Functions as Data & Recursion III

Functions as Data

1. A higher order function is a function that takes in a _____ as input.

2. What does the grey ring in Snap! do?

_____

_____

3. What does the 

block do? What does the 

block do? What is the difference between them?

_____

_____

_____

4. What does each of the following calls to “map” report?

a. 

_____

b. 

_____

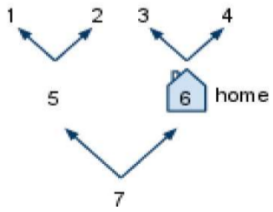
c. 

_____

Two Roads Diverged in a Wood

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.

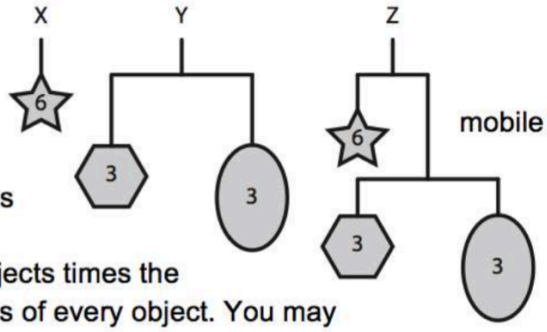
Here are some example calls to the helper blocks, based on the map above.

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

Now, try writing path-home.

```
path-home(place):
```

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 the code to find the force of a mobile:

```
Force(mobile):
```

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

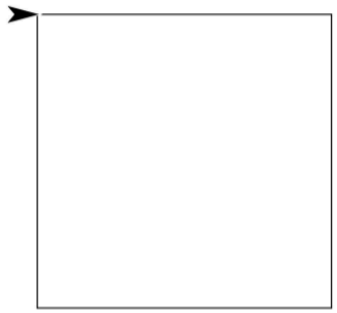
c. Your solution above was either iterative or recursive. Could you have written it the other way?

Diamonds in the Rough

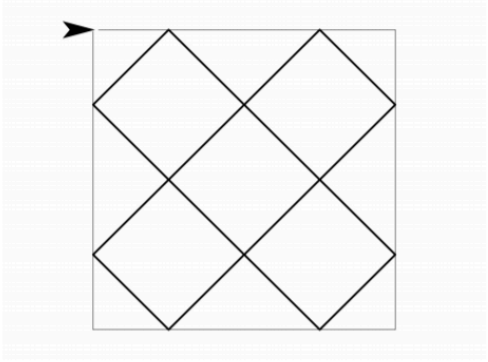
In the following exercise, you will be coding the square fractal below. We have provided the first four levels of the fractal. The recursive cases for each level are represented by bold lines.

Level 1 is 300 pixels on each side. Each of the sides of the four smaller diamonds are the length of the side of the larger square divided by the square root of 8. You may assume that the sprite starts off at the top-left corner of each level, facing right and ends in the same position. (Hint: The repeat block will be very useful for both the base case and the recursive case)

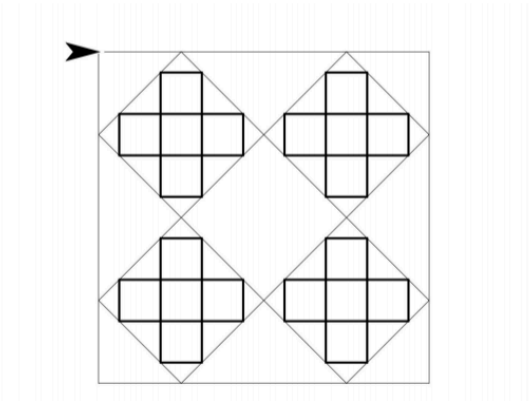
Level 1:



Level 2:



Level 3:



Level 4: