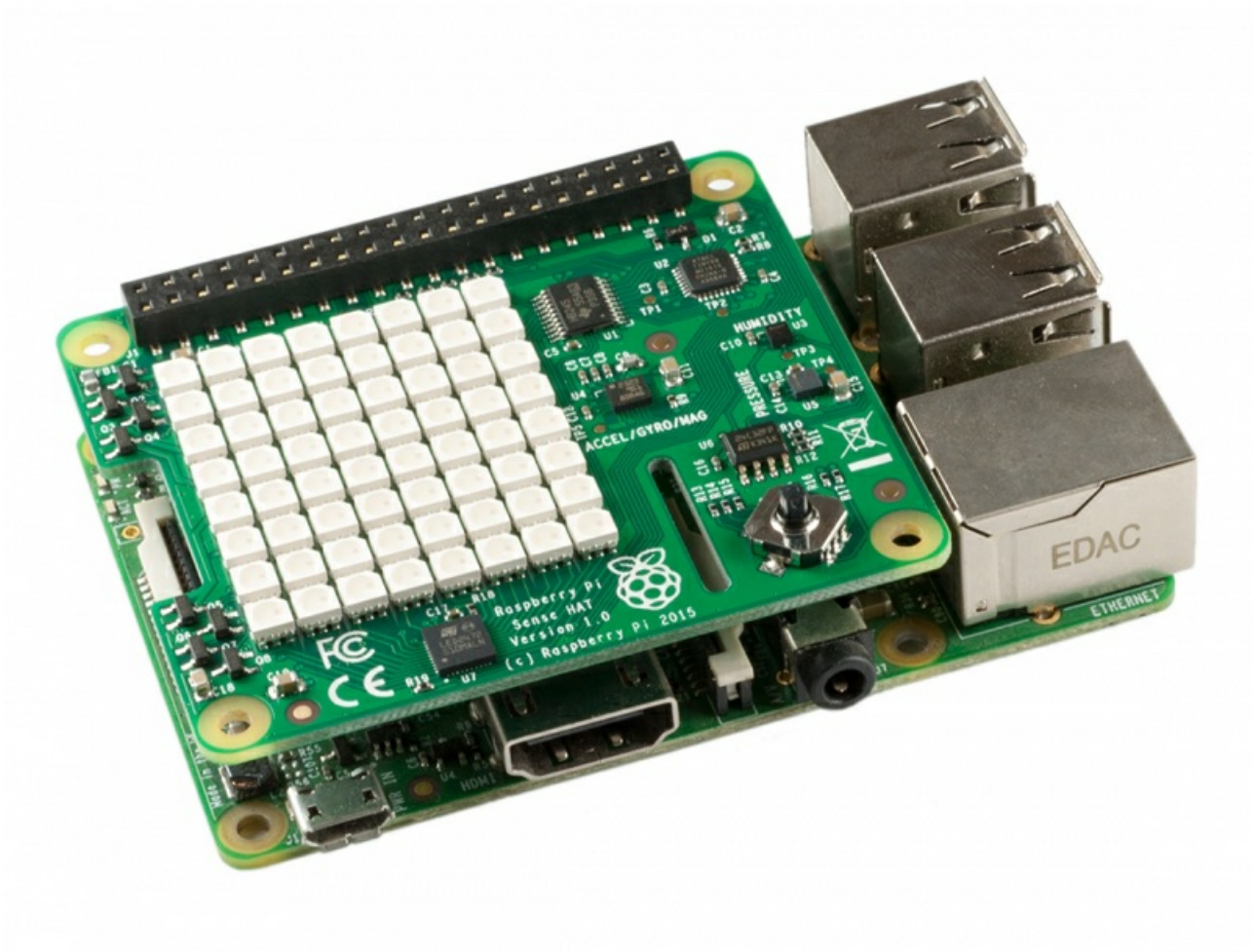
The end goal of this lecture is to get you ready to start doing some programming of your own, in the various projects we have prepared for you.

The lecture start right from the beginning, so you don't need to have done programming before. If you have, then use this lecture as a helpful reminder.

---

1. Introduction to the Sense HAT

All the projects that we're going to do during the summer school will be using something called the *Sense HAT*. It looks like this:



The white squares that you see in the image are LED-lights, and using Python we can make them light up to any color we like. There are 8 columns and 8 rows of these LED-lights, which means that there are 64 in total.

On the bottom-right of the device, you can also see that there's a *joystick*. Using this we can, for example, create various games.

The Sense HAT can also, among other things, measure the current temperature, the pressure in the room and can act as a compass!

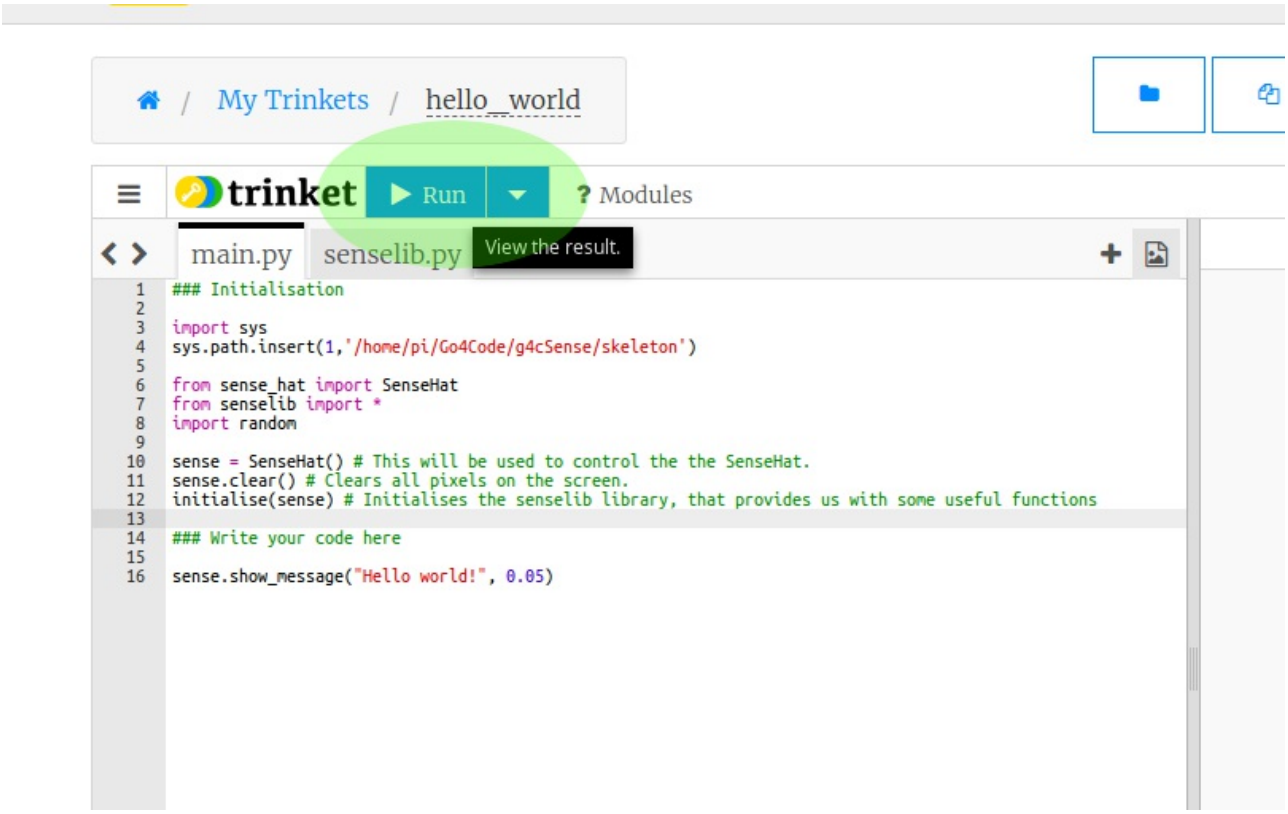
There are a *ton* of stuff you can do on a Sense HAT. Much more than we can cover during this summer school. That's why we're going to let you take your Sense HATs home with you after the end!

1. Trinket, and writing your first program

In your projects, you'll have your own Sense HAT to play around with. But since you'll have to test around and tinker with your projects for quite some time before its ready, we're going to be testing all our code on a *virtual* Sense HAT on a website called *Trinket*.

Open up the following link:
https://goo.gl/JRpQfC (https://goo.gl/JRpQfC)

This opens up an example project that we've prepared. The left side of the page has some *Python* code already. Don't worry if it all looks like gibberish right now!



The right side of the page shows nothing in the beginning, but if you press the *Run* button, a virtual Sense HAT should appear on the screen. It won't do much right now, because there's no code to run!

Let's create out first program. Copy the following into the Python code pane:

```
sense.show_message("Hello world!", 0.05)
```

After you have done that, click Run again. The message, "Hello World!" should scroll by on the screen. You've written your first Python program!
You're now *officially* a Python programmer.

The red text you see in the Python code is what we call a *string*. A string is basically a piece of text. In Python, we can write strings by either enclosing text in double quotation marks like this

```
"This is a string!"
```

or single quotation marks, like this

```
'This is a string!'
```

Now, we're going to play around with this program that you've made.

Try changing the message to something else. Also, you might notice that there's a number after the comma inside the brackets.

Exercise 1: Change the message to something else. Try doing it using single-quotation marks as well.

Exercise 2: You might have noticed that there's a number after the string.
Try changing the value of that number. Find out what that number does.

2. Colours in Python: RGB-values

Let's change your program a bit more.

Replace the line in your code with the following:

```
sense.show_message("Hello world!", 0.05, text_colour=[255, 0, 0])
```

Run it again, and see what happens. The text should now be red. The last part of the line that you just wrote `text_colour=[255, 0, 0]` decides which color the text should be in. You can change these three numbers to change the color of the text.

We call these triplets of numbers RGB-values, and this is the reason:

- The first number decides how *red* the text should be,
- the second number decides how *green* the text should be,
- and the third number decides how *blue* the text should be.

The maximum redness/greenness/blueness is 255, and the lowest is 0. As you saw above, the RGB-value for red is [255, 0, 0].
And, for example, the RGB-value for black is [0, 0, 0].

Exercise 3: Make the text flash by in the following colors:

- Red
- Blue
- Green
- White
- Gray
- Purple
- Yellow
- Orange

You'll have to experiment to try to find the right color-combinations!

3. Commenting

Before we get going with some more interesting code, let's remove the *sense.show_message* line so that it won't disturb us later on. But instead of just erasing the line completely, we can *comment it out*.

To do this, simply add a hashtag # at the start of the line, like this:

```
# sense.show_message("Hello world!", 0.05, text_colour=[255, 0, 0])
```

Try running the code after you've commented the code out. No text should appear on the screen. In Python we often use comment symbol # to add explanations to our code, for example:

```
# This line displays some text on the screen
sense.show_message("Hello world!", 0.05, text_colour=[255, 0, 0])
```

4. Drawing stuff on the Sense HAT: *sense.set_pixel*

It's really easy to draw things on the Sense HAT screen. We can do this using the function *sense.set_pixel* (if you're confused about how I just used the word "*function*", don't worry I'll explain it in the next section.). Try adding the following lines to your code

```
sense.clear()
sense.set_pixel(2, 2, 255, 255, 255)
sense.set_pixel(4, 2, 255, 255, 255)
sense.set_pixel(3, 3, 255, 255, 255)
sense.set_pixel(2, 4, 255, 255, 255)
sense.set_pixel(4, 4, 255, 255, 255)
```

It should draw a white cross on the screen. The first line, *sense.clear*, resets all the pixels on the screen to black. This is just to avoid any clutter on the screen if you make mistakes.

Let's see how the *sense.set_pixel* function works.

You'll recognize the last three numbers of each line, those set the colors, as we learned previously. The first two numbers decide *where* we'll draw the pixel.

The first number is what column on the screen the pixel should be drawn at. We usually call this the *x-coordinate*.

The second number is what row on the screen the pixel should be drawn at. We usually call this the *y-coordinate*.

Exercise 4: Play around with the code, and try do draw something of your own. Maybe the first letter of your name?

Exercise 5: Try placing the sense.clear() line at the end of the code. What happens? Can you explain why? After that, place the line in the middle of the code. Again, what happens, and can you explain why that happens?

5. Functions

Before we move on, I'll explain (as promised) what a *function* is. A function is simply something that we use in a code to do something for us, whether it is drawing text on the screen, or clearing the screen of any drawings, or drawing individual pixels.

Let's look a bit closer at the *sense.set_pixel* function.

```
sense.set_pixel(4, 4, 255, 255, 255)
```

The name of the function is *sense.set_pixel*. When you use a function, it's often referred to as *calling* the function.

What about all those numbers inside the brackets? Those are called *arguments*. Sometimes when we call a function, we want to specify exactly what we want it to do, we do this using *arguments*.

In this case, the arguments are the numbers that specify the position of the the pixel (the first two numbers), and the color of the pixel (the last three).

In the first example we used, we saw:

```
sense.show_message("Hello world!", 0.05)
```

The first argument is the message we display, and the second how fast the message should display.

Don't worry if all of this sounds a bit complicated. It's a lot of names for something that's quite simple. If you have any questions about this, discuss it with a supervisor.

Before we move on, we'll introduce the *print* function. The print function is very simple, it just displays some text. The important difference here is that it doesn't print out text on the screen, but in the *terminal* underneath the Sense HAT screen. It's easiest to see for yourself, copy the following code:

```
print("Here's some text!")
```

You should see some text appear in the lower-right corner of the page.

6. Using Python as a calculator

Next thing we'll do is try out some simple mathematical operations in Python.
All the main math signs you know from school exist in Python.
Using Python we can add, subtract, multiply or divide numbers.
Copy the following code to try it for yourself:

```
print(3+5)
print(2-7)
print(2*5)
print(4/2)
```

Run the code and see what the results are. You can also add really large numbers

```
print(1234567890000 + 987654321111)
```

Or do several mathematical operations in a row

```
print(23*52*13 + 52*611 - 412 )
```

We can also use brackets "*()*" when calculating things

```
print((2+3)*5)
print(6 + ((100 - 1)*42)/32)
print(((2 * (23 - 3) / (12 * 23)) - 32)*62)
```

Try changing the numbers, and try to do different calculations.

Exercise 6: In *Trinket*, translate the following into Python code:

```
_(four times ( six minus two ) plus five) divided by two_
```

and use *print* to show the result.

7. Variables

Python allows us to store information in *variables*. We can use the assignment operator (the *equal-to* sign '=') to assign a value to a variable.

There are three components to variable assignment.

- First we need to decide on the name of our variable, like my_variable.
- Second, we set the variable name equal to something using the *equal-to* sign '='.
- Third, we write the value that the variable should be equal to.

Like this:

```
my_variable = 123
```

We can also store strings in a variable:

```
my_variable = "Hello!"
```

Note that the variable name must be written as *one single set of characters* without spaces. So, for example, *my variable* would not be a valid name for a variable.

The code below shows how easy it is to define and display a variable.

```
myNumber = 5*7
someText = "Five times seven is "
myMessage = someText + str(myNumber)

sense.show_message(myMessage, 0.05)
```

Can you look at this code and figure it out what it does without actually running it?
After you have guessed, write the code in your Trinket project and run it.

Explanation: We have used the *str* function to convert the number 5*7 to a *string* "35". Then we added the string "Five times seven is " with "35" to form "Five times seven is 35".

Exercise 7: Write a program that displays your full name and age on the screen.
Store your full name in a variable called *myName*, and store your age in a variable called *myAge*. Remember that you have to convert numbers to strings using the *str* function.

8. A basic program

Variables are quite useful, as they allow us to remember numbers (or other types of data, like *strings*) in terms of more memorable variable names. We can use these variables in our programs to compute things. Below is an example of a basic program that calculates how many minutes there are in a week.

```
minutes_in_an_hour = 60 # storing the number of minutes in an hour into a
variable
hours_in_a_day = 24 # Storing hours in a day into a variable
days_in_a_week = 7

# calculate the minutes in the week

minutes_in_a_week = minutes_in_an_hour*hours_in_a_day*days_in_a_week

print("There are" , minutes_in_a_week , "minutes in a week")
```

You don't have to copy this code, you can go on the following link to see the code in action: [https://goo.gl/CZH1UX (https://goo.gl/CZH1UX)](https://goo.gl/CZH1UX).

But before you run it, try to just read the code and figure out what it does yourself. It's good practice!

9. If-and-else statements

This next part is where things get more interesting (and a bit more complicated)!

The *if*-statement allows us to make a program that reacts in different ways, under different conditions. Think of it as a way to ask a computer questions.

Look at the code below, can you guess what it does?

```
myName = "Lukas"

if myName == "Ishan":
  sense.show_message("Your name is Ishan", 0.03)
else:
  sense.show_message("Your name is not Ishan", 0.03)
```

Copy the code into your project. How can you change the variable *myName* so that the result of the program is different?

Note that the code inside the *if* and the *else* clause are indented (2 spaces away from the left), the indentation tells the computer which parts of the code are inside the 'if' and the 'else' statements. To add an *indentation* to your code, press the *Tab* key on your keyboard.

Essentially, the *if*-statement checks whether something is true or not. The dobuble-equals sign "==" means "is equal to". So the code above translates to (in English):

```
If *myName* is equal to "Ishan", show "Your name is Ishan". If *myName* is
not equal to "Ishan", show "Your name is not Ishan".
```

The code that is in the *true* part of the code, is on the lines below the *if*.
The code that is in the *false* part of the code, is on the lines below the *else*.

We can also create *if*-statements using numbers:

```
my_variable = 10

if my_variable < 20:
    print("The number is less than 20.")

if my_variable < 1:
    print("The number is less than 1.")

if my_variable > 5:
    print("The number is larger than 5.")

if my_variable == 11:
    print("The number is equal to 11.")

if my_variable > 55:
    print("The number is larger than 55.")

if my_variable == 10:
    print("The number is equal to 10.")
```

You don't havet copy this code, you can see it here: https://goo.gl/jpmjLj (https://goo.gl/jpmjLj). Before running
it, read the code and guess what it does.

In the code above, we were able to compare numbers in the if statements using the <, > and == symbols. Here's
an explanation for what these symbols mean:

- **a > b** a is greater than b.
- **a < b** a is less than b.
- **a >= b** a is greater than or equal to b.
- **a <= b** a is less than or equal to b.
- **a == b** a is equal to b. We use double equals because single equals is used to assign values to variables.

This is all quite a lot of information already, so if you find it confusing, try to dicsuss it with a supervisor or the person sitting next to you. Don't worry if you don't understand everything yet!

Exercise 8: Write a program that does the following:

- Define a variable called *myNumber*, and assign it any number you like.
- Using if-statements, make the program say whether the number is positive, negative or equal to zero.

10. Lists

In Python, we can not only store numbers and strings in variables, we can also store a *list* of items in a variable.

```
myList = [42, 51, 62, "Hello", 61, 123, "World"]
```

We already saw a list early on in the lecture. The RGB colors were stored in lists.

If we want to access individual elements in the list, we can do it like this:

```
myList = [42, 51, 62, "Hello", 61, 123, "World"]

# The first element in the list
print(myList[0])

# The third element in the list
print(myList[2])

# The fifth element in the list
print(myList[4])
```

Copy this code and see what it does. Note that the *first* element is given by the number *0* in the list, and the second by *1* and so on...

Exercise 8: Repeat Exercise 7, but instead of storing your name and age is separate variables, store them in a *single* list.

11. For-loops

Sometimes you might want to repeat the same piece of code several times. Loops allow us to accomplish this in fewer lines of code, by *repeating* code until our operation is complete.

Look at the code below, what do you think it does? See if you can figure it out before you run it.

```
for n in range(1,10):
  print("The number is", n )
```

Now copy the code and run it. Did you guess correctly?

How about this code?

```
for n in range(1, 5):
  sense.set_pixel(n, 0, 0, 0, 255)
```

This one is more complicated. Try to imagine what this could do. After that, copy it and see for yourself.

Play around with the code and see if you can make it do more complicated!

Exercise 9: Write a program that *prints* the numbers from 50 to 100.

Exercise 10: Write a program that draws a line of yellow pixels from the top-left corner, to the bottom-right corner.

12. While-loops

Another type of loop that we can use in Python is the *while loop*.

The while loop repeats its code as long as the expression after the *while* command evaluates to *true*. You can think of it as a mix between an *if-statement* and a *for-loop*.

The previous explanation might be a bit complicated to follow, but if you look at the code below you can see that it's quite intuitive.

```
n = 0 # Initialising n
while n<10: # Starting the loop with condition that n must be less than 10
    n = n+1 # redefines n as n+1
    print(n) # print the new value of n
```

As usual, look at the code and try to figure out what it does. After that copy it and see for yourself!

What do you think will happen if you run the following code?

```
while True:
  print("Hello!")
```

Don't try to run it! What will happen is that the code will run *forever*, can you see why? It'll print out "Hello!" again and again and it wont stop! Although never-ending codes like this might seem useless to you, it does actually come in handy, and we'll often use them in our projects!

Exercise 11: Redo exercise 10, but with a *while*-loop, instead of a *for*-loop.

13. What's next?

The next thing is to ask yourself if you've understood everything!
We covered quite a lot, and very quickly, so it might be worth to just have a quick read-through of everything again.

After that, it's time to start choosing a project. You should try to start off with one of the easier projects, before attempting any harder ones. Remember that you can always ask for help within your group, or by a supervisor, during the entirety of the summer school.

We discussed some useful functions in this lecture, like *sense.set_pixel* and *print*. There are many more functions you can use, and the best way to check them all out is by reading the *Function Reference* document. Use that document as a reference on how to use various Python functions.

Good luck!

Author: Lukas Kikuchi

Date: August 09, 2017

Debugging Go4Code markdown & Testing scripts on RPi

General comments

Make it clear that if you want to override a file the files must be written
exactly the same
filenames should not have spaces
Code blocks sometimes cut off end of code -- see snake line205

Project Draw a picture

Markdown

Corrections to be made

~~line91 row does not show as italics~~

~~line171 when the user holds the button to blink (make more explicit)~~

Suggestions

~~Remind students that python starts/counts from 0~~

~~Split paragraph (line109-line114) up, it can be simplified - maybe give an piece of code~~

~~line145/146 rewrite~~

RPi testing

no bugs detected

Project Magic eight ball

Markdown

Corrections to be made

~~When references are made to pressing the joystick on trinket~~

~~explicitly say the user must press enter on keyboard~~

line39-41 should be put either on the homepage on the introduction lecture (NOTE this is found on all the project pages) **Lukas: Why? The skeleton codes are specific for each project.** *referring to comments on trinket account*

~~Rewrite line46-47~~

~~line189 rewrite~~

Suggestions

~~line19 change so to therefore~~

~~Rewrite lines 24-26~~

~~line151-153 possibly explain in more detail, could add an example for clarity~~

RPi testing

~~script doesn't repeat question "ask ball a question after first shake"~~ **Should be fine**

Project Paint

Markdown

Corrections to be made

~~line89 random #~~

~~Change color to colour~~

~~line167-176 consider rewriting, various grammar mistakes~~

Suggestions

~~line94: change to "In this section we will define some variables which we will use later on"~~

Change variables to cursor_x and cursor_y: easier to read **I try to avoid the underscore as it slows down the typing-speed for the kids**

RPi testing

no bugs found

Project MarbleMaze

Markdown

Corrections to be made

Trinket simulator hard to control motion sensor- get supervisor to demonstrate on their raspberry pi (supervisor should be prepared to do this for all project)

~~line114 make things happen .... change to something more specific~~

~~More information is needed on the functions, for example, what arguments can they take, how to use them, i.e. does drawWall() place wall in same place everytime?~~ **Added some more explanations. Although in those sections we only ask the kids to copy code** *resolved*

Sections are not in order - this will cause confusion **This is unavoidable. The document explains that we'll jump from section to section, but it is important that the supervisors
are aware of this as well. Maybe Ishan should go through this in the introductory lecture as well.** *resolved*

Suggestions

~~Reference yaw and pitch again in Sec 2.1~~

In general the project is quite difficult to follow - Sec 2.2 was a bit confusing

RPi testing

overall game a bit buggy, ball doesn't always move

pixels flicker

Project Snake

Markdown

Corrections to be made

~~Slow down speed of game for first trinket example - quite hard to control for first go on trinket~~

~~Colour - need to define RGB e.g 0- 255~~ **This is done in the introductory lecture**
resolved- may be worth repeating

~~Code block line205 cuts off end sentence~~

~~Trinket code has full solution (only supposed to be skeleton)~~ **Are you sure? Can you check again?**
resolved

Suggestions

~~in general harder to follow due to difficulty of script~~

~~Encourage students to test code out after each section - not always clear~~

RPi testing

Doesn't work - snake lib is not in skeleton folder on RPi config (checked on github)

Project Countdown Clock

Markdown

Corrections to be made

~~line104: missing 'should'~~

Suggestions

RPi testing

in general works - however numbers flicker

Project: Magic eight ball

Difficulty level: Easy

Description

In this project we'll be creating a *Magic 8 ball* using the Sense HAT. If you've never seen one before, you might remember it from Toy Story: [https://www.youtube.com/watch?v=mFOracFClBg (https://www.youtube.com/watch?v=mFOracFClBg)](https://www.youtube.com/watch?v=mFOracFClBg).

It's a very simple toy: You ask it a question, you shake the ball (or Sense HAT), and it'll give you an answer.

You can try an example solution of the project here:

To get the answer, press the middle joystick button (ENTER on your keyboard).

In the version you will make, you will shake the Sense HAT to get the answer. Unfortunately you can't shake the virtual Sense HAT on Trinket, so in the version above we used the joystick instead.

Project Manual

This project guide will tell you step-by-step the main things you have to do in order to create a *magic eight ball*. For some of the steps, you'll have to use your own creativity to proceed, good luck!

Introducing the project

The first thing you should do is open the *skeleton code* for the project.
In programming, skeleton code means code that only has the basic elements of a program. It is up to you to fill in the rest!

You can find the skeleton code on Trinket, here:
https://goo.gl/bzqdNi (https://goo.gl/bzqdNi)

If you can, you should also create and account and log in to Trinket. This will allow you to save the Trinket projects. Otherwise you have to copy the code on to your computer to save it.

On Trinket, you'll be able to test your code on a *virtual* Sense HAT, before you try your code on the real thing.

As you might see, the skeleton code is split up into sections, divided by the headlines. For example:

```
#### 2. Code section
```

This guide will go through the various sections (not necessarily in order), and help you write your code. **Very important note:** *You should add the code in the specified section in your skeleton code as you follow this guide.*

The next part of this guide will explain the stuff that's already in the skeleton code when you first open it.

Explanation of the skeleton code

Before we get on to the coding, it's worth looking over the *skeleton code* and make sure you are familiar with it.

The first few lines in the script are:

```
#### 1.1 Import libraries

import sys
sys.path.insert(1,'/home/pi/Go4Code/g4cSense/skeleton')

from sense_hat import SenseHat
from senselib import *
```

Without going into detail, these lines are called *import statements*. They are used to *import* code from other Python files into your own file. This is useful because you can use other people's code to simplify your own.

The next part of the code (Sec. 1.2) creates some important *Objects* (don't worry if you're not sure what that means) that we'll use in the later on.

Sec. 1.3 is where you will write all the messages that the eight-ball can give.

Sec. 2 is where you will write the code that checks if the Sense HAT is shaken.

Writing the code

(Sec. 1.3) Write answers

When you shake the Sense HAT, we want it to give an answer. In this section you will write the answers that the magic eight ball can give.

If you look at the section, you'll see the following code:

```
answers = []
```

You'll have to add the answers as *strings* in the list, for example:

```
answers = ["yes", "no"]
```

A string is a piece of text enclosed within either double " or single ' quotation marks. The strings in the list have to be separated by commas (**,**).

This is an empty list, that you should fill with possible answers. Here are some typical magic eight ball answers that you can use:

- Outlook good
- Yes
- No
- Ask again later
- Better not tell you now
- Concentrate and ask again
- Don't count on it

But instead of using these, you should try to come up with your own!

(Sec. 2) Main program code

Before we start coding, we'll explain a bit what's going on in Sec. 2.

In this part of the code, we'll sense if the user has shaken the Sense HAT.
If the user shakes it, the program should display one of the answers we made up in the previous section by random.

Code explanation: If we look at the skeleton code, we see that the section is within a *while*-loop.
Remember that code inside a while loop runs again and again, until we tell it to stop. The reason why we want our program to be in a loop is because we want to continually check if the user has shaken the device.

Because the code is in the *while*-loop, remember that you have to add a *Tab* at every line, to make it indented. Like this:

```
while True:
    # write your code like this,
    # with a tab at the start of the line.
```

(Sec. 2.1) Check if the user shaken the Sense HAT)

This part is a bit complicated, so pay attention! Don't be afraid to ask a supervisor if you don't understand something.

To see if the user has shaken the Sense HAT, we'll have to use the *accelerometer* inside the device. The
accelerometer basically tells us how fast the device is accelerating.

If you don't know what the word *acceleration* means, the next bit will give you a short lesson in physics.

> **Acceleration:**
>
> If *speed* is how fast
> something is moving, *acceleration* is how fast the speed is changing. For example,
> let's say you drop a ball from a cliff. At first, when you first drop the ball, its speed is 0 (it's not moving). But due to gravity, it'll *accelerate* and gain speed. By the time the ball has reached the ground, it's gained a lot of speed.

To get the information from the accelerator, use the following code:

```
ac = sense.get_accelerometer_raw()
x = ac["x"]
y = ac["y"]
z = ac["z"]
```

Now, in the *x*, *y* and *z* variables, we have information recorded by the accelerometer. This is all a bit complicated, but the basic gist of it is this, calculate this:

```
shake = x*x + y*y + z*z
```

We have stored *x times x plus y times y plus z times z* into the variable *shake*. Now, the larger the variable *shake* is, the more has the Sense HAT been shaken.
We're going to say that if *shake* is *larger* than *5*, the Sense HAT has been shaken. To do this, you're going to have to use an *if*-statement.

If you have detected that the Sense HAT has been shaken (in other words, if *shake* is larger than 5) you want to display a one of the messages we wrote earlier by random. To do this, we use the *random.choice* function, like this:

```
random_message = random.choice(answers)
```

To then display the message stored in *random_message*, we use the functions *sense.show_message*. Check the *Function Reference* to see how to use it.

This might seem complicated, but it's fairly straightforward. If something confuses you, ask a supervisor for help.

To break it down, what you need to do is this:

- Get the accelerometer information.
- Calculate *shake*, as shown above.
- If *shake* is greater than 5, show a random message.

Finished!

If it's all done, correctly, the game should now work! Don't worry if it doesn't, things often go wrong in programming. Errors in code are usually called *bugs*. If you have a bug in your code, you'll have to *debug* it!

If it works, congratulations! You can either move on to another project or try to come up with new things to add to the current project. Use your creativity! You can discuss any ideas you have with a supervisor.

Author: Lukas Kikuchi

Date: August 09, 2017

Project: Marble Maze

Difficulty level: Medium

Description

In this project you'll be creating a digital version of Marble Maze (see the picture if you don't know what it is).
The Sense HAT can tell whether you're tilting your device, using this feature we can create the game. You'll also get to design your own maze in this project.

Check out an example solution of the game here:

To control the ball on Trinket, drag the Sense HAT around in the different directions with your mouse.

Project Manual

This project guide will tell you step-by-step the main things you have to do in order to create *Marble Maze*. For some of the steps, you'll have to figure out how to proceed yourself, good luck!

Introducing the project

The first thing you should do is open the *skeleton code* for the project.
In programming, skeleton code means code that only has the basic elements of a program. It is up to you to fill in the rest!

You can find the skeleton code on Trinket, here:
https://goo.gl/TEVxE8 (https://goo.gl/TEVxE8)

If you can, you should also create and account and log in to Trinket. This will allow you to save the Trinket projects. Otherwise you have to copy the code on to your computer to save it.

On Trinket, you'll be able to test your code on a *virtual* Sense HAT, before you try your code on the real thing.

As you might see, the skeleton code is split up into sections, divided by the headlines. For example:

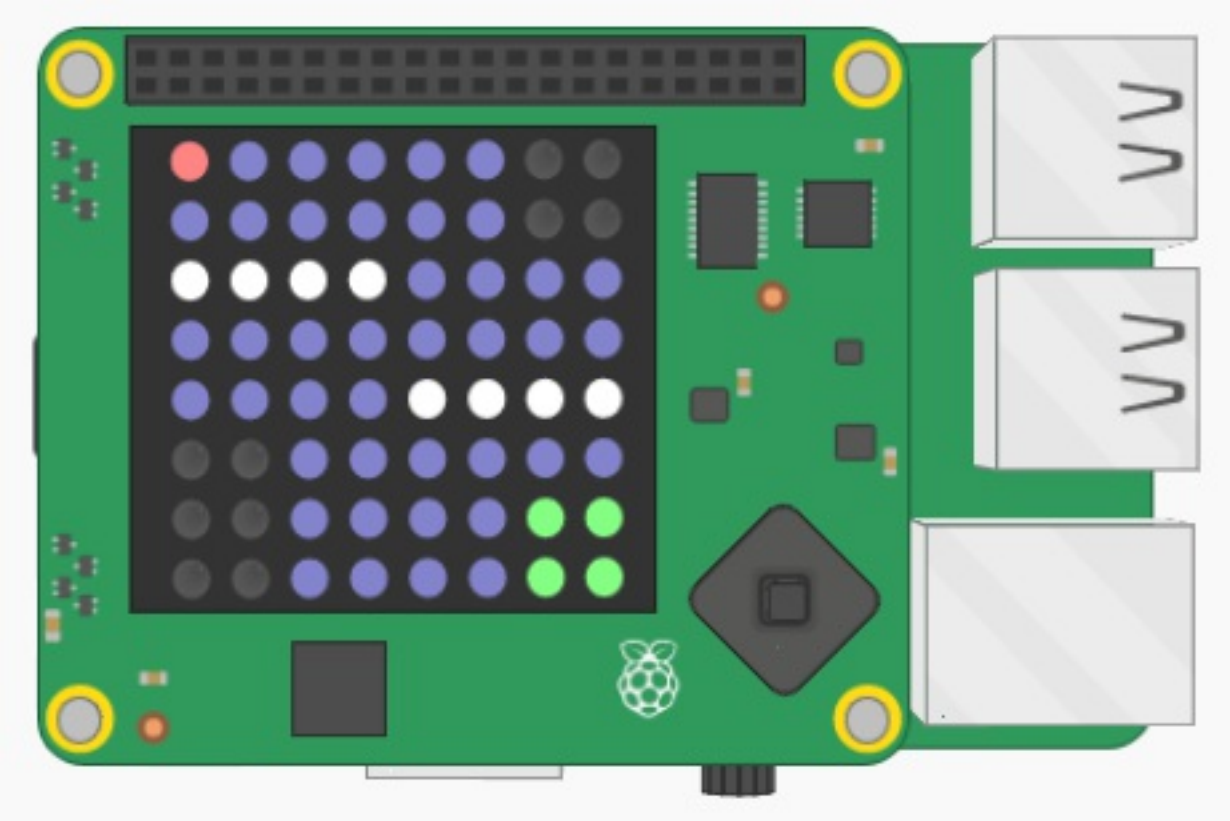
```
#### 2. Code section
```

This guide will go through the various sections (not necessarily in order), and help you write your code. **Very important note:** *You should add the code in the specified section in your skeleton code as you follow this guide.*

The next part of this guide will explain the stuff that's already in the skeleton code when you first open it.

Explanation of the game

The game you'll be working works very similarly to an actual Marble Maze game. Here's a screenshot of the game:



You have a ball (red dot in the picture) that you control move around by tilting the Sense HAT. The ball can collide and
be stopped by the walls (white dots in the picture) of the maze. The player loses if the ball goes in the holes (black dots). And the player wins if the ball goes in the goal (green dots).

Explanation of the skeleton code

Before we get on to the coding, it's worth looking over the *skeleton code* and make sure you are familiar with it.

The first few lines in the script are:

```
#### 1.1 Import libraries

import sys
sys.path.insert(1,'/home/pi/Go4Code/g4cSense/skeleton')

from sense_hat import SenseHat
from marble_maze_lib import MarbleMaze
import random
from senselib import *
```

Without going into detail, these lines are called *import statements*. They are used to *import* code from other Python files into your own file. This is useful because you can use other people's code to simplify your own.

The next part of the code (Sec. 1.2) creates some important *Objects* (don't worry if you're not sure what that means) that we'll use in the game code.

Sec. 1.3 is where we set up the initial properties of the game is set up. Like the starting position of the ball, the colour of the ball, the colour of the wall, the goal and the holes and so on. You'll need to edit these values later on in order to customize the game.

Sec. 1.4 is where you'll create your maze!

Sec. 2 is the part where you'll get your hands dirty with some real python game programming. If you look at it now, there's not much there:

```
#### 2. Main game code

while True:

    #### 2.1 Make the ball move

    #### 2.2 Draw the walls, holes, goals and the ball

    #### 2.3 Update the game

    #### 2.4 Check if the player has won or lost
```

The most important thing to note about the code is the following:

```
while True:

    # ... The rest of the code ...
```

The *while* part of the code is what we call the *main loop*. All the code that's *inside* the while-clause will be repeated again and again until the game ends. That's why we call it a loop!

Writing the code

(Sec. 1.3) Set up the game variables

In this section we'll be defining some variables that we will use in the game. A lot of programming is just about knowing what information to store, and where to store it. For example, some very important information to store is the position of the marble ball on the screen. To set the ball position, you have to edit this line:

```
game.setBallPosition(0, 0)
```

Replace the first number with the *x*-position of the ball, and the second number with the *y*-position. If you don't know what that means, the x-position is how many squares the ball is away from the left side of the screen, and the y-position is how many squares the ball is away from the top side of the screen.

The rest of the lines in this section decides the colours of all the things in the game. Currently all the colours are set to black, so you'll definitely have to change them to something, although you can decide which colours yourself.

Colours in Python are not called "purple", "yellow" or "brown". Instead they are given by *three* numbers. The first number is how *red* the colour is, the second number is how *green* it is, and the third how *blue* it is. The value of the redness/greenness/blueness can be from 0 to 255 (where 255 is the maximum redness/greenness/blueness). For example to set the ball to be purple, we set it to be maximum red and maximum blue, like this:

```
game.ballColour = [255, 0, 255]
```

Confusing? Try setting the colours to different things later to get the hang of it.

(Sec. 2.2) Draw the walls, the holes, the goals and the ball

In this section we'll write code that draws stuff on the Sense HAT.

The first thing we want to do is to draw the floor. You do this using the function:

sense.clear(red, green, blue)

You can replace *red*, *green* and *blue* with any numbers that you like. You should add this code to the start of the section.

The next thing is to add code that will draw the walls, the holes and the goal blocks. There's already some code that will do this for you, all you need to do is to add the following lines of code to the section:

```
game.drawWalls()
game.drawHoles()
game.drawGoals()
```

These functions will draw all the walls, the holes and the goals on the screen.

The final thing is to draw the ball. To do this you use the function:

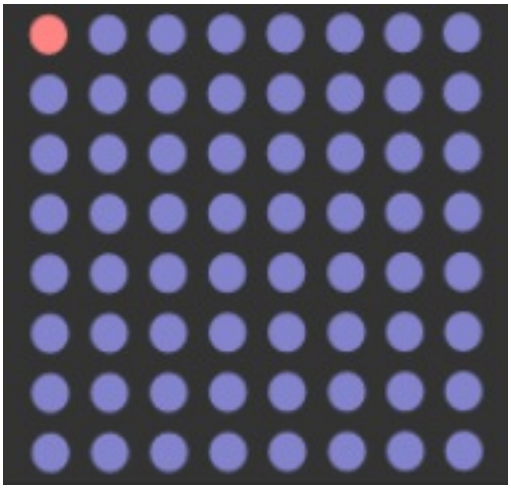
```
sense.set_pixel(x, y, red, green, blue)
```

Where *x* and *y* is the position of the ball, and *red*, *green* and *blue* the colour of the ball. You can get the *x* and *y* positions of the ball using:

```
game.getBallX()
game.getBallY()
```

You'll have to use these functions within the *sense.set_pixel*. The colour of the ball is in the *game.ballColour* variable. For more information about how the function works, you can check the *Function Reference* document.

After you've added all of this, all the stuff should be drawn on the scren if you start it! Because you haven't yet added anything, it should look similar to this:



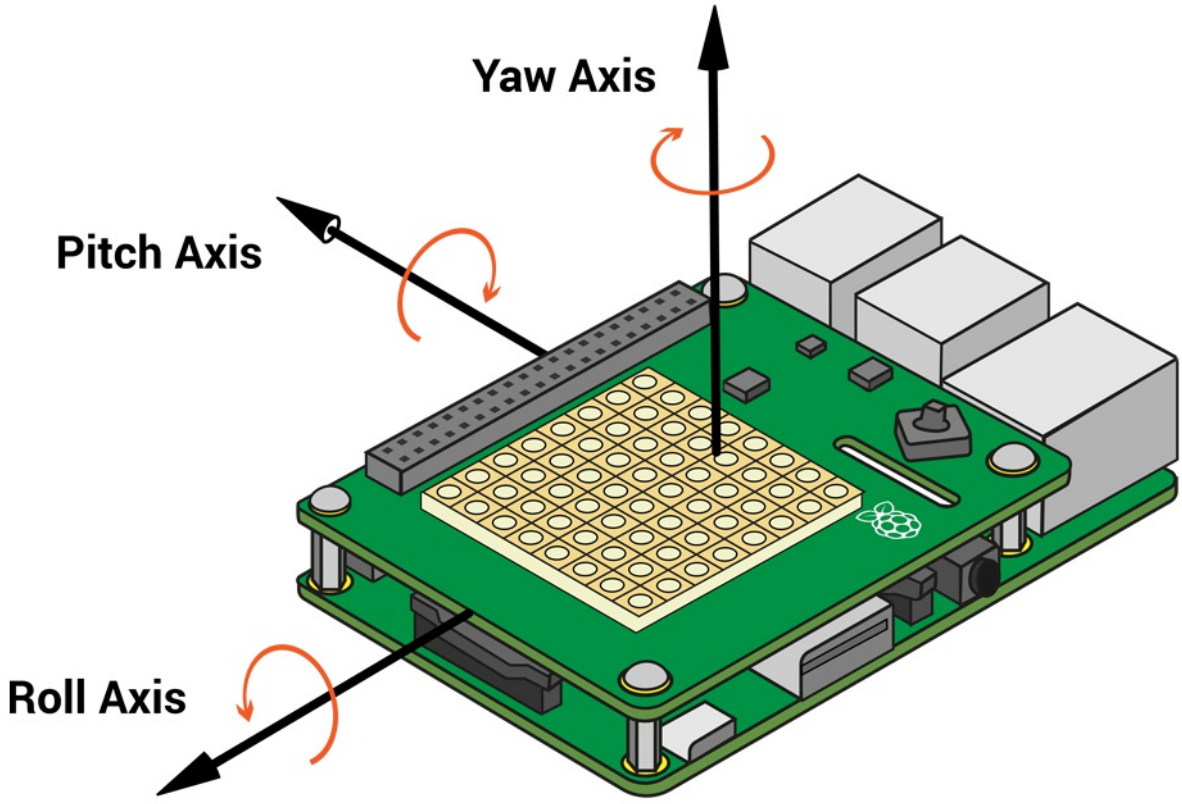
(Sec. 2.1) Make the ball move

In this section, you'll be making the ball move by tilting the Sense HAT.

Using the *game.moveBallHorizontally* and *game.moveBallVertically* functions, we can tell the game how tilted the Sense HAT is, and how fast the ball should move across the screen. For example, to tell the game that HAT is tilted *20 degrees* horizontally, and *40 degrees* vertically:

```
game.moveBallHorizontally(20)
game.moveBallVertically(40)
```

The technical terms for these kinds of "*tilts*" are *yaw*, *pitch* and *roll*. Look at the image below to get an idea for how they work:



To get how much the Sense HAT is tilted horizontally ("pitch"), use:

```
sense.get_orientation()["pitch"]
```

To get how much the Sense HAT is tilted vertically ("roll"), use:

```
sense.get_orientation()["roll"]
```

After you've written this code correctly, run the game. The ball should move across the screen as you tilt the Sense HAT.

(Sec. 1.4) Create the maze!

This is a fun section! Here, you'll be creating the maze.

To create a wall pixel:

```
game.placeWall(x, y)
```

where *x* and *y* should be replaced by numbers. To create a hole pixel:

```
game.placeHole(x, y)
```

if the ball falls in the hole, the player will lose. To create a goal pixel:

```
game.placeGoal(x, y)
```

At the start, all the pixels are automatically floor pixels, so you don't need to add them.

After you've made the maze, try running the game. All the things you placed should be on the screen!

(Sec. 2.3) Make things happen

This one is simple, just add the following line:

```
game.update()
```

This line should make sure that the ball collides properly with the walls.

(Sec. 2.4) Check if the player has won or lost

As we mentioned earlier, you can get the *x* and *y* positions of the ball using:

```
game.getBallX()
game.getBallY()
```

The following function can be used to check whether there's a hole at a particular position:

```
game.isHole(x, y)
```

The function will return *True* if there's a hole at the position (*x*, *y*).
The following function can be used to check whether there's a goal at a particular position:

```
game.isGoal(x, y)
```

The function will return *True* if there's a goal at the position (*x*, *y*).

It might seem a bit tricky but using these functions you should be able to check whether a player has lost (if the ball's position is on a hole) or has won (if the ball's position is on a goal).

The way to do this is using an *if*-statement. If you forgot what that was, check the introduction lecture from earlier, or ask a supervisor.

If the player has lost or won, the ball should be placed in the starting position again, so the player can play again.

Finished!

If it's all done, correctly, the game should now work! Don't worry if it doesn't, things often go wrong in programming. Errors in code are usually called *bugs*. If you have a bug in your code, you'll have to *debug* it!

If it works, congratulations! You can either move on to another project or try to come up with new things to add to the current project. Use your creativity! You can discuss any ideas you have with a supervisor.

Author: Lukas Kikuchi

Date: August 09, 2017

Academic Leads Help Sheet

Go4Code Coding Summer School 22-24 August 2017

This sheet will guide you through what you should expect on the day and some possible troubleshooting answers.

Timetable

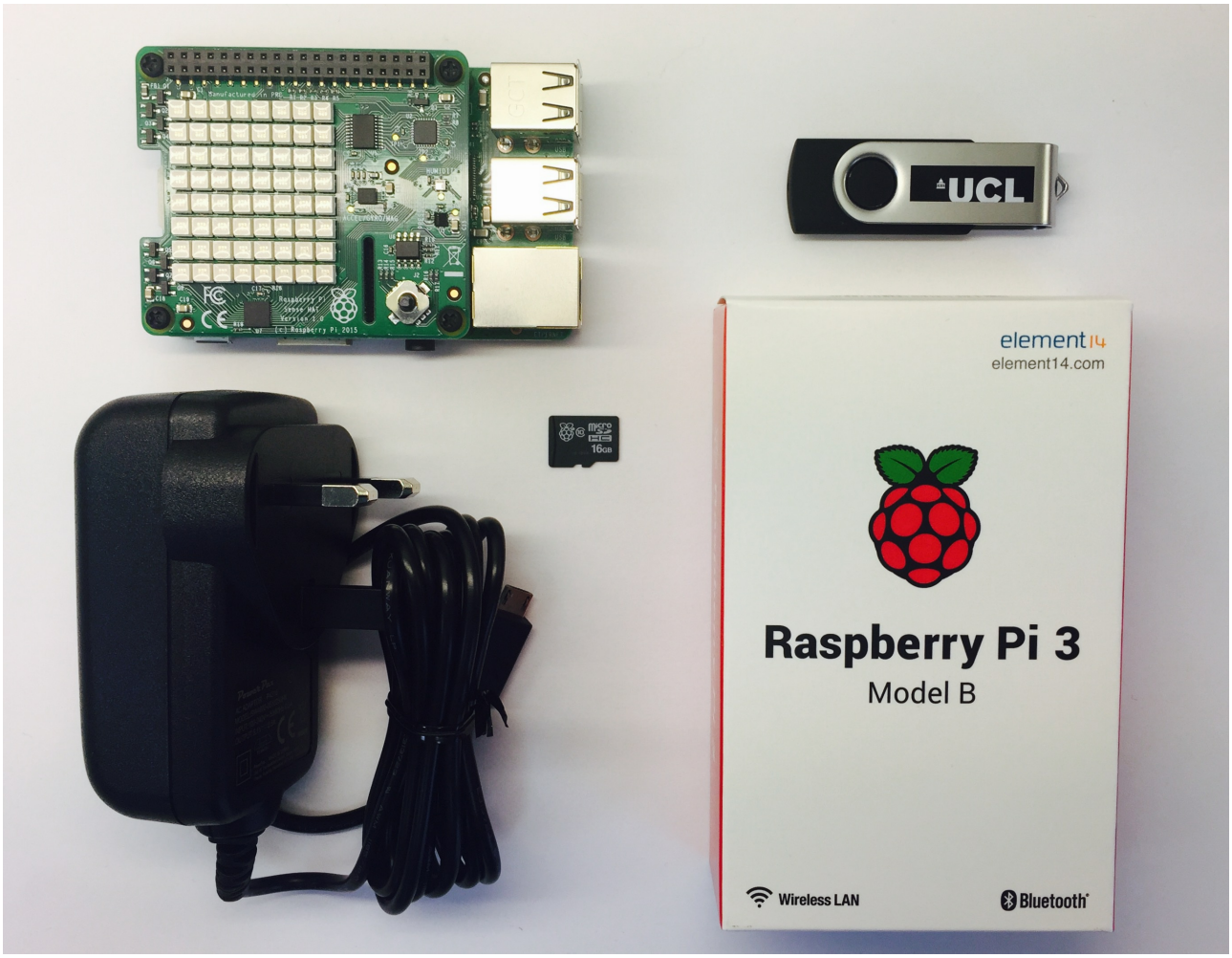
Day 1	Day 2	Day 3
10:00 - 11:00 Arrivals	09:45 - 12:30 Project Work	09:45 - 12:15 Project Work
11:00 - 11:45 Welcome Speeches and meet the ambassadors	12:30 - 13:30 Lunch	12:15 - 12:30 Group Portrait

12:30 - 13:00 Ice breakers	13:30 - 16:30 Project Work	12:30 - 13:15 Lunch with parents
13:00 - 14:00 Introduction Lecture by Ishan Khurana	16:15 Students depart	13:15 - 14:00 Science Fair
14:00- 16:00 Project Work		14:00 - 15:00 Panel Discussion with Jenny Sivapalan (Guardian), Jay (UCL ExoPlanets), Simon Jolly (UCL Proton Beam Therapy)
16:15 Students depart		15:00-15:30 Evaluation and Certificates

Equipment

Each student will have:

- Raspberry Pi 3
- Sense HAT
- Power supply
- Pre-programmed SD card
- USB memory stick



The Raspberry Pi's and Sense HATs will be given pre-assembled to the students. The SD cards will already be on their holder in the Pi.

Each academic lead will have their own equipment so they can demonstrate the projects to the students. There is spare equipment if any appear faulty.

Project Work

- Each academic lead will be look after a group of 5-6 students
- Programming language - Python 3
- Projects are available on: Coding Summer School Webpage (https://codingsummerschool.github.io/codingsummerschool)
- Full solutions are available at: Solutions (https://codingsummerschool.github.io/codingsummerschool/teachers/trinket_links.html
    - Solutions are labelled "*Solution (with code)*"
- Projects are split into difficulties: easy, medium and hard

Students will be writing scripts on the Trinket Emulator to develop and debug using the skeleton code provided. To run code on the RPi they must create a python file by copying the code in *main.py* file in Trinket to a new file in IDLE and save this file as a *.py* file. The names mustn't contain spaces.

The RPi has been configured to copy the scripts from the memory stick over. If scripts with the same name are already on the Pi they will be overwritten.

Scripts will then be available to select their scripts from a digital list.

Please familiarise yourself with the project scripts, this will help both the students and yourself!

Libraries and Skeleton Code

The Function Reference Webpage (https://codingsummerschool.github.io/codingsummerschool/docs/function_reference.html) is a list of useful functions that students may wish to use.

You should be familiar with the function in the document above and the python API at: pythonhosted.org/sense-hat/api/ (https://pythonhosted.org/sense-hat/api/)

Skeleton code and object classes for projects will already be pre-loaded onto the Pi's.

Day 1

- The programme will be introduced by Ishan at 13:00 in the Darwin Lecture Theatre.
- You must arrive at your resective cluster room by 13:30 to help the student ambassadors login to the computers and set up the room.
- Students will first work through the webpage. Introduction to Python and Sense HAT page (https://codingsummerschool.github.io/codingsummerschool/docs/SenseHatIntro.html). It is important that students go through and understand the webpage as it is required to complete the projects.
- Variables and conditional statements are important concepts that the students need to be familiar with.
- The aim is to get each student to the end of the webpage by the end of session, so

they are ready to start the projects the next day - students can start projects if they find the worksheet easy.

Day 2

- Full day of project work.
- All students should of finished the introduction to programming webpage.

Day 3

- Two hours of project work to finish projects.
- In the afternoon, the students will be showcasing their work in a Science fair to their parents.
- A panel discussion with 2 UCL academics and a software developer from the Guardian - this is a Q&A session between the students and the panel guests. Feel free to ask questions as well.

Troubleshooting

Here is a list of potential problems that could occur:

- Script appears on the Raspberry Pi but does run the file as expected:
    - Check the script has the following lines at the start:

```
import sys (1,'home''/home/pi/.Go4Code/g4cSense/pong/skeleton')
from sense_hat import SenseHat
from pong import Pong
import random
from senselib import *
```

    - Sense HAT libraries need to be imported
    - Make sure indentation is correct. This is a very common error. If 2 space indentation doesn't work try 4 space.
    - This example comes from the pong project, which has a library and class in the skeleton code
    - Skeleton code can be found on the Trinket emulator which is on the project script page
- Encourage students to try things out by themselves
- Please either *text* Ishan (+44 7528692298) or *call* Laura (+44 7891138825) if you encounter any difficulties

Project: Board for Snakes and Ladders

Difficulty level: Medium/Hard

Description

In this project you will create a *Board* for the classic board game snakes and ladders.

You can try out the complete version of the project here:

Use the up, left, right and left arrows to move the brush. Press the middle button (ENTER) to paint a pixel.

Project Manual

This project guide will tell you step-by-step the main things you have to do in order to create a *countdown clock*. For some of the steps, you'll have to use your own creativity to proceed, good luck!

Introducing the project

The first thing you should do is open the *skeleton code* for the project.
In programming, skeleton code means code that only has the basic elements of a program. It is up to you to fill in the rest!

You can find the skeleton code on Trinket, here:
https://goo.gl/rXr3vQ (https://goo.gl/rXr3vQ)

If you can, you should also create and account and log in to Trinket. This will allow you to save the Trinket projects. Otherwise you have to copy the code on to your computer to save it.

On Trinket, you'll be able to test your code on a *virtual* Sense HAT, before you try your code on the real thing.

As you might see, the skeleton code is split up into sections, divided by the headlines.
For example:

```
#### 2. Code section
```

This guide will go through the various sections (not necessarily in order), and help you write your code. **Very important note:** *You should add the code in the specified section in your skeleton code as you follow this guide.*
The next part of this guide will explain the stuff that's already in the skeleton code when you first open it.

Explanation of the skeleton code

Before we get on to the coding, it's worth looking over the *skeleton code* and make sure you are familiar with it.

The first few lines in the script are:

```
#### 1.1 Import libraries

import sys
sys.path.insert(1,'/home/pi/Go4Code/g4cSense/skeleton')
sys.path.insert(1,'/home/pi/Go4Code/g4cSense/graphics')
from sense_hat import SenseHat
import random
from senselib import *
```

Without going into detail, these lines are called *import statements*. They are used to *import* code from other Python files into your own file. This is useful because you can use other people's code to simplify your own.

The next part of the code (Sec. 1.2) creates some important *Objects* (don't worry if you're not sure what that means) that we'll use in the later on.

Sec. 1.3 is where the initial variables of the project are set up.
You'll need to edit these values later on in order to customize the it.

Sec. 2 is where all the main coding will take place.

Sec. 2.1 is where we'll write code that controls the *brush*, that places paint on the screen.

Sec. 2.2 is where we'll write code that changes the colour of the brush.

Writing the code

(Sec. 1.3) Set up the variables

In this section we will define some variables which we will use later on.

The starting position of the paint brush is in the variables *cursorx* and *cursory*.

The possible paint brush colours are stored in the list *colours*. Your first job will be to add some colours to this list. As you might remember, in programming a colour is usually represented as *three* numbers *[r, g, b]*. Where the first number is how red the colour is, the second how green it is, the third how blue it is.
For example, to add the colours red, green and blue to the available brush colours:

```
colours = [
      [255, 0, 0],
      [0, 255, 0]
      [0, 0, 255]
    ]
```

The final variable is *currentColour*. This variable is a bit trickier to understand. currentColour is a number that decides which colour the paint brush is currently drawing. For example, if we set the *colours* list to what was shown above:

- *currentColour = 0* would mean the brush is red.
- *currentColour = 1* would mean the brush is green.
- *currentColour = 2* would mean the brush is blue.

Later on we'll let the user change the brush colour by shaking the Sense HAT, this will be achieved by randomly changing the value of *currentColour*.

(Sec. 2) Main program code

Before we start coding, we'll explain a bit what's going on in Sec. 2.

In this part of the code, we'll do all the main programming.
If we look at the skeleton code, we see that the section is within a *while*-loop. Remember that code inside a while loop runs again and again, until we tell it to stop. The reason why we want our program to be in a loop is because we want to continually check if the user has shaken the device.

Because the code is in the *while*-loop, remember that you have to add a *Tab* at every line, to make it indented. Like this:

```
while True:
    # write your code like this,
    # with a tab at the start of the line.
```

(Sec 2.1) Control the paint cursor and place colours on the screen

Inside the while-loop, you should write the following:

```
for event in sense.stick.get_events():
    if event.action == "pressed":

        if event.direction == "up":
            # Fill in with your own code

        elif event.direction == "down":
            # Fill in with your own code

        elif event.direction == "left":
            # Fill in with your own code

        elif event.direction == "right":
            # Fill in with your own code

        elif event.direction == "middle":
            # Fill in with your own code
```

This code will check if the user has pressed the *up*, *down*,*left*, *right* or the *middle* buttons on the joystick. Don't mind the complicated structure of the code above, try to figure out how to use the code by looking at it.

Inside the *if*-statements ou'll find the comments *# Fill in with your own code*. This is where you should write your code.

If the user presses *up*, *down*,*left* or *right* on the joystick, the variables *cursorx* and *cursory* should be changed. We'll give you a hint to start off with: If the user presses the up button, cursory should *decrease* by 1.

Bonus: One thing that you might want to add is that if the user pushes the brush off the screen, it should pop back up on the other side of the screen. You can do this using further if-statements.

Finally, if the user presses the *middle* button, the brush should paint a pixel at the position of the brush. Remember that the available brush colours are in the variable *colours*, and the current brush colour is given by *currentColour*. So you can get the current colour, and then draw it on the screen, using:

```
c = colours[currentColour]
sense.set_pixel(cursorx, cursory, c[0], c[1], c[2])
```

2.2 Change colour by shaking

This part is a bit complicated, so pay attention! Don't be afraid to ask a supervisor if you don't understand something.

To see if the user has shaken the Sense HAT, we'll have to use the *accelerometer* inside the device. The accelerometer basically tells us how fast the device has been accelerating. To get the information from the accelerator, use the following code:

```
ac = sense.get_accelerometer_raw()
x = ac["x"]
y = ac["y"]
z = ac["z"]
```

Now, in the *x*, *y* and *z* variables, we have information recorded by the accelerometer. This is all a bit complicated, but the basic gist of it is this, calculate this:

```
shake = x*x + y*y + z*z
```

We have stored *x times x plus y times y plus z times z* into the variable *shake*. Now, the larger the variable *shake* is, the more has the Sense HAT been shaken.
We're going to say that if *shake* is *larger* than *5*, the Sense HAT has been shaken. To do this, you're going to have to use an *if*-statement.

If you have detected that the Sense HAT has been shaken (in other words, if *shake* is larger than 5) you want to randomly change the current brush colour.

Remember that the current brush colour is given by *currentColour*. We can generate random numbers using the function *random.randint*. You can read about in the *Function Reference* document. If we have 5 different colours to choose from, we should randomly assign *currentColour* with a number from 0 to 4 (in Python, we usually start counting from 0, rather than from 1).

To get the length of a list (like *colours*), you can use the *len* function. You can use this to generate a new brush colour every time the user has shaken the Sense HAT:

```
currentColour = random.randint(0, len(colours) - 1)
```

Finished!

If it's all done, correctly, the game should now work! Don't worry if it doesn't, things often go wrong in programming. Errors in code are usually called *bugs*. If you have a bug in your code, you'll have to *debug*it!

If it works, congratulations! You can either move on to another project or try to come up with new things to add to the current project. Use your creativity! You can discuss any ideas you have with a supervisor.

Bonus: Add a visible cursor on the screen

Currently, when you draw on the screen, you can't see the brush as you move it. If you want a real challenge, figure out how you can have the brush always be visible on the screen.

The main issue that complicates this is that you don't want the brush leaving a trail as you move it on the screen. You'll have to somehow *save* how the screen looks before you draw the brush, and then when you move the brush again, restore that picture. It's a bit confusing, so try reading it again or discuss it with a supervisor.

Two functions that you'll have to use is the *sense.get_pixels* and *sense.set_pixels* functions (note that the latter function is *sense.set_pixel*, but with an *s* at the end). The first function saves the pixels on the screen into a variable, and the second one can redraw those pixels at a later stage.

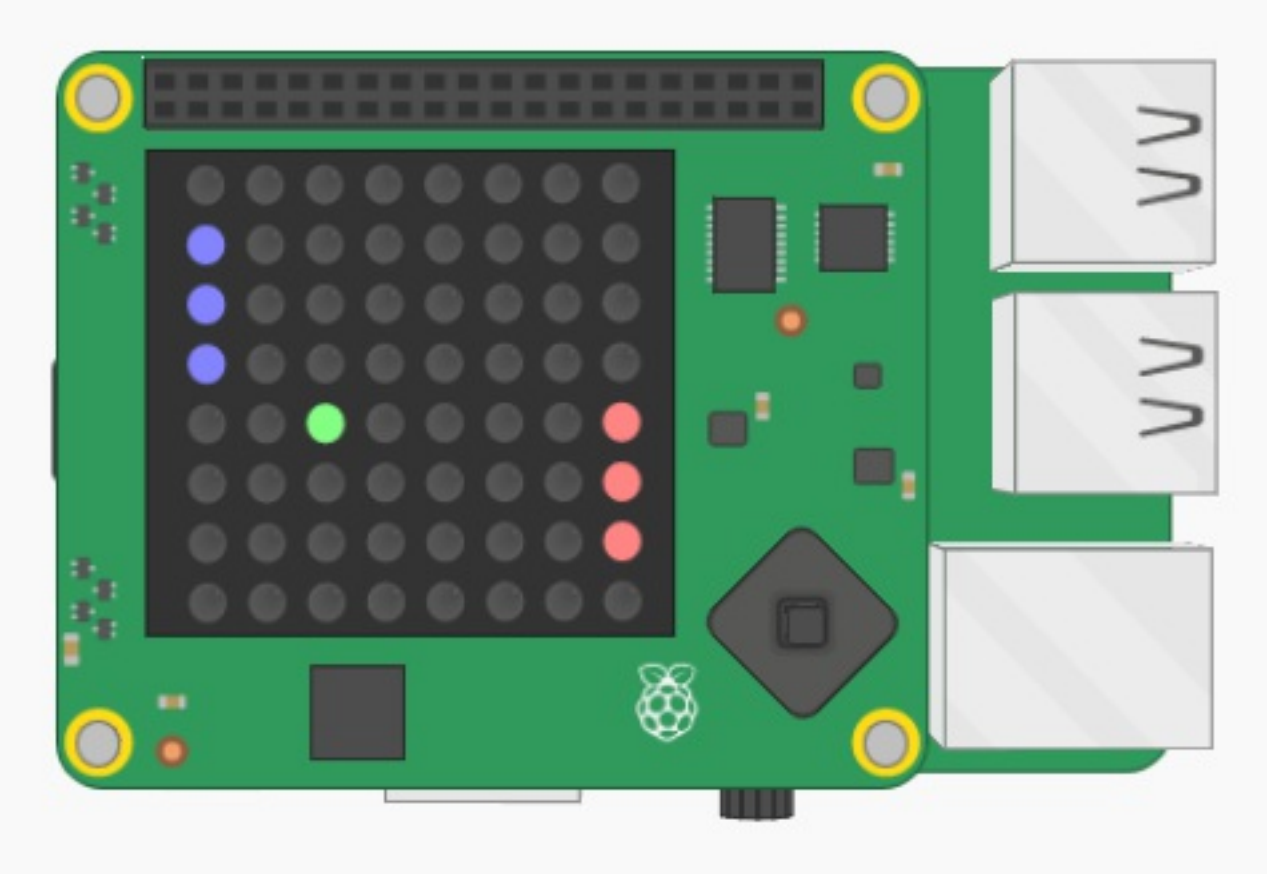
```
image = sense.get_pixels() # Save the pixels on the screen

sense.get_pixel(1, 1, 255, 255, 55) # Draw some stuff
sense.get_pixel(7, 3, 77, 34, 0) # Draw some stuff
sense.get_pixel(2, 5, 22, 222, 111) # Draw some stuff

sense.set_pixels(image) # The screen is as it was from the beginning!
```

Author: Lukas Kikuchi

Date: August 09, 2017

Project: Pong

Difficulty level: Very hard



Description

Note: This project is very hard, so you might want to attempt an easier project before doing this one.

In this project you'll be creating the classic game *Pong* on the Sense HAT. The game will be made from ground-up, so it's definitely one of the harder projects to do.

You can try out the complete version of the game here:

Use the up and down keys to move your paddle.

Project Manual

This project guide will tell you step-by-step the main things you have to do in order to create *Pong*. For some of the steps, you'll have to figure out how to proceed yourself, good luck!

Introducing the project

The first thing you should do is open the *skeleton code* for the project.
In programming, skeleton code means code that only has the basic elements of a program. It is up to you to fill in the rest!

You can find the skeleton code on Trinket, here:
https://goo.gl/77PmJ8 (https://goo.gl/77PmJ8)

If you can, you should also create and account and log in to Trinket. This will allow you to save the Trinket projects. Otherwise you have to copy the code on to your computer to save it.

On Trinket, you'll be able to test your code on a *virtual* Sense HAT, before you try your code on the real thing.

As you might see, the skeleton code is split up into sections, divided by the headlines. For example:

```
#### 2. Code section
```

This guide will go through the various sections (not necessarily in order), and help you write your code. **Very important note:** *You should add the code in the specified section in your skeleton code as you follow this guide.*

The next part of this guide will explain the stuff that's already in the skeleton code when you first open it.

Explanation of the skeleton code

Before we get on to the coding, it's worth looking over the *skeleton code* and make sure you are familiar with it.

The first few lines in the script are:

```
import sys
sys.path.insert(1,'/home/pi/.Go4Code/g4cSense/pong/skeleton')


from sense_hat import SenseHat
from pong import Pong
import random
from senselib import *
```

Without going into detail, these lines are called *import statements*. They are used to *import* code from other Python files into your own file. This is useful because you can use other people's code to simplify your own.

The next part of the code (Sec. 1.2) creates some important *Objects* (don't worry if you're not sure what that means) that we'll use in the game code. This part of the code also displays a message to the user when first starting the game:

sense.show_message("PONG!") # Display an intro message for the viewer.

Sec. 1.3 is where we set up the initial properties of the game,
like the position of the ball, the positions of the player and the computer,
the colour that the should drawn in, and so on. This is the first part of the skeleton
code that you will be writing in.

Sec. 2 is the part where you'll get your hands dirty with some real python game
programming! If you look at it now, there's not much there:

Main game code

```
while True:

  #### 2.1 Check for player1 movement (joystick)

  #### 2.2 Let computer control player2

  #### 2.3 Find new ball position

  #### 2.4 Check if new ball position has hit the player pads

  #### 2.5 Check if new ball position is in goal

  #### 2.6 Check if new ball position has collided with the top or bottom walls

  #### 2.7 Update ball position

  #### 2.8 Clear screen

  #### 2.9 Draw player1 and player2

  #### 2.10 Draw ball

  #### 2.11 Wait a little bit before the next frame
```

That's a lot of code to fill in! Don't worry though, this guide will take you
through each step of the way. The most important thing to note is

```
while True:

    # ... The rest of the code ...
```

The *while* part of the code is what we call the *main loop*. All the code that's
inside the while-clause will be repeated again and again until the game ends.
That's why we call it a loop!

Writing the code

(Sec. 1.3) Set up the game variables

In this section we'll be defining some variables that we will use in the game.
A lot of programming is just about knowing what information to store, and where
to store it. For example, some very important information to store is the position

of the Pong ball on the screen.

You see that we have two players: player1 and player2. Player 1 is the (human) user playing the game, and player 2 is controller by a computer. Player 1 is on the left side of the screen, and player 2 is on the right.

If you look at the values in the skeleton code, you'll notice that they're all set to zero. You'll have to think of some more appropriate values to set them to! For the colours of the players and the ball, change them to whatever you want them to look like. If you leave the colours like they are, you won't see them on the screen, as they'll be coloured black!

By reading the comments of the variables, you can figure out what they mean (if you don't, ask a supervisor!). There are two variables that might look a bit strange to you at first:

```
pong.ballVelocityX = random.choice([-1, 1]) # Randomizes the vertical velocity
of the ball.
pong.ballVelocityY = random.choice([-1, 1]) # Randomizes the horizontal
velocity of the ball.
```

These variables are the horizontal and vertical velocities of the ball (horizontal means the left and right directions, and vertical means the up and down directions).
If, for example, *pong.ballVelocityX = -1* and *pong.ballVelocityY = 1* ,the ball is moving to the left and down at the same time (so the ball is moving diagonally to the bottom-left).

The following piece of code

```
random.choice([-1, 1])
```

randomizes the initial velocity of the ball. The result of that is *either* -1 or 1, by random.

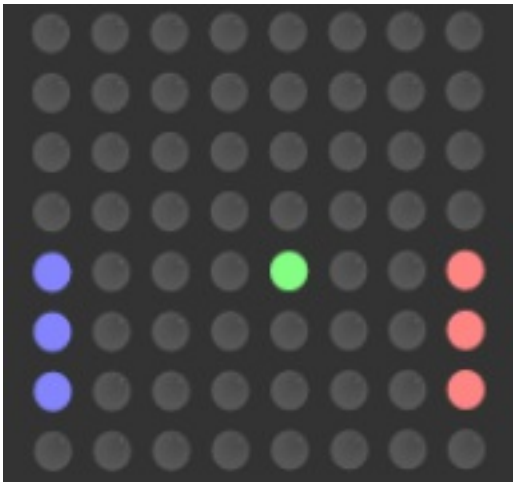
(Sec. 2.8-2.10) Draw the players and the ball

These sections are very straightforward. Use the *sense.set_pixel* and the *drawVerticalLine* functions to draw the players and the ball on the screen. Check the *Function reference* document
to see how you use these functions.

The position of the ball is given by *pong.ballX* and *pong.ballY*. The colour of the ball is in *pong.ballColour*. Since the ball is only one pixel, you should *sense.set_pixel*.

Similarly, you should use the *drawVerticalLine* function to draw the player1 and player2 paddles. Player1 should be on the left side of the screen, so its x-coordinate should be 0, whilst player2 is on the right side of the screen, so its x-coordinate should be 7.

Once you've done this, you can try running the game. The screen should look something like



In Sec. 2.8, add the following line, in order to clear the screen each game loop. This is important, as otherwise if the ball moves across the screen, it'll leave a trail!

```
sense.clear()
```

(Sec. 2.1) Control player1

In this section you'll add some code that lets the user control player1. Read the *Checking the joystick* section in the *Function reference* document to see how you could go about doing this.

What you want to do is to check if the user presses up on the joystick, in which case you'll decrease the value of pong.player1pos by 1.

```
pong.player1pos = pong.player1pos - 1
```

Or if the user presses down

```
pong.player1pos = pong.player1pos + 1
```

(Sec. 2.2) Let computer control player2

This part is easy. We have prepared some code, that will control the player2 paddle for us. Just add the following line to the

```
pong.updatePlayer2()
```

If you really want a challenge, you could skip using our code and try to do it yourself!

(Sec. 2.3) Ball movement

From now on the programming gets a bit trickier, so pay attention to every detail!

Every frame, the ball should move a step (otherwise Pong wouldn't be much of a game). The direction of the movement is given by the velocity variables we went through earlier: *pong.ballVelocityX* and *pong.ballVelocityY*.

In Sec. 2.3 the code should calculate the new position of the ball. The code should look something like this:

```
newBallX = # ... Your code here
newBallY = # ... Your code here
```

Figure out what you should write after the equal signs!

(Sec. 2.4) Check if new ball position has hit the player paddles

This part is tricky! We now have to see if the if the ball has bounced off of the player paddles. I'll go into detail about how we would check this for player1, and then you can figure out how we'd do it for player2 yourself.

If the ball has hit the player1 paddle, newBallX should equal to 0 (it should be on the leftmost part of the screen). For the y-coordinate of the ball, it should be such that it's aligned with the y-coordinate of the paddle, adjusting for its size.

If the ball collides with the paddle, the pong.ballVelocityX should be reversed (if it's negative, it should become positive. If it's positive it should become negative).

Here's the code that would accomplish this:

```
if newBallX == 0:
  if pong.ballY >= pong.player1pos and pong.ballY <= (pong.player1pos +
pong.player1size):
    pong.ballVelocityX = -pong.ballVelocityX
    newBallX = pong.ballX + pong.ballVelocityX
    newBallY = pong.ballY + pong.ballVelocityY
```

(Sec. 2.5) Check if new ball position is in goal

Now we have to check ball has landed in either of they player's goals. Once again, I'll go through the code for player1, and you'll figure it out for player2 yourself.

It's confusing, but the way we'll check if the ball has landed in the goal is if the x-coordinate of the ball is -1. That is, if it's gone beyond the boundary of the screen. If it has, it is safe to assume that player2 has scored.

If the ball has gone into the goal, there are a couple of things we need to do:

- Reset the ball position to the middle of the game.
- Add a point to the player score.
- Show a victory message (use the function sense.show_message)

(Sec. 2.6) Check if new ball position has collided with the top or bottom walls

This bit might seem tricky, but it's actually quite simple. If the ball collides with either the top or the bottom part of the game screen, the pong.ballVelocityY has to be reversed, like this:

```
pong.ballVelocityY = -pong.ballVelocityY
```

(Sec 2.7) Update ball position

You need to update the variables that store the ball position with newBallX and newBallY here, otherwise the ball won't move at all!

(Sec. 2.11) Add a delay to each game loop

To prevent the game from moving too fast, we're going to add some time-delay to each loop of the game. We'll basically tell the computer to wait *X* amount of time before starting another loop. You can do this using the *wait* function. Try to find an appropriate amount of time to wait.

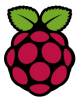
Finished!

If it's all done, correctly, the game should now work! Don't worry if it doesn't, things often go wrong in programming. Errors in code are usually called *bugs*. If you have a bug in your code, you'll have to *debug* it!

If it works, congratulations! You can either move on to another project or try to come up with new things to add to the current project. Use your creativity! You can discuss any ideas you have with a supervisor.

Author: Lukas Kikuchi

Date: August 09, 2017



Using the Sense HAT

Go4Code

Your Sense HATs have been pre-programmed by the Go4Code team to do some clever things in the background so that that you can easily run python files without needing to plug the Raspberry Pi into a TV or monitor. All you need to do is put a python file with your code on a USB stick and plug it into the Raspberry Pi!

In this guide we will go through the process of transferring code written in Trinket on the desktop to running it on the Raspberry Pi.

Writing Code in  Trinket

Each project script has a link to a Trinket Emulator where you will develop code to complete the projects. Now that you have gone through the introduction to programming session, you should be slightly familiar with programming the Sense HAT.

To demonstrate how to run code on your Sense HATs we will be displaying a simple message on the Sense HAT screen.

Click on this link to open the Trinket Emulator for this guide: https://trinket.io/python/bf3cadd959. This should open the Trinket Emulator web-page.

If you like you can create a Trinket account by clicking the *Sign Up* icon in the top left corner, it takes 2 minutes and will allow you to save your projects so you can access them from anywhere.

Task: In the Trinket Emulator on the web-page, write code that will display the message "Hello World" on the Sense HAT's display.

Click the run button below to see what your code should do:

Trinket Emulator

If your code displays the "Hello World" message as shown above, you are ready to transfer it to the Raspberry Pi!

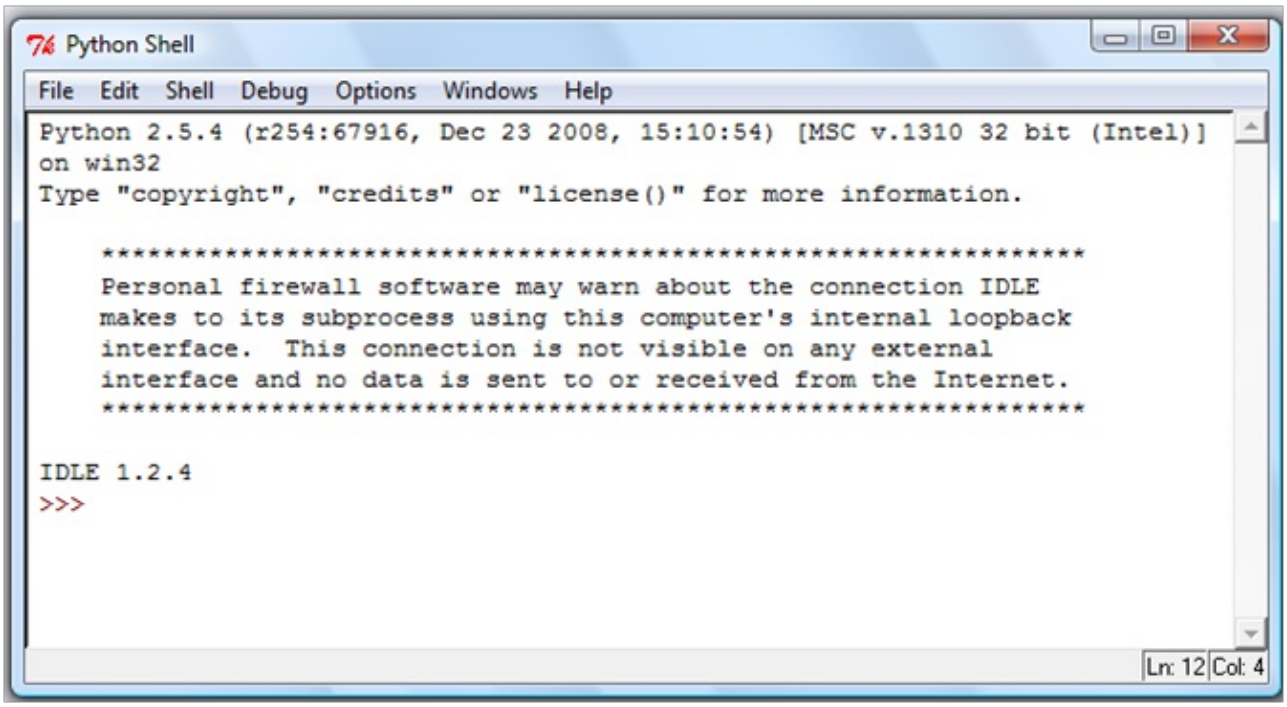
Creating python files in IDLE

Some of you may be familiar with IDLE. Don't worry if you've never seen it before, it is just a program in which we can write and run python code. We will be using it to create python files.

Before we start, go to documents and create a folder called Go4Code. You should save all your python files in this folder.

Let's open the IDLE program on your computers. Click start and type IDLE in the search bar and press enter.

This will open up a window like this:



The Python Shell

Click file -> new file. This should open up a window like the one below:

Copy your code in here as shown in the images below:

Now go to *File* and click *Save As* and save the file as test.py in . You can call your files anything really, as long as the names do not have spaces and end the with *.py* extension. This tells the computer that the file we have created is a python file.

Run Your Code on the Raspberry Pi

If you don't understand anything in this section, feel free to ask your supervisor
Now copy the python file you just created, *test.py*, onto the memory stick. Now **safely remove** the USB drive and plug it into the Raspberry Pi.

The Raspberry Pi will now copy the code from the USB stick onto it's own memory and will display the name of your file on the screen.

This is the Main Menu on our Raspberry Pi Sense HAT interface. It has all the python files that we copy on to the Raspberry Pi using the memory stick. If you move the joystick to the right, the screen will show the next file in the folder, if you move it to the left it will show the previous

file. To select and run a file press the middle button on the joystick on the Sense HAT.

Recap

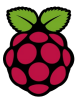
So, just to go over things again. To write code and run it on your Raspberry Pi:

1. Write and test your code on the Trinket Emulator.
2. Open IDLE and copy the code from Trinket into a new file.
3. Save the file with your code with any name as long as it ends with the *.py* extension.
4. Copy the python file (*.py* file) onto your USB stick.
5. Safely remove your USB stick and plug it into your Raspberry Pi.
6. Press the middle button to run a file or press left and right to scroll through and find a different file.

Great now you are ready to start on your own project!

Author: Ishan Khurana

Date: August 10, 2017

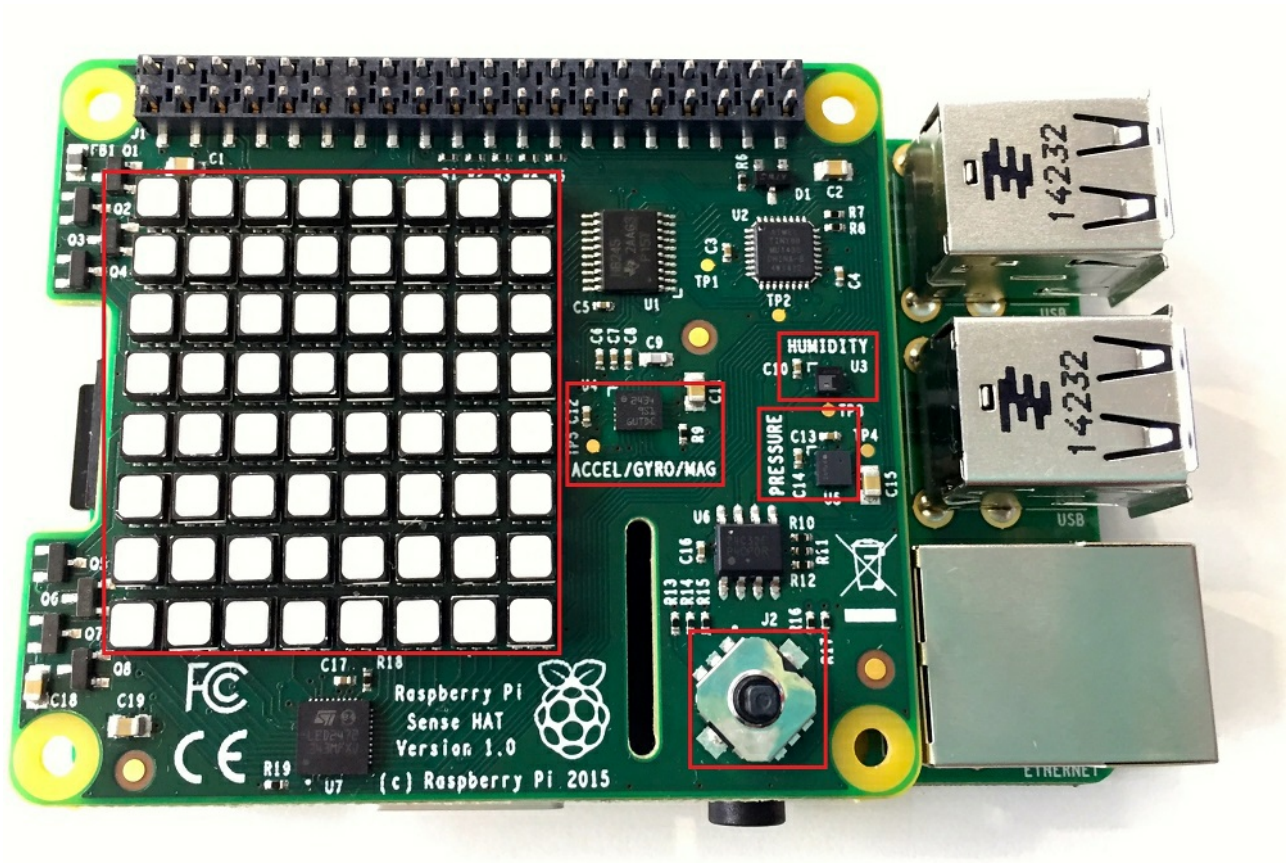


Raspberry Pi Sense HAT

Go4Code

Introduction

The Sense HAT is an add-on board for the Raspberry Pi. It gives us the ability to measure all kinds of things with the Raspberry Pi, make games and display things with the LED lights. Let's firstSe go through what is actually on the board!



The Sense HAT: The main features are outlined in red.

Features:

- The LED Matrix on the left has 64 (LEDs) arranged in an 8 x 8 grid. They can be used to display shapes, icons and messages. The screen is one of the ways the Raspberry Pi can communicate with us, which is very important for making games!
- The small microchip labelled **ACCEL/GYRO/MAG** is what is known as the inertial measurement unit (IMU). This actually has three sensors!
    - The first sensor is the **accelerometer**. This measures how acceleration and will be useful in detecting how fast the Raspberry Pi has been moved.
    - The second sensor is a **gyroscope**. This sensor basically measures if the Raspberry Pi has been rotated. We can measure rotation in three directions, but more on this later.
    - The third sensor is a **magnetometer**. This measures the magnetic field and can be used to detect the magnetic field caused by the rotation of the Earth.
    - Next is a **humidity sensor**. This will allow you to measure air moisture. It's the small chip just below the text HUMIDITY. You can also use it to measure ambient temperature.
- There is also a **pressure sensor** for measuring air pressure, something which is certainly important in space. It's the chip to the right of the text PRESSURE.

- Last but not least is the **joystick**. The Raspberry Pi cannot be connected to a USB keyboard or mouse in space, so the Sense HAT has its own five button joystick. This is the silver rectangle in the bottom right corner with the small stick poking out of the top. It can move up, down, left, right, and allow middle-clicks.

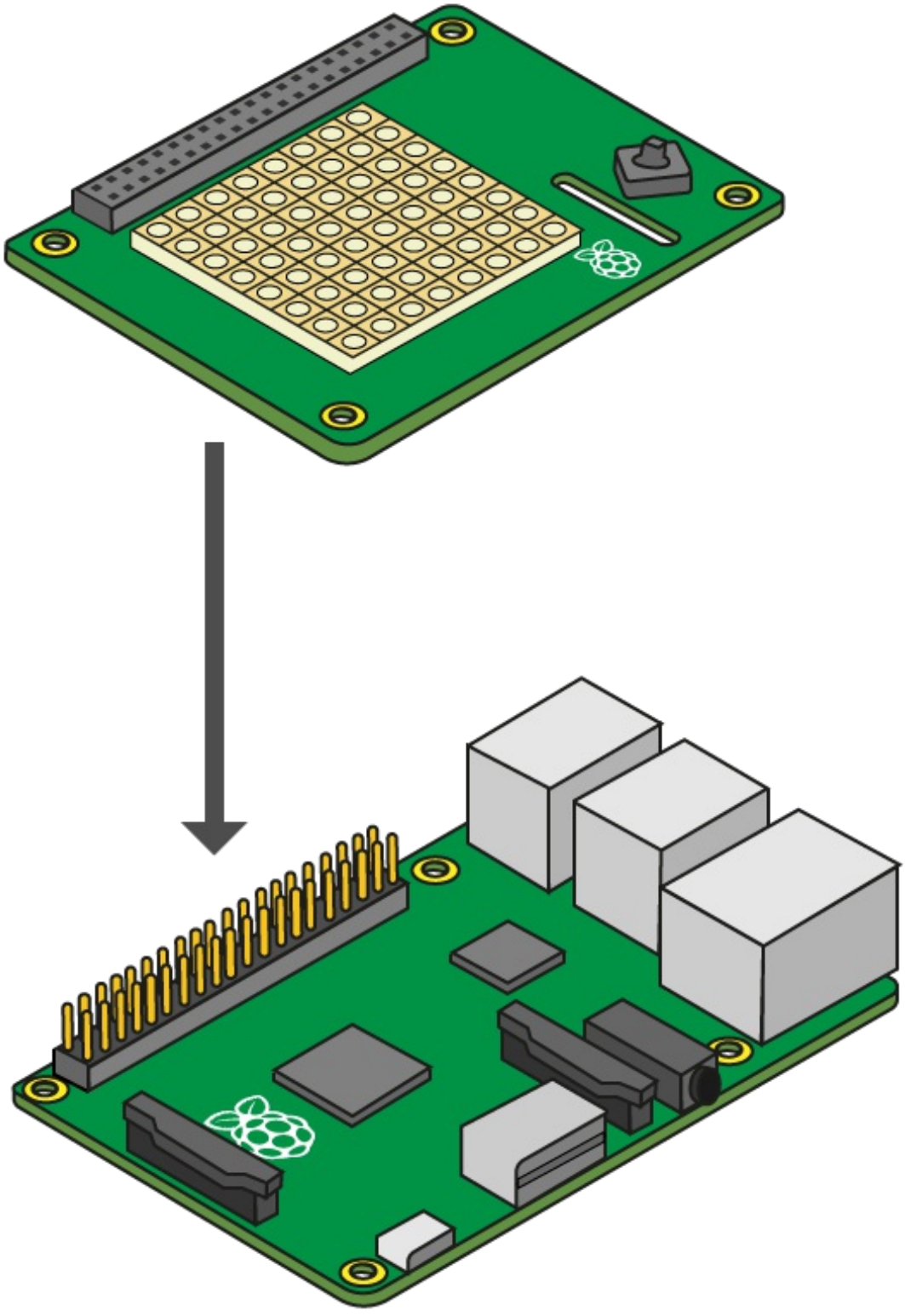
These simple features allow us to do countless things with the Sense HAT, we have come up with a few projects that you will do. To let you really express your imagination and creativity, you get to keep the Sense HAT and Raspberry Pi so you can come up with your own gadgets!

The Assembly

This section is optional. You may skip to the next section

The Sense HAT is just one type of HAT (Hardware Attached on Top) that you can get for your Raspberry Pi. Some more examples of HATs are listed here (https://shop.pimoroni.com/collections/hats).

The Sense HAT you have for the summer school has been preassembled by the Go4Code team. But if you wanted to take it apart and use a different HAT, it would be good to know how it is put together.



The Sense HAT Assembly

- The Sense HAT connector is first attached into the GPIO pinouts on the Raspberry Pi (the golden pins in the image above).
- Then four hexagonal stands are placed between the Sense HAT and the Raspberry Pi on

circular holes.
- These stands are then screwed on to securely attach the Sense HAT to the Raspberry Pi

Programming the Sense HAT

We will be using the *Python* programming language to write code that will run on the Raspberry Pi and control the Sense HAT. This guide will take you through the basics of Python programming.

The end goal is to get you ready to start doing some programming of your own, in the various projects we have prepared for you.

We start right from the beginning, so you don't need to have done programming before. If you have, then use this guide as a helpful reminder.

 The Trinket Emulator

In your projects, you'll have your own Sense HAT to play around with. But since you'll have to test around and tinker with your projects for quite some time before its ready, we're going to be testing all our code on a *virtual* Sense HAT on a website called *Trinket*.

In your project scripts you will get a trinket page to test and develop your code before you transfer it on the the Raspberry Pi.

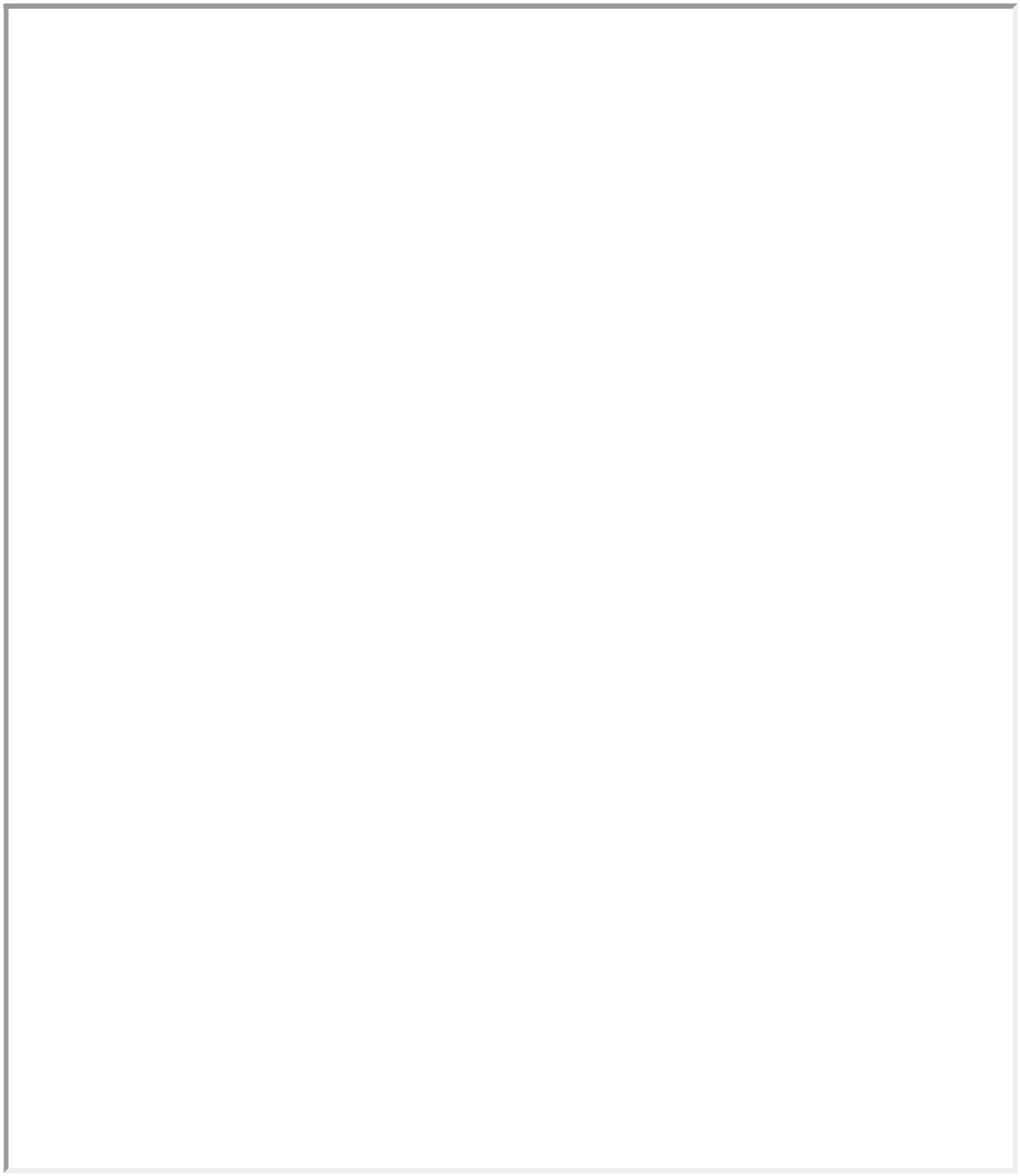
We have embedded the Trinket Emulator into this document so you can learn to code here before you start developing your own code.

Trinket Emulator

The left side of the page has some Python code already. The right side of the page shows nothing in the beginning, but if you press the Run button ▶ in the top left corner, a virtual Sense HAT should appear on the screen. The code in the there right now just starts the Sense HAT, but doesn't actually tell it to do anything. Let's create out first program!

1. Showing Messages

In this section we will go over showing messages on the Sense HAT. You should use the Emulator below to test out the code.



Trinket Emulator

Copy the following into the Python code pane in the Emulator above:

```
sense.show_message("Hello world!", 0.05)
```

After you have done that, click Run again. The message, "Hello World!" should scroll by on the screen. You've written your first Python program!
You're now *officially* a Python programmer.

The red text you see in the Python code is what we call a *string*. A string is basically a piece of text. In Python, we can write strings by either enclosing text in double quotation marks like this

```
"This is a string!"
```

or single quotation marks, like this

```
'This is a string!'
```

The *show_message()* function (don't worry if you don't know what a function is yet, we'll go over this later) displays whatever *string* you put inside the bracket, on the Sense HAT.

Now, we're going to play around with this program that you've made.

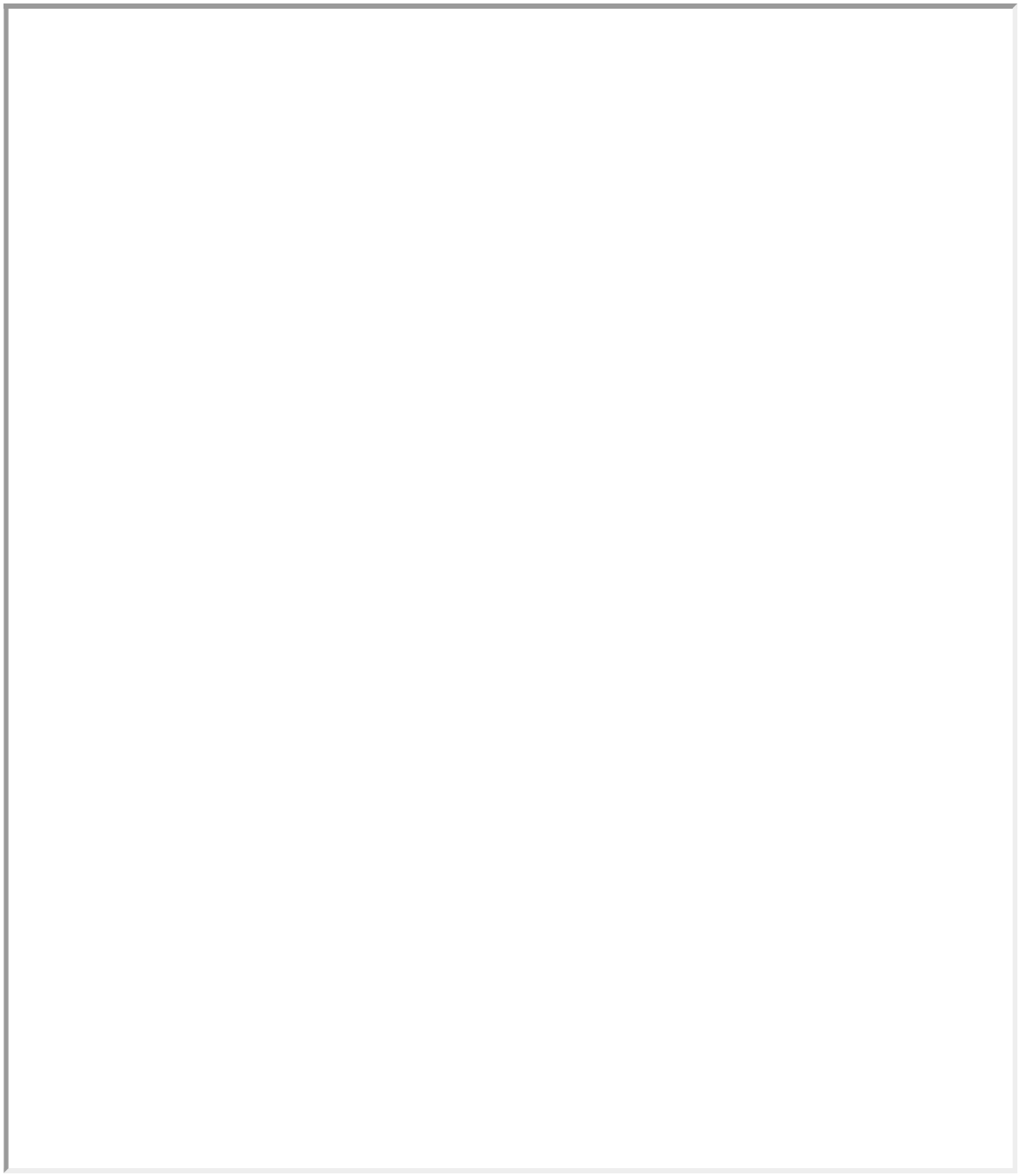
Exercise 1: Change the message to something else. Try doing it using single-quotation marks as well.

Exercise 2: You might have noticed that there's a number after the string.
Try changing the value of that number. Find out what that number does.

2. Colours in Python: RGB -values

In this section we will be changing the colour of the light emitted by the LEDs.

Run the code in the emulator below:



Trinket Emulator

Run it again, and see what happens. The text should now be red. The last part of the line that you just wrote `text_colour=[255, 0, 0]` decides which color the text should be in. You can change these three numbers to change the color of the text. The numbers in the square brackets specify the colour of the light emitted from the pixel.

We call triplets of numbers that specify colour RGB-values, and this is the reason:

- The first number decides how *red* the light should be,
- the second number decides how *green* the light should be,
- and the third number decides how *blue* the light should be.

The pixels on the Sense HAT are made up of tiny LEDs that emit red, green and blue light. You may remember from the talk that light is made up of different colours and that we can combine light of different colours to make new colours.

The maximum redness/greenness/blueness in RGB format 255, and the lowest is 0. As you saw above, the RGB-value for red is [255, 0, 0].
And, for example, the RGB-value for black is [0, 0, 0]. Note: the pixels won't actually show black as there is no such thing as black light. Black is just the lack of light. By entering an RGB value of [0,0,0] we just turn the LEDs off.

We can also set the back colour using the code below. Replace the show_message() function in emulator with the one below:

```
sense.show_message("Hello world!", 0.05, text_colour=[255, 0, 0], back_colour =
[0,255,255])
```

Use what you have just learnt to complete the exercise below. Your code should go in the Trinket emulator above. If you don't understand something, feel free to ask your supervisor.

Exercise 3: Make the text flash by in the following colors:

- Red
- Blue
- Green
- White
- Purple
- Yellow
- Orange

You'll have to experiment with the RGB-values to try to find the right color-combinations!

Commenting

Before we get going with some more interesting code, let's remove the *sense.show_message* line so that it won't disturb us later on. But instead of just erasing the line completely, we can *comment it out*.

To do this, simply add a hashtag # at the start of the line, like this:

```
# sense.show_message("Hello world!", 0.05, text_colour=[255, 0, 0])
```

Try running the code after you've commented the code out. No text should appear on the screen. In Python we often use comment symbol # to add explanations to our code, for example:

```
# This line displays some text on the screen
sense.show_message("Hello world!", 0.05, text_colour=[255, 0, 0])
```

3. Functions

Before we move on, I'll explain (as promised) what a *function* is. A function is simply something that we use in a code to do something for us. For instance we used the *show_message* function earlier to display text on the Sense Hat's LED matrix.

Let's look a bit closer at the *show_message* function.

```
sense.show_message("Some String", 0.05, text_colour = [255,0,0],back_colour =
[0,255,255])
```

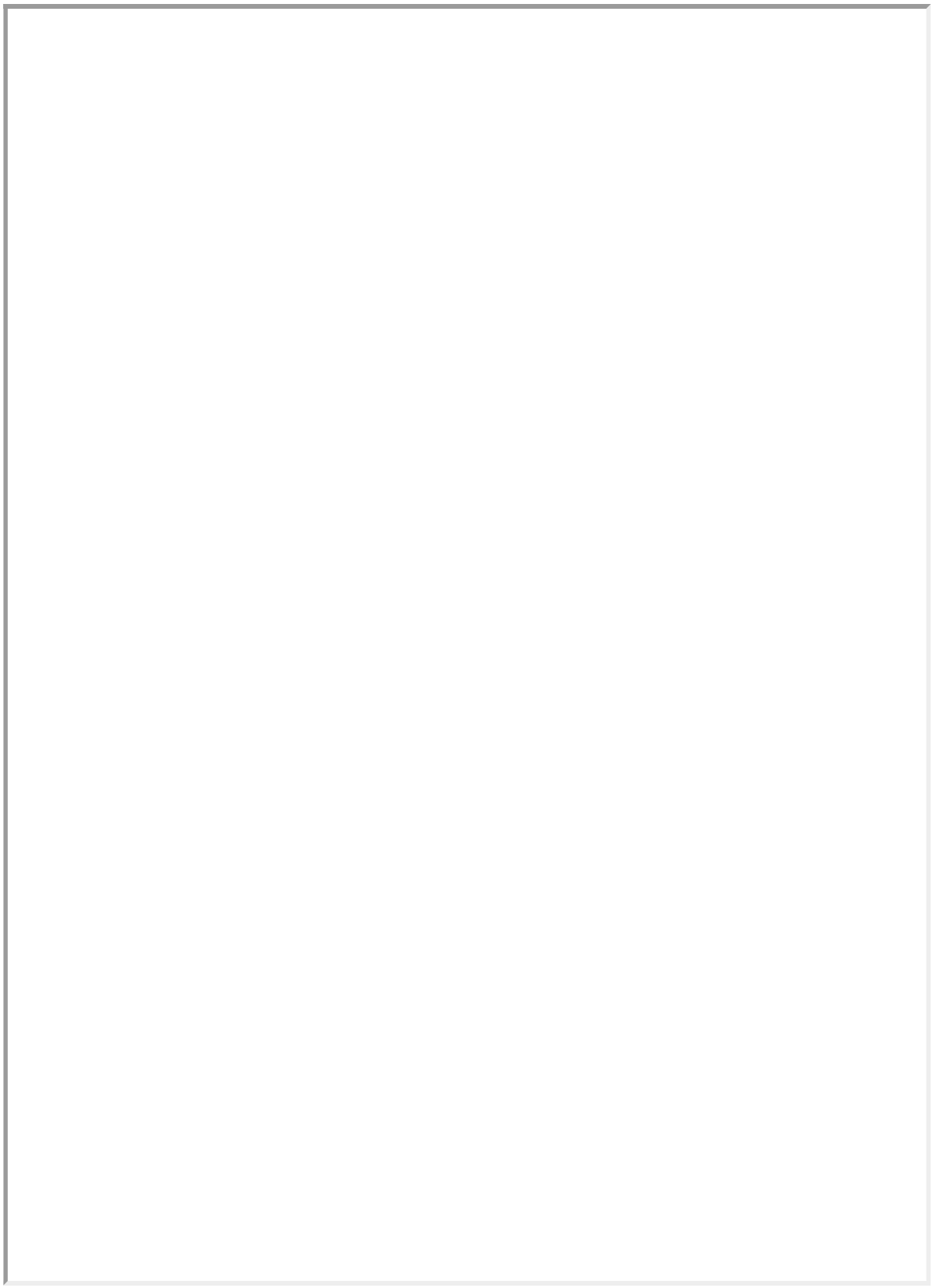
The name of the function is *show_message*. When you use a function, it's often referred to as *calling* the function. We are calling the function on our Sense HAT (we have named the Sense HAT *sense*) by writing *sense.show_message*.

What about all those numbers inside the brackets? Those are called *arguments*. Sometimes when we call a function, we want to specify exactly what we want it to do, we do this using *arguments*.

In this case, the arguments are: the message that will be displayed on the screen, the scroll speed of the message, the text colour and the back colour.

Don't worry if all of this sounds a bit complicated. It's a lot of names for something that's quite simple. If you have any questions about this, discuss it with a supervisor.

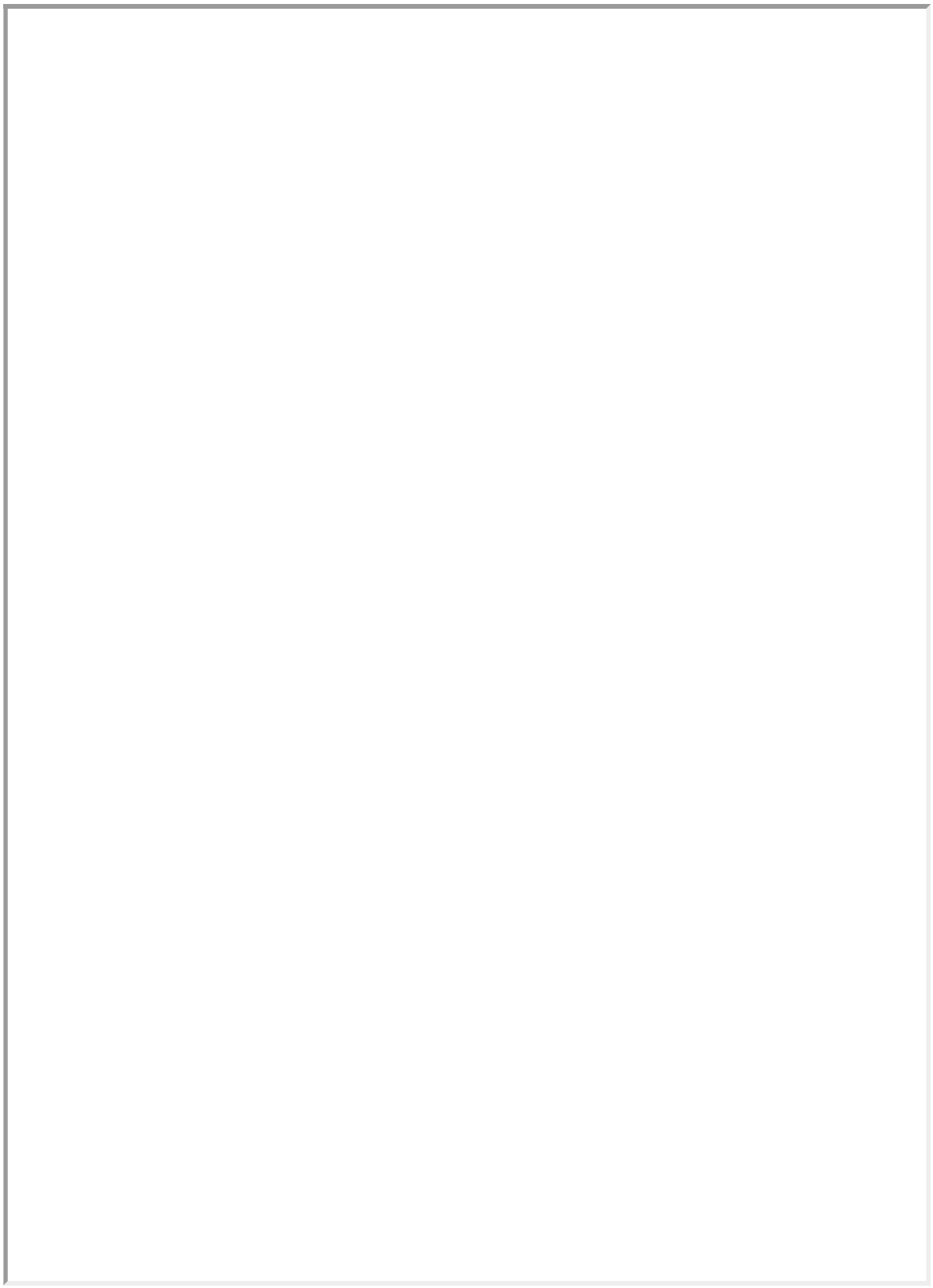
Before we move on, we'll introduce the *print* function. The print function is very simple, it just displays some text. The important difference here is that it doesn't print out text on the screen, but in the *terminal* underneath the Sense HAT screen on Trinket. It is easiest for you to see for yourself:



Trinket Emulator

You should see some text appear in the lower-right corner of the page. Try changing it to make it print something else.

4. Using Python as a calculator



Trinket Emulator

Next thing we'll do is try out some simple mathematical operations in Python.
All the main maths signs you know from school exist in Python. Using Python we can add, subtract, multiply or divide numbers. Copy the following code into the emulator above to try it for yourself:

```
print(3+5)
print(2-7)
print(2*5)
print(4/2)
```

Run the code and see what the results are. You can also add really large numbers

```
print(1234567890000 + 987654321111)
```

Or do several mathematical operations in a row

```
print(23*52*13 + 52*611 - 412 )
```

We can also use brackets "*()*" when calculating things

```
print((2+3)*5)
print(6 + ((100 - 1)*42)/32)
print(((2 * (23 - 3) / (12 * 23)) - 32)*62)
```

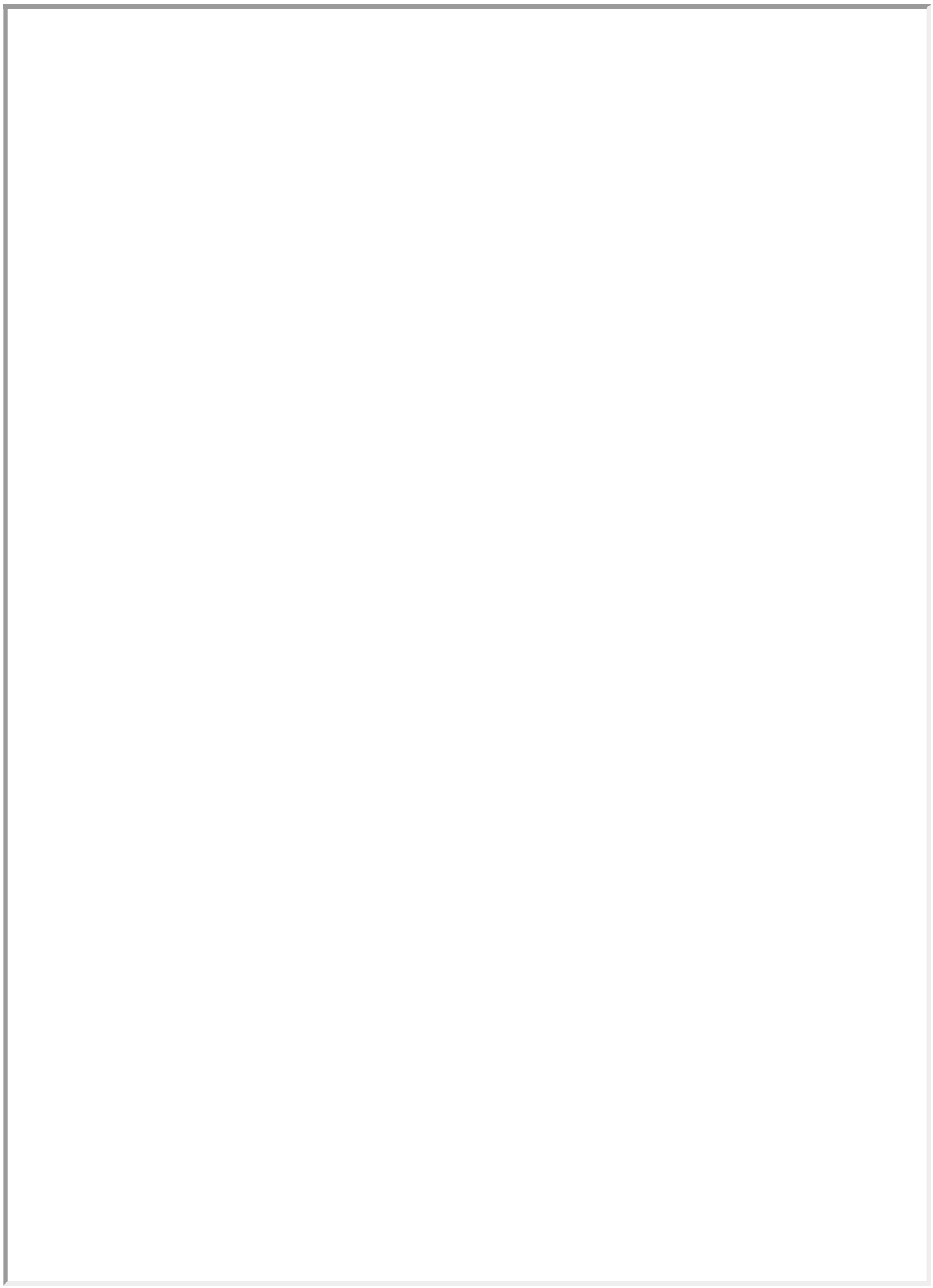
Try changing the numbers, and try to do different calculations.

Exercise 4: In *Trinket*, translate the following into Python code:

```
_(four times ( six minus two ) plus five) divided by two_
```

and use *print* to show the result.

5. Variables



Trinket Emulator

Python allows us to store information in *variables*. We can use the assignment operator (the *equal-to* sign '=') to assign a value to a variable.

There are three components to variable assignment.

- First we need to decide on the name of our variable, like my_variable.

- Second, we set the variable name equal to something using the *equal-to* sign '='.
- Third, we write the value that the variable should be equal to.

Like this:

```
my_variable = 123
```

We can also store strings in a variable:

```
my_variable = "Hello!"
```

Note that the variable name must be written as *one single set of characters* without spaces. So, for example, *my variable* would not be a valid name for a variable.

The code below shows how easy it is to define and display a variable.

```
myNumber = 5*7
someText = "Five times seven is "
myMessage = someText + str(myNumber)

sense.show_message(myMessage, 0.05)
```

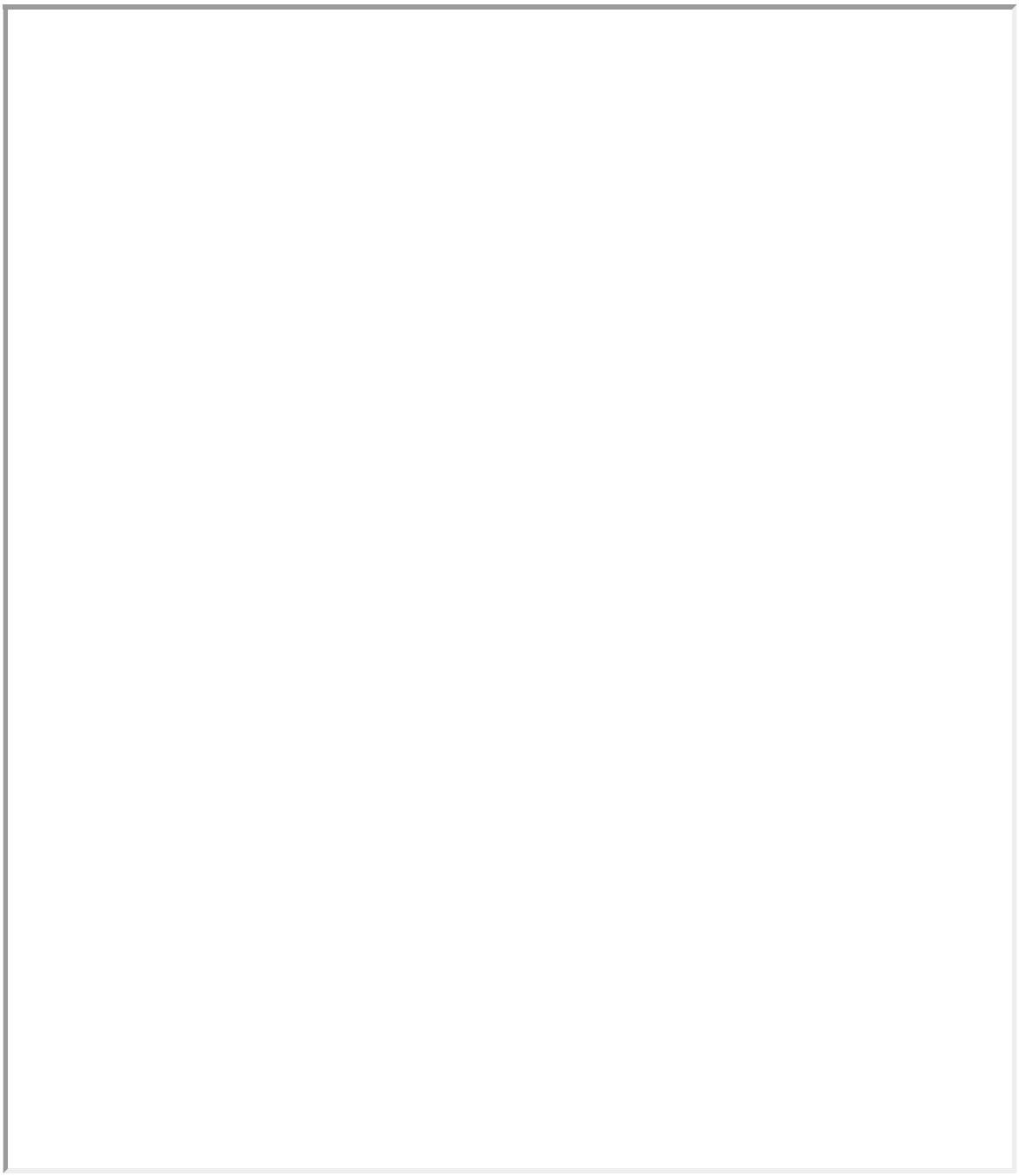
Can you look at this code and figure it out what it does without actually running it?
After you have guessed, write the code in the Trinket Emulator above and run it.

Explanation: We have used the *str* function to convert the number 5*7 to a *string* "35". Then we added the string "Five times seven is " with "35" to form "Five times seven is 35".

Exercise 5: Write a program that displays your full name and age on the screen. Store your full name in a variable called *myName*, and store your age in a variable called *myAge*. Remember that you have to convert numbers to strings using the *str* function.

6. A basic program

Variables are quite useful, as they allow us to remember numbers (or other types of data, like *strings*) in terms of more memorable variable names. We can use these variables in our programs to compute things. In the Trinket below is an example of a basic program that calculates how many minutes there are in a week.



Trinket Emulator

You don't have to copy this code, you can go on the following link to see the code in action: https://goo.gl/CZH1UX (https://goo.gl/CZH1UX).

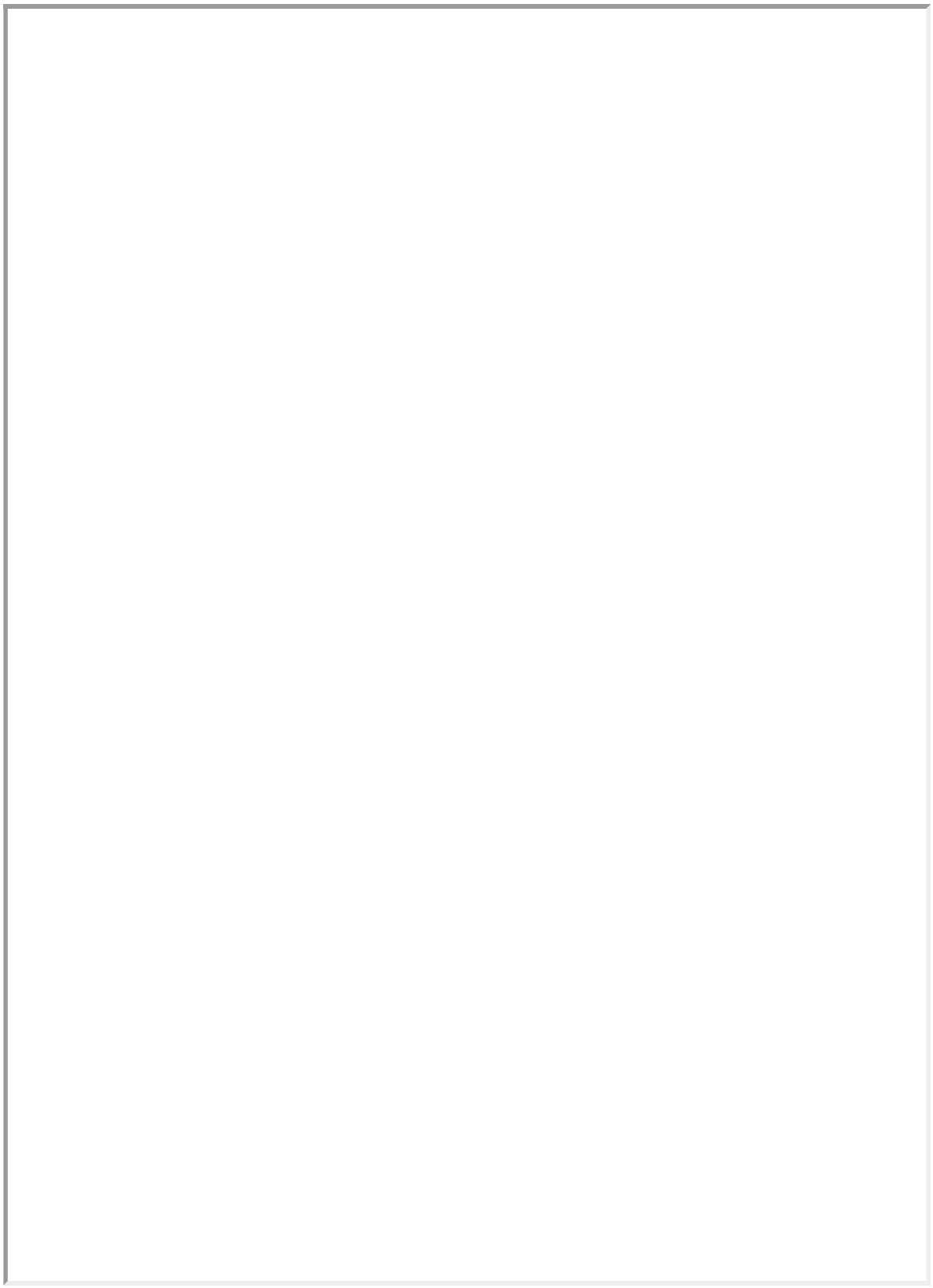
But before you run it, try to just read the code and figure out what it does yourself. It's good practice and will help you become a better programmer!

Exercise 6: Alter the code in the Trinket above to show the number of minutes in a week on the Sense Hat's display. Hint: the *show_message* function displays messages given as *strings*.

7. Drawing stuff on the Sense HAT: *sense.set_pixel*

It's really easy to draw things on the Sense HAT screen. We can do this using the function *sense.set_pixel* Try adding the following lines to your code in the emulator.

```
sense.clear()
sense.set_pixel(2, 2, 255, 255, 255)
sense.set_pixel(4, 2, 255, 255, 255)
sense.set_pixel(3, 3, 255, 255, 255)
sense.set_pixel(2, 4, 255, 255, 255)
sense.set_pixel(4, 4, 255, 255, 255)
```



Trinket Emulator

It should draw a white cross on the screen. The first line, *sense.clear*, resets all the pixels on the screen to black. This is just to avoid any clutter on the screen if you make mistakes.

Let's see how the *sense.set_pixel* function works.

You'll recognize the last three numbers of each line, those set the colors, as we learned previously. The first two numbers decide *where* we'll draw the pixel.

The first number is what column on the screen the pixel should be drawn at. We usually call this the *x-coordinate*.

The second number is what row on the screen the pixel should be drawn at. We usually call this the *y-coordinate*.

Exercise 7: Play around with the code, and try do draw something of your own. Maybe the first letter of your name?

Exercise 8: Try placing the sense.clear() line at the end of the code. What happens? Can you explain why? After that, place the line in the middle of the code. Again, what happens, and can you explain why that happens?

8. If-and-else statements

This next part is where things get more interesting (and a bit more complicated)!

The *if*-statement allows us to make a program that reacts in different ways, under different conditions. Think of it as a way to ask a computer questions.

Look at the code below, can you guess what it does?

```
myName = "Lukas"

if myName == "Ishan":
  sense.show_message("Your name is Ishan", 0.03)
else:
  sense.show_message("Your name is not Ishan", 0.03)
```

Copy the code into your project. How can you change the variable *myName* so that the result of the program is different?

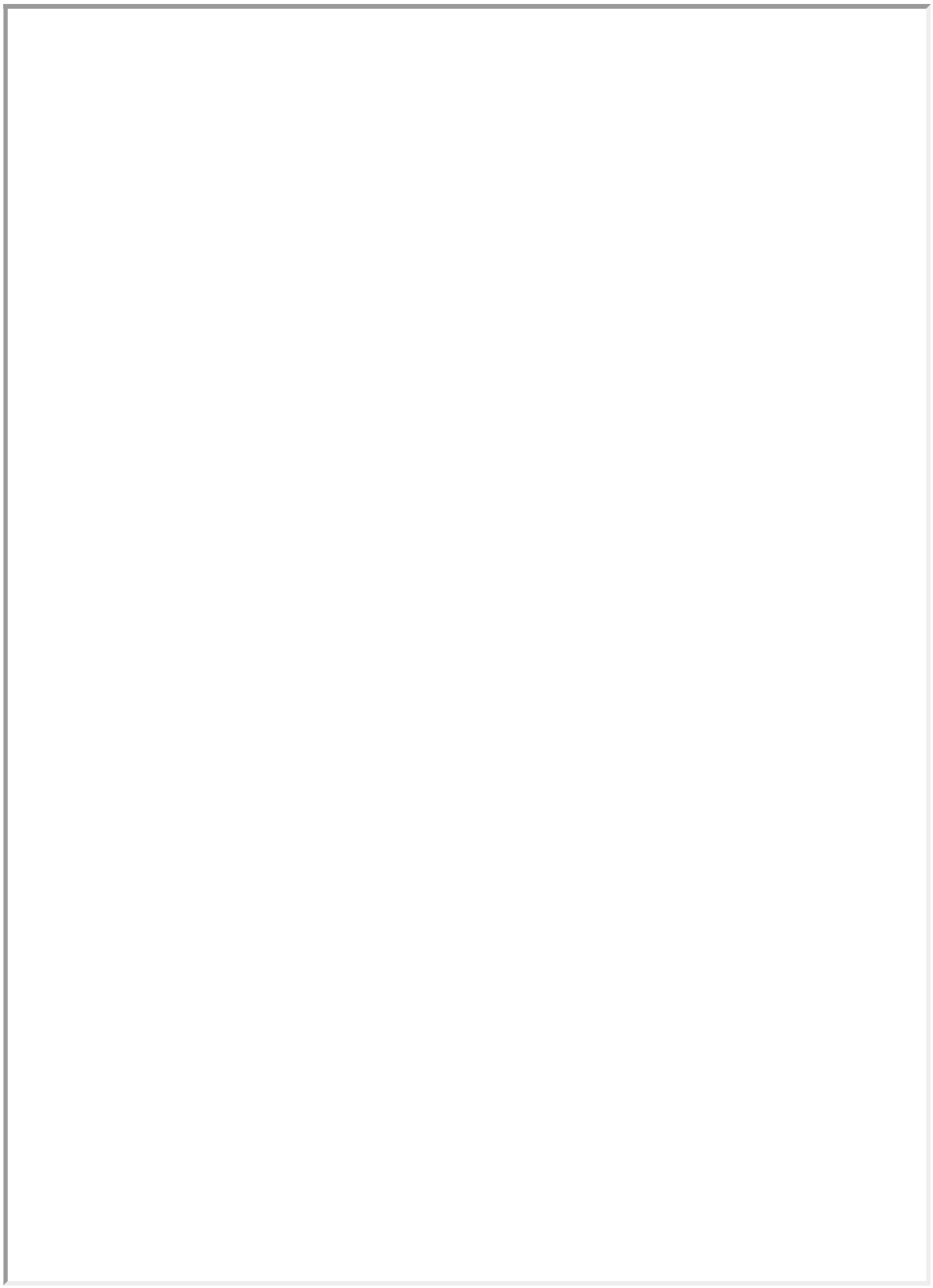
Note that the code inside the *if* and the *else* clause are indented (2 spaces away from the left), the indentation tells the computer which parts of the code are inside the 'if' and the 'else' statements. To add an *indentation* to your code, press the *Tab* key on your keyboard.

Essentially, the *if*-statement checks whether something is true or not. The dobuble-equals sign "==" means "is equal to". So the code above translates to (in English):

```
If *myName* is equal to "Ishan", show "Your name is Ishan". If *myName* is
not equal to "Ishan", show "Your name is not Ishan".
```

The code that is in the *true* part of the code, is on the lines below the *if*.
The code that is in the *false* part of the code, is on the lines below the *else*.

We can also create *if*-statements using numbers like in the Trinket below. Before running it, read the code and guess what it does.



Trinket Emulator

In the code above, we were able to compare numbers in the if statements using the <, > and == symbols. Here's
an explanation for what these symbols mean:

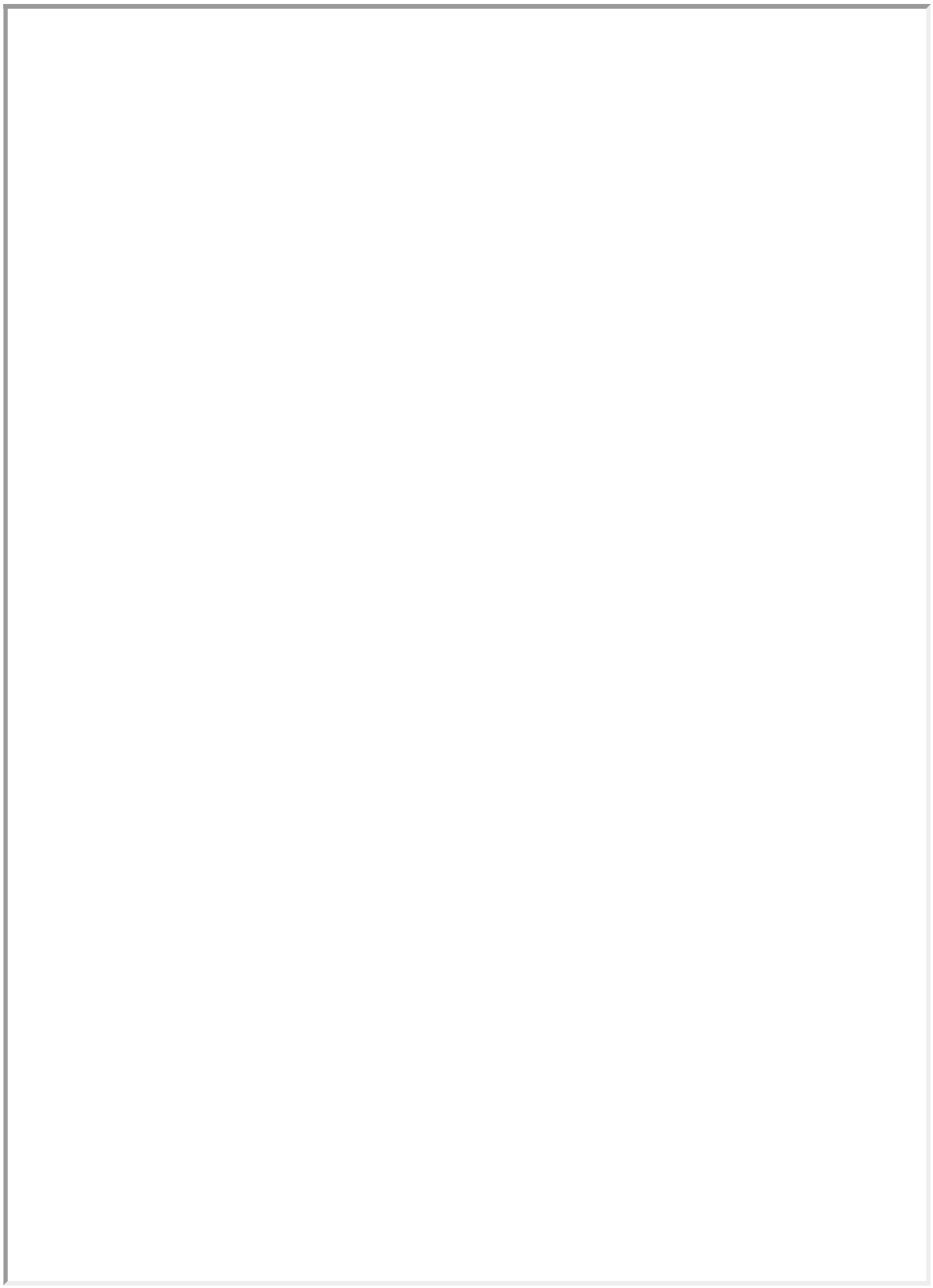
- **a > b** a is greater than b.
- **a < b** a is less than b.

- **a >= b** a is greater than or equal to b.
- **a <= b** a is less than or equal to b.
- **a == b** a is equal to b. We use double equals because single equals is used to assign values to variables.

This is all quite a lot of information already, so if you find it confusing, try to discuss it with a supervisor or the person sitting next to you. Don't worry if you don't understand everything yet!

Exercise 9: In the Trinket below write a program that does the following:

- Define a variable called *myNumber*, and assign it any number you like.
- Using if-statements, make the program say whether the number is positive, negative or equal to zero.



Trinket Emulator

10. Lists

In Python, we can not only store numbers and strings in variables, we can also store a *list* of items in a variable.

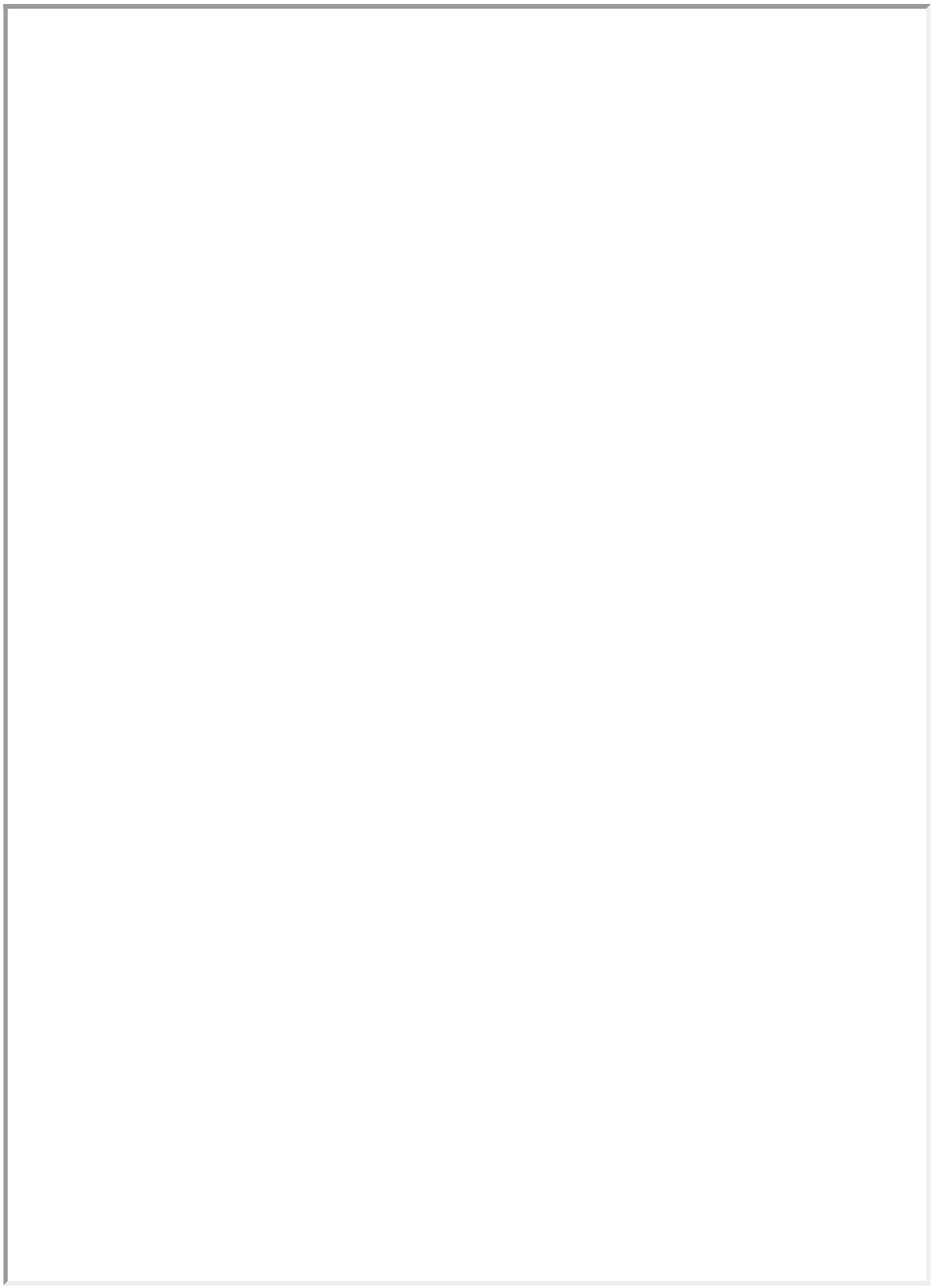
```
myList = [42, 51, 62, "Hello", 61, 123, "World"]
```

We already saw a list early on in the lecture. The RGB colors were stored in lists.

If we want to access individual elements in the list, we can do it like this:

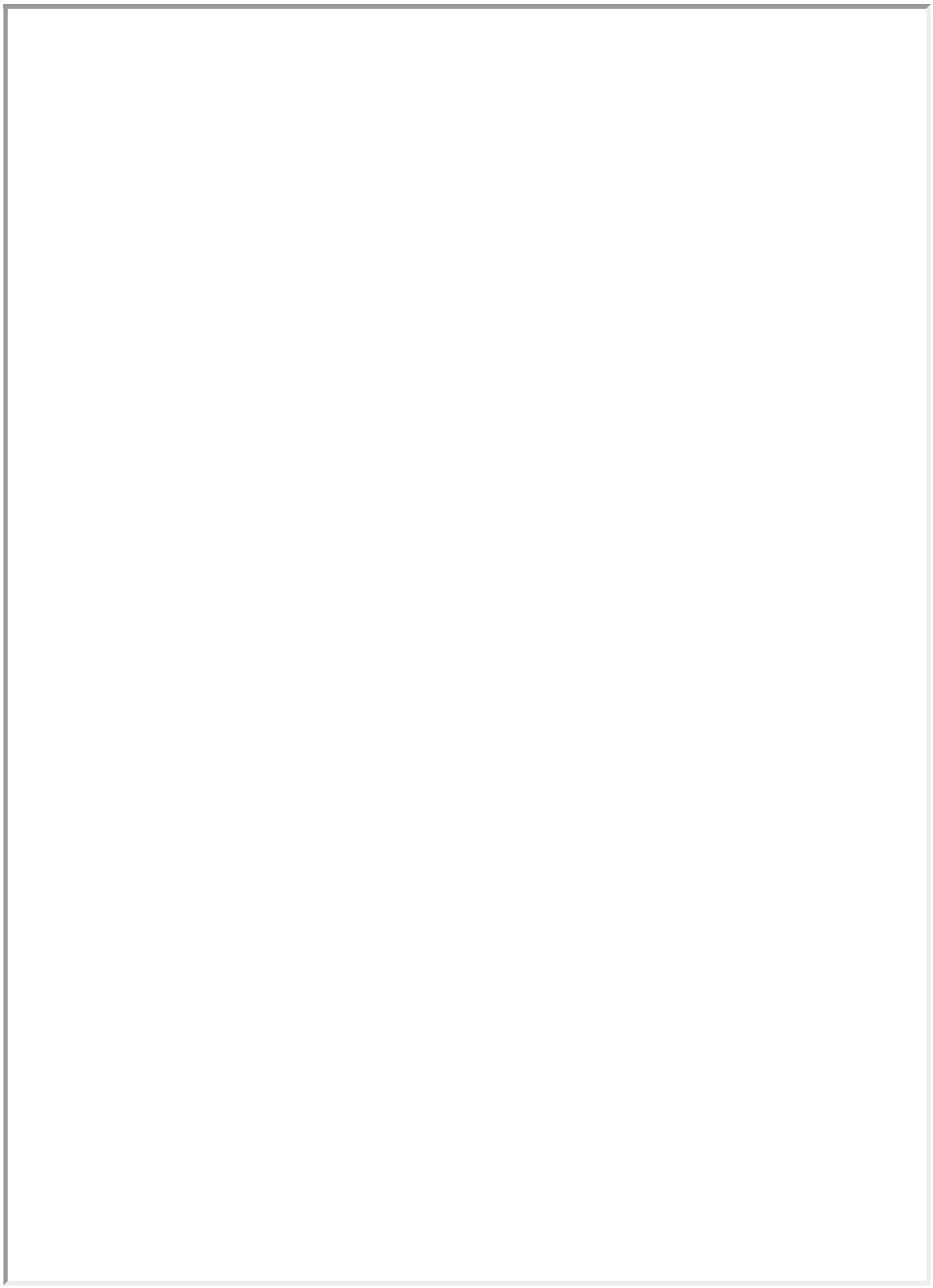
```
myList = [42, 51, 62, "Hello", 61, 123, "World"]

# The first element in the list
print(myList[0])

# The third element in the list
print(myList[2])

# The fifth element in the list
print(myList[4])
```

Copy this code and see what it does. Note that the *first* element is given by the number *0* in the list, and the second by *1* and so on...



Trinket Emulator

Exercise 10: In the Trinket below repeat Exercise 7, but instead of storing your name and age is separate variables, store them in a *single* list.



Trinket Emulator

11. For-loops

Sometimes you might want to repeat the same piece of code several times. Loops allow us to accomplish this in fewer lines of code, by *repeating* code until our operation is complete.

Look at the code below, what do you think it does? See if you can figure it out before you run it.

```
for n in range(1,10):
  print("The number is", n )
```

Now copy the code and run it. Did you guess correctly?

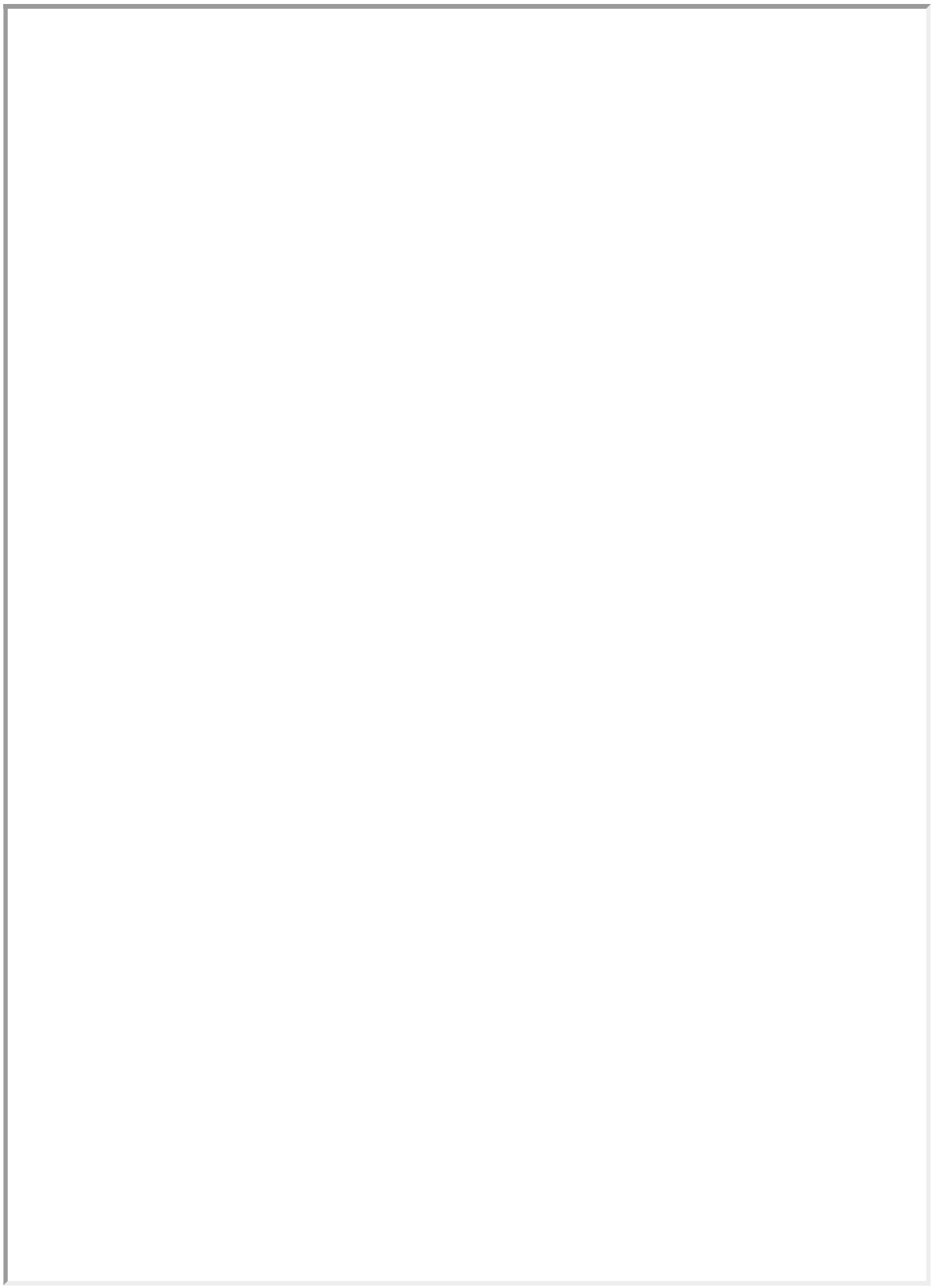
How about this code?

```
for n in range(1, 5):
  sense.set_pixel(n, 0, 0, 0, 255)
```

This one is more complicated. Try to imagine what this could do. After that, copy it and see for yourself.

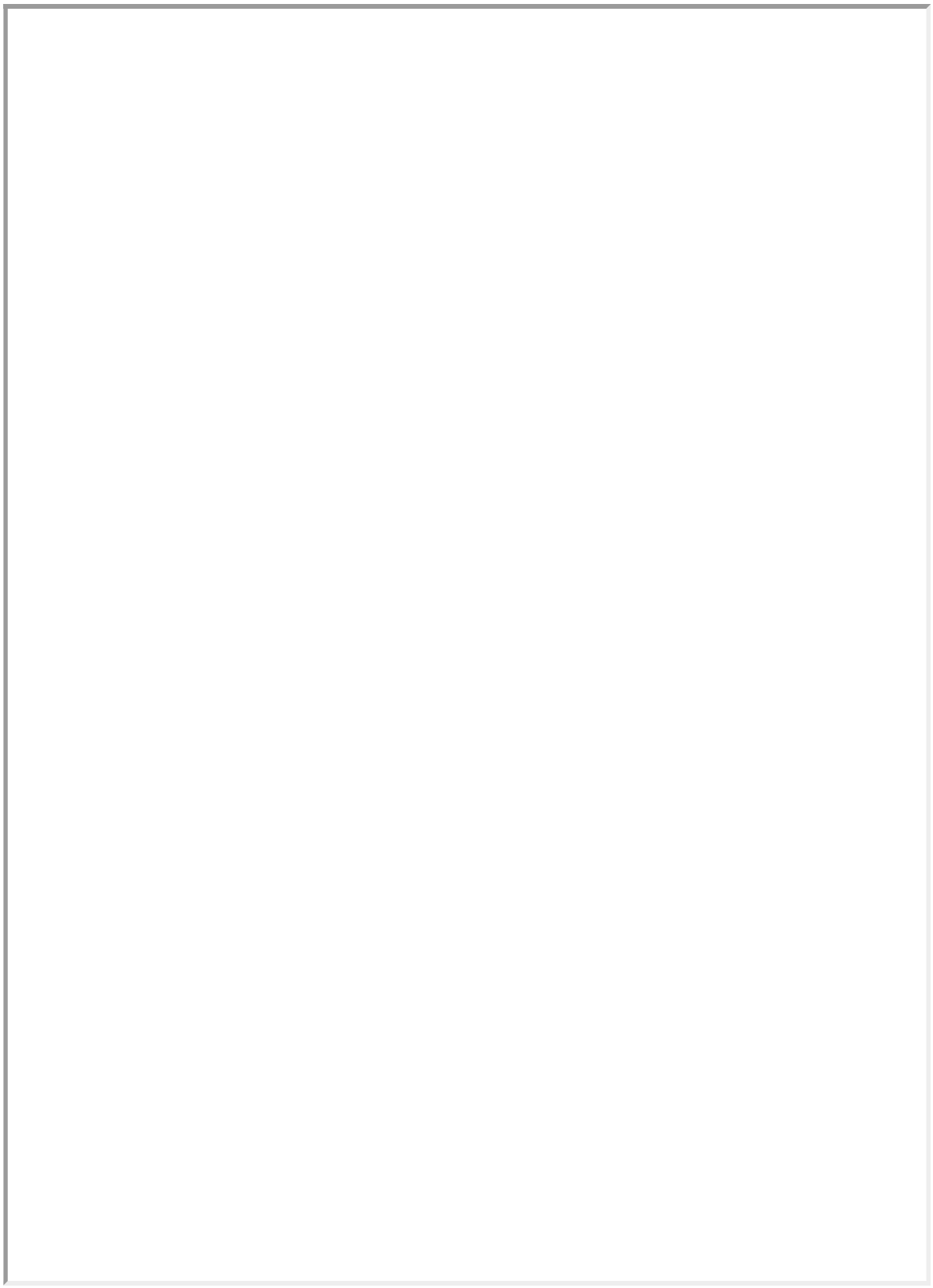
Play around with the code and see if you can make it do more complicated!

Exercise 10: Write a program that *prints* the numbers from 50 to 100.



Trinket Emulator

Exercise 11: Write a program that draws a line of yellow pixels from the top-left corner, to the bottom-right corner.



Trinket Emulator

12. While-loops

Another type of loop that we can use in Python is the *while loop*.

The while loop repeats its code as long as the expression after the *while* command evaluates to *true*. You can think of it as a mix between an *if-statement* and a *for-loop*.

The previous explanation might be a bit complicated to follow, but if you look at the code below you can see that it's quite intuitive.

```
n = 0 # Initialising n
while n<10: # Starting the loop with condition that n must be less than 10
    n = n+1 # redefines n as n+1
    print(n) # print the new value of n
```

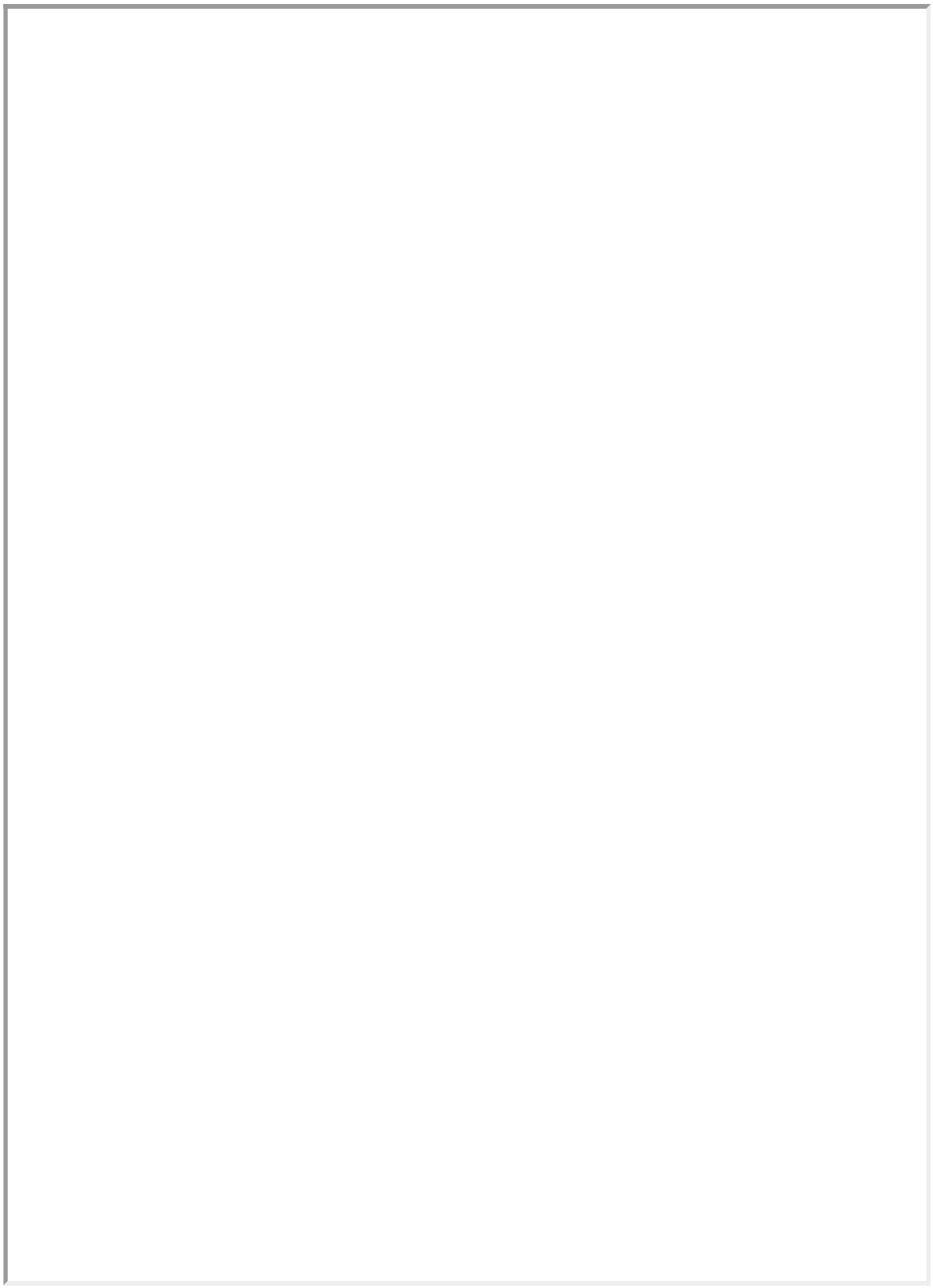
As usual, look at the code and try to figure out what it does. After that copy it and see for yourself!

What do you think will happen if you run the following code?

```
while True:
  print("Hello!")
```

Don't try to run it! What will happen is that the code will run *forever*, can you see why? It'll print out "Hello!" again and again and it wont stop! Although never-ending codes like this might seem useless to you, it does actually come in handy, and we'll often use them in our projects!

Exercise 12: Redo exercise 10, but with a *while*-loop, instead of a *for*-loop.



Trinket Emulator

13. What's next?

The next thing is to ask yourself if you've understood everything!
We covered quite a lot, and very quickly, so it might be worth to just have a quick read-through of everything again.

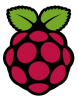
After that, it's time to start choosing a project. You should try to start off with one of the easier projects, before attempting any harder ones. Remember that you can always ask for help within your group, or by a supervisor, during the entirety of the summer school.

We discussed some useful functions in this lecture, like *sense.set_pixel* and *print*. There are many more functions you can use, and the best way to check them all out is by reading the *Function Reference* document. Use that document as a reference on how to use various Python functions.

Good luck!

Authors: Lukas Kikuchi & Ishan Khurana

Date: August 09, 2017

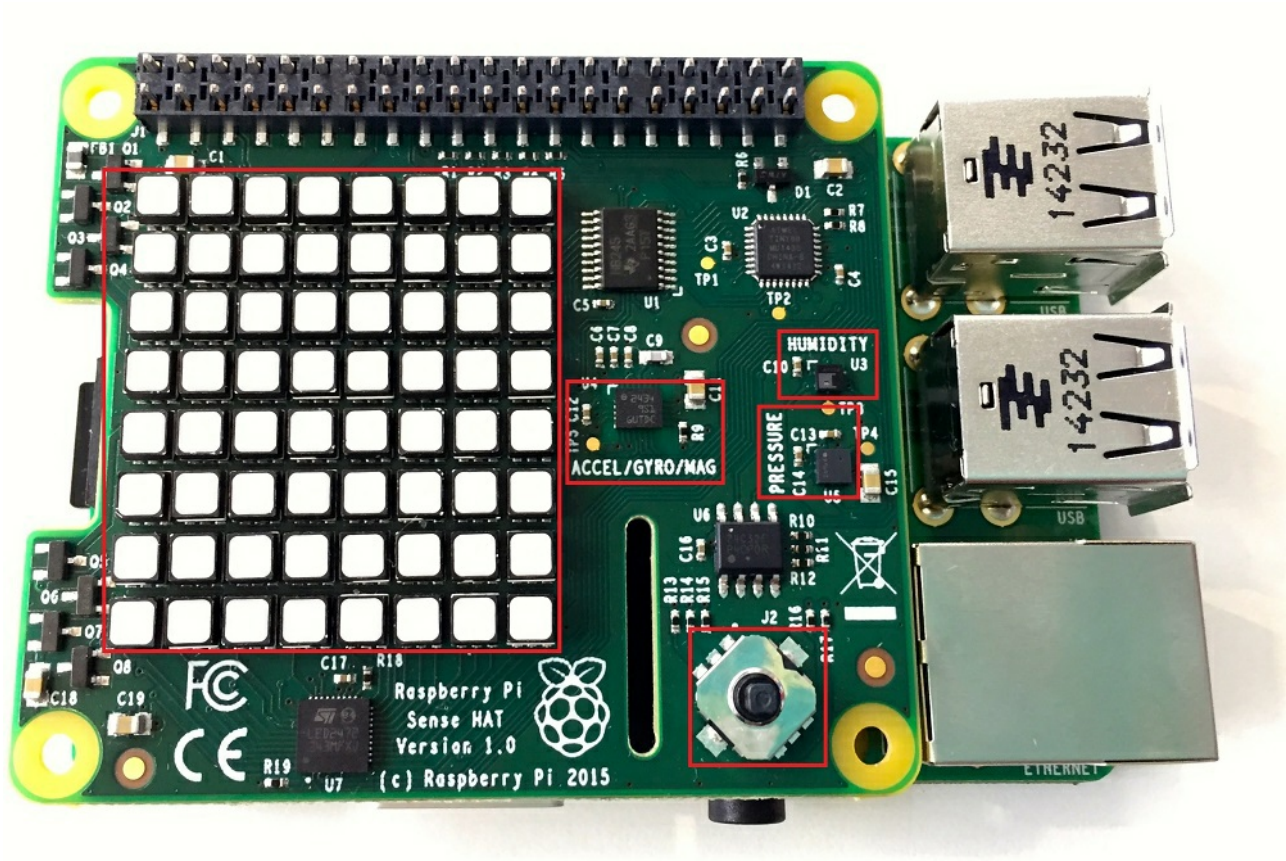


Raspberry Pi Sense HAT

Go4Code

Introduction

The Sense HAT is an add-on board for the Raspberry Pi. It gives us the ability to measure all kinds of things with the Raspberry Pi, make games and display things with the LED lights. Let's firstSe go through what is actually on the board!



The Sense HAT: The main features are outlined in red.

Features:

- The LED Matrix on the left has 64 (LEDs) arranged in an 8 x 8 grid. They can be used to display shapes, icons and messages. The screen is one of the ways the Raspberry Pi can communicate with us, which is very important for making games!
- The small microchip labelled **ACCEL/GYRO/MAG** is what is known as the inertial measurement unit (IMU). This actually has three sensors!
    - The first sensor is the **accelerometer**. This measures how acceleration and will be useful in detecting how fast the Raspberry Pi has been moved.
    - The second sensor is a **gyroscope**. This sensor basically measures if the Raspberry Pi has been rotated. We can measure rotation in three directions, but more on this later.
    - The third sensor is a **magnetometer**. This measures the magnetic field and can be used to detect the magnetic field caused by the rotation of the Earth.
    - Next is a **humidity sensor**. This will allow you to measure air moisture. It's the small chip just below the text HUMIDITY. You can also use it to measure ambient temperature.
- There is also a **pressure sensor** for measuring air pressure, something which is certainly important in space. It's the chip to the right of the text PRESSURE.

- Last but not least is the **joystick**. The Raspberry Pi cannot be connected to a USB keyboard or mouse in space, so the Sense HAT has its own five button joystick. This is the silver rectangle in the bottom right corner with the small stick poking out of the top. It can move up, down, left, right, and allow middle-clicks.

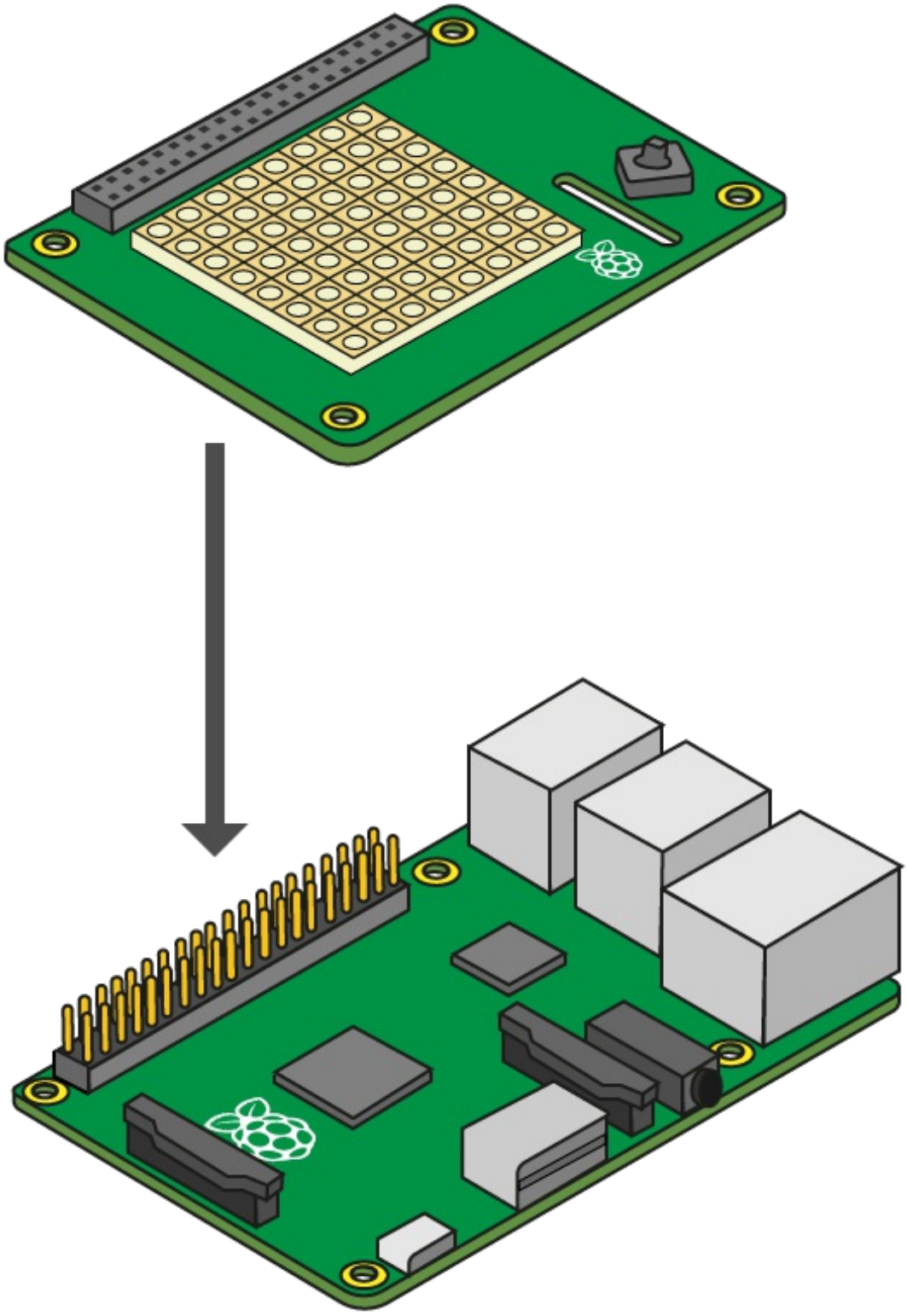
These simple features allow us to do countless things with the Sense HAT, we have come up with a few projects that you will do. To let you really express your imagination and creativity, you get to keep the Sense HAT and Raspberry Pi so you can come up with your own gadgets!

The Assembly

This section is optional. You may skip to the next section

The Sense HAT is just one type of HAT (Hardware Attached on Top) that you can get for your Raspberry Pi. Some more examples of HATs are listed here (https://shop.pimoroni.com/collections/hats).

The Sense HAT you have for the summer school has been preassembled by the Go4Code team. But if you wanted to take it apart and use a different HAT, it would be good to know how it is put together.



The Sense HAT Assembly

- The Sense HAT connector is first attached into the GPIO pinouts on the Raspberry Pi (the golden pins in the image above).
- Then four hexagonal stands are placed between the Sense HAT and the Raspberry Pi on

circular holes.
- These stands are then screwed on to securely attach the Sense HAT to the Raspberry Pi

Programming the Sense HAT

We will be using the *Python* programming language to write code that will run on the Raspberry Pi and control the Sense HAT. This guide will take you through the basics of Python programming.

The end goal is to get you ready to start doing some programming of your own, in the various projects we have prepared for you.

We start right from the beginning, so you don't need to have done programming before. If you have, then use this guide as a helpful reminder.

 The Trinket Emulator

In your projects, you'll have your own Sense HAT to play around with. But since you'll have to test around and tinker with your projects for quite some time before its ready, we're going to be testing all our code on a *virtual* Sense HAT on a website called *Trinket*.

In your project scripts you will get a trinket page to test and develop your code before you transfer it on the the Raspberry Pi.

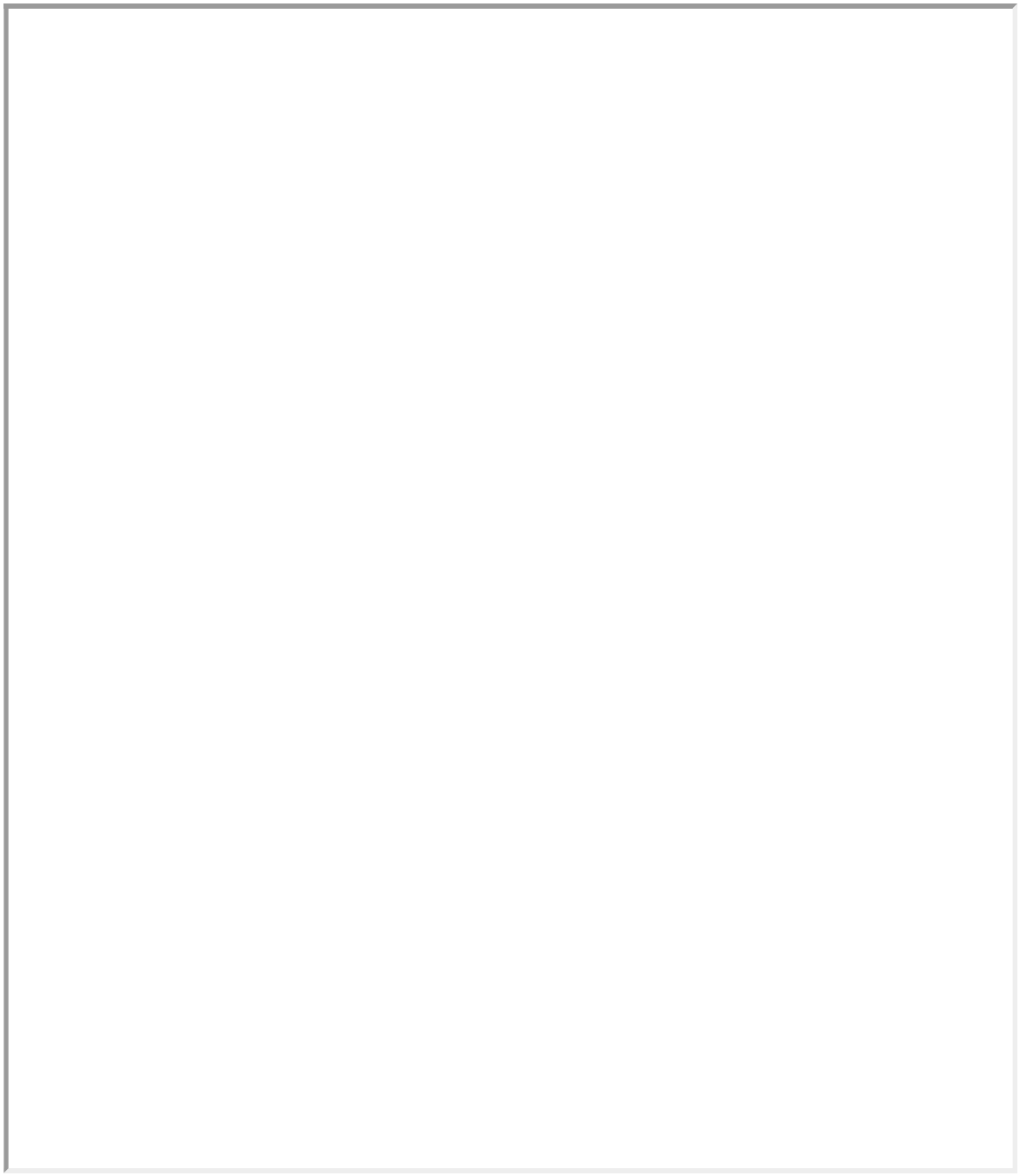
We have embedded the Trinket Emulator into this document so you can learn to code here before you start developing your own code.

Trinket Emulator

The left side of the page has some Python code already. The right side of the page shows nothing in the beginning, but if you press the Run button ▶ in the top left corner, a virtual Sense HAT should appear on the screen. The code in the there right now just starts the Sense HAT, but doesn't actually tell it to do anything. Let's create out first program!

1. Showing Messages

In this section we will go over showing messages on the Sense HAT. You should use the Emulator below to test out the code.



Trinket Emulator

Copy the following into the Python code pane in the Emulator above:

```
sense.show_message("Hello world!", 0.05)
```

After you have done that, click Run again. The message, "Hello World!" should scroll by on the screen. You've written your first Python program!
You're now *officially* a Python programmer.

The red text you see in the Python code is what we call a *string*. A string is basically a piece of text. In Python, we can write strings by either enclosing text in double quotation marks like this

```
"This is a string!"
```

or single quotation marks, like this

```
'This is a string!'
```

The *show_message()* function (don't worry if you don't know what a function is yet, we'll go over this later) displays whatever *string* you put inside the bracket, on the Sense HAT.

Now, we're going to play around with this program that you've made.

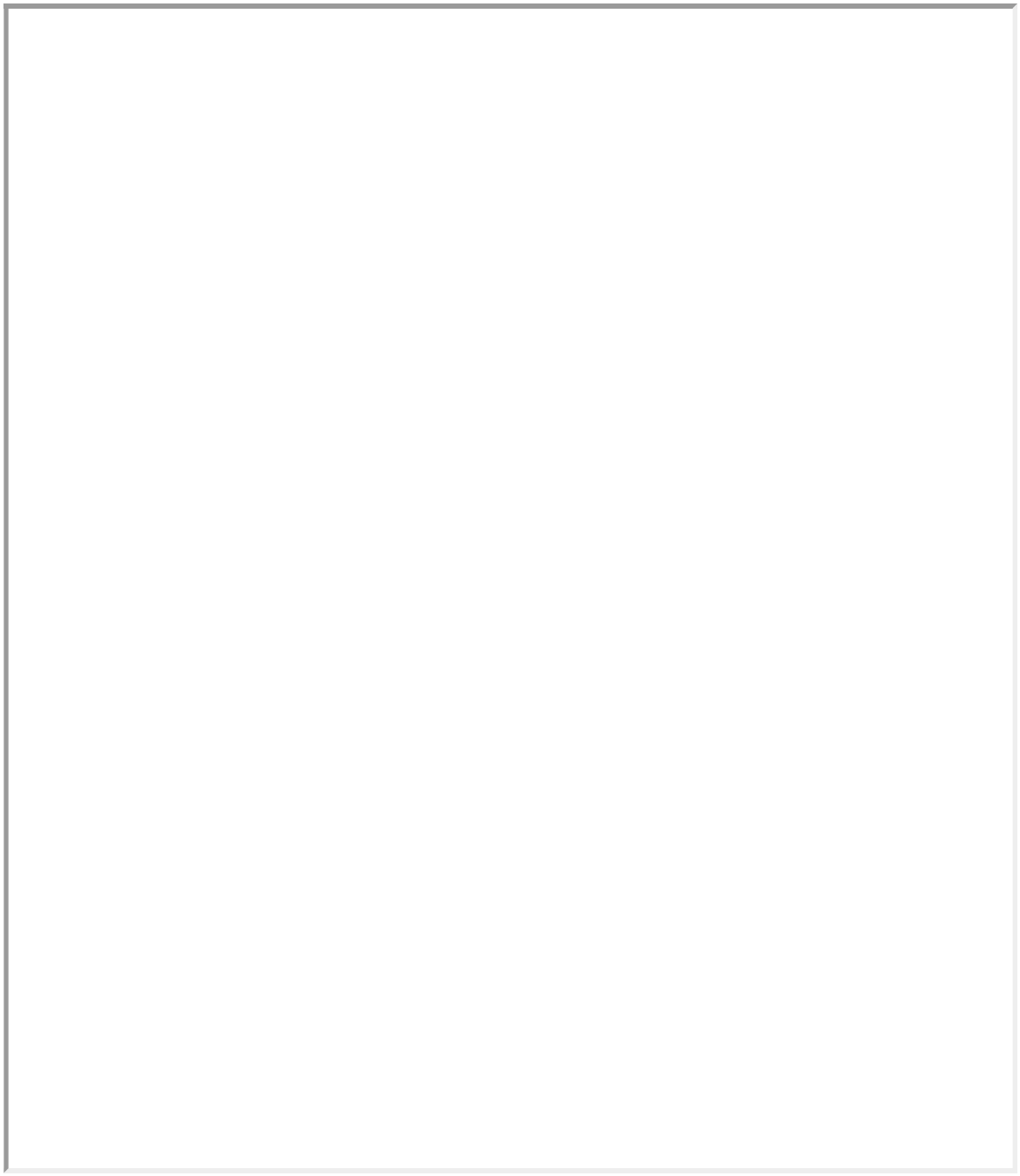
Exercise 1: Change the message to something else. Try doing it using single-quotation marks as well.

Exercise 2: You might have noticed that there's a number after the string.
Try changing the value of that number. Find out what that number does.

2. Colours in Python: RGB -values

In this section we will be changing the colour of the light emitted by the LEDs.

Run the code in the emulator below:



Trinket Emulator

Run it again, and see what happens. The text should now be red. The last part of the line that you just wrote `text_colour=[255, 0, 0]` decides which color the text should be in. You can change these three numbers to change the color of the text. The numbers in the square brackets specify the colour of the light emitted from the pixel.

We call triplets of numbers that specify colour RGB-values, and this is the reason:

- The first number decides how *red* the light should be,
- the second number decides how *green* the light should be,
- and the third number decides how *blue* the light should be.

The pixels on the Sense HAT are made up of tiny LEDs that emit red, green and blue light. You may remember from the talk that light is made up of different colours and that we can combine light of different colours to make new colours.

The maximum redness/greenness/blueness in RGB format 255, and the lowest is 0. As you saw above, the RGB-value for red is [255, 0, 0].
And, for example, the RGB-value for black is [0, 0, 0]. Note: the pixels won't actually show black as there is no such thing as black light. Black is just the lack of light. By entering an RGB value of [0,0,0] we just turn the LEDs off.

We can also set the back colour using the code below. Replace the show_message() function in emulator with the one below:

```
sense.show_message("Hello world!", 0.05, text_colour=[255, 0, 0], back_colour =
[0,255,255])
```

Use what you have just learnt to complete the exercise below. Your code should go in the Trinket emulator above. If you don't understand something, feel free to ask your supervisor.

Exercise 3: Make the text flash by in the following colors:

- Red
- Blue
- Green
- White
- Purple
- Yellow
- Orange

You'll have to experiment with the RGB-values to try to find the right color-combinations!

Commenting

Before we get going with some more interesting code, let's remove the *sense.show_message* line so that it won't disturb us later on. But instead of just erasing the line completely, we can *comment it out*.

To do this, simply add a hashtag # at the start of the line, like this:

```
# sense.show_message("Hello world!", 0.05, text_colour=[255, 0, 0])
```

Try running the code after you've commented the code out. No text should appear on the screen. In Python we often use comment symbol # to add explanations to our code, for example:

```
# This line displays some text on the screen
sense.show_message("Hello world!", 0.05, text_colour=[255, 0, 0])
```

3. Functions

Before we move on, I'll explain (as promised) what a *function* is. A function is simply something that we use in a code to do something for us. For instance we used the *show_message* function earlier to display text on the Sense Hat's LED matrix.

Let's look a bit closer at the *show_message* function.

```
sense.show_message("Some String", 0.05, text_colour = [255,0,0],back_colour =
[0,255,255])
```

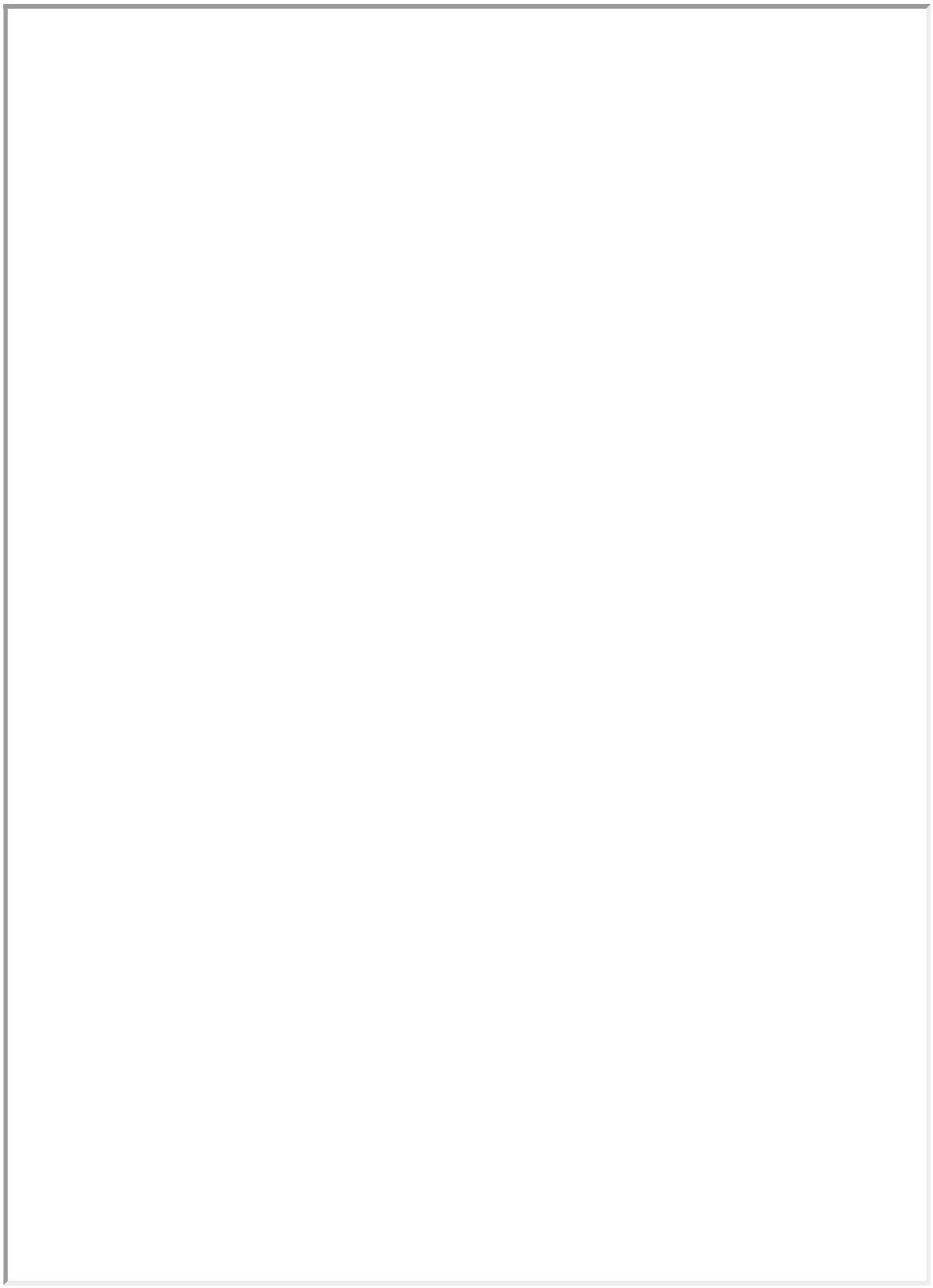
The name of the function is *show_message*. When you use a function, it's often referred to as *calling* the function. We are calling the function on our Sense HAT (we have named the Sense HAT *sense*) by writing *sense.show_message*.

What about all those numbers inside the brackets? Those are called *arguments*. Sometimes when we call a function, we want to specify exactly what we want it to do, we do this using *arguments*.

In this case, the arguments are: the message that will be displayed on the screen, the scroll speed of the message, the text colour and the back colour.

Don't worry if all of this sounds a bit complicated. It's a lot of names for something that's quite simple. If you have any questions about this, discuss it with a supervisor.

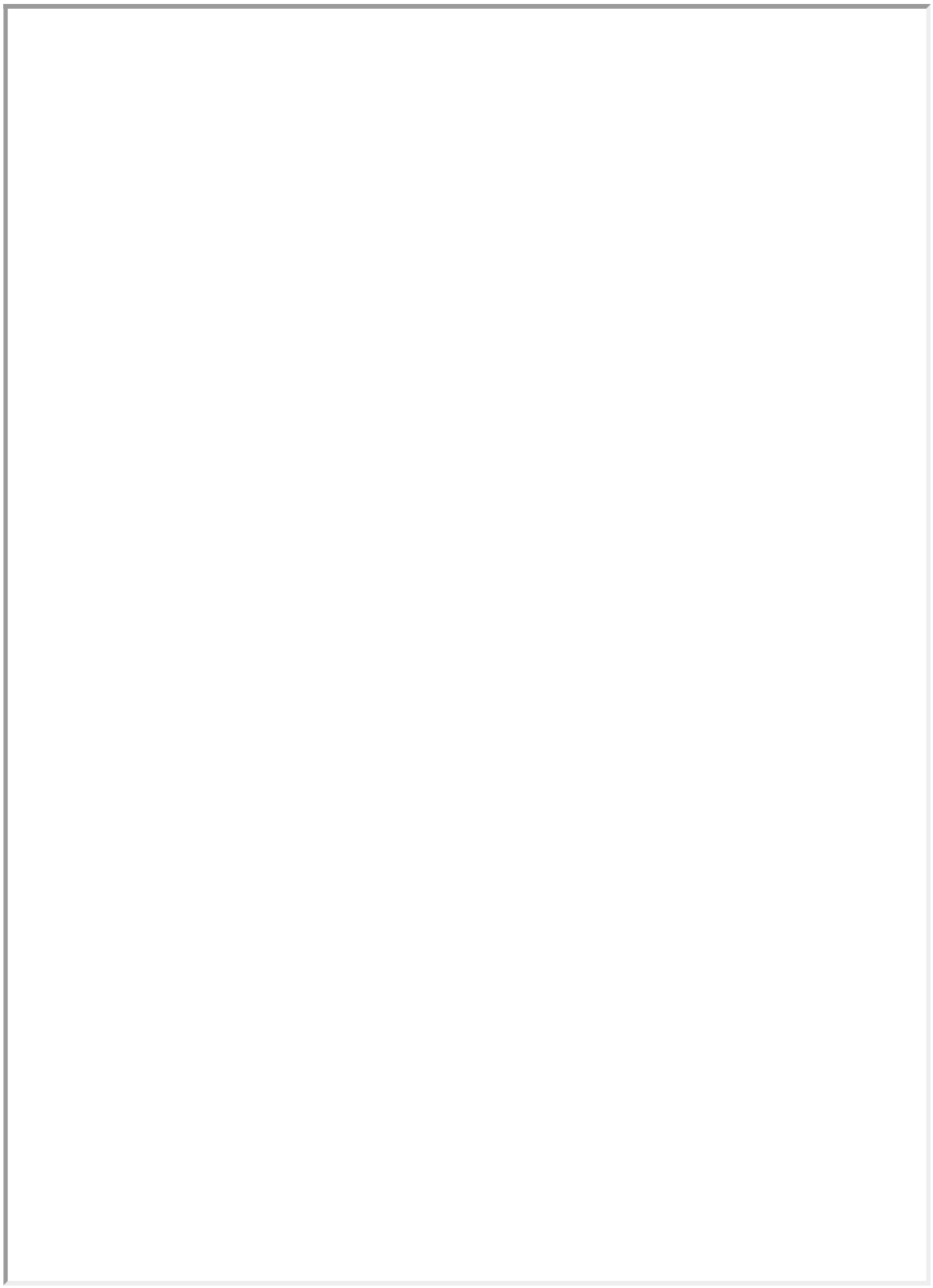
Before we move on, we'll introduce the *print* function. The print function is very simple, it just displays some text. The important difference here is that it doesn't print out text on the screen, but in the *terminal* underneath the Sense HAT screen on Trinket. It is easiest for you to see for yourself:



Trinket Emulator

You should see some text appear in the lower-right corner of the page. Try changing it to make it print something else.

4. Using Python as a calculator



Trinket Emulator

Next thing we'll do is try out some simple mathematical operations in Python.
All the main maths signs you know from school exist in Python. Using Python we can add, subtract, multiply or divide numbers. Copy the following code into the emulator above to try it for yourself:

```
print(3+5)
print(2-7)
print(2*5)
print(4/2)
```

Run the code and see what the results are. You can also add really large numbers

```
print(1234567890000 + 987654321111)
```

Or do several mathematical operations in a row

```
print(23*52*13 + 52*611 - 412 )
```

We can also use brackets "*()*" when calculating things

```
print((2+3)*5)
print(6 + ((100 - 1)*42)/32)
print(((2 * (23 - 3) / (12 * 23)) - 32)*62)
```

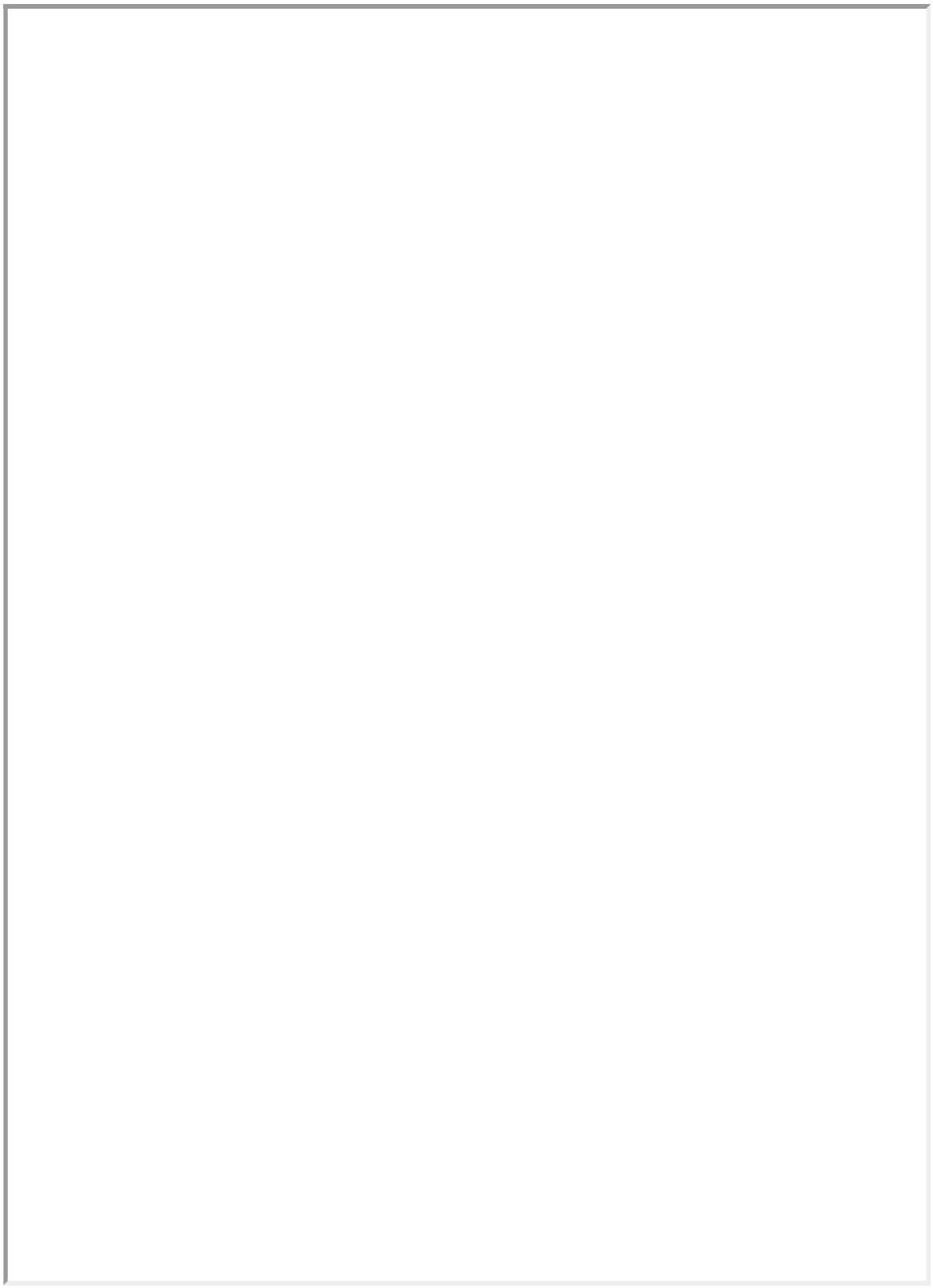
Try changing the numbers, and try to do different calculations.

Exercise 4: In *Trinket*, translate the following into Python code:

```
_(four times ( six minus two ) plus five) divided by two_
```

and use *print* to show the result.

5. Variables



Trinket Emulator

Python allows us to store information in *variables*. We can use the assignment operator (the *equal-to* sign '=') to assign a value to a variable.

There are three components to variable assignment.

- First we need to decide on the name of our variable, like my_variable.

- Second, we set the variable name equal to something using the *equal-to* sign '='.
- Third, we write the value that the variable should be equal to.

Like this:

```
my_variable = 123
```

We can also store strings in a variable:

```
my_variable = "Hello!"
```

Note that the variable name must be written as *one single set of characters* without spaces. So, for example, *my variable* would not be a valid name for a variable.

The code below shows how easy it is to define and display a variable.

```
myNumber = 5*7
someText = "Five times seven is "
myMessage = someText + str(myNumber)

sense.show_message(myMessage, 0.05)
```

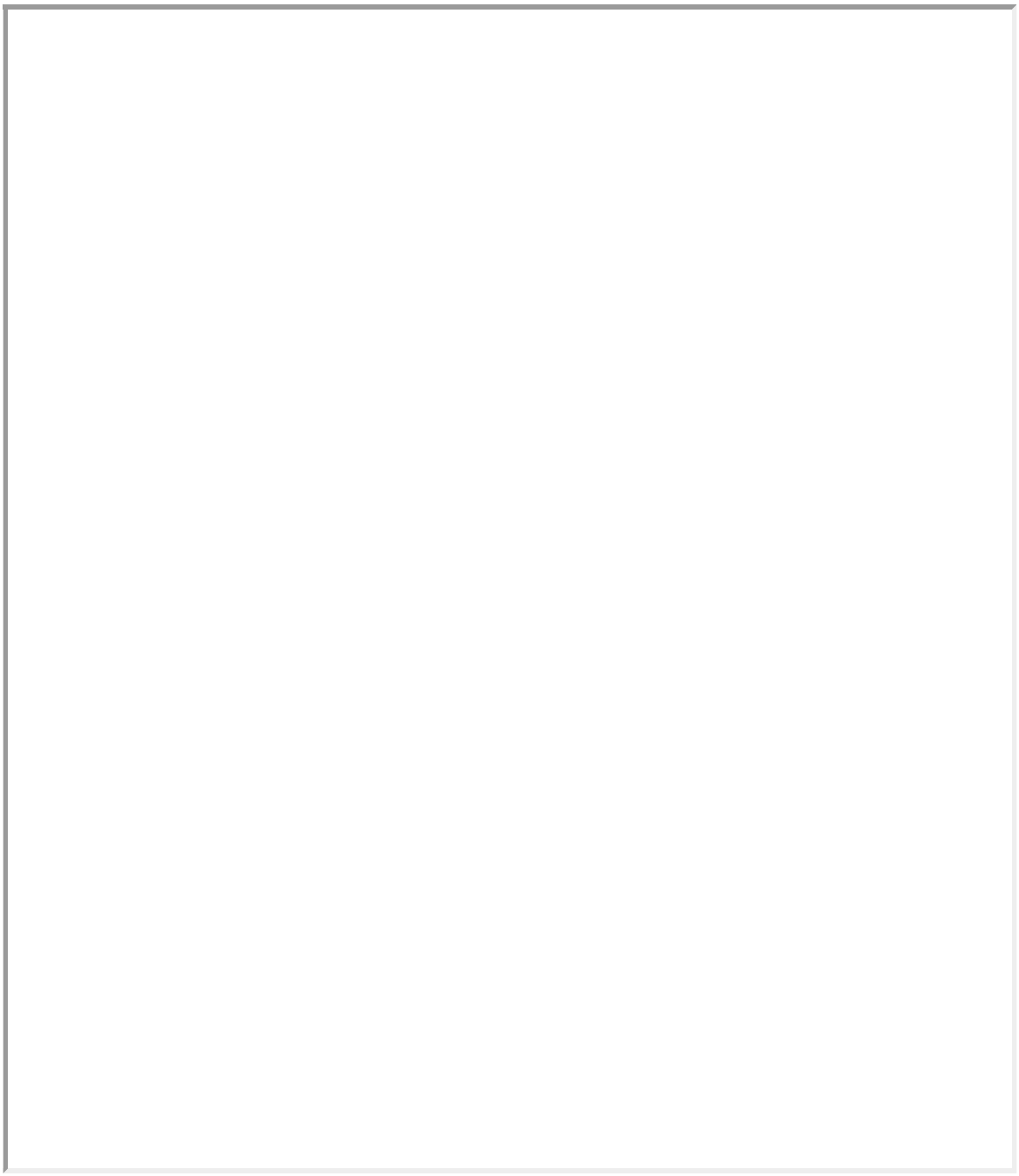
Can you look at this code and figure it out what it does without actually running it?
After you have guessed, write the code in the Trinket Emulator above and run it.

Explanation: We have used the *str* function to convert the number 5*7 to a *string* "35". Then we added the string "Five times seven is " with "35" to form "Five times seven is 35".

Exercise 5: Write a program that displays your full name and age on the screen. Store your full name in a variable called *myName*, and store your age in a variable called *myAge*. Remember that you have to convert numbers to strings using the *str* function.

6. A basic program

Variables are quite useful, as they allow us to remember numbers (or other types of data, like *strings*) in terms of more memorable variable names. We can use these variables in our programs to compute things. In the Trinket below is an example of a basic program that calculates how many minutes there are in a week.



Trinket Emulator

You don't have to copy this code, you can go on the following link to see the code in action: https://goo.gl/CZH1UX (https://goo.gl/CZH1UX).

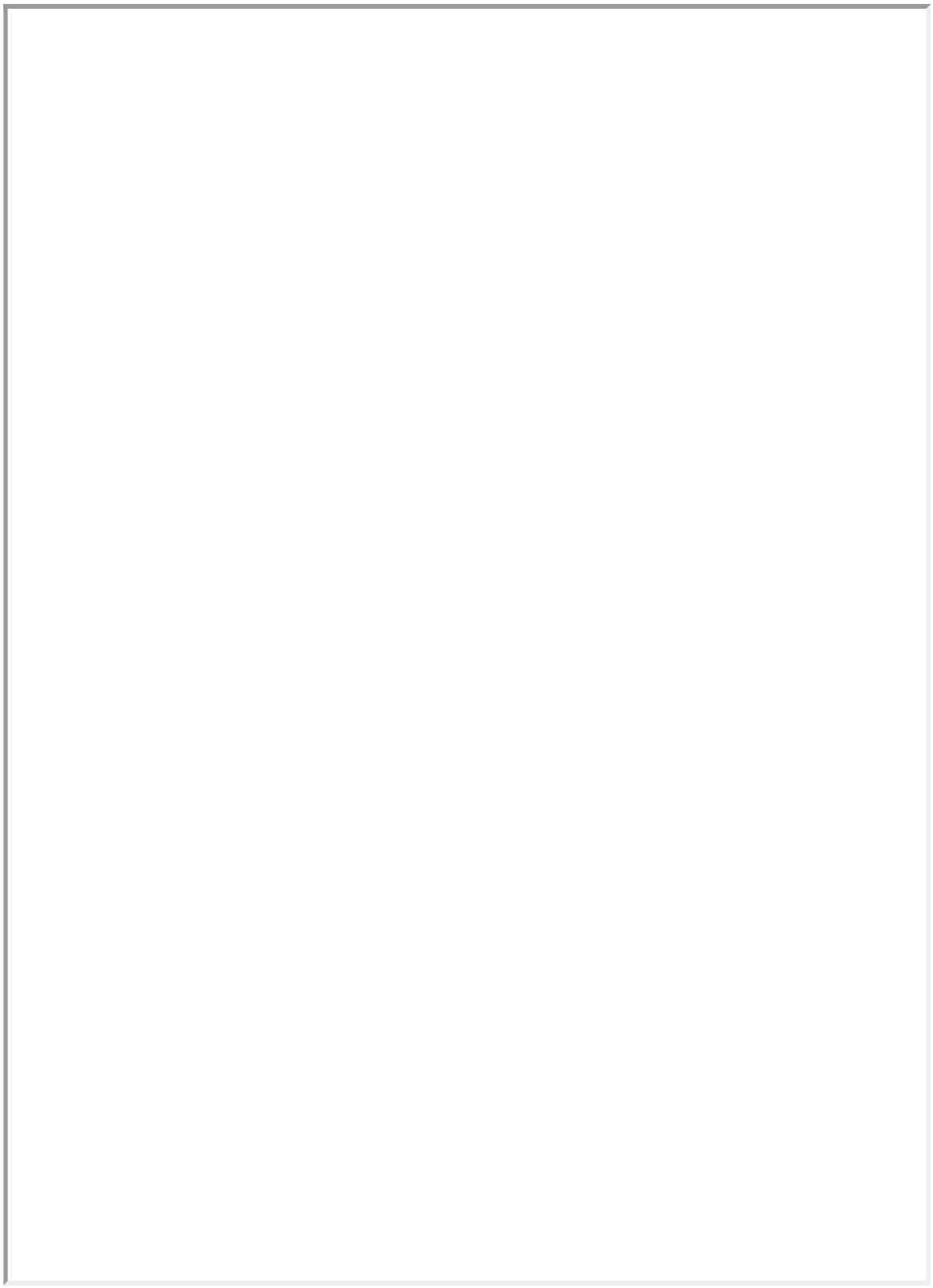
But before you run it, try to just read the code and figure out what it does yourself. It's good practice and will help you become a better programmer!

Exercise 6: Alter the code in the Trinket above to show the number of minutes in a week on the Sense Hat's display. Hint: the *show_message* function displays messages given as *strings*.

7. Drawing stuff on the Sense HAT: *sense.set_pixel*

It's really easy to draw things on the Sense HAT screen. We can do this using the function *sense.set_pixel* Try adding the following lines to your code in the emulator.

```
sense.clear()
sense.set_pixel(2, 2, 255, 255, 255)
sense.set_pixel(4, 2, 255, 255, 255)
sense.set_pixel(3, 3, 255, 255, 255)
sense.set_pixel(2, 4, 255, 255, 255)
sense.set_pixel(4, 4, 255, 255, 255)
```



Trinket Emulator

It should draw a white cross on the screen. The first line, *sense.clear*, resets all the pixels on the screen to black. This is just to avoid any clutter on the screen if you make mistakes.

Let's see how the *sense.set_pixel* function works.

You'll recognize the last three numbers of each line, those set the colors, as we learned previously. The first two numbers decide *where* we'll draw the pixel.

The first number is what column on the screen the pixel should be drawn at. We usually call this the *x-coordinate*.

The second number is what row on the screen the pixel should be drawn at. We usually call this the *y-coordinate*.

Exercise 7: Play around with the code, and try do draw something of your own. Maybe the first letter of your name?

Exercise 8: Try placing the sense.clear() line at the end of the code. What happens? Can you explain why? After that, place the line in the middle of the code. Again, what happens, and can you explain why that happens?

8. If-and-else statements

This next part is where things get more interesting (and a bit more complicated)!

The *if*-statement allows us to make a program that reacts in different ways, under different conditions. Think of it as a way to ask a computer questions.

Look at the code below, can you guess what it does?

```
myName = "Lukas"

if myName == "Ishan":
  sense.show_message("Your name is Ishan", 0.03)
else:
  sense.show_message("Your name is not Ishan", 0.03)
```

Copy the code into your project. How can you change the variable *myName* so that the result of the program is different?

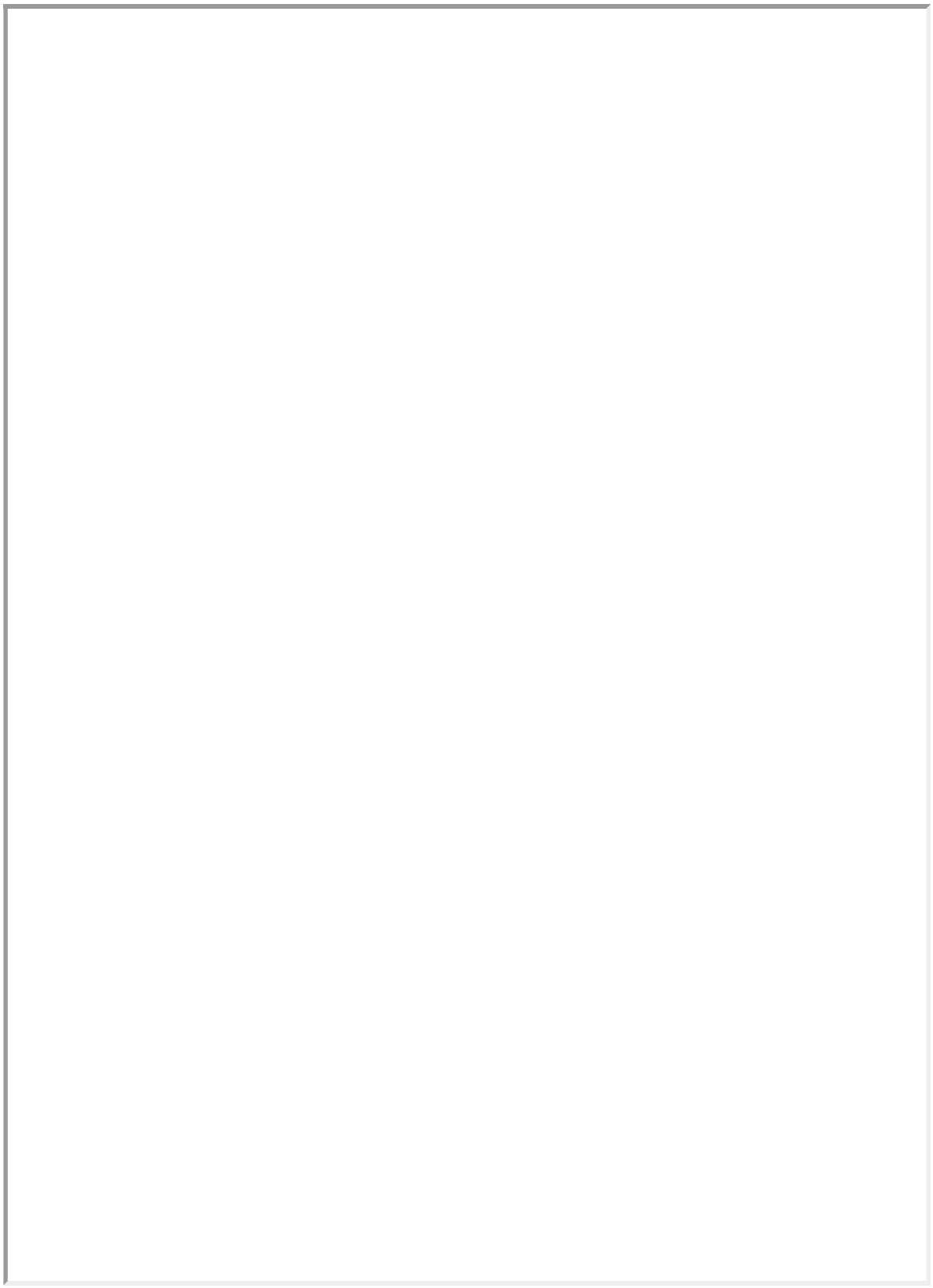
Note that the code inside the *if* and the *else* clause are indented (2 spaces away from the left), the indentation tells the computer which parts of the code are inside the 'if' and the 'else' statements. To add an *indentation* to your code, press the *Tab* key on your keyboard.

Essentially, the *if*-statement checks whether something is true or not. The dobuble-equals sign "==" means "is equal to". So the code above translates to (in English):

```
If *myName* is equal to "Ishan", show "Your name is Ishan". If *myName* is
not equal to "Ishan", show "Your name is not Ishan".
```

The code that is in the *true* part of the code, is on the lines below the *if*.
The code that is in the *false* part of the code, is on the lines below the *else*.

We can also create *if*-statements using numbers like in the Trinket below. Before running it, read the code and guess what it does.



Trinket Emulator

In the code above, we were able to compare numbers in the if statements using the <, > and == symbols. Here's
an explanation for what these symbols mean:

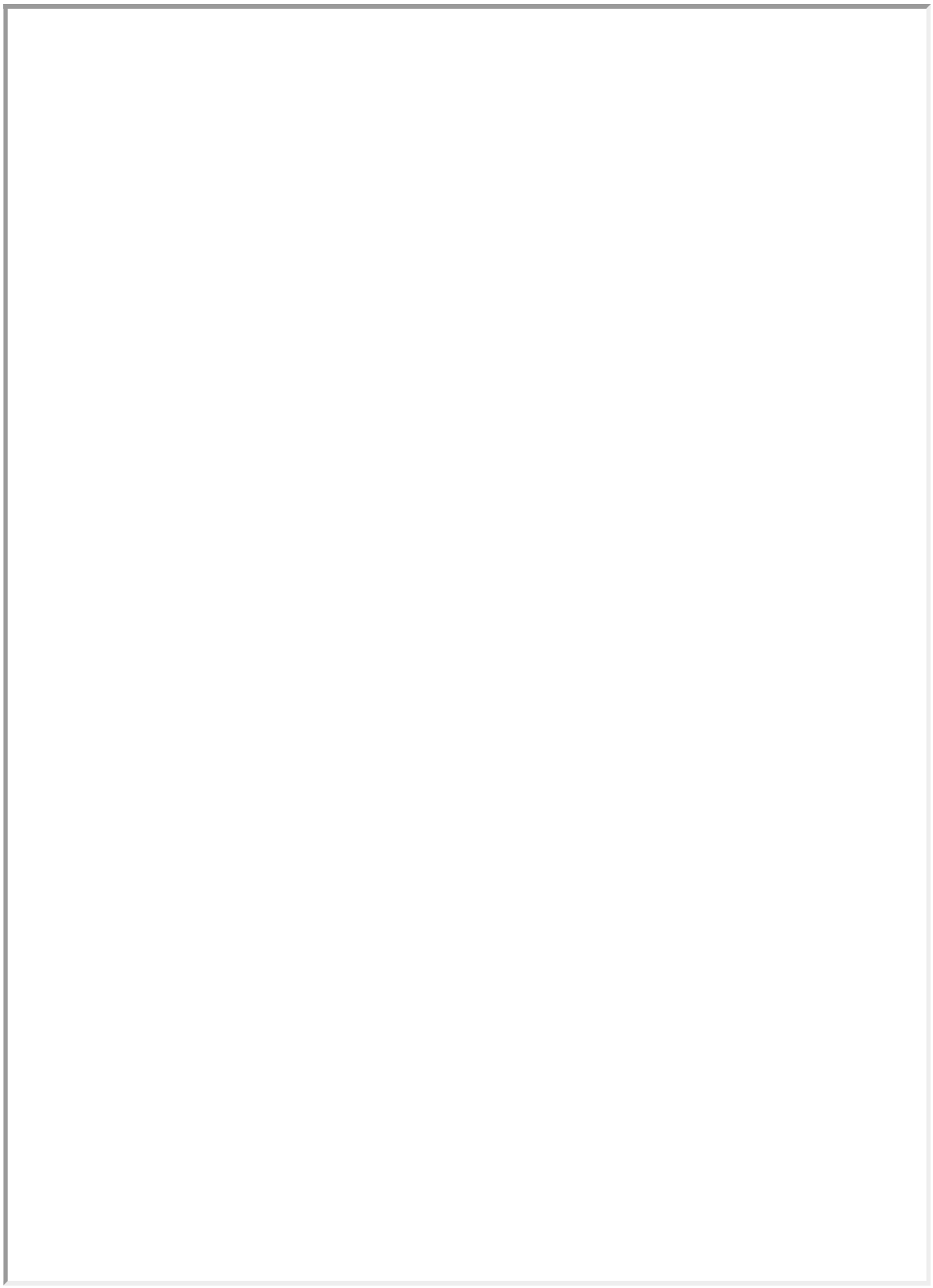
- **a > b** a is greater than b.
- **a < b** a is less than b.

- **a >= b** a is greater than or equal to b.
- **a <= b** a is less than or equal to b.
- **a == b** a is equal to b. We use double equals because single equals is used to assign values to variables.

This is all quite a lot of information already, so if you find it confusing, try to discuss it with a supervisor or the person sitting next to you. Don't worry if you don't understand everything yet!

Exercise 9: In the Trinket below write a program that does the following:

- Define a variable called *myNumber*, and assign it any number you like.
- Using if-statements, make the program say whether the number is positive, negative or equal to zero.



Trinket Emulator

10. Lists

In Python, we can not only store numbers and strings in variables, we can also store a *list* of items in a variable.

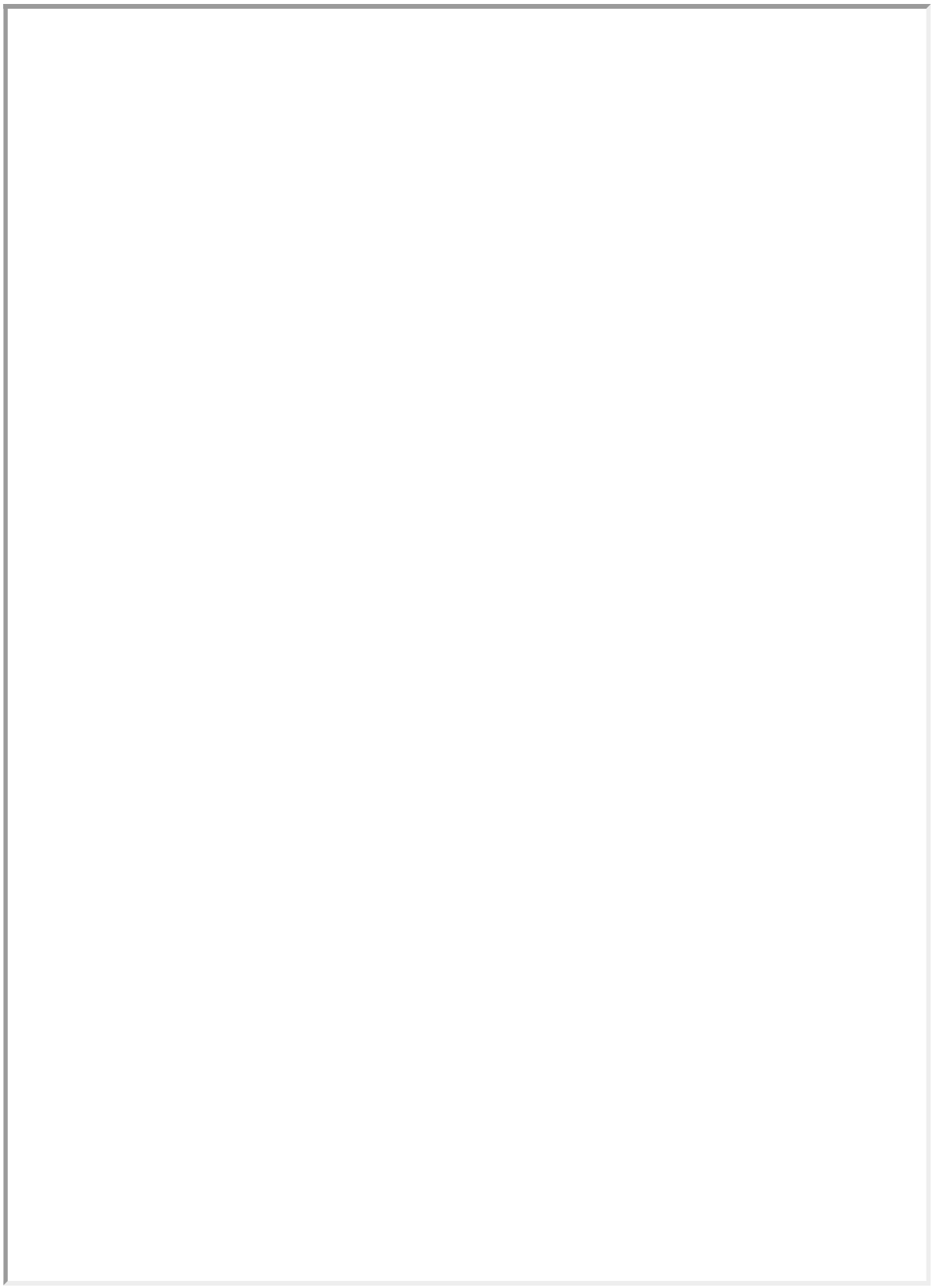
```
myList = [42, 51, 62, "Hello", 61, 123, "World"]
```

We already saw a list early on in the lecture. The RGB colors were stored in lists.

If we want to access individual elements in the list, we can do it like this:

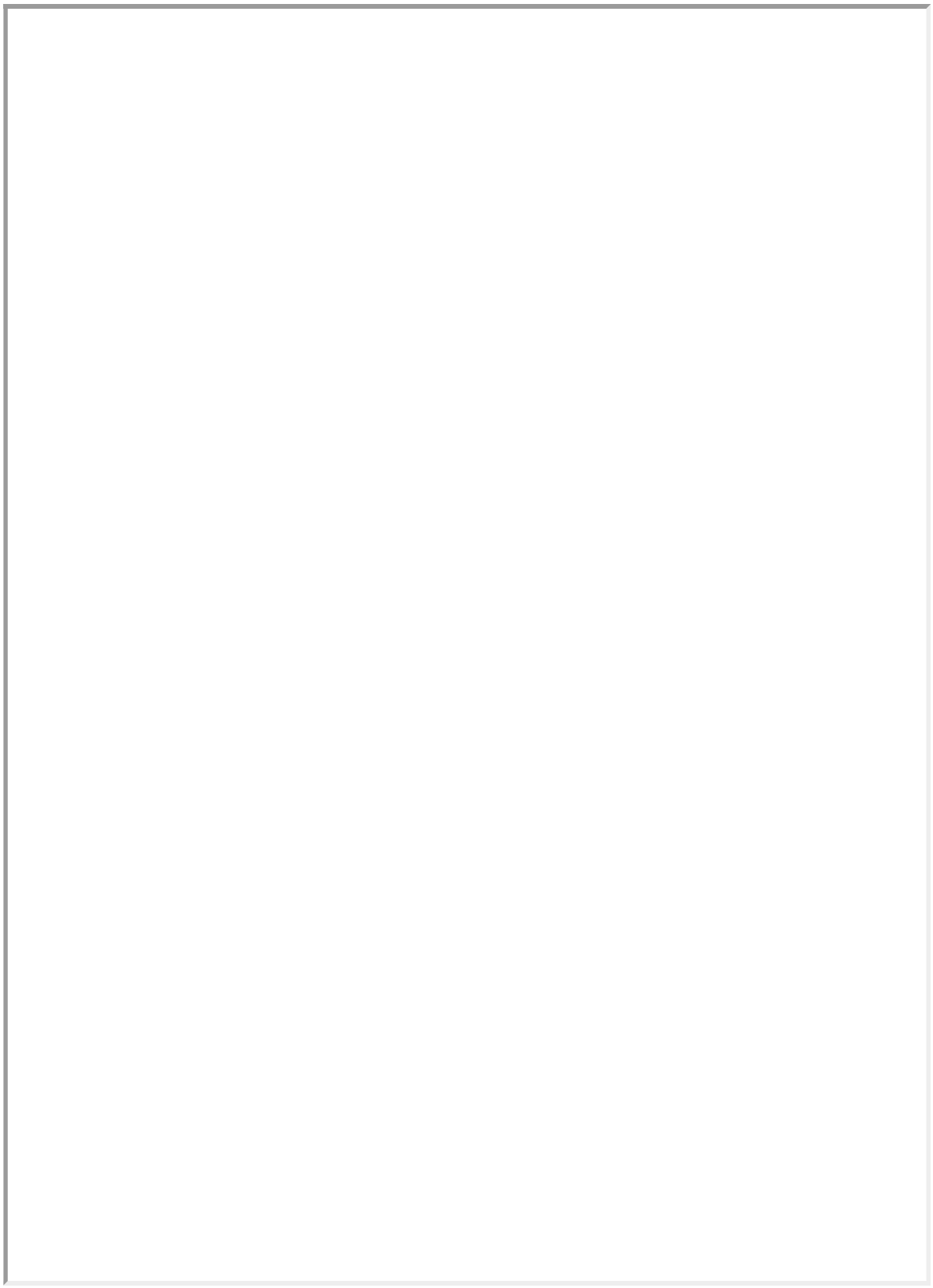
```
myList = [42, 51, 62, "Hello", 61, 123, "World"]

# The first element in the list
print(myList[0])

# The third element in the list
print(myList[2])

# The fifth element in the list
print(myList[4])
```

Copy this code and see what it does. Note that the *first* element is given by the number *0* in the list, and the second by *1* and so on...



Trinket Emulator

Exercise 10: In the Trinket below repeat Exercise 7, but instead of storing your name and age is separate variables, store them in a *single* list.



Trinket Emulator

11. For-loops

Sometimes you might want to repeat the same piece of code several times. Loops allow us to accomplish this in fewer lines of code, by *repeating* code until our operation is complete.

Look at the code below, what do you think it does? See if you can figure it out before you run it.

```
for n in range(1,10):
  print("The number is", n )
```

Now copy the code and run it. Did you guess correctly?

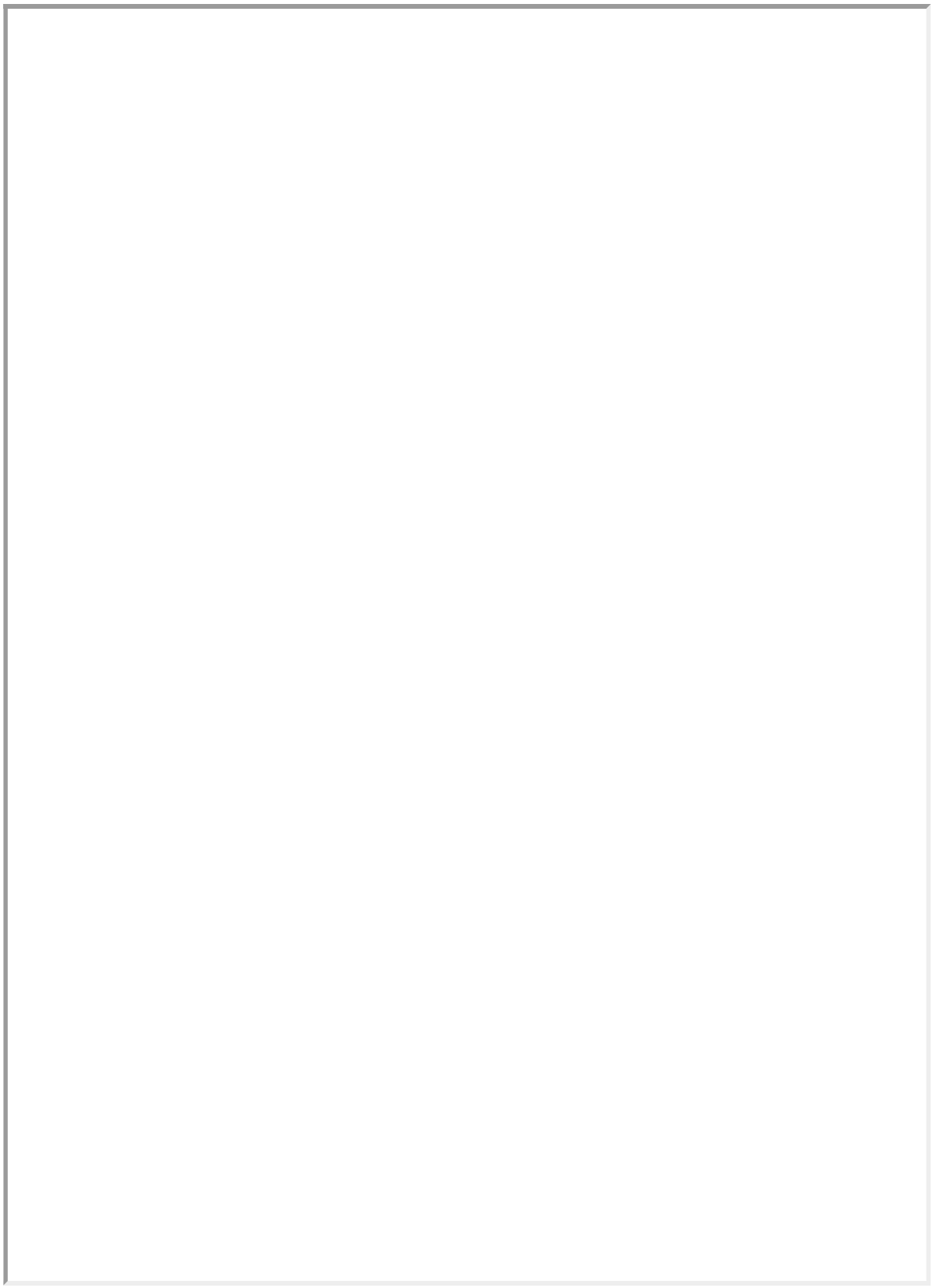
How about this code?

```
for n in range(1, 5):
  sense.set_pixel(n, 0, 0, 0, 255)
```

This one is more complicated. Try to imagine what this could do. After that, copy it and see for yourself.

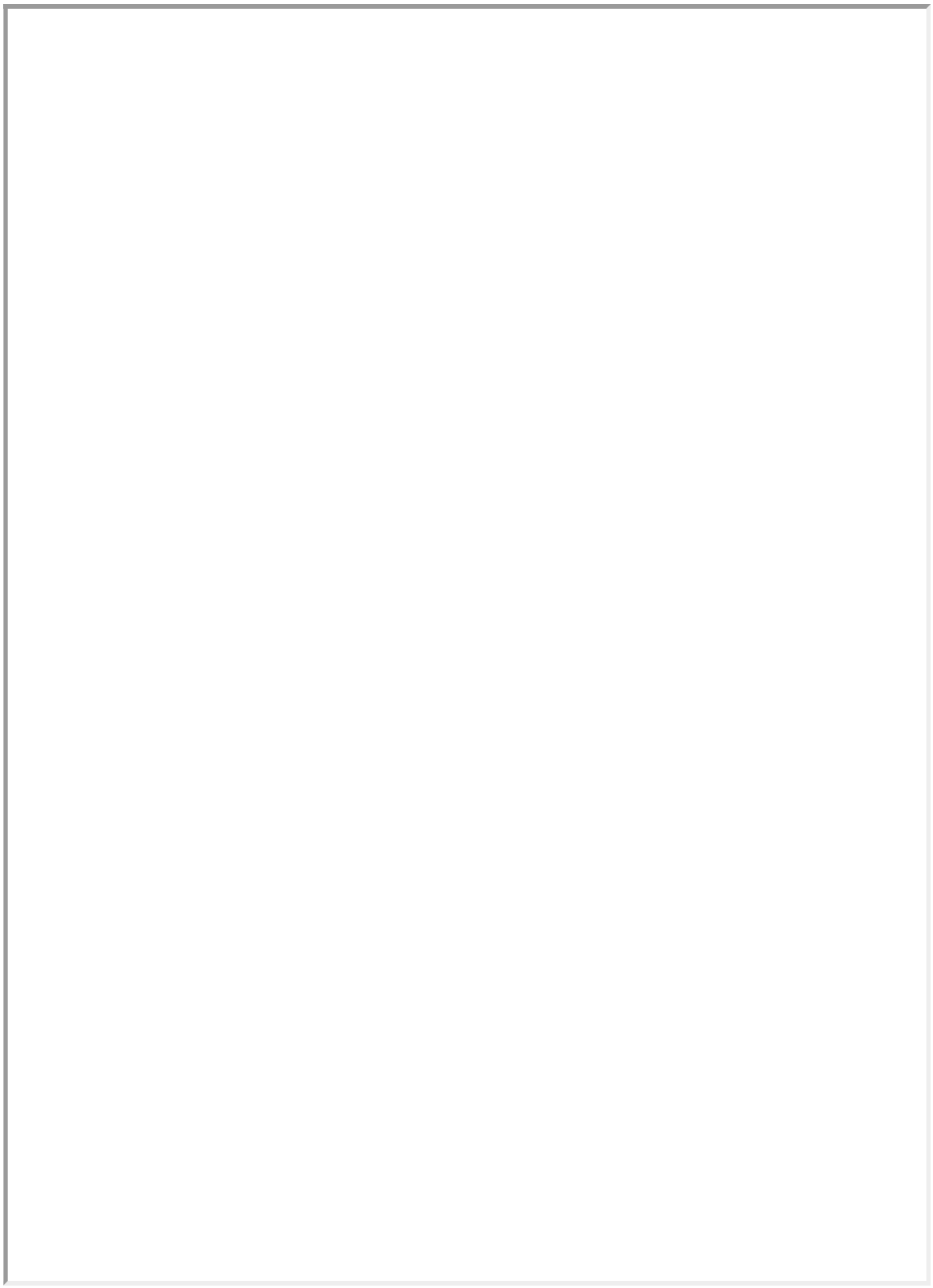
Play around with the code and see if you can make it do more complicated!

Exercise 10: Write a program that *prints* the numbers from 50 to 100.



Trinket Emulator

Exercise 11: Write a program that draws a line of yellow pixels from the top-left corner, to the bottom-right corner.



Trinket Emulator

12. While-loops

Another type of loop that we can use in Python is the *while loop*.

The while loop repeats its code as long as the expression after the *while* command evaluates to *true*. You can think of it as a mix between an *if-statement* and a *for-loop*.

The previous explanation might be a bit complicated to follow, but if you look at the code below you can see that it's quite intuitive.

```
n = 0 # Initialising n
while n<10: # Starting the loop with condition that n must be less than 10
    n = n+1 # redefines n as n+1
    print(n) # print the new value of n
```

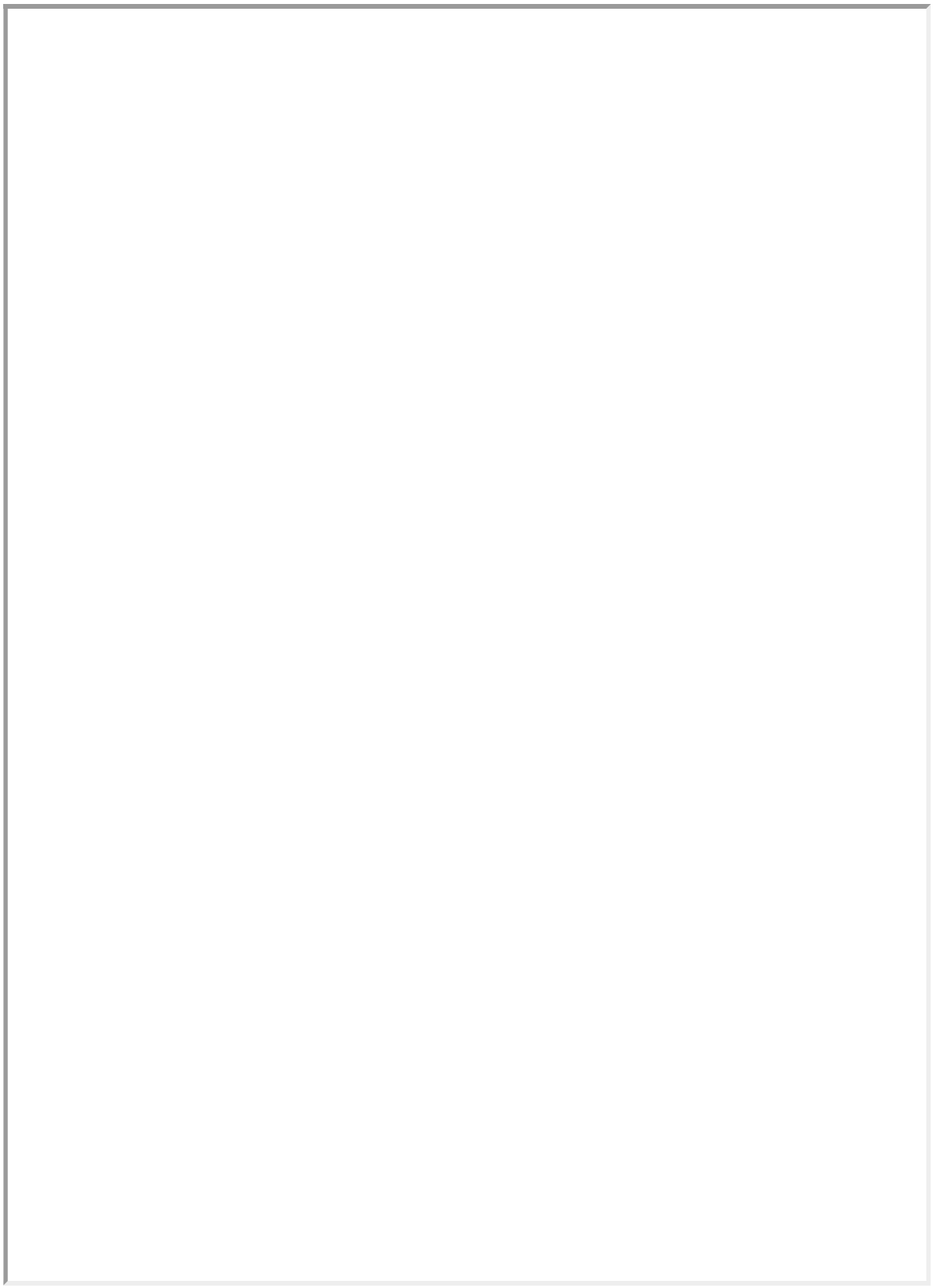
As usual, look at the code and try to figure out what it does. After that copy it and see for yourself!

What do you think will happen if you run the following code?

```
while True:
  print("Hello!")
```

Don't try to run it! What will happen is that the code will run *forever*, can you see why? It'll print out "Hello!" again and again and it wont stop! Although never-ending codes like this might seem useless to you, it does actually come in handy, and we'll often use them in our projects!

Exercise 12: Redo exercise 10, but with a *while*-loop, instead of a *for*-loop.



Trinket Emulator

13. What's next?

The next thing is to ask yourself if you've understood everything!
We covered quite a lot, and very quickly, so it might be worth to just have a quick read-through of everything again.

After that, it's time to start choosing a project. You should try to start off with one of the easier projects, before attempting any harder ones. Remember that you can always ask for help within your group, or by a supervisor, during the entirety of the summer school.

We discussed some useful functions in this lecture, like *sense.set_pixel* and *print*. There are many more functions you can use, and the best way to check them all out is by reading the *Function Reference* document. Use that document as a reference on how to use various Python functions.

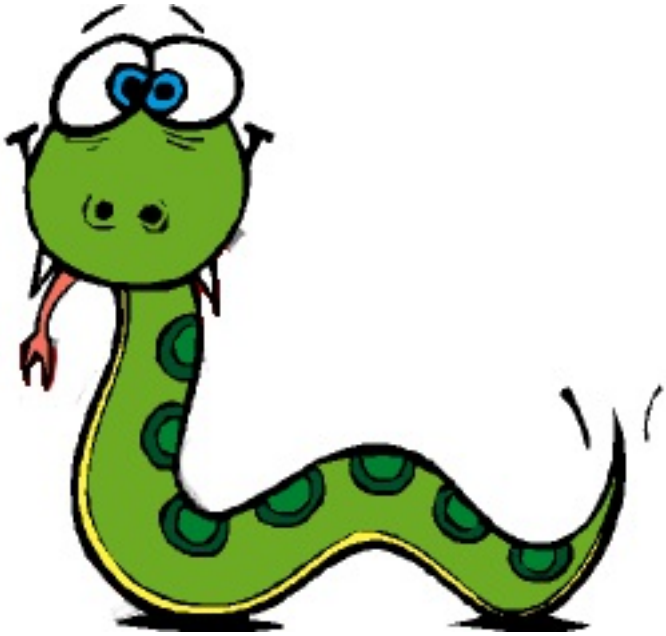
Good luck!

Authors: Lukas Kikuchi & Ishan Khurana

Date: August 09, 2017

Project: Snake

Difficulty level: Medium



Description

In this project, you'll recreate the classic game *Snake*.

You can try out the complete version of the project here:

Use the up, down, left and right keys to move the snake around. Eat the apples to grow.

Project Manual

This project guide will tell you step-by-step the main things you have to do in order to create the game *snake*. For some of the steps, you'll have to use your own creativity to proceed, good luck!

Introducing the project

The first thing you should do is open the *skeleton code* for the project.
In programming, skeleton code means code that only has the basic elements of a program. It is up to you to fill in the rest!

You can find the skeleton code on Trinket, here:
https://goo.gl/VkEkW1 (https://goo.gl/VkEkW1)

If you can, you should also create and account and log in to Trinket. This will allow you to save the Trinket projects. Otherwise you have to copy the code on to your computer to save it.

On Trinket, you'll be able to test your code on a *virtual* Sense HAT, before you try your code on the real thing.

As you might see, the skeleton code is split up into sections, divided by the headlines. For example:

```
#### 2. Code section
```

This guide will go through the various sections (not necessarily in order), and help you write your code. **Very important note:** *You should add the code in the specified section in your skeleton code as you follow this guide.*

The next part of this guide will explain the stuff that's already in the skeleton code when you first open it.

Explanation of the skeleton code

Before we get on to the coding, it's worth looking over the *skeleton code* and make sure you are familiar with it.

The first few lines in the script are:

1.1 Import libraries

```
import sys
sys.path.insert(1,'/home/pi/Go4Code/g4cSense/skeleton')

from sense_hat import SenseHat
from snake_lib import Snake
import random
from senselib import *
```

Without going into detail, these lines are called *import statements*. They are used to *import* code from other Python files into your own file. This is useful because you can use other people's code to simplify your own.

The next part of the code (Sec. 1.2) creates some important *Objects* (don't worry if you're not sure what that means) that we'll use in the later on.

Sec. 1.3 is where the initial variables of the project are set up.

Sec. 2.1 is where we'll write code that lets the user control the snake.

Sec. 2.2 is where we'll draw the snake on the screen.

Sec. 2.3 is where we'll make the snake move.

Sec. 2.4 is where we'll check if the snake has collided with itself, in which case the player will lose the game.

Sec. 2.5 is where we'll check if the snake has eaten an apple, in which case the snake should grow.

Sec. 2.6 we add some delay into the game, so that it won't move too fast.

Writing the code

(Sec. 1.3) Set up the game variables

In this section we'll be defining some variables that we will use in the game. A lot of programming is just about knowing what information to store, and where to store it. For example, one important piece of information to store is the position of the apple on the screen. If you look at the skeleton code, you see that there's a variable called *applePosition*.

```
applePosition = [1, 1]
```

The first number is its x-coordinate, and the second number is the y-coordinate. If you don't know what that means, the x-position is how many squares the ball is away from the left side of the screen, and the y-position is how many squares the ball is away from the top side of the screen.

Every time the snake eats the apple, we'll set a new position in *applePosition*.

The line

```
snake = Snake([3,3])
```

creates the snake *object* that we see on the screen. The two numbers you see is also the starting position of the snake.

The final two variables are the colours that we will draw the snake and the apple in

```
snakeColour = [0, 255, 0] # Colour of the snake
appleColour = [255, 0, 0] # Colour of the apple
```

Colours in Python are not called "purple", "yellow" or "brown". Instead they are determined by *three* numbers. The first number is how *red* the colour is, the second number is how *green* it is, and the third how *blue* it is. We've currently set the snake to be green, and the apple to be red, but feel free to change that!

(Sec. 2) Main program code

Before we start coding, we'll explain a bit what's going on in Sec. 2.

In this part of the code, we'll do all the main programming.
If we look at the skeleton code, we see that the section is within a *while*-loop. Remember that code inside a while loop runs again and again, until we tell it to stop. The reason why we want our program to be in a loop is because we want to continually check if the user has shaken the device.

Because the code is in the *while*-loop, remember that you have to add a *Tab* at every line, to make it indented. Like this:

```
while True:
    # write your code like this,
    # with a tab at the start of the line.
```

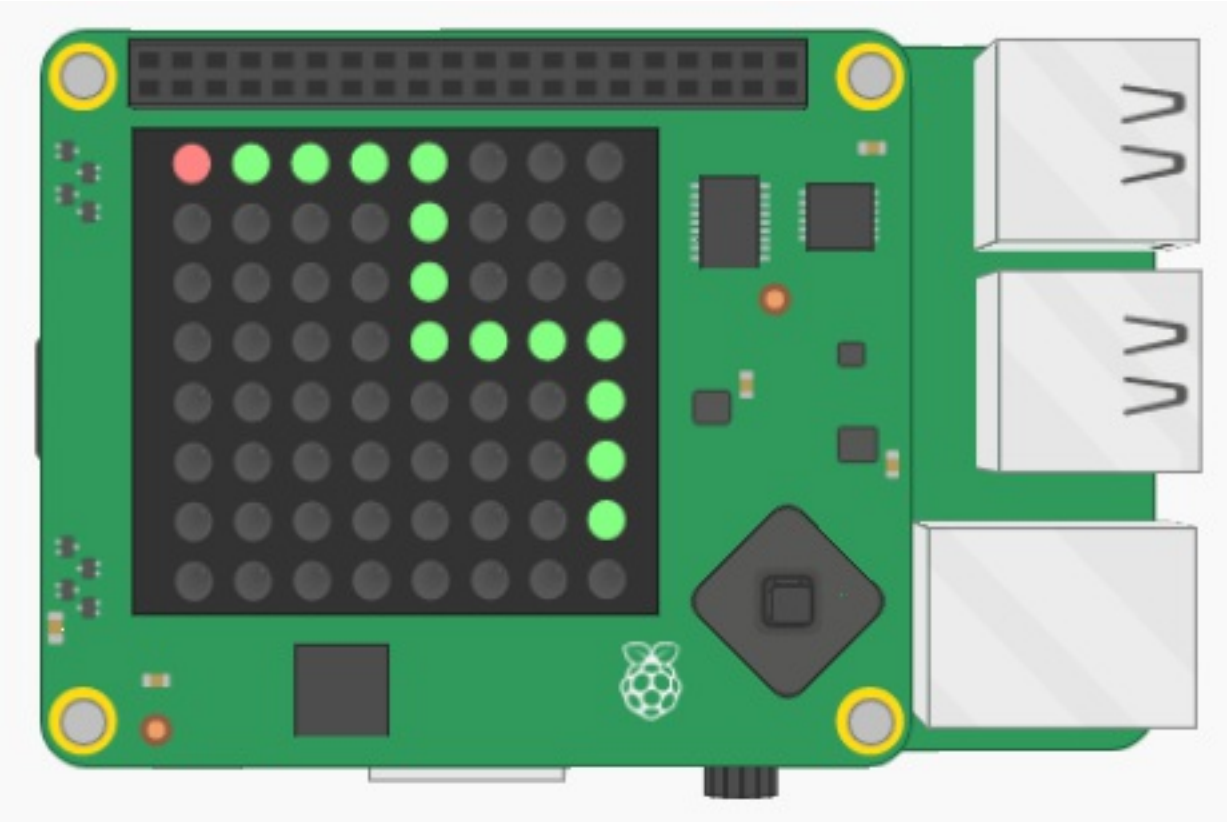
(Sec. 2.2) Draw the snake and the apple

Clearing the screen

The first thing you want to do before drawing anything is to clear the screen of any drawings. This is important, because otherwise all drawings will be stacked on top of each other. You can do this using the *sense.clear()* function.

```
sense.clear() # Clears the screen
```

Drawing the snake



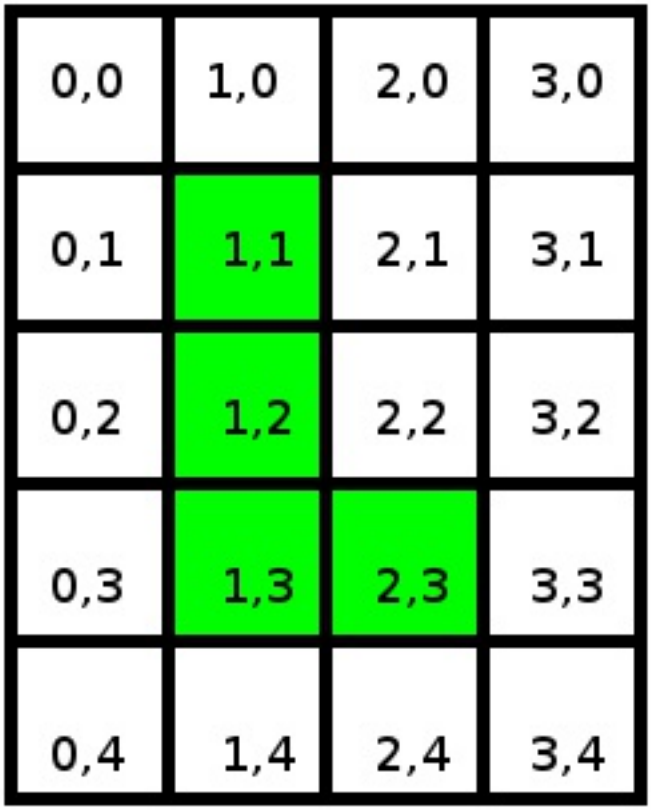
If you look at the image, you see that the snake's body can twist and turn into quite a complicated shape! Luckily, we have already prepared some code that keeps track of the snake's body behind-the-scenes. It's now your job to draw the snake.

This part bit is a bit complicated! So we'll start with some background explanation, before we get to how to do this part of the code.

The snake's body is stored in a *list*, where the *elements* of the list are *x*- and *y*-coordinates. For example, the body of a snake that is in an *L*-shape could be represented like this:

```
snake_body = [ [1, 1], [1, 2], [1, 3], [2, 3] ]
```

This is shown in the picture below:



Look at the numbers to get a feeling of how the *x* and *y*-coordinates work.
If you're having trouble understanding how this works, ask a supervisor about it!

Now that we've explained how the snake's body is stored in a list, we can go on to explain how to draw the body. As we mentioned earlier, the snake's body is kept track by code that we've already pre-written. So all you need to do is to call a function that we've made that *returns* the snake's body.

```
snake_body = snake.get_body() # Gets the body of the snake, containing all the positions.
```

Now that we have a list of all the points that the snake consists of, we can start drawing it. We're going to do this using the *for*-loop you learned about earlier. We can use a *for*-loop to go through each element in a list in order. For example, the following code

```
snake_body = [ [1, 1], [1, 2], [1, 3], [2, 3] ]

for bodypart in snake_body:
    print(bodypart)
```

Will print out:

```
[1,1]
[1,2]
[1,3]
[2,3]
```

Now instead of *printing* each body part, we want to *draw* it. We can do this using the *sense.set_pixel* function. Read about it in the *Function Reference*, and use it to draw the snake! Remember that the colour of the snake is stored in the *snakeColour* variable.

Draw the apple

This step is much simpler. Use the *sense.set_pixel* pixel to draw the apple on the screen. The position of the apple is in the *applePosition* variable, and the colour is in *appleColour*.

Try running the game

If you have completed these steps, you should try running the game. Unless there were any mistakes, you should now see one red and one green dot on the screen. Since in the beginning of the game the snake hasn't grown yet, it's only a single pixel.

(Sec. 2.3) Move the snake

This step is really simple. Every *frame* we want the snake to move a step forward. We can do this by simply calling the following function

```
snake.move_forward()
```

Try running the game again, you should see the snake moving across the screen. You might notice that the snake is moving too fast, in the next section we'll fix that.

(Sec. 2.6) Add some delay

To make the game run a bit slower, add the following line to the section to slow the game down a little

```
wait(0.2)
```

The number is how long the delay should be, so you can change it if you'd like the game to go faster or slower. Experiment! Try running the game with different waiting times.

(Sec. 2.1) Let the user control the snake

At the beginning of the while-loop, you should write the following:

```
for event in sense.stick.get_events():
    if event.action == "pressed":

        if event.direction == "up":
            # Fill in with your own code

        elif event.direction == "down":
            # Fill in with your own code

        elif event.direction == "left":
            # Fill in with your own code

        elif event.direction == "right":
            # Fill in with your own code
```

This code will check if the user has pressed either the *up*, *down*,*left* or the *right* button on the joystick. Never mind the complicated structure, try to look at the code and figure out how to use it.

Inside the code you find the comments *# Fill in with your own code*. This is where you should write your code.

If the user presses *up*, *down*,*left* or *right* on the joystick, we want to turn the snake. Remember that earlier in (Sec. 1.3) we created a *Snake* object? Well it provides us with some useful functions that we can use to control the snake:

```
snake.turn_up() # Turns the snake up
snake.turn_down() # Turns the snake downwards
snake.turn_left() # Turns the snake to the left
snake.turn_right() # Turns the snake to the right
```

You can use these functions along with the code provided above to allow the user to control the snake.

After you've written the code in this section, try running it. If it's done correctly, you should be able to control the snake.

(Sec. 2.4) Check if apple ate an apple

If the *head* of the snake touches an apple, we want it to *eat* it and then grow one pixel. We also then want to create an apple at another position on the screen.

To check if the snake has touched the apple, we use an *if*-statement:

```
if snake.get_position() == applePosition:
    # Write code
```

The code inside the if-statement will only run, if the snake's head is on the apple position.

Inside the if-statement, we want to do two things:

- Grow the snake. We can do this using the *snake.grow()* function:

  snake.grow()

- Set a new position for the apple. This can be done using the *snake.get_new_apple_position()* function:

  applePosition = snake.get_new_apple_position()

Try running the code now. Every time the snake eats an apple, it should now grow.

(Sec. 2.5) Check if snake has collided with itself

We want the user to lose the game if the snake tries to "*eat*" itself (or in other words, if it collides with itself).

There are three things we want to do if the user loses the game:

- Show a "Game Over" message. We can do this using the *sense.show_message* function (again, check the *Function Reference* to see how to use it).
- Reset the snake so that it's small again. We do this by calling the *snake.reset()* function:

```
snake.reset()
```

- Set a new position for the apple, the same way we did it in the previous section.

To check if the snake has collided with itself, you can call the function *snake.has_collided_with_self()*. If it has, the function will return *True*, and if it hasn't it will return *False*. For example, the following code

```
if snake.has_collided_with_self():
  print("You lost!")
```

will only run if the snake has collided with itself.

Using this information, you should be able to write code that checks if the snake has collided with itself.

Finished!

If it's all done, correctly, the game should now work! Don't worry if it doesn't, things often go wrong in programming. Errors in code are usually called *bugs*. If you have a bug in your code, you'll have to *debug* it!

If it works, congratulations! You can either move on to another project or try to come up with new things to add to the current project. Use your creativity! You can discuss any ideas you have with a supervisor.

Author: Lukas Kikuchi

Date: August 09, 2017

Project: Snake & Ladders

Difficulty: Medium

Description

In this project you will be **working in pairs** to make the classic board game snakes and ladders on the Sense Hat.

The aim of the game is simple. Start from the bottom left corner, roll the dice and move along the board. You win once you reach the top left corner of the board. If you land on the bottom of the ladder you go up to the top of the ladder, but if you land on the head of a snake, you get bitten and go back to the tail!

If you are unsure of what this will look like on the Sense Hat, ask a supervisor to show you the game.

Project Manual

To complete this project we are going to break it down into two separate projects, the board and the dice.

There are two ways you can proceed. Either you both work together on the *dice* and then on the *board*, or one of you works on the *dice* and the other works on the *board*. Programming is often done in teams and programmers work together to create the software we use in our day to day lives. So if you decide to work independently on the *dice* and *board* make sure you help your partner if you finish first.

Game features

In this game, one Raspberry Pi is used to make *Dice* and the another Raspberry Pi will have the board.

Dice

- The dice are to be shake activated. Which means the raspberry pi displays random numbers from 1 to 6 when a user shakes it.
- If you are working on the ***Dice*** for the game, go to the folder labelled 'Dice' and follow the instructions in the dice project script.

Board

- You must design the board by placing snakes and ladders in positions of your choosing.
- The board has a moveable cursor that allows players to place their piece along the board depending on the number they get on the dice.
- If the piece lands on the head of a snake, it moves back down to the tail.
- If a piece lands on the foot of a ladder, it moves to the top.
- If the player lands on the top left corner, they win.
- If you are making the ***board*** for snakes and ladders, go to the folder labelled 'Board'

and follow the instructions

Go4Code Summer School
Student Guide

Welcome to the UCL Go4Code Summer School. You have all been given a Raspberry Pi and a Sense HAT. During your three days at UCL you will be programming your Raspberry Pi in *python* to control the Sense HAT.

We have prepared some projects for you to work on and once you get comfortable with coding you can design your own projects! All the material is on our website which is open on the computers in front of you and you can access this from anywhere.

We want you to be able to continue tinkering with code once you go back home and that is why the Raspberry Pi and Sense HAT are yours to keep!

Supervisors

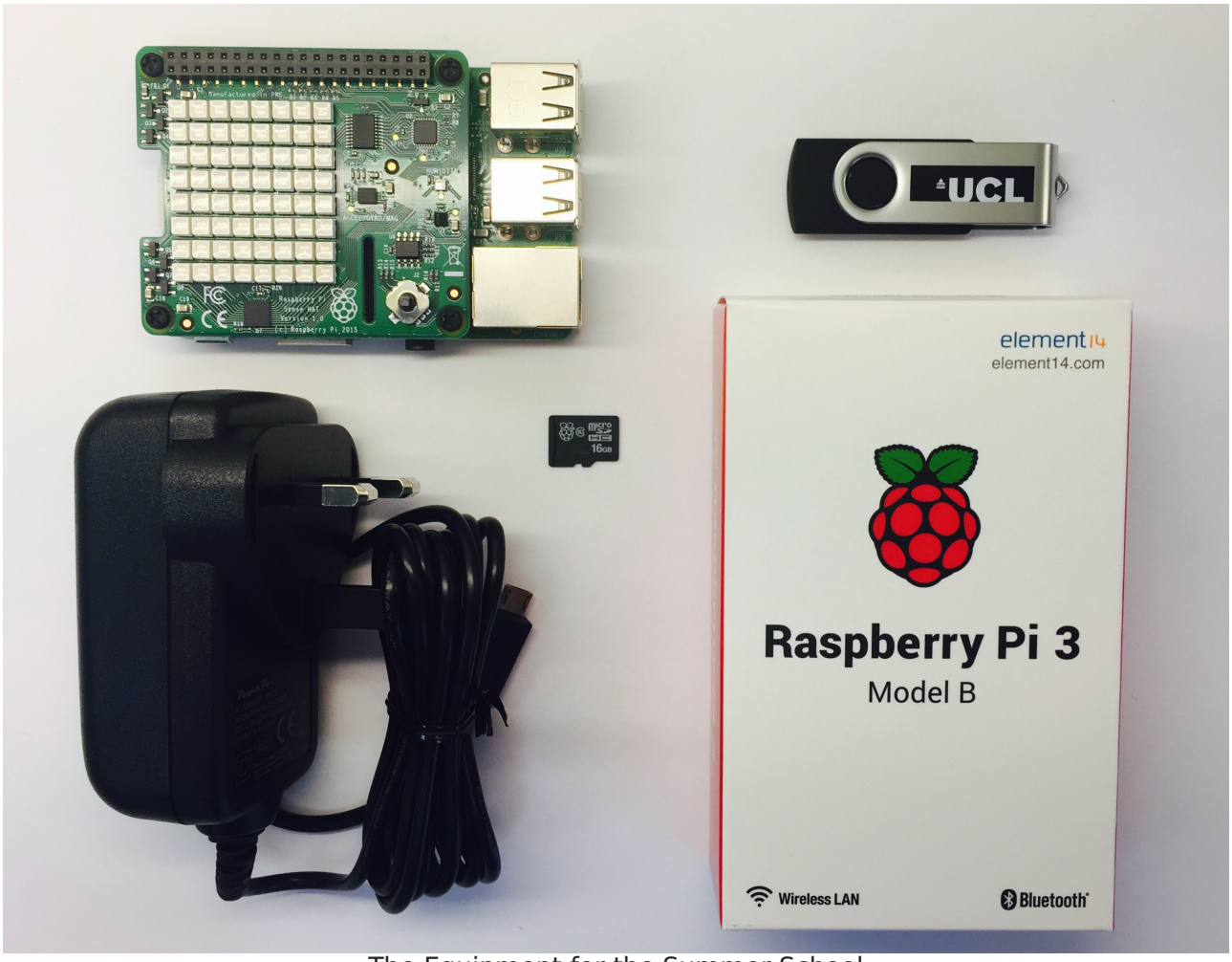
You have all been sorted into groups of six. Each group has a **supervisor** and two **mentors** who are here to help you get through the summer school. If at any point you need any help with the programming you should ask your mentors or supervisor for help.

Your supevisor has a Raspberry Pi and Sense HAT and will be able to show you how it works along with what the projects should look like when completed

Equipment

You will all have the following equipment in front of you:

- Raspberry Pi 3
- Sense HAT
- Power supply
- Pre-programmed micro SD card
- USB memory stick



The Equipment for the Summer School

Note: The Raspberry Pi and the Sense HAT have been pre-assembled and the micro SD card has already been inserted in the Raspberry Pi.

To turn the Raspberry Pi on, simply plug in the *power supply* into a plug socket and connect the *micro USB* cable into the Raspberry Pi. When you do this, you will see the lights on the Raspberry Pi flash in the colours of the Rainbow.

If you are missing any items or your Raspberry PI does not turn on, **ask your supervisor** to get you a replacement.

Starting out

When you first arrive at the computer cluster, you will need to find the computer which has your name card. This computer will already be logged into your UCL computer account and will have the Go4Code Coding Summer School website open.

In case it is not already open, the URL for the website is:

[codingsummerschool.github.io/codingsummerschool (https://codingsummerschool.github.io/codingsummerschool)](https://codingsummerschool.github.io/codingsummerschool)

To start with you will go through the [Introduction to Python and Sense HAT page (https://codingsummerschool.github.io/codingsummerschool/docs/SenseHatIntro.html)](https://codingsummerschool.github.io/codingsummerschool/docs/SenseHatIntro.html). This will introduce you to the features of the Sense HAT and important coding concepts that you will be

required to use in your projects.

Once you have made your way through this tutorial you will have the basic knowledge to start on your own Sense HAT project!

Projects

The projects are split into 3 categories:

- Easy
- Medium
- Hard

You are encouraged to start with the *easy* projects moving to *medium* then *hard*. You can do as many projects as you would like and can even create your own.

Once you have selected a project, discuss it with your supervisor, and ask them to show you what the final version should like.

You will be developing your code on the Trinket Emulator on the desktop and then transferring your code to the Raspberry Pi via the USB stick provided.

The instructions on how to do this are in the Running Files on the Sense HAT (https://codingsummerschool.github.io/codingsummerschool/docs/Running-Files-On-the-Sense-HAT.html).

Programing is about creativity. So feel free to get creative and add your own twist onto the suggested projects. If you have completed a few of the ones we have prepared and you have an idea that you want to build, you can create you own project! Ask you supervisor to help you with this.

For the future (parents read this section!)

All the equipment is yours to keep! This way you can carry on working on your projects and further develop your programming skills.

The Raspberry Pi is your own computer, all you need to complete the package is:

- HDMI cable
- USB Mouse and Keyboard
- An HDMI TV/Monitor

Useful websites to help develop your coding skills:

- codecademy.com (https://www.codecademy.com)
- raspberrypi.org (https://www.raspberrypi.org)

Examples of more projects you can do with your Pi's:

- Motion sensitive security camera
- Retro game console
- Smart lights
- Music Player

The skills that you will learn over the next couple of days are used by programmers all over the world! This is just the start of your coding adventures.

Author: Laura Hargreaves & Ishan Khurana

Date: August 15, 2017

<p align = "center", style = "line-height:1.3" >

Go4Code Summer School <font size = "5", face = "Courier"> Student Guide

Welcome to the UCL Go4Code Summer School. You have all been given a Raspberry Pi and a Sense HAT. During your three days at UCL you will be programming your Raspberry Pi in *python* to control the Sense HAT.

We have prepared some projects for you to work on and once you get comfortable with coding you can design your own projects! All the material is on our website which is open on the computers in front of you and you can access this from anywhere.

We want you to be able to continue tinkering with code once you go back home and that is why the Raspberry Pi and Sense HAT are yours to keep!

Supervisors

You have all been sorted into groups of six. Each group has a **supervisor** and two **mentors** who are here to help you get through the summer school. If at any point you need any help with the programming you should ask your mentors or supervisor for help.

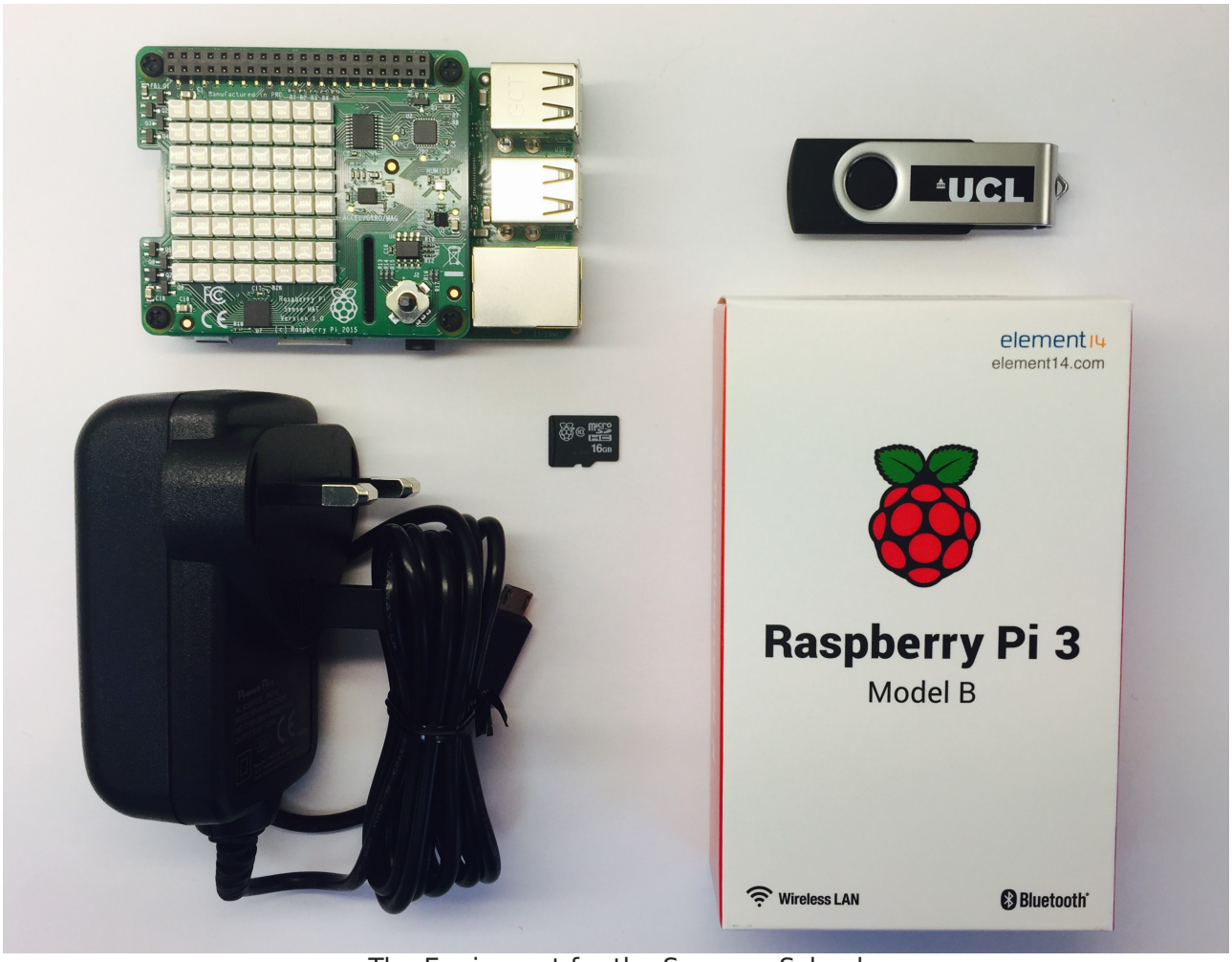
Your supevisor has a Raspberry Pi and Sense HAT and will be able to show you how it works along with what the projects should look like when completed

Equipment

You will all have the following equipment in front of you:

- Raspberry Pi 3
- Sense HAT
- Power supply

- Pre-programmed micro SD card
- USB memory stick



The Equipment for the Summer School

Note: The Raspberry Pi and the Sense HAT have been pre-assembled and the micro SD card has already been inserted in the Raspberry Pi.

To turn the Raspberry Pi on, simply plug in the *power supply* into a plug socket and connect the *micro USB* cable into the Raspberry Pi. When you do this, you will see the lights on the Raspberry Pi flash in the colours of the Rainbow.

If you are missing any items or your Raspberry PI does not turn on, **ask your supervisor** to get you a replacement.

Starting out

When you first arrive at the computer cluster, you will need to find the computer which has your name card. This computer will already be logged into your UCL computer account and will have the Go4Code Coding Summer School website open.

In case it is not already open, the URL for the website is:

[codingsummerschool.github.io/codingsummerschool (https://codingsummerschool.github.io/codingsummerschool)](https://codingsummerschool.github.io/codingsummerschool)

To start with you will go through the Introduction to Python and Sense HAT page (https://codingsummerschool.github.io/codingsummerschool/docs/SenseHatIntro.html). This will introduce you to the features of the Sense HAT and important coding concepts that you will be required to use in your projects.

Once you have made your way through this tutorial you will have the basic knowledge to start on your own Sense HAT project!

Projects

The projects are split into 3 categories:

- Easy
- Medium
- Hard

You are encouraged to start with the *easy* projects moving to *medium* then *hard*. You can do as many projects as you would like and can even create your own.

Once you have selected a project, discuss it with your supervisor, and ask them to show you what the final version should like.

You will be developing your code on the Trinket Emulator on the desktop and then transferring your code to the Raspberry Pi via the USB stick provided.

The instructions on how to do this are in the Running Files on the Sense HAT (https://codingsummerschool.github.io/codingsummerschool/docs/Running-Files-On-the-Sense-HAT.html).

Programing is about creativity. So feel free to get creative and add your own twist onto the suggested projects. If you have completed a few of the ones we have prepared and you have an idea that you want to build, you can create you own project! Ask you supervisor to help you with this.

For the future (parents read this section!)

All the equipment is yours to keep! This way you can carry on working on your projects and further develop your programming skills.

The Raspberry Pi is your own computer, all you need to complete the package is:

- HDMI cable
- USB Mouse and Keyboard
- An HDMI TV/Monitor

Useful websites to help develop your coding skills:

- codecademy.com (https://www.codecademy.com)
- raspberrypi.org (https://www.raspberrypi.org)

Examples of more projects you can do with your Pi's:

- Motion sensitive security camera
- Retro game console
- Smart lights

- Music Player

The skills that you will learn over the next couple of days are used by programmers all over the world! This is just the start of your coding adventures.
Title: Project Plan - Marble Maze
Author: Lukas Kikuchi
Date: August 07, 2017

Project: Marble Maze

Difficulty level: Medium

Description

In this project you'll be creating a digital version of Marble Maze (see the picture if you don't know what it is).
The Sense HAT can tell whether you're tilting your device, using this feature we can create the game. You'll also get to design your own maze in this project.

Project Manual

This project guide will tell you step-by-step the main things you have to do in order to create *Marble Maze*. For some of the steps, you'll have to figure out how to proceed yourself, good luck!

Introducing the project

The first thing you have to do is open the file you'll be programming in. In the *Marble Maze* project folder, open the file called *main.py*. The code in this file is not enough to actually run Marble Maze, you'll have to fill in the blanks! In programming, we call this *skeleton code*.

As you might see, the skeleton code is split up into sections, divided by the headlines. For example:

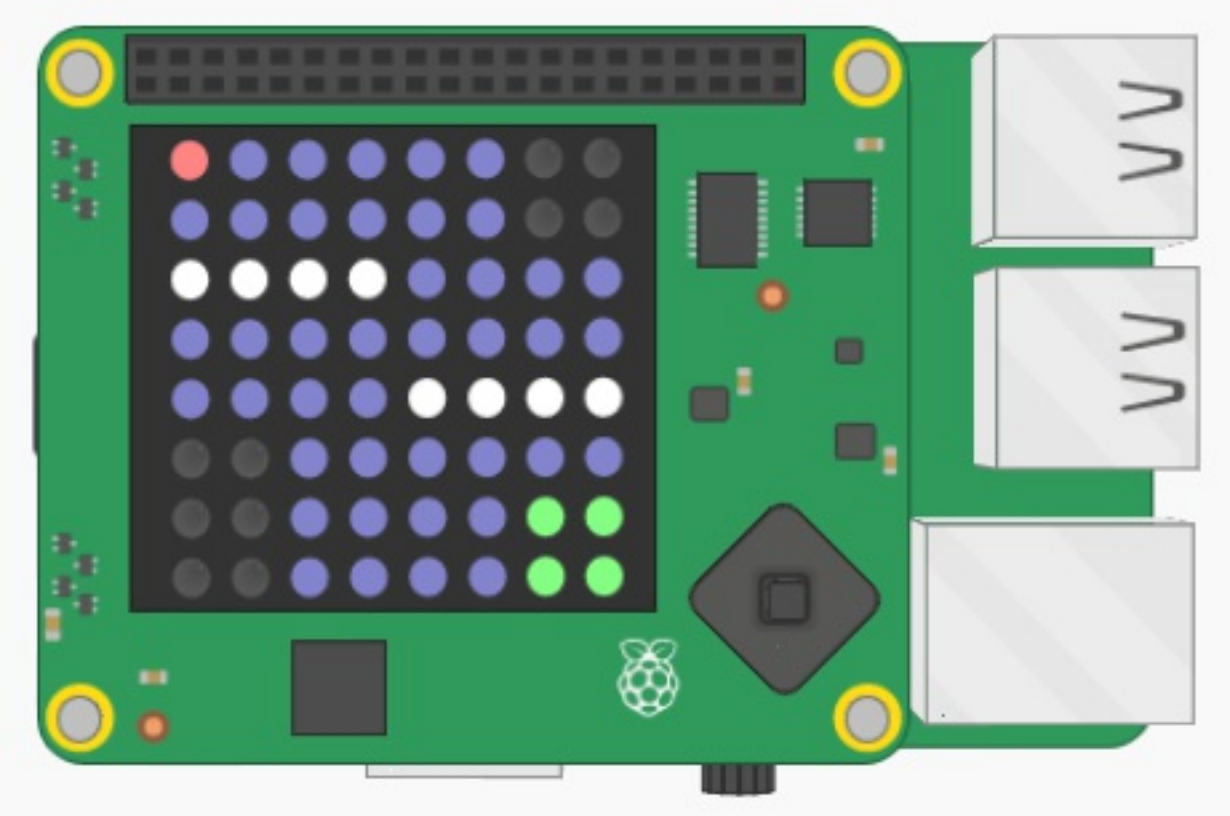
```
#### 2. Code section
```

This guide will go through the various sections (not necessarily in order), and help you write your code. **Very important note:** *You should add the code in the specified section in your skeleton code as you follow this guide.*

The next part of this guide will explain the stuff that's already in the skeleton code when you first open it.

Explanation of the game

The game you'll be working works very similarly to an actual Marble Maze game. Here's a screenshot of the game:



You have a ball (red dot in the picture) that you control move around by tilting the Sense HAT. The ball can collide and
be stopped by the walls (white dots in the picture) of the maze. The player loses if the ball goes in the holes (black dots). And the player wins if the ball goes in the goal (green dots).

You can try out the complete version of the game here:

https://goo.gl/9XABaX (https://goo.gl/9XABaX)

Coding in Trinket

You should write and develop your code on the website https://trinket.io/ (https://trinket.io/). There, you'll be able to test out your code on a *virtual* Sense HAT, before you try out your code on the real thing. Go on to the website, and create an account.

You should copy all of the code in *main.py*, as well as the other files in the project directory, on to a new Trinket project.

Explanation of the skeleton code

Before we get on to the coding, it's worth looking over the *skeleton code* and make sure you are familiar with it.

The first few lines in the script are:

```
#### 1.1 Import libraries

import sys
sys.path.insert(1,'/home/pi/.Go4Code/g4cSense/pong/skeleton')

from sense_hat import SenseHat
from marble_maze import MarbleMaze
import random
from senselib import *
```

Without going into detail, these lines are called *import statements*. They are used to *import* code from other Python files into your own file. This is useful because you can use other people's code to simplify your own.

The next part of the code (Sec. 1.2) creates some important *Objects* (don't worry if you're not sure what that means) that we'll use in the game code.

Sec. 1.3 is where we set up the initial properties of the game is set up. Like the starting position of the ball, the color of the ball, the color of the wall, the goal and the holes and so on. You'll need to edit these values later on in order to customize the game.

Sec. 1.4 is where you'll create your maze!

Sec. 2 is the part where you'll get your hands dirty with some real python game programming. If you look at it now, there's not much there:

```
#### 2. Main game code

while True:

    #### 2.1 Make the ball move

    #### 2.2 Draw the walls, holes, goals and the ball

    #### 2.3 Make things happen

    #### 2.4
```

You'll be filling in each section with your own code. The most important thing to note is

```
while True:

    # ... The rest of the code ...
```

The *while* part of the code is what we call the *main loop*. All the code that's *inside* the while-clause will be repeated again and again until the game ends. That's why we call it a loop!

Writing the code

(Sec. 1.3) Set up the game variables

In this section we'll be defining some variables that we will use in the game. A lot of programming is just about knowing what information to store, and where to store it. For example, some very important information to store is the position of the marble ball on the screen. To set the ball position, you have to edit this line:

```
game.setBallPosition(0, 0)
```

Replace the first number with the *x*-position of the ball, and the second number with the *y*-position. If you don't know what that means, the x-position is how many squares the ball is away from the left side of the screen, and the y-position is how many squares the ball is away from the top side of the screen.

The rest of the lines in this section decides the colors of all the things in the game. Currently all the colors are set to black, so you'll definitely have to change them to something, although you can decide which colors yourself.

Colours in Python are not called "purple", "yellow" or "brown". Instead they are given by *three* numbers. The first number is how *red* the color is, the second number is how *green* it is, and the third how *blue* it is. The value of the redness/greenness/blueness can be from 0 to 255 (where 255 is the maximum redness/greenness/blueness). For example to set the ball to be purple, we set it to be maximum red and maximum blue, like this:

```
game.ballColour = [255, 0, 255]
```

Confusing? Try setting the colors to different things later to get the hang of it.

(Sec. 2.2) Draw the walls, the holes, the goals and the ball

In this section we'll write code that draws stuff on the Sense HAT.

The first thing we want to do is to draw the floor. You do this using the function:

```
sense.clear(red, green, blue)
```

You can replace *red*, *green* and *blue* with any numbers that you like. You should add this code to the start of the section.

The next thing is to add code that will draw the walls, the holes and the goal blocks. In order to do this, you simply have to add the following lines of code:

```
game.drawWalls()
game.drawHoles()
game.drawGoals()
```

The final thing is to draw the ball. To do this you use the function:

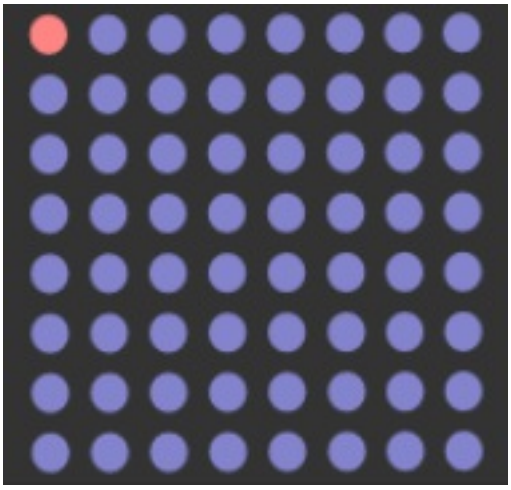
```
sense.set_pixel(x, y, red, green, blue)
```

Where *x* and *y* is the position of the ball, and *red*, *green* and *blue* the color of the ball. You can get the *x* and *y* positions of the ball using:

```
game.getBallX()
game.getBallY()
```

You'll have to use these functions within the *sense.set_pixel*. The color of the ball is in the *game.ballColour* variable. For more information about how the function works, you can check the *Function Reference* document.

After you've added all of this, all the stuff should be drawn on the scren if you start it! Because you haven't yet added anything, it should look similar to this:



(Sec. 2.1) Make the ball move

In this section, you'll be making the ball move by tilting the Sense HAT.

Using the *game.moveBallHorizontally* and *game.moveBallVertically* functions, we can tell the game how tilted the Sense HAT is, and how fast the ball should move across the screen. For example, to tell the game that HAT is tilted *20 degrees* horizontally, and *40 degrees* vertically:

```
game.moveBallHorizontally(20)
game.moveBallVertically(40)
```

To get how much the Sense HAT is tilted horizontally, use:

```
sense.get_orientation()["pitch"]
```

To get how much the Sense HAT is tilted vertically, use:

```
sense.get_orientation()["roll"]
```

If you run the game now, the ball should move across the screen as you tilt the Sense HAT.

(Sec. 1.4) Create the maze!

This is a fun section! Here, you'll be creating the maze.

To create a wall pixel:

```
game.placeWall(x, y)
```

where *x* and *y* should be replaced by numbers. To create a hole pixel:

```
game.placeHole(x, y)
```

if the ball falls in the hole, the player will lose. To create a goal pixel:

```
game.placeGoal(x, y)
```

At the start, all the pixels are automatically floor pixels, so you don't need to add them.

After you've made the maze, try running the game. All the things you placed should be on the screen!

(Sec. 2.3) Make things happen

This one is simple, just add the following line:

```
game.update()
```

This line should make sure that the ball collides properly with the walls.

(Sec. 2.4) Check if the player has won or lost

As we mentioned earlier, you can get the *x* and *y* positions of the ball using:

```
game.getBallX()
game.getBallY()
```

The following function can be used to check whether there's a hole at a particular position:

```
game.isHole(x, y)
```

The function will return *True* if there's a hole at the position (*x*, *y*).
The following function can be used to check whether there's a goal at a particular position:

```
game.isGoal(x, y)
```

The function will return *True* if there's a goal at the position (*x*, *y*).

It might seem a bit tricky but using these functions you should be able to check whether a player has lost (if the ball's position is on a hole) or has won (if the ball's position is on a goal).

The way to do this is using an *if*-statement. If you forgot what that was, check the introduction lecture from earlier, or ask a supervisor.

If the player has lost or won, the ball should be placed in the starting position again, so the player can play again.

Run the game

If it's all done, correctly, the game should now work! Don't worry if it doesn't, things often go wrong in programming. Errors in code are usually called *bugs*. If you have a bug in your code, you'll have to *debug* it!

Once you've completed the project, feel free to try to improve or change it by adding any new code to it. Use your imagination!

Full Test

This document will produced notes on my practice test of the coding website