Python & Minecraft Worksheet

Introduction

Welcome to Minecraft! But not as you know it. Games, like Minecraft, are more than just the images you see on screen, they are all built out of code. The following steps will teach you how to change the minecraft world using Python code.

Setup : Getting ready to code

Step 1 : The code can be downloaded to your Raspberry Pi from https://github.com/deep3/lesson-python-pi-minecraft/archive/master.zip

Step 2 : Unzip the folder. You can either double click on it and click Extract, or use the command line window and run the command. It doesn't matter when you unzip it too.

```
unzip lesson-python-pi-minecraft-master.zip
```

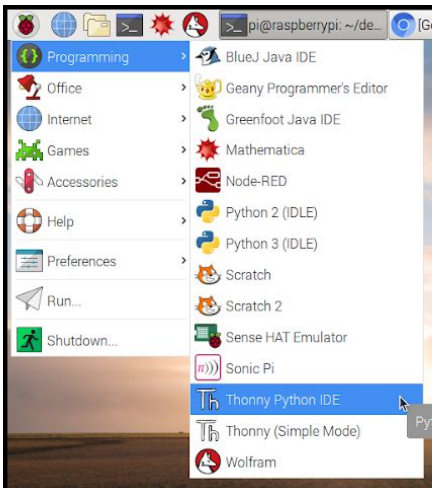
Step 3 : Run the SETUP.sh script. To do this use the command line, get to the folder where you unzipped the folder. you will need to use the 'cd' command for this.

Then enter the command

```
. SETUP.sh
```

you will then be asked a set of questions. just follow the instructions. When Minecraft opens, click start, and open the existing world. use the tab key to switch to the coding app that will have also opened.

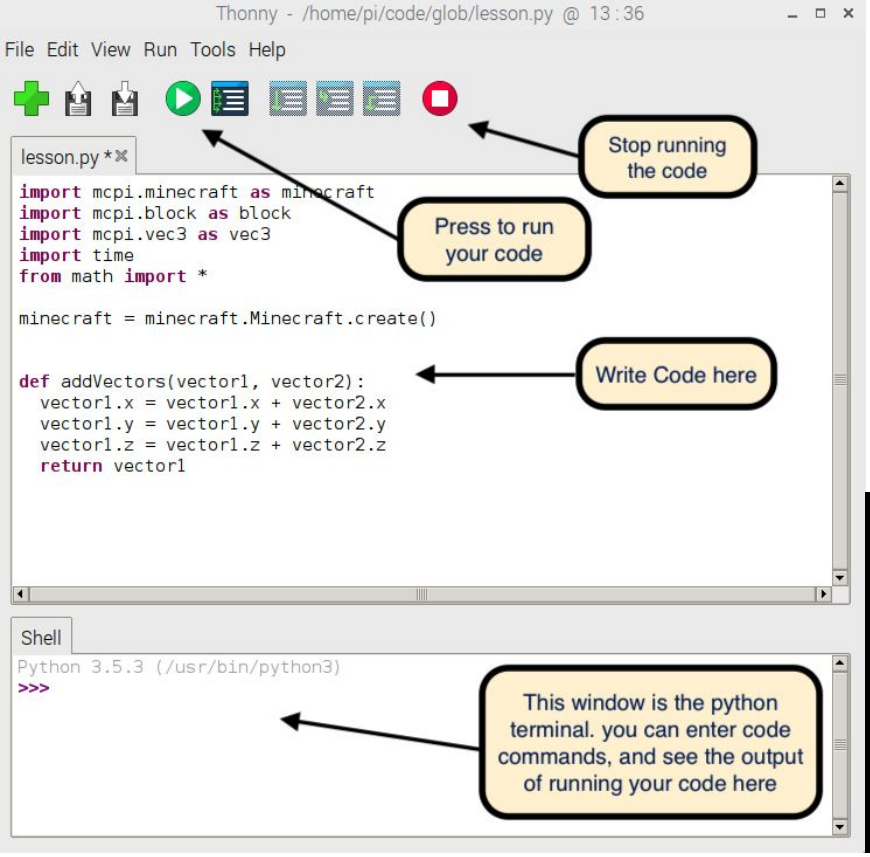
The Development Space : How to use the tools



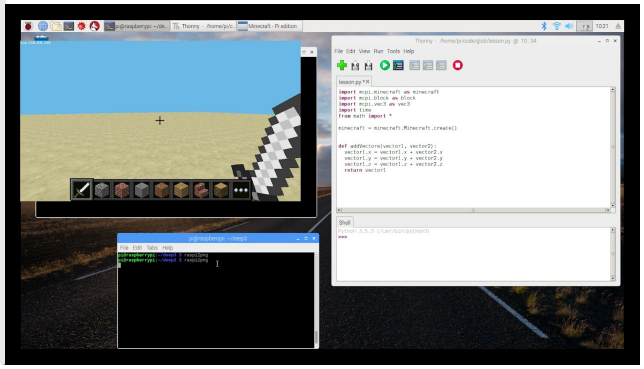
Writing code into a machine is simple, and there are lots of ways to do it! If you already have a prefered way then stick to it.

For this lesson, we recommend using **Thonny Python IDE**. This will open up automatically when running the SETUP.sh script, but you can also open it from the drop down menu. You will also need minecraft, which can be opened from the games drop down.

Thonny gives an easy way to write and run code. Write your code in the top portion, then run it and see the output in the bottom. The bottom section is a python terminal, which means you can run individual commands in it, and it will show the result. For this lesson well will just look at it to fix problems.



Once you have the editor open, you will find it easier to arrange your screen so you can see minecraft and the editor at the same time. When you hit the green run button, you will see what it does to the minecraft world!



Using minecraft is simple. the basic controls are:

- Move using WASD
- Look around with the mouse
- press TAB to stop controlling minecraft
- SPACE lets you jump.
- If you double tap SPACE you switch into flying mode. In flying mode you can press space to rise and shift to fall. This will help you see bigger things you build. doubler tap again to go back to walking

Lesson 1 : Imports

1.1 Imports

First we need to get all the bits, known as modules, to start affecting the minecraft world. We do this using the "`import`" keyword and tell Python what we want to import. We then use the "as" keyword to call it something we can use later as shown in the example below.

Here is an example:

```
import deep3.util as util
```

One of the first things to get is the code to connect to minecraft. We can get this from the file system as `mcpi.minecraft` . So that we can use it, let's call it `minecraft` using the "`as`" keyword.

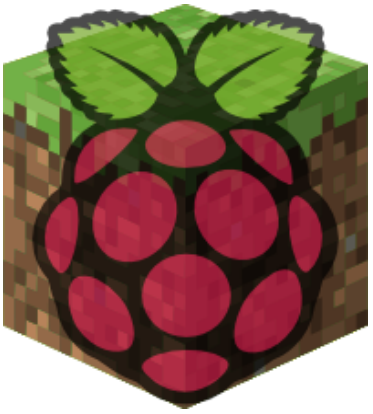
Step 1 : Write your first line of code to import minecraft into your script.

```
import mcpi.minecraft as minecraft
```

Step 2 : We need some more modules for use later. Add these imports to your file too.

```
import deep3.util as util
import mcpi.block as block
import mcpi.vec3 as vector
import time
```

1.2 Connect to the world



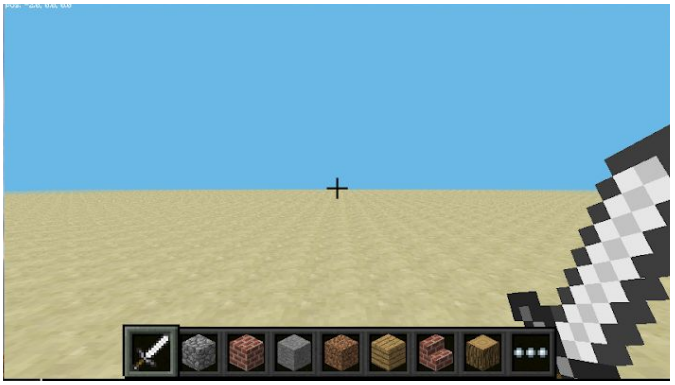
To make things happen inside the minecraft world, we need to connect to it. We can now get `minecraft` that we **import**ed earlier to do something. Make a new **variable** to use whenever we want to interact with the game.

Step 3 : Add the following line to your code underneath the import statements. Make sure to leave a few lines of space before starting.

```
minecraft = minecraft.Minecraft.create()
```

We are now ready to start changing the minecraft world!

Lesson 2: Magic Block



Now we have a lovely empty world, but wouldn't it be better with something in it?

Let's make something happen! The world is very flat and boring at the moment, so lets create a block.

2.1 First Block

We can change things in the game using the `minecraft` **variable**. There is a **method** for minecraft called '.setBlock()' which creates a block in the game. This **method** needs to be told some things before it can be used, which are the coordinates and the type of block you want to place. X position, Y position, Z position, Block Number. These are called the methods parameters.

If we use the minecraft variable, we can use it with a line like this:

```
    minecraft.setBlock(x, y, z, blockNumber)
```

Step 4 : Add the '.setBlock' command to your file using the minecraft variable you have already created. Replace x with 0, y with 1, z with 0 and blockNumber with 1.

Step 5 : Run your code! In your code editor press the green play button at the top. What happens? You should now see your first Block!

Now you have a nice stone block, lets see what else we can do.

2.2 Block Types

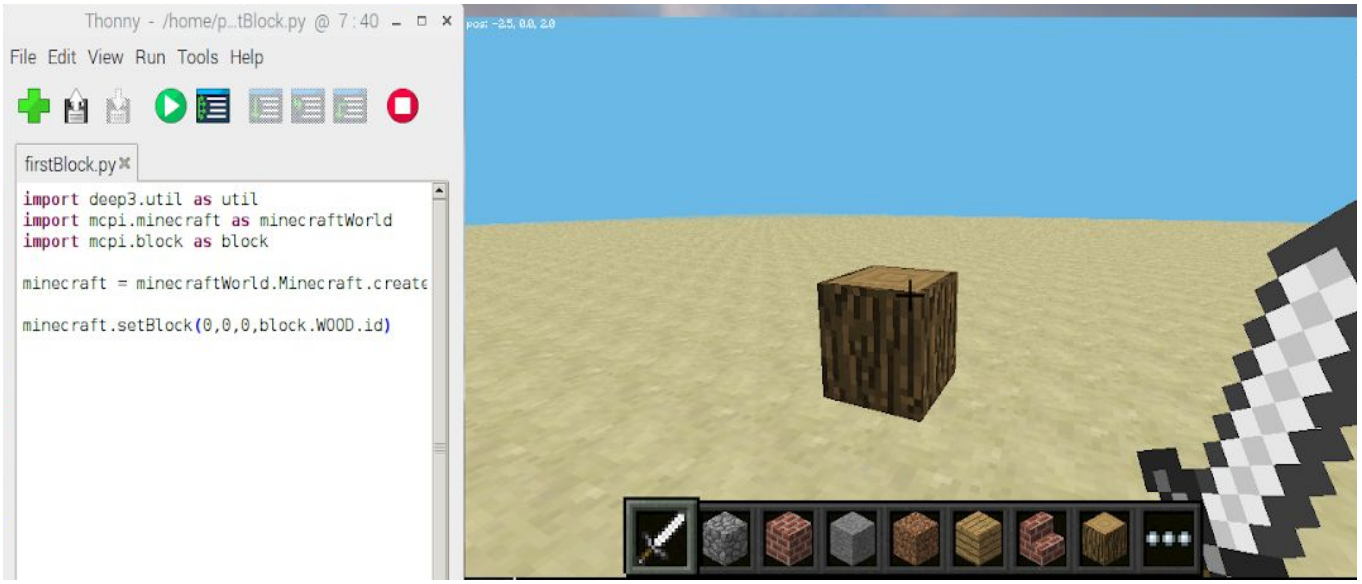
Why make a stone block when you can make diamonds! Each type of minecraft block has a number. We can use “mcpi.block” that we imported earlier to get all the different block texture options without having to remember the numbers. The “as” keyword after this allows us to call this module by simply using the word “block” (or the word we put after the “as”). This is done by asking for a blocks name and then getting its number by using .id on the end.

Here is what it looks like:

```
block.WOOD.id
```

Step 6 : Change the block type to wood and run the script again.

Once your done, you should have a world with your own little block in it shown in the image on the next page.

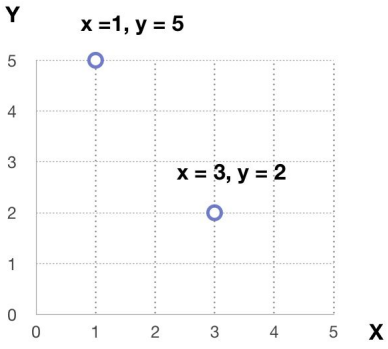


There are lots of blocks to choose from. Have a look at the Minecraft Python API document to see a full list of all the options.

Step 7 : Change the block to any other block type you like

Extension 1: Can you work out how to put more than 1 block in the world? We will learn more about this in the next lesson.

Lesson 3 : Geometry



To make sure we are making blocks in the positions we want, we need to understand coordinates .

coordinates are a way of marking points with numbers. In the graph on the left, you can see how an X and Y value mark different locations.

Minecraft does the same thing, but in 3D! This is all the same, but you also use Z.

- **Y** is how high and Deep
- **X** is East and West
- **Z** is like South and North

To help show how this works in the minecraft world, we have put in the Axis.

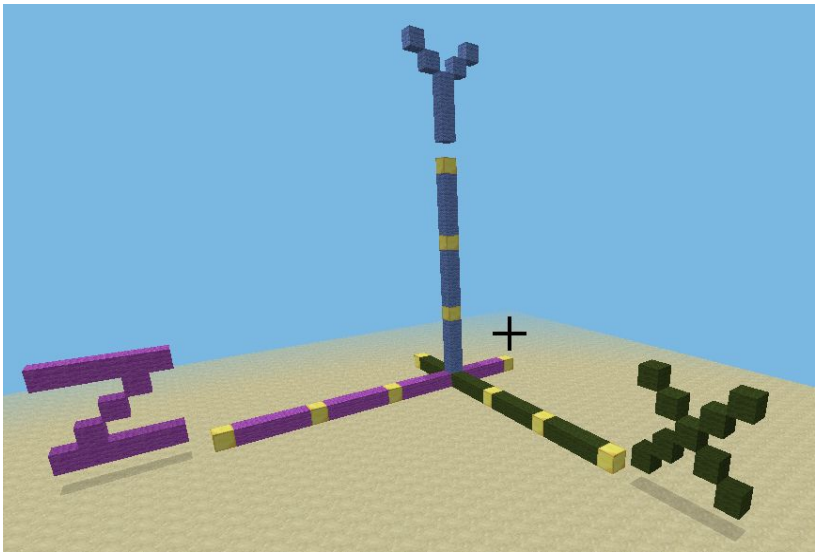
Step 1 : To show an axis in your world, use the following method from the utils class. Put it at the start just under your imports.

```
util.drawAxis()
```

Now when you run your script, you will see a large axis appear.

The Axis takes up the space of 0 in each direction. Be careful not to place blocks in the same location.

The notation of coordinates is (x, y, z). The 2D graph above would have 2 coordinates (1, 5) and (3, 2).



Step 2 : Make some blocks in the world with the axis. Look at one of the coordinates below, and guess where it is going to end up. Then set a block in that location in the same way as Lesson 1. Does it get created in the place you expected?

A. (10, 0, 10)
B. (10, 5, 10)
C. (-5, 0, 10)

Step 3 : Now you know how to place blocks, make a picture or sculpture. Start with something simple like a smiley face, our the letter of your first name.

Extension : If you make a block in location (10, -5, 10), where does it go?

Hint: If you want to turn the Axis off, just remove the drawAxis() line

Lesson 4 : Wall

In this lesson we are going to learn how to build a wall.

To do this, we are going to write two new methods. They will be called “spawnLine” and “spawnWall”.

We need to define the first method “spawnLine”, as we will reuse this in our “spawnWall” method. You’ll understand why later.

Step 1 : Let’s start by writing the signature for our new method “spawnLine”. This method will spawn us a line of bricks from a set position, in a certain direction, with a given length and the texture we want applied to the blocks. This will effectively give us a small wall, but only one block high, so for now it’s just a line of blocks.

To do this we need to use the “def” keyword.

Write the method signature out as shown below, underneath your previous code.

```
    def spawnLine(position,direction,length,blockId):
```

The “def” keyword tells Python that we are defining a new method (aka, a function).

We can then call a method whatever we like, but the name should ALWAYS be relevant to what the method is doing, so that other people who read or want to use our code can have a good idea what it is doing. In this instance, as we’ve said before, we are creating a line of blocks, so we’ve called it “spawnLine”.

Now inside the brackets “(“ or in development terms, the parentheses, we have what seems to be a list of words: position, direction, length, blockId. These are known as our method’s parameters. Parameters are the values that we must pass into our method in order for it to do its job properly.

Step 2 : Add the following lines of code underneath your signature:

```
    for i in range(1,length):
        time.sleep(0.1)
        position = util.addVectors(position,direction)
        minecraft.setBlock(position.x,position.y,position.z, blockId)
    return position
```

This adds the functionality of the method. You MUST make sure that there is a 4 space indent for the first line of code underneath the signature. These indents are needed as this is how Python works.

The code explained:

`for i in range(1,length): -` This line of code is known as a “for loop”. In this example, this line of code will cause the code to repeat every line of code **indented** underneath the for loop statement, starting at 1 and counting up until it reaches the given integer value of “length”. The letter i you can see represents an integer value to loop through the range of 1-length.

`time.sleep(0.1) -` This line of code will make the program slow down slightly making it easier for us to visualise the for loop in our minecraft world. We have told it to sleep for 100ms.

`position = util.addVectors(position,direction) -` This line of code adds the vectors of direction and position together in order to get the next position for the next block to draw. (You don’t need to worry too much about this for now).

`minecraft.setBlock(position.x,position.y,position.z, blockId)`
This line of code draws a new block (as we’ve learnt in previous lessons) using the new position that was made on the previous line.

This line of code will return us the final position created in the for-loop, so that we can use this value to do more work with it. This will be used later.

Your method should now look **exactly** like this:

```
    def spawnLine(position,direction,length,blockId):
        for i in range(1,length):
            time.sleep(0.1)
            position = util.addVectors(position,direction)
            minecraft.setBlock(position.x,position.y,position.z, blockId)
        return position
```

Step 3 : Go to the very bottom of your Python file and add the following line of code:

```
position = vector.Vec3(0,1,0)
```

Here we have declared a variable called “position” and set it to a vector. We’ll use this in a moment.

Step 4 : Underneath the line we wrote in step 3, declare a variable called “direction” and set it to a vector like we did in step 3 with the coordinates (1,0,0)

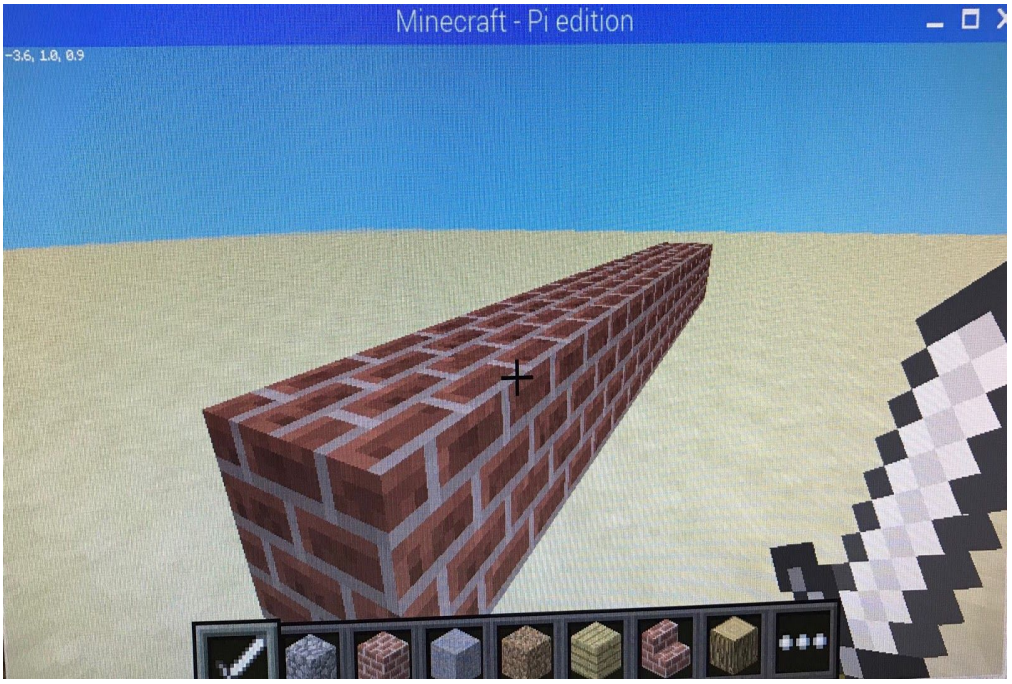
Step 5 : Underneath the line we wrote in step 4, declare a variable called “length” and set it to equal 10.

Step 6 : Now under the line from step 5, declare a variable called “blockId” and set it to a block type id of your choice using the API sheet provided. HINT: use the keyword “block” to access these.

Step 7 : Lets now call the function we’ve created using all the variables we’ve just declared. Underneath our new variables, call the spawnLine method and pass in the variables we’ve just written. HINT it's almost like defining the “spawnLine” method again except we **don’t** need the “def” keyword or the colon at the end.

Step 8 : Run the program and see the line of blocks get created in the minecraft world!

You should now see a line of blocks like so:



YOU SHOULD NOW DELETE THE CALL YOU WROTE IN STEP 7.

Our line of blocks is essentially a small wall with a height of only 1 block. So now we'll create a new method to spawn a proper wall, that is much higher.

Step 9 : Create a method called "spawnWall" that takes the **same** parameters as our spawnLine method with **one additional** parameter called "height".

Step 10 : Now inside the method, create variable called **playerPos** and set it to equal **position**

Step 11 : Now underneath the this line of code, **create** a variable called **holdPos** to equal the following:

```
vector.Vec3(
            minecraft.player.getPos().x,
            minecraft.player.getPos().y,
            minecraft.player.getPos().z
           )
```

Step 12 : Under this line create a new for loop that iterates from 0 to the value of height. Have a go at trying to write this for loop statement on your own.

HINT: have a look at the for loop inside the spawnLine function and try to think how yours will be different.

Step 13 : Now **inside the for loop** make a **call** to our **spawnLine** function passing it the parameters it needs.

HINT use the playerPos, direction, length and blockId as the parameters. **Remember** we need to indent by 4 spaces to put code inside a for loop.

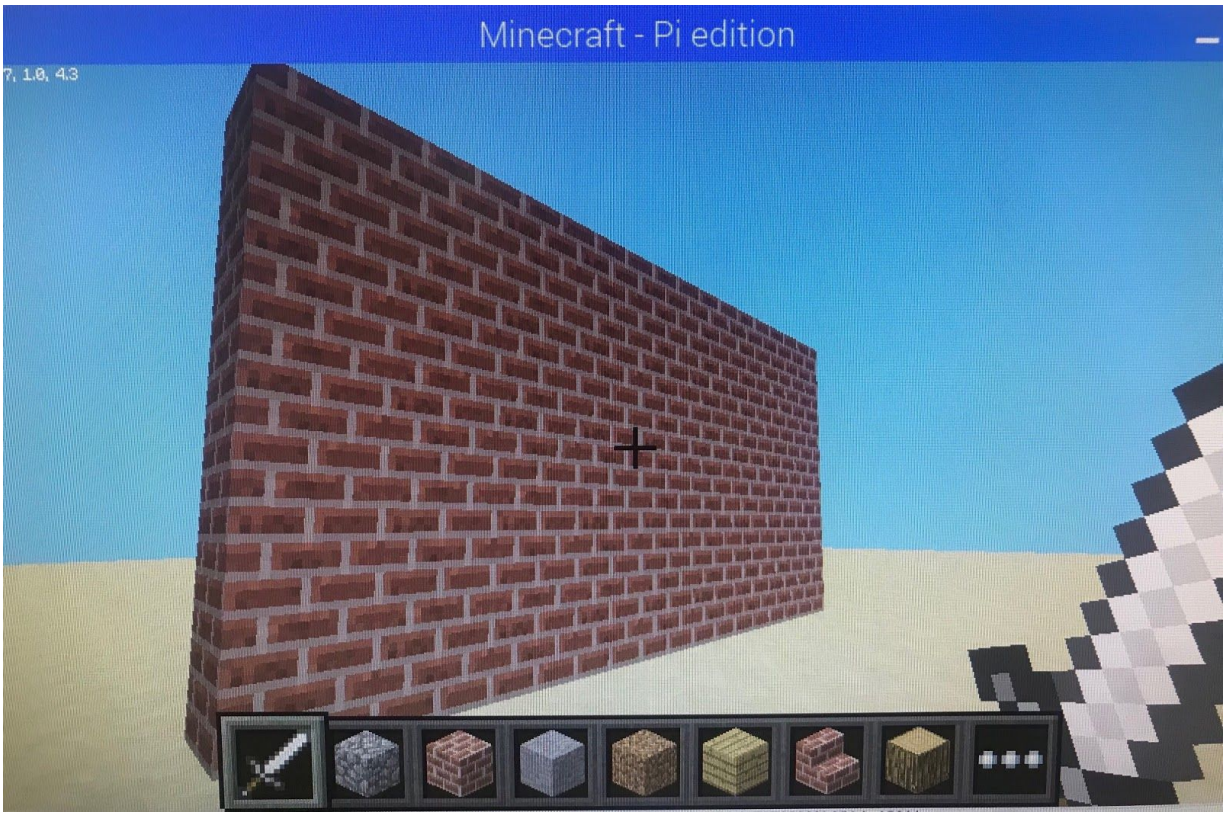
Step 14 : For the final few lines of code in our for loop set the "y" values of the playerPos variable to "+= 1", then the value of its "x" variable to equal holdPos.x and the value of its "z" to holdPos.z.

Your spawnWall method should look like below:

```
    def spawnWall(position, direction, length, blockId, height):
        # get the players position
        playerPos = position
        # set a holding variable to deal with reference values
        # IMPORTANT this should all be on one line
        holdPos = vector.Vec3(minecraft.player.getPos().x,
    minecraft.player.getPos().y, minecraft.player.getPos().z)
            # allows a pupil to give the height
            for j in range (0, height):
                spawnLine(playerPos,direction,length,blockId)
                # adds the height on to the next layer of the wall
                playerPos.y += 1
                playerPos.x = holdPos.x
                playerPos.z = holdPos.z
```

Step 13 : Now call your new method by declaring a height integer next to your other variables near the bottom of your python file and by replacing your spawnLine method call with the spawnWall method. Set position to the players position, the direction to a new vector with x = 1, y = 0 and z = 0. Use whatever blockId you like.

Step 14 : Run the code and you should see a wall being built!



Lesson 4 : House

In this lesson we are going to build a house. For now we're not going to worry about windows and doors. We'll create a house with 4 walls and a flat roof.

Step 1 : Define a method signature called spawnWalls that takes the following parameters:

- position
- direction
- height
- length

Step 2 : Create a for loop with a range of 0 to our height variable, using a variable "j" as the iterator. This will allow us to set the height of the wall.

HINT: Look at a for loop you've written before. All you need to do is replace the variables with the ones mentioned above.

Step 3 : Immediately create another for loop within the one we created above, but this time use a variable called 'i' as the iterator. Set the range from 0 to 4. We are doing this because, there's going to be 4 walls that make the outside of our house.

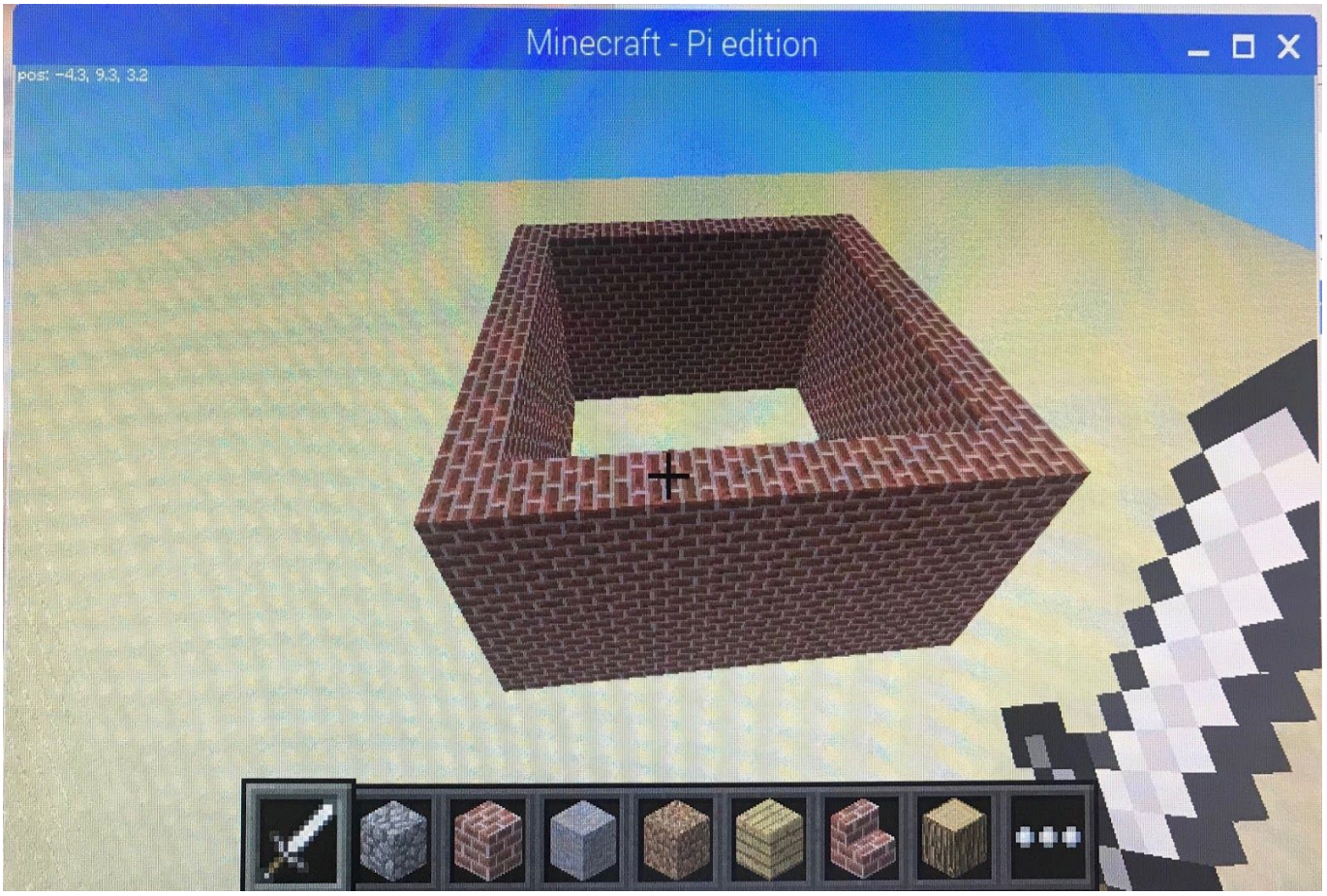
Step 4 : Within the for loop created above, set a variable called position to the result of calling our spawnLine method we created earlier. Set the parameters to position, direction, length and pass the variable we need for blockId as block.BRICK_BLOCK.id.

Step 5 : On the next line set a variable called direction and set it to the have the following value:

```
util.turnVectorClockwise(direction)
```

Step 6 : On the next line, outside of the second for loop, but inside the first (so this line of code should be indented inline with the second for loop) set the value of the position's y value to += 1.

Step 7 : Make a call to our new function by creating some suitable variables or using the old ones. Set the blockId to block.BRICK_BLOCK.id and then run the program. We should see the square of 4 walls! Our house is nearly complete. Now all we need is a roof!



Ok, let's create a really simple flat roof and then a final method to combine both of our new methods that can create all 4 walls and the roof in one method call! **NOTE** You should now delete any previous method calls before continuing.

Step 8 : Create another method signature called spawnRoof that takes the parameters position and length.

Step 9 : Now create a for loop that iterates from the range from 0 to length and inside that for loop, write the following code:

```
spawnLine(vec3.Vec3(position.x+i,position.y,position.z-1), vec3.Vec3(0,0,1), 11,
block.WOOD_PLANKS.id)
```

Step 10 : Create a new method called spawnHouse that takes the parameters position, height and length.

Step 11 : Inside our new method call your spawnWalls method and give it the parameters that it needs.

Step 12 : Below the above call create a new direction variable and set it to the following value:

```
vec3.Vec3(1,0,0)
```

Step 13 : Then below that call the spawnRoof function with the parameters it needs. Give the position value a new vector where:

- x = position.x
- y = position.y + height
- z = position.z

HINT: Use the example in Step 12 if you need assistance.

Give the direction value, the value of your variable created in step 12.

Set the length value to length +1.

Finally set the block id to blockid.

Step 14 : Try and call the new function using the things you've learnt from this lesson and previous lessons.

Voilå, there you have it! When you run your code, you should now see a house slowly being built.

The legal bit

This lesson and associated code is under the Apache 2.0 license. This license was picked for teachers and education, so you can copy, distribute, and modify it without worry.