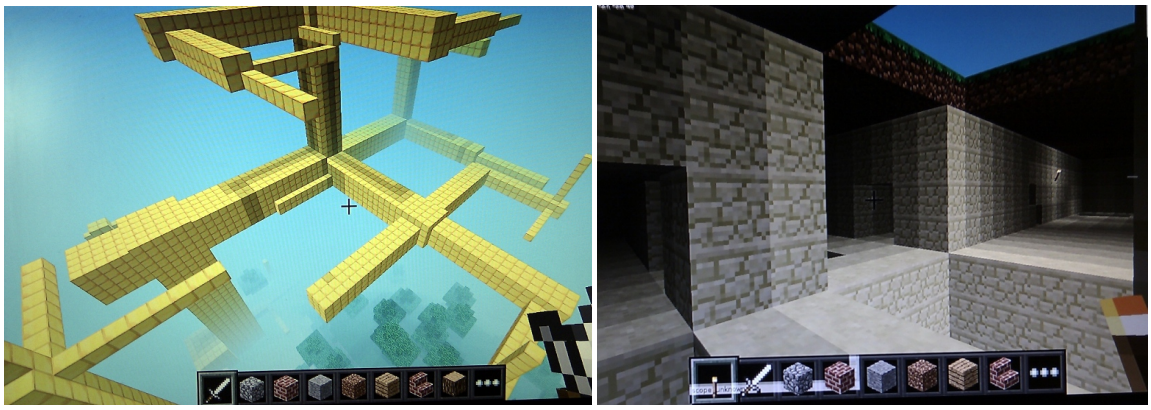
Huge skyscrapers and dungeons



Before you start:

- Launch Minecraft and create a new world.
- Launch Geany

Step 1 Make a function that calls itself

- In Geany, from the file menu select "new", and then save as `skyscraper.py` in the "pi" directory.
- We need to import the dBsCode commands we'll be using and clear an area in the Minecraft world for working in. Create this program:

```
from dbscode_minecraft import *
bulldoze()
```

- **Test** Press F5 to run the program (this will also save your program for you). After a few seconds you should see a "flatworld" type of environment.
- In this project, we're going to see how "recursion" can be used to make very complex structures without very much code. Lets start with a row of blocks.

```
def row(pos,count):
    if count>0:
        box(SANDSTONE,pos,point(1,1,1))
        row(pos+point(3,0,0),count-1)
```

This function is a bit like a for loop, in that it will repeat the box command as many times as count is set to at the start. If we pass 10 in, it checks it with the if, draws a cube and calls itself at the end with the count as 9 - and so on until it reaches 0, when it's finished (the "if" only runs it's following code running if count is greater than 0).

Let's try it:

```
    row(point(0,0,0),5)
```

You should see 10 blocks. If the program seems to be running forever, click in the section at the bottom of the geany window and press "ctrl" and "c" to stop it and check the code.

Step 2 Make a tree

We can use the same principle to do more interesting things. Change your `row` function into a tree one:

```
def tree(pos,count):
    if count>0:
        box(SANDSTONE,pos,point(1,count,1))
        tree(pos+point(-1,count,0),count-1)
        tree(pos+point(1,count,0),count-1)

tree(point(0,0,0),5)
```

Here we are calling ourselves twice at the end, you can think of these like branches first moving to the left then the right, with -1 and 1 being added to the X position in the tree call. We're also using the current count to change how big the box is as it goes down the tree.

Step 3 Test out the third dimension

As well as moving -1 and 1 in the X, lets add it to the Y as well - add this inside the the tree function, after the 2 others:

```
        tree(pos+point(0,count,-1),count-1)
        tree(pos+point(0,count,1),count-1)
```

Try increasing the 1 and -1's in the points to 2 and -2 - this will explode the tree - your function should now look like this:

```
    def tree(pos,count):
        if count>0:
            box(SANDSTONE,pos,point(1,count,1))
            tree(pos+point(-2,count,0),count-1)
            tree(pos+point(2,count,0),count-1)
            tree(pos+point(0,count,-2),count-1)
            tree(pos+point(0,count,2),count-1)
```

- Try change the amount it explodes, and making different for different dimensions.

Step 4 Adding direction and randomness

Let's briefly go back to 2 dimensions and try adding direction (a `d` parameter), this is needed so we can make longer branches that point in different directions. We'll add `d` to the `pos` in the tree call and to the size of the box too:

```
    def tree(pos,d,count):
        if count>0:
            box(SANDSTONE,pos,d+point(1,1,1))
```

```
            tree(pos+d,point(-2,0,0),count-1)
            tree(pos+d,point(2,0,0),count-1)
            tree(pos+d,point(0,2,0),count-1)

    tree(point(0,0,0),point(0,1,0),5)
```

Try running this, it doesn't look too interesting - the problem is that there are far too many branches being drawn all on top of each other. We can remove some of them using a random chance, let's write a quick function to make this easier:

```
    def chance(percent):
        return random_range(0,100)<percent
```

This returns `True` if the parameter is less than a random number between 0 and 100. 50 will result in a 50% chance. We'll use it with `if` to give out tree a 50% chance of branching in any direction:

```
    def tree(pos,d,count):
        if count>0:
            box(SANDSTONE,pos,d+point(1,1,1))
            if chance(50): tree(pos+d,point(-2*count,0,0),count-1)
            if chance(50): tree(pos+d,point(2*count,0,0),count-1)
            if chance(50): tree(pos+d,point(0,2*count,0),count-1)
```

This will be different each time you run the program, so try it a few times as some of the trees will be better than others.

Step 5 Going large!

Lets make it all much bigger and add a thickness to the branches.

- Put `count` in the box size.
- Change the 2 and -2's with 4 and -4 to make it go further.
- Add back the third dimension with two more branches.

```
def chance(percent):
    return random_range(0,100)<percent

def tree(pos,d,count):
    if count>0:
        box(SANDSTONE,pos,d+point(count,count,count))
        if chance(50): tree(pos+d,point(-4*count,0,0),count-1)
        if chance(50): tree(pos+d,point(4*count,0,0),count-1)
        if chance(50): tree(pos+d,point(0,0,-4*count),count-1)
        if chance(50): tree(pos+d,point(0,0,4*count),count-1)
        if chance(50): tree(pos+d,point(0,4*count,0),count-1)

tree(point(0,0,0),point(0,1,0),7)
```

Step 6 Convert it into a dungeon

Nearly all the shapes we've been making are made by placing blocks into the world, but it's just as easy to make shapes by taking them away. With 4 changes we can make this into a dungeon generator:

- Change `SANDSTONE` to `AIR`
- Change `point(0,4*count,0)` in the last tree branch to `point(0,-4*count,0)`. This will make it go down instead of up.
- Change 1 to -5 where the tree is first called.
- Add a huge box to clear the ground `box(SANDSTONE,point(-100,-101,-100),point(200,100,200))`

```
def chance(percent):
    return random_range(0,100)<percent

def tree(pos,d,count):
    if count>0:
        box(AIR,pos,d+point(count,count,count))
        if chance(50): tree(pos+d,point(-4*count,0,0),count-1)
        if chance(50): tree(pos+d,point(4*count,0,0),count-1)
        if chance(50): tree(pos+d,point(0,0,-4*count),count-1)
        if chance(50): tree(pos+d,point(0,0,4*count),count-1)
        if chance(50): tree(pos+d,point(0,-4*count,0),count-1)

box(SANDSTONE,point(-100,-101,-100),point(200,100,200))
tree(point(0,0,0),point(0,-5,0),5)
```

Challenges

- What happens if you remove the bulldoze or huge SANDSTONE box and repeatedly run the program?
- We haven't tried changing the block types - can you make the tree structure change material as it gets built?
- Can you add lights and decoration to the dungeon?
- Can you find a way to combine both above and below ground structures?