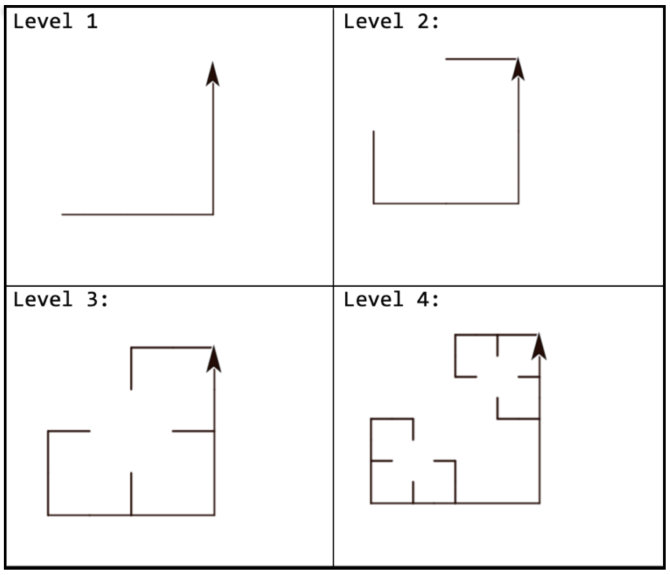
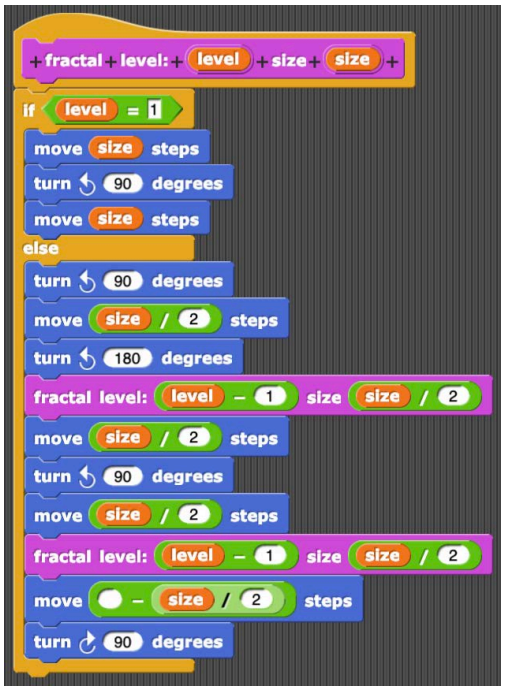
Discussion 7: Midterm Prep

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 image is ¼ of its original size every time we make a recursive call.




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



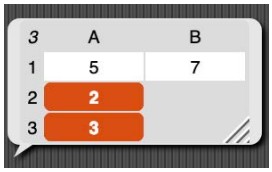
Boolean Logic

You're comfy in bed. You get up if you're thirsty. You *also* get up if you're hungry. Which expression(s) capture when you *stay in bed?*

- ● not thirsty? or hungry?
- ☐ not thirsty? and hungry?
- ☐ not thirsty? or not hungry?
- ● not thirsty? and not hungry?
- ☐ none of these

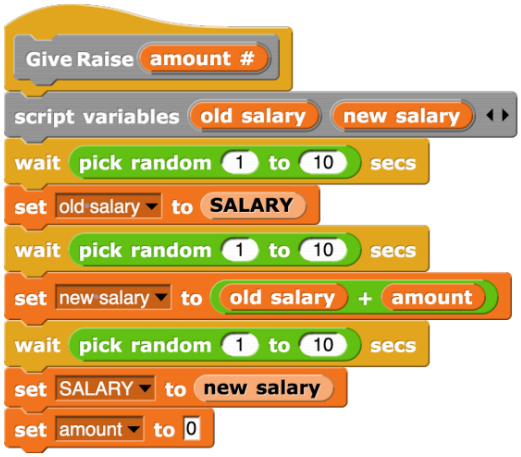
Mutability

What is the value of `data` after we run the following script?



Concurrency

Below is the definition for a block, `Give Raise`, which changes the value of the global variable `SALARY`. Below each script on the right, write ALL the possible values of `SALARY` after `Give Raise` has run to completion. (The `launch` block allows its input scripts to run in parallel, and does not wait for its input to be finished running before it moves to the block below it).



☐ 10 ☐ 11 ☐ 12 ☐ 13 ● 14 ☐ 15	☐ 10 ● 11 ☐ 12 ● 13 ● 14 ☐ 15

Programming Paradigms

1. Write down the programming paradigm that **best** fits the following descriptions:

   a. One sprite tells a second sprite to run some code. The second sprite does it.
   Object-Oriented Programming (OOP)

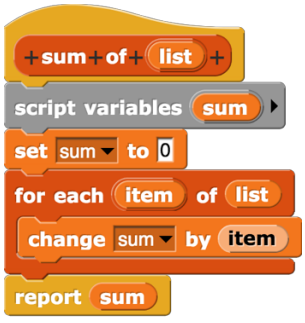
   b. You input a global list into a block. It reports a new list with different values, without modifying the input list.
   Functional

   c. You give a program a condition as an input and it uses this condition to remove numbers from a list. You input a list and it removes items.
   Declarative

   d. You have a global variable set to a secret word. You change the secret word every time you ask a player for a new secret word.
   Imperative

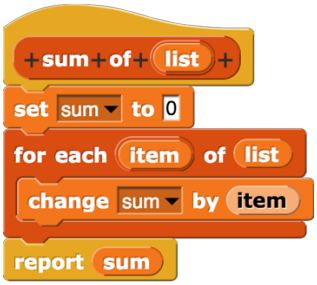
2. Match each of the following scripts to a programming paradigm. Note that each script may match multiple programming paradigms- please list the one it best fits.

Functional



Imperative

Assume `sum` is a global variable

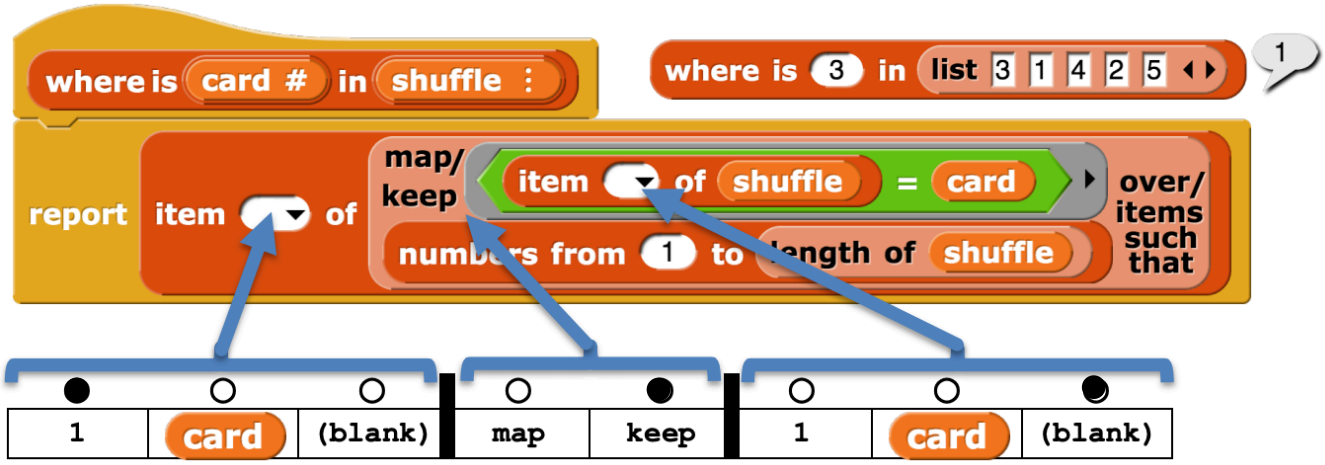


Object-Oriented

Higher Order Functions

You have cards numbered 1-N, which are shuffled (their order is scrambled) and placed into a list.

1. Fill in the circles to fill in `where is card in shuffle`, whose job is to report the index of a particular card in `shuffle`.



2. We change numbers from 1 to length of shuffle to shuffle. What would the block do now?

○ Crash	○ Run Forever	● Return the same value as before	○ Return a random value, depending on the shuffle

Algorithmic Complexity

Airlines feed their passenger in a unique way. First, the steward walks down the aisle and takes passengers' orders, one-by-one. Then, they walk back to the kitchen to prepare the meals, and walks back and forth with a single tray in hand to serve the passenger (serving one passenger at a time).

For this problem, we will ask you to measure the steward's steps, by which we mean the number of steps they walked. You may assume that each row has the same number of passengers.

Here is a formula that may be useful: $1 + 2 + \ldots + N = \frac{N(N+1)}{2}$

1. How many steps are required to take the orders as a function of the number of passengers?

○ Constant	○ Logarithmic	● Linear	○ Quadratic	○ Cubic	○ Exponential

2. How many steps are required to deliver the meals as a function of the number of passengers?

○ Constant	○ Logarithmic	○ Linear	● Quadratic	○ Cubic	○ Exponential

3. Sometimes, when a passenger receives their meal, they also ask for a pair of chopsticks. In this case, the steward needs to walk back to the kitchen, pick up the chopsticks, deliver them to the passenger, and then return to the kitchen to get the next passenger's meal. Now, how many steps are required to deliver the meals as a function of the number of passengers?

○ Constant	○ Logarithmic	○ Linear	● Quadratic	○ Cubic	○ Exponential

4. Every person sitting on the window seat has the option of hitting the new electronic shutter which makes the window opaque (light can't go in or out) or clear. If you look at the windows of the plane from the outside, it looks like a binary number (clear = light = 1, opaque = dark = 0). How many different binary numbers can be made, as a function of the number of passengers?

○ Constant	○ Logarithmic	○ Linear	○ Quadratic	○ Cubic	● Exponential

5. Does the answer to question (a) change if instead they hire a second steward to help who starts from the back, with the idea that there is an identical galley in the back with the same food as in the front, and the stewards stop when they meet each other somewhere near the middle?

○ Yes	● No

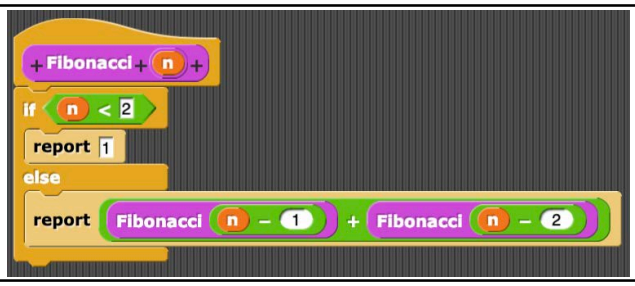
Recursion

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):`



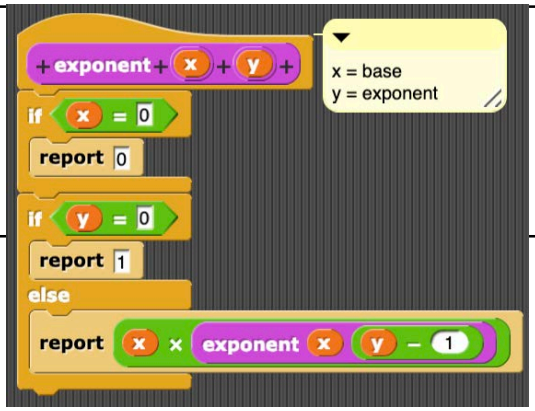
*Worked with the assumption that Fibonacci starts at 0.
> Fibonacci(0) = 1
> Fibonacci(1) = 1
> Fibonacci(2) = 2
> ...

b. What is the runtime of `Fibonacci`? exponential time

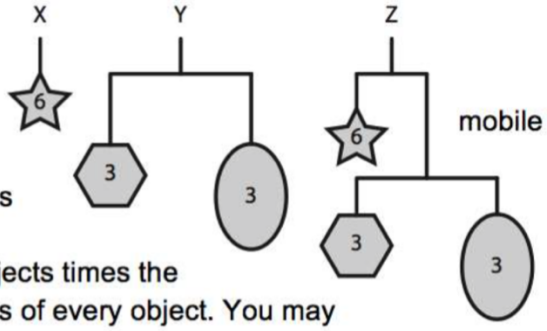
Exponent

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

`exponent(x, y):`



A Little Town in Alabama...



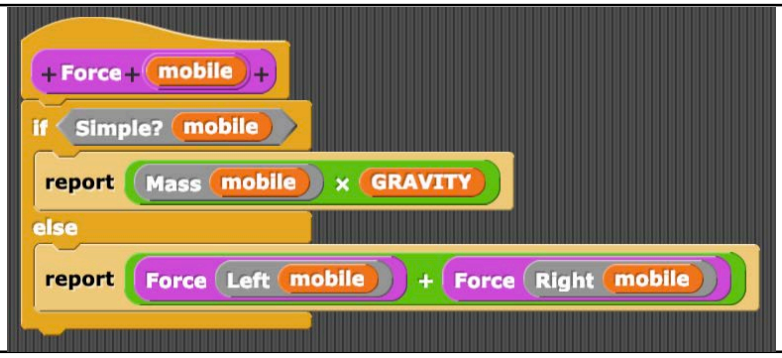
You fondly remember the *mobiles* hanging above your crib, but you always wondered what force it took to hold them up. You wish to write **`Force(mobile)`** to answer that question. A mobile is either *simple* (has only a single object hanging from it), or *complex* (has a horizontal "inverted-T" rod 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`?

Linear! To convince yourself of this, make a table to keep track of how many steps it takes to run this block for different numbers of objects.

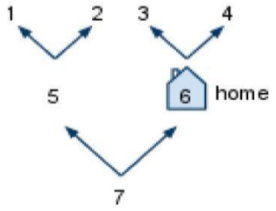
c. Your solution was recursive. Could you have written it iteratively?

Yes! Anything that can be written iteratively can be written recursively, and vice versa.

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

In the box below, write `Path Home` recursively.

```
Path Home (place):
```