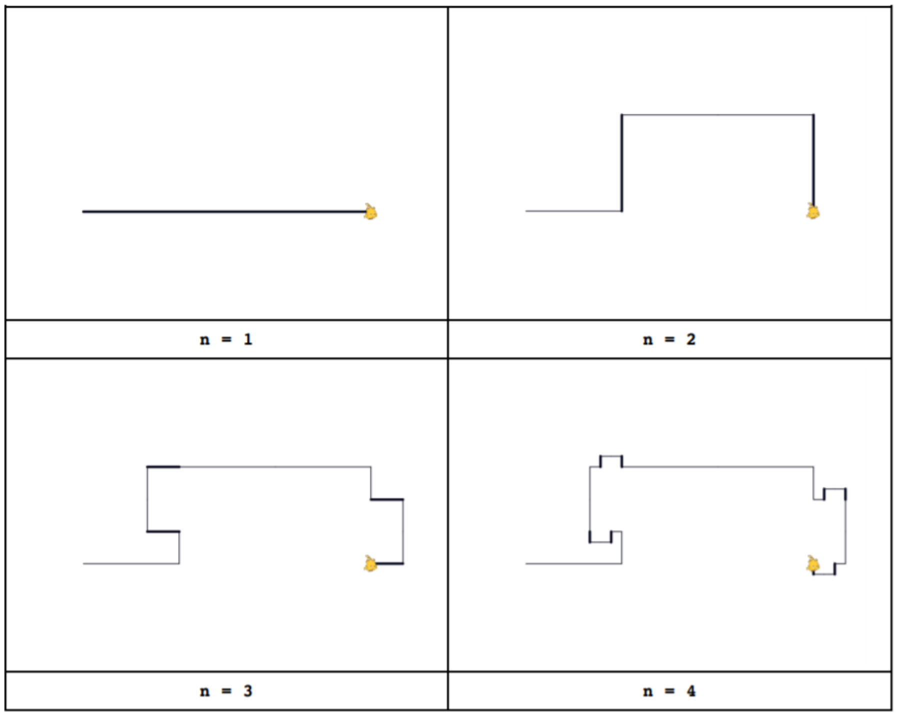
Discussion 7: More Recursion

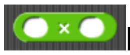
Fractals

Write out code to draw the fractal below. The sprite starts out at the bottom left corner of the image facing right and ends in the position the sprite is in in the picture. The rectangle is always ⅔ of the length of the entire line and ⅓ the height.



```
Fractal level (level) size (size):
```

Recursion

Multiply
In the box below, write a recursive function, multiply(x, y), that returns x multiplied by y **WITHOUT** using the  block. You may use any other math reporter.

```
multiply(x, y):
```

Exponent
In the box below, write a recursive function, `exponent(x, y)`, that returns x raised to the power of y.

```
exponent(x, y):
```

Fibonacci
The Fibonacci sequence is a mathematical sequence where each number in the sequence is defined as the sum of the two previous numbers. Here are the first few digits of the Fibonacci sequence: 1, 1, 2, 3, 5, 8, 13, 21, ...

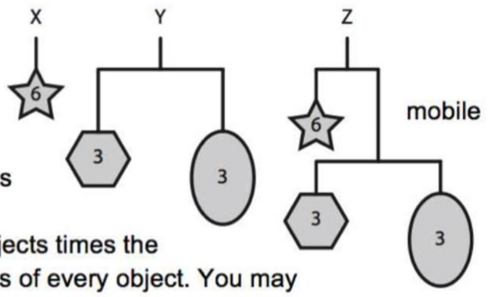
a. In the box below, write `Fibonacci(n)` recursively so it returns the nth Fibonacci number.

```
Fibonacci(n):
```

b. What is the runtime of `Fibonacci`?

____

A Little Town in Alabama...



You fondly remember the *mobiles* hanging above your crib, but you always wondered what force it took to hold them up. You wish to write **`Force(mobile)`** to answer that question. A mobile is either *simple* (has only a single object hanging from it), or *complex* (has a horizontal "inverted-T" rod that balances two mobiles on its left and right). Each object has a mass (the numbers in the examples on the right), and the *total* force is the *total* mass of all objects times the GRAVITY constant, also computed as the sum of the individual forces of every object. You may assume the vertical strings and horizontal "inverted-T" rods are weightless. From our example, **`Force(X)`** = **`Force(Y)`** = 6 * GRAVITY, and **`Force(Z)`** is double that.

Here are 4 helper blocks you'll need to use:

Block	Description
Simple? Mobile	Report if **`Mobile`** is *simple*, **`true`** for X above, **`false`** for Y and Z.
Left Complex Mobile	Reports the mobile on the *left* of the topmost "inverted-T" junction. Calling this function is an error if the mobile is simple. Example: **`Left(Z)`** would report a mobile identical to X.
Right Complex Mobile	Reports the mobile on the *right* of the topmost "inverted-T" junction. Calling this function is an error if the mobile is simple. Example: **`Right(Z)`** would report a mobile identical to Y.
Mass Simple Mobile	Reports the mass of the simple mobile. Calling this function is an error if the mobile is complex. Examples: **`Mass(X)`** would report 6, and **`Mass(Left(Y))`** would report 3.

a. Write a recursive function to find the force of a mobile.

```
Force(mobile):
```

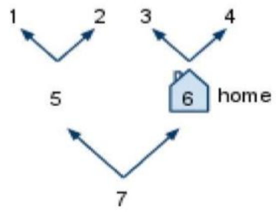
b. As a function of the number of *objects* in a mobile, what is the runtime of `Force`?

c. Your solution was recursive. Could you have written it iteratively?

Path Home

In this problem, you are given a map and a starting location, and it is your task to figure out whether you can reach home from your starting position.

For example, in the map to the right, where arrows represent paths you can take from one place, path-home would return true if your starting position is 7, and false if your starting position is 5.



You are given the following helper blocks:

- home? place : returns true if the input place is your home
- dead-end? place : returns true if the input place is a dead end
- go left place : returns the place you will be at if you go left from the input place. Gives an error if the input place is a dead end.
- go right place : returns the place you will be at if you go right from the input place. Gives an error if the input place is a dead end.

Here are some example calls to the helper blocks, based on the map above.

place	home? place	dead-end? place	go left place	go right place	path-home? place
1	false	true	ERROR	ERROR	false
2	false	true	ERROR	ERROR	false
3	false	true	ERROR	ERROR	false
4	false	true	ERROR	ERROR	false
5	false	false	1	2	false
6	true	false	3	4	true
7	false	false	5	6	true

Below, write `path_home` recursively.

path_home(place):