Raspberry Pi®

CoderDojo Belfast

Python Pi Project #2: House

Level: Middling

In this project we're going to do a little house building in Minecraft. It's not a fancy house – it just has four walls and a door. Now, one wall is pretty much the same as another wall – it's just a big rectangular pile of blocks. But it would be nice if we can write the program to build the house without having to write the code to build a wall four times – once for each wall. We can indeed avoid doing this, if we use *functions*.

Coding skills needed

- defining a function:

```
def wall(p1, p2) :
    plane(p1, p2, block.STONE.id)
```

- data input:

```
width = raw_input("width: ")
```

...

- A little bit of maths!

Vector arithmetic

Pretty much every programming language has some sort of idea of functions in it. A function is basically a way of bundling some code together in a way that it can be re-used, so you only have to write it once, but can use it many times, from different places in your program, or from different programs altogether.

A quick look at functions

Here's what making a function looks like in Python:

```
def wall(p1, p2) :
        """Build a stone wall from p1 to p2"""
        plane(p1, p2, block.STONE.id)
```

'def' is short for 'define'. The function has a *name*, "wall". Then we define a couple of inputs, known as *parameters*, to the function, 'p1' and 'p2'. Some functions have parameters, others don't need them. Then, like a lot else in Python, we add a colon and indent the following lines, which are the *body* of the function, and which say what it does.

Our example above starts with a comment explaining what the function does. Then the only other thing it does is call another function!

One of the things you can put in a function is a 'return' statement, which says what the the *result* of the function is (its *output*). You don't have to have a return if you don't need one. Here's one that does -

```
def add(x, y) :
        """adds x and y!"""
        return x + y
```

So how do you use these? Just write the name of the function, and then the input

values, the *arguments*, in parentheses:

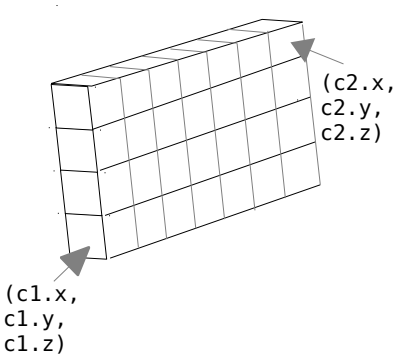
```
wall(groundNE, roofSE) # call the 'wall' function
```

If the function has a return value you can assign the return value to a variable:

```
sum = add(2, 3)                    # sum is now 5
```

How to build a house

Let's think about what functions we'll want. The idea is to build a wall using Minecraft's 'setBlocks' function, which lets us build a pile of blocks between two corners, whose co-ordinates we have to supply. If we keep the corners in the same straight line, N/S or E/W, the pile will be just one block thick, and will do nicely as a wall.



We could build four walls by working out the corners of each one (let's call them a, b, c... for short) and writing

```
mc.setBlocks(a.x, a.y, a.z, b.x, b.y, b.z, block.STONE.id)
mc.setBlocks(c.x, c.y, c.z, d.x, d.y, d.z, block.STONE.id)
mc.setBlocks(e.x, e.y, e.z, f.x, f.y, f.z, block.STONE.id)
mc.setBlocks(g.x, g.y, g.z, h.x, h.y, h.z, block.STONE.id)
```

but that already looks a bit yucky. And what about when we want a window or a door? We don't want to have to write even more lines like that, only with glass and wood instead of stone.

So what about if we have a function that we can pass two positions to, and that will do the setBlocks for us?

```
def wall_v1(c1, c2):
 mc.setBlocks(c1.x, c1.y, c1.z, c2.x, c2.y, c2.z, block.STONE.id)
```

Then the wall building becomes

```
wall_v1(a, b)
wall_v1(c, d)
wall_v1(e, f)
wall_v1(g, h)
```

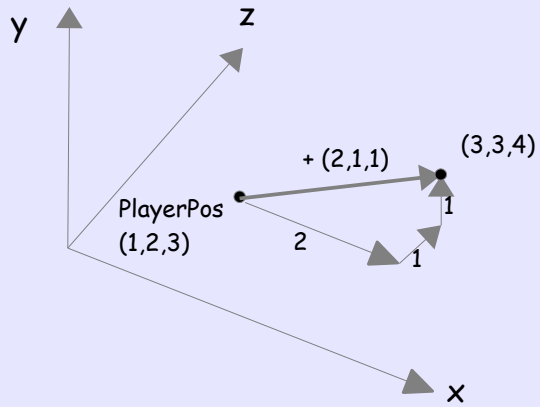
which already looks a lot nicer.

That already shows how functions can make things cleaner by re-using code. But for our house we'll go one better by deciding that a wall, a window and a door are all just examples of 'planes' (that's flat land, not aero!) made of different stuff. So we'll make our function that does setBlocks create a 'plane' of some building material. We'll tell it what material by supplying the 'id' of the material (glass, wood...) as another parameter to the function along with the co-ordinates. See “The complete program” below.

We'll use a bit of maths - “vectors” - to work out the positions of the corners. Don't worry if you've not done vectors yet, it's not complicated – see the side bar below for details.

Adding Vectors

A player position in Minecraft is not just a position, it's a vector – which just means you can get from it to another position by adding on another vector. This is just a way of adding the x, y and z co-ordinates all at once:



So if you're at (1,2,3) and add (2,1,1) you add the xs, ys and zs to get to (3,3,4).

The last new bit of Python that we'll use is a little data input, to ask the user for the dimensions of the house (how wide, deep, and tall it is).

```
width = raw_input("width: ")
```

This prints the word “width:” to the screen where you started the program (not on the Minecraft screen), and waits for you to type in a number, for how many blocks wide the house is.

Dojo Challenge:

Add a roof to the house. Can you add some windows to the walls?

The complete program

```
#!/usr/bin/env python
# A progam to build (some of) a house.  Uses functions to do the work.

import time
import sys

import mcpi.minecraft as minecraft
import mcpi.block as block
from mcpi.vec3 import Vec3

# a global variable - not normally recommended
mc = minecraft.Minecraft.create()

def plane(c1, c2, material):
        """Build from one corner to another using the material given"""
        mc.setBlocks(c1.x, c1.y, c1.z, c2.x, c2.y, c2.z, material)

def wall(p1, p2) :
        """Build a stone wall from p1 to p2"""
        plane(p1, p2, block.STONE.id)

def door(p1) :
        """Place a door (height 2 blocks) at the given position"""
        top = Vec3(0, 1, 0)
        plane(p1, p1 + top, block.DOOR_WOOD.id)

def house(location, east, south, height):
        """
        Build a house - 'location' becomes the north east corner,
        and the house is 'east' blocks wide toward the east, and
        'south' blocks long toward the south, and 'height' blocks high.
        There is a door in the middle of the north wall.
        """
        groundNE = location                                # ground north east
        roofSE = groundNE + Vec3(0, height, south)         # roof south east
        roofNW = groundNE + Vec3(east, height, 0)          # roof north west
        groundSW = groundNE + Vec3(east, 0, south)         # ground south west
        wall(groundNE, roofSE)                             # build east wall
        wall(groundNE, roofNW)                             # build north wall
        wall(roofSE, groundSW)                             # build south wall
        wall(roofNW, groundSW)                             # build west wall
        northCentre = Vec3(int(east/2), 0, 0 )             # half way 'east'
        door(groundNE + northCentre)                 # add door in north wall


# Build a house just south of the player.
# Uses input function to ask for the dimensions of the house
corner = mc.player.getTilePos() + Vec3(0, 0, 3)
width = raw_input("width: ")
depth = raw_input("depth: ")
height = raw_input("height: ")
house(corner, int(width), int(depth), int(height))
```