Raspberry Pi® CoderDojo Belfast

Python Pi Project #3: Magic Sword

Level: Middling

In Project 2 we saw how functions simplify things and let you re-use code from more than one place in your program. But suppose you want to use your function from more than one program?

Coding skills needed

- writing our own modules
- module test code

```
if __name__ == "__main__":
  handle_events(test_spell)
```

- event driven programming

```
hits = mc.events.pollBlockHits()
if hits:
        for hit in hits:
                spell(mc, hit)
```

No, you don't have to write it again and again in each different program. Python lets us put functions we want to re-use into a file, called a *module*. Then any program that wants some of those functions can *import* the module, which reads in the functions in it so you can use them.

Imports

We've already seen this in action in our programs so far when we use Minecraft code:

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

The “import mcpi.minecraft” means “read the file `minecraft.py` from the directory `mcpi`”, and call it module minecraft.

If we don't tell python differently it will force us to use the whole path to the module as the prefix for each of the functions in it. That way our 'create' call would look like

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

We use the “as minecraft” to tell python that we'll just call the module 'minecraft', so the line becomes just

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

A magic sword

Let's write a program to turn the sword in Minecraft into a magic sword, one that will do some piece of magic when you right-click it. What will it do? Well, it could be anything, we'll just write some python code to cast whatever spell we want.

What we don't want to have to do is re-write the 'magic-sword' stuff every time we want a new spell. So let's put the 'magic-sword' code into its own module. Then every program we write that needs a magic sword (and, let's face it, they're often very handy!) can import the module and use it with whatever 'magic spell' code it needs to.

To make the module all we do is create a separate file to put the code in. The name of the file becomes the name of

the module. We'll call ours `sword.py`.

Our module is just going to have one function in it, handle_events.

```
def handle_events(spell, host="localhost",  mc=None):
    ...
```

This is going to take advantage of something that Minecraft provides, which it calls 'Block Hits'. Whenever you hit some-thing with a sword (with right-click), Minecraft remembers this event. In a Python program you can ask Minecraft to give you these 'hits' with its `pollBlockHits` function, and you can keep doing that in a loop using '`for`' as long as you like. Every time you get one of these 'hits' you can do something. This way of doing things is known as 'event driven programming' and it's very common in real world programs.

Each 'hit' contains a `pos` property, which is its position (with x, y and z co-ordinates), as well as a `face` that says which face of the block you hit (top, bottom, north etc.)

The clever bit of our module is that it will leave it up to whoever's using it to decide what to do with each block hit. The first parameter to `handle_events` is called `spell`, and it's going to be a function that our module can call on the hits. We'll insist that the function takes two parameters, one for our general Minecraft variable, 'mc', and one for the hit.

```
for hit in hits:
    spell(mc, hit)
```

Now someone can import the module and call the handle_events function, passing in a function to do whatever spell they want.

```
handle_events(build_tower_beside_me)
```

Hang on, shouldn't there be three arguments in there?

Well, yes, handle_events does have three arguments, which let you supply your own minecraft object ('mc') or the address ('host') of a different machine where your Minecraft is running. But two of them have default values – so you don't need to pass them in if you don't want to. If you leave them out, the function assumes they have the values given in the definition. If you do want to supply one or other – say you've already connected to Minecraft so you have your own Minecraft object, '`my_mc`' - you can do so like this:

```
handle_events(build_tower_beside_me, mc=my_mc)
```

Testing the module.

I lied earlier when I said the module has just one function. It actually does have a small function to let us test the module without having another program that imports it. When you run the module on its own as a program, Python gives it a name (in the "`__name__`" variable) of "`__main__`". So most modules have a bit of code in them that looks like this:

```
if __name__ == "__main__":
        # do some test stuff in here...
```

A Tower Building Spell

Our example spell, in `sword_tower.py`, is pretty basic – it just builds a small pile of bricks beside the player.

Dojo Challenge:

Can you conjure up a spell that does something more interesting than just build a tower?

sword.py: The sword module

```
#!/usr/bin/env python

import mcpi.minecraft as minecraft
import mcpi.block as block

# A 'Magic Sword' module.
# This defines a function 'handle_events' that will continually poll
# Minecraft for block hits (right mouse button) with the sword, and
# do some spell on each block hit received.  The spell gets passed
# in to the function.

def handle_events(spell, host="localhost",  mc=None):
        """
        Does the 'spell' given to it on every block hit.
        The spell is a function such as 'test_spell' below, which
        must have two arguments, one for the minecraft object and one
        for the block 'hit'.
        If you don't pass in 'mc', this will create it, using the 'host'
        parameter, which you can also pass in.
        """
        if not mc :
                mc = minecraft.Minecraft.create(host)
                mc.postToChat("Magic Sword on")
        try:
                while True:
                        hits = mc.events.pollBlockHits()
                        if hits:
                                for hit in hits:
                                        spell(mc, hit)
        except KeyboardInterrupt:
                mc.postToChat("Magic Sword off")
                print("Magic Sword off")


def test_spell(mc, hit):
        print("hit: " + str(hit.pos) + str(hit.face))

if __name__ == "__main__":
        handle_events(test_spell)
```

sword_tower.py: A program using the sword module

```
#!/usr/bin/env python
import mcpi.block as block

# import our module!
from sword import handle_events

# Uses the 'sword' module, and builds a small tower of blocks
# on each sword hit, a couple of paces to the east of the player.

def build_tower(mc, pos, block) :
        mc.setBlock(pos.x+2, pos.y, pos.z, block)
        mc.setBlock(pos.x+2, pos.y+1, pos.z, block)
        mc.setBlock(pos.x+2, pos.y+2, pos.z, block)

def build_tower_beside_me(mc, hit):
        print("hit at: " + str(hit.pos) + ", face: " + str(hit.face))
        mc.postToChat("Shazam!")
        pos = mc.player.getPos()
        build_tower(mc, pos, block.STONE)

handle_events(build_tower_beside_me)
```