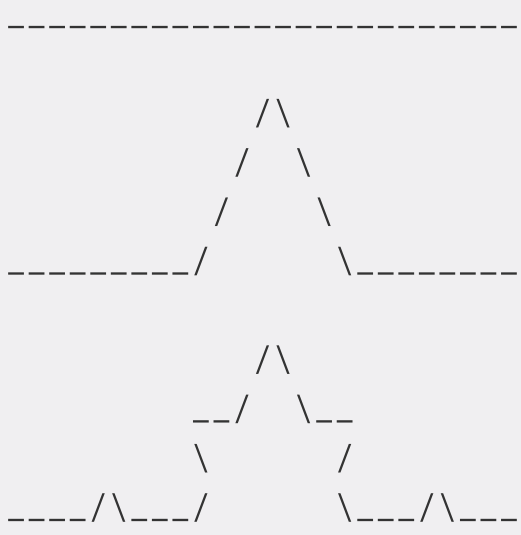
04—Turtles all the way down

STEP 1: DRAWING A MOUNTAIN

Pick up your turtles, open IDLE, it's time to draw again. But first, look at the following three shapes:
how would we draw them in python?



1. Drawing The first is easy. Like we always do, the first line of the file is always `from turtle import *` so Python knows we want to draw. Copy the program into a new file

```
from turtle import *

def first():
    forward(30)

first()
```

I've put it in a function because I like functions. We're going to put all the shapes in functions. The second one looks easy too, let's add it to the file we're writing.

2. Edit your code so it looks like the following:

```
from turtle import *

def first():
    forward(30)

def second():
    forward(30)
    left(60)
    forward(30)
    right(120)
    forward(30)
    left(60)
    forward(30)

second()
```

3. Run the code and see what it does, does it draw the right shape?. It should look like this

```
           /\
          /  \
         /    \
_________/      \________
```

STEP 2: A MOUNTAIN FROM MOUNTAINS

But how about the third shape? Although it's rather involved if we wrote out all the steps, it's actually drawing the second mountain, four times.

```
           /\
        __/  \__
        \      /
____/\___/      \___/\___
```

You can see we draw the second shape, turn, draw it again, turn, draw it again, turn and draw it one last time.

1. Instead of writing all of the smaller movements, let's draw it by calling `second`

```
from turtle import *

def second():
    forward(30)
    left(60)
    forward(30)
    right(120)
    forward(30)
    left(60)
    forward(30)

def third():
    second()
    left(60)
    second()
    right(120)
    second()
    left(60)
    second()

third()
```

`third` looks very much like `second`, but instead of calling `forward`, we're calling `second`. If we wanted to, we could even write the fourth one, and call `third` instead. This seems like an awful lot of work still, surely we can get the computer to understand what we're doing.

This shape is a special shape, which you draw by drawing it over and over: The third is made up of the second, the second is made up of the first. What we really want to do is tell the computer to keep drawing it over and over again until it is done.

STEP 3: OVER AND OVER.

We do this by splitting up the problem into two: The simple case, and the special case. The simple case is easy: it's just forward(100). The special case is a little harder, it needs to say 'Do the special case, but one less, until you get to the simple case'. It's easier to look at the code.

1. Open a new file and write in the following code:

```
from turtle import *

def mountain(depth):
    if depth == 1:
            forward(10)
    else:
        newdepth = depth -1
        mountain(newdepth)
        left(60)
        mountain(newdepth)
        right(120)
        mountain(newdepth)
        left(60)
        mountain(newdepth)

mountain(3)
```

   We can see that we've used code very similar to `first`, `second` and `third`. We use an `if` to work out if we should draw the simple case, or the special one. In the special case, we ask to draw a mountain, like how `third` called `second`, but we ask it to draw a simpler one each time, at a new depth, one less than what we started with.

2. Run it and see what happens. What happens if you try `mountain(1)`, `mountain(2)`, or `mountain(4)`?

STEP 4: DRAWING A SNOWFLAKE FROM THREE MOUNTAINS

1. Let's just add one last thing to the mountain file, change it to look like the following, adding a new function

`snowflake`:

```
from turtle import *

def mountain(depth, length):
    if depth == 1:
            forward(length)
    else:
        newdepth = depth -1
        mountain(newdepth, length)
        left(60)
        mountain(newdepth, length)
        right(120)
        mountain(newdepth, length)
        left(60)
        mountain(newdepth, length)

def snowflake(depth, length):
    mountain(depth, length)
    right(120)
    mountain(depth, length)
    right(120)
    mountain(depth, length)
    right(120)

snowflake(4,5)
```

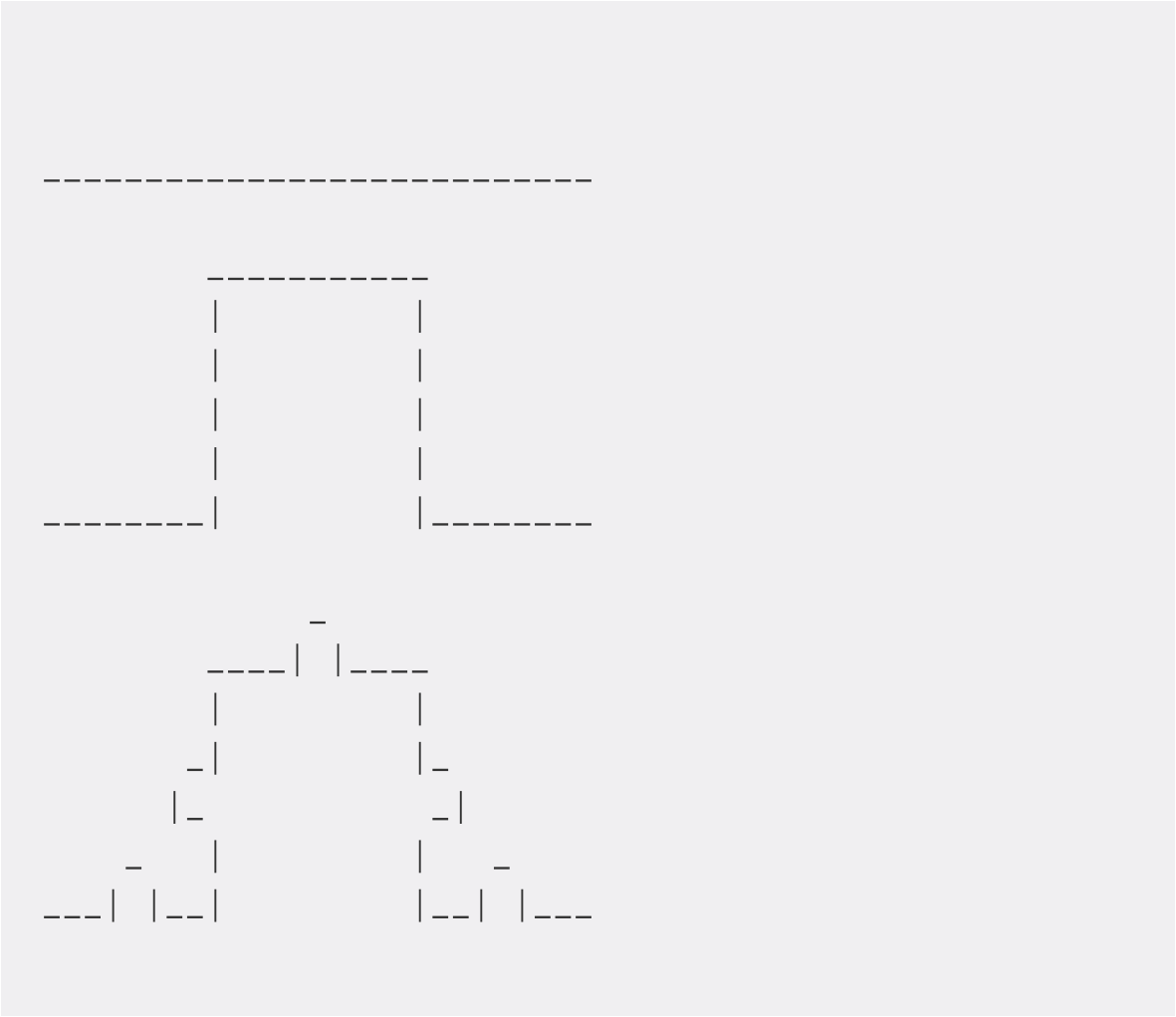
This picture is known as the Koch Snowflake. If you like, try playing with the angles and seeing what happens.

This is called a fractal, because the larger picture is made up of smaller versions of itself.

2. Try running `snowflake(1, 50)`, `snowflake(2, 25)`, `snowflake(3, 20)`. The more depth, the longer it takes to draw, so remember to put `speed(11)` in to make the turtle rush!

STEP 5: BOXES, MORE BOXES, EVEN MORE BOXES.

Let's look at another shape, that is very much like the snowflake, but with boxes instead of mountains.



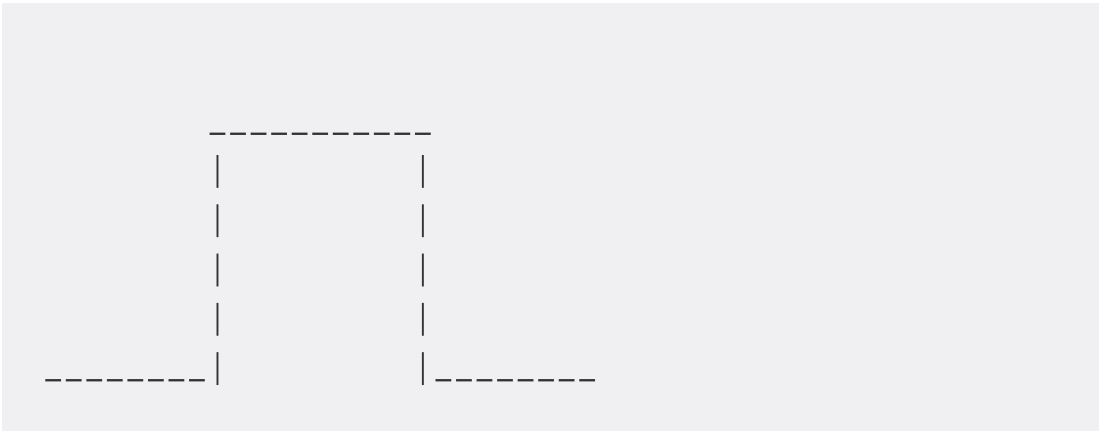
Like before with the mountain, we have a simple case: a straight line, and a special case: draw a line with a square bump in it. We can see the third one is just like before, drawing the second one a few times.

1. Let's open a new file and try to draw the second picture, which is what we'll be repeating:

```
from turtle import *

forward(30)
left(90)
forward(30)
right(90)
forward(30)
right(90)
forward(30)
left(90)
forward(30)
```

2. Run it, and you should get this:



We have the simple case `forward(100)`, and we know how to draw the bumpy line, so let's skip straight to the drawing!

STEP 6: BUMPY BOXES

1. Open up a new file in IDLE and copy the following code into it:

```
from turtle import *

def box(depth, length):
    if depth == 1:
            forward(length)
    else:
        newdepth = depth -1

        # What should go here?
        # Copy the bumpy line code here, but
        # use box(newdepth, length) instead of forward(100)
        # Ask for help if you are unsure

def xcurve(depth, length):
    box(depth, length)
    left(90)
    box(depth, length)
    left(90)
    box(depth, length)
    left(90)
    box(depth, length)
    left(90)

xcurve(4,5)
```

2. We've left the special case out for you to fill in; it should be trying to draw the bumpy line,
but calling `box(newdepth, length)` to do it. Your code should look very much like the mountain and snowflake.

STEP 7: A SACRED RELIC

Let's draw one last fractal, and like before we have a simple case and a special case.

The first three versions look like this. We Draw a triangle, then draw it as three triangles together.

```
       /\
      /  \
     /    \
    /      \
   /        \
  /          \
 /            \
----------------

       /\
      /  \
     /    \
    --------
   /\      /\
  /  \    /  \
 /    \  /    \
----------------

       /\
      ----
     /\  /\
    --------
   /\      /\
  ----    ----
 /\  /\  /\  /\
----------------
```

1. Let's open a new file and try it out.

```
from python import *

def triforce(depth, length):

    if depth == 0:
        pendown()
        forward(length)
        left(120)
        forward(length)
        left(120)
        forward(length)
        left(120)
        penup()
    else:
        penup()
        newlength = length/2
        newdepth = depth - 1

        triforce(newdepth, newlength)
        forward(newlength)
        triforce(newdepth, newlength)

        left(120)
        forward(newlength)
        right(120)
        triforce(newdepth, newlength)

        right(120)
        forward(newlength)
        left(120)


speed(11)
penup()
setpos(-255,-255)
triforce(7, 512)
```

You might have noticed, we're using a new command `setpos` to move the turtle robot to the corner of the screen.

2. Run it and see what happens. We can see the simple case is just drawing a triangle, but the special case is drawing three little triangles.

3. Try changing the values passed into `triforce()`, change the last line in the file to `triforce(5, 300)`, what does it do?

STEP 8: A BUBBLY RELIC

1. If you like, instead of triangles we can draw it with circles! Open up a new file and try the following code:

```
from turtle import *

def bubble(depth, length):
    if depth == 0:
        pendown()
        circle(length/2)
        penup()
    else:
        penup()
        newlength = length/2
        newdepth = depth - 1
        bubble(newdepth, newlength)
        forward(newlength)
        bubble(newdepth, newlength)
        left(120)
        forward(newlength)
        right(120)
        bubble(newdepth, newlength)
        right(120)
        forward(newlength)
        left(120)

speed(11)
penup()
setpos(-255,-255)
bubble(6, 512)
```

2. What happens? What does it look like? We've used the `circle` command to draw a circle on the screen, which takes a radius.

3. Try changing the circle's radius, replace `circle(length/2)` with `circle(length)`, this will draw a bigger circle instead.
