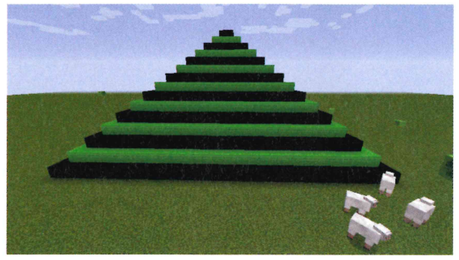
Create 3D art pieces in Minecraft using Python loops

Taking our Minecraft pixel art into the third dimension, this time we'll build pyramids with Python loops

Calvin Robinson is head of computing and network manager at an all-through state school. Specialising in computer science, Calvin also consults with schools all over London.

Resources

- **Minecraft** www.mojang.com/games
- **Python** www.python.org
- **McPiFoMo** http://rogerthat.co.uk/McPiFoMo.rar
- **Block IDs:** http://bit.ly/MinecraftIDList



Last issue, we started our series on Minecraft pixel art, by coding some 2D art blocks. This time we're taking a different approach to our art, by implementing an additional axis and therefore bringing it into the third dimension. Well, technically, everything is 3D in Minecraft, but last issue we built some rather convincing 'flat' pixel art. Now we're taking it to the next level.

We're going to build a pyramid using `while` and `for` loops in Python. This will save us typing lots of similar lines of code. Spawning our coded creations is so much faster than placing each individual block manually in Minecraft's Creative mode.

If you're using Minecraft Pi edition on a Raspberry Pi, no additional software is necessary. We've also put together a number of tools to ensure this hack works on Linux, with a retail version of Minecraft. Therefore as a prerequisite, we assume you've installed McPiFoMo from our previous three issues. McPiFoMo includes MCPiPy by 'fleap' and 'bluepillRabbit' of MCPiPy.com; and Raspberry Jam, developed by Alexander Pruss.

All Python scripts should be saved in the directory **~/home/.minecraft/mcpipy/**, regardless of whether you're running Minecraft Pi edition or retail Linux Minecraft. Be sure to run Minecraft with the Forge 1.8 profile that's included in our McPiFoMo package.

01 Starter code and variables

Here's the first block of code to start with.

```
from mcpi import minecraft
mc = minecraft.Minecraft.create()
pos = mc.player.getTilePos()
x = pos.x + 2
y = pos.y
z = pos.z
```

```
height = 29
count = 0
blockID = 24
blockType = 1
```

02 Starter pyramid

First, we're going to initiate a bunch of variables, collecting the player's standing position with `pos = mc.player.getTilePos()`, and breaking that down into x, y, z coordinates with `x = pos.x + 2`, `y = pos.y` and `z = pos.z`. We've also got variables for the height of our pyramid; we'll get to shortly. Below that we have `blockID` and `blockType`, which will affect the blocks used to create our pyramid. Block ID 24:1 would be broken down to blockID 24, blockType 1. We felt Chiseled Sandstone looked quite 'pyramid-y' to begin with.

03 Adding the loops

Let's create some `for` loops nested within a `while` loop.

```
while height - (2 * count) > 0:
    for block in range(height - (2 * count)):
        for row in range(height - (2 * count)):
            blockX = x + block + count
            blockY = y + count
            blockZ = z + row + count
            mc.setBlock(blockX, blockY, blockZ,
blockID, blockType)
    count += 1
```

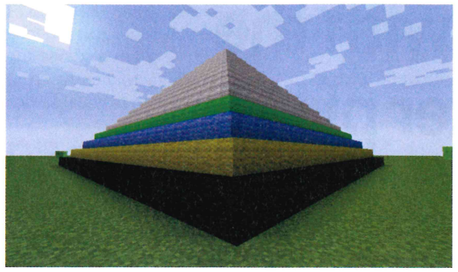
04 Customising our pyramid

We've already initialised variables for `height` and `count`. By changing the `height`, we can make our pyramid larger or smaller. The `count` variable is used in the loop, to make sure our pyramid stops on the number of rows we specified in height. We start at 0 and increment upwards with each cycle of the loop.

If we wrap our `setBlock` command in conditional (`if ... else`) statements, we can alternate rows, using the `count` variable to see if the row is an even number or an odd. We can place a different `blockID` on each row.

05 Wrap with conditional statements

Replace the current `mc.setBlock(blockX, blockY, blockZ, blockID, blockType)` line with the following code:



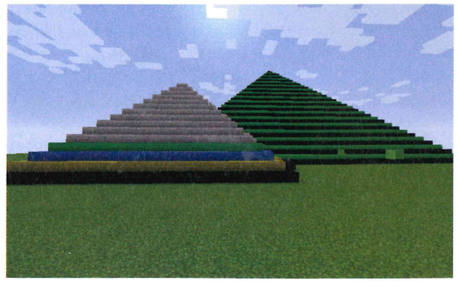
```
if count % 2 == 0:
    mc.setBlock(blockX, blockY, blockZ,
woolBlockBlack, woolBlockBlackType)
else:
    mc.setBlock(blockX, blockY, blockZ,
woolBlockGreen, woolBlockGreenType)
```

It just so happens we're using different coloured wool blocks. You might want to name your variables differently.

06 Alternating colours on our pyramid

Rather than setting the colour based on odds/evens, we can get more creative and make each row a different colour. Alter the `if` statement to include `elif`s for each row that you'd like to colour, but don't forget to initialise `blockID`/`blockType` variables:

```
woolBlock = 35
woolBlockWhiteType = 0
woolBlockGreenType = 5
```



07 Coding multicoloured lines

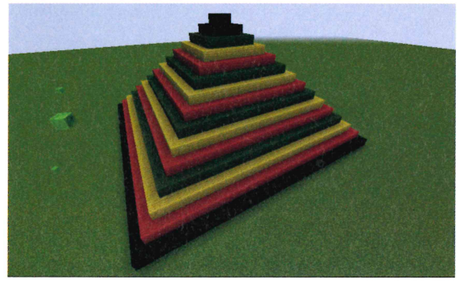
Now let's build layers upon layers.

```
if count == 0:
    mc.setBlock(blockX, blockY, blockZ,
woolBlock, woolBlockBlackType)
elif count == 1:
    mc.setBlock(blockX, blockY, blockZ,
woolBlock, woolBlockYellowType)
elif count == 2:
    mc.setBlock(blockX, blockY, blockZ,
woolBlock, woolBlockBlueType)
elif count == 3:
    mc.setBlock(blockX, blockY, blockZ,
woolBlock, woolBlockGreenType)
else:
    mc.setBlock(blockX, blockY, blockZ,
woolBlock, woolBlockWhiteType)
```

08 Variables for the above code

Having a fancy `if` statement is great, but don't forget the variables:

```
woolBlock = 35
woolBlockGreenType = 5
woolBlockBlackType = 15
woolBlockYellowType = 4
woolBlockBlueType = 3
woolBlockWhiteType = 0
```



09 Egyptian influence

We should now be in a position to alter the height of our pyramids and the types of block used for each row, or to alternate the colour of rows depending on what suits our needs. Now is the time to get really creative and put that together to produce something original.

We'd love to see what you come up with, so please do tweet us a screenshot of your pyramids and accompanying Python code to **@linuxusermag**.

You could also try reversing the `for` loops, to create an inverted pyramid. ■

Creative coding

Minecraft provides a space for gamers to be creative. Using the tools provided in this series of tutorials, we can hook directly into Minecraft with Python scripts, allowing us even more control over our virtual environment. Instead of individually placing blocks around our world, we can write a simple Python script to spawn them. This also means we can do far more complex calculations than we'd ever be able to do in our heads, to spawn creations that boggle the mind.

With each issue of **LU&D** we take a deeper look into coding Python for Minecraft, with the aims of both improving our Python programming skills and also gaining a better understanding of what goes on under the hood of everyone's favourite voxel game.