The Join operation

Here is an alternative explanation to Monads than what was in the video. We'll start by talking about the join operation. Observe these four examples.

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>join [[2,3],[4,5,6]]</td>
<td>[2,3,4,5,6]</td>
</tr>
<tr>
<td>join [[3,1],[],[6,4]]</td>
<td>[3,1,6,4]</td>
</tr>
<tr>
<td>join (Just Nothing)</td>
<td>Nothing</td>
</tr>
<tr>
<td>join (Just (Just 340))</td>
<td>Just 340</td>
</tr>
</tbody>
</table>

**Problem 1)** What is the purpose of the above code? Not just what it does, but why we might want to do it?

**Problem 2)** Give an implementation of `join` for these two types.

**Problem 3)** Here is the implementation of `bind` for List and Maybe.

```haskell
1 x >>= f = concatMap f x
2 Nothing >>= f = Nothing
4 Just x >>= f = f x
```

How is this related to `join` above?
What will this do?

The point of these examples is to help you become more familiar with how monads behave.

```haskell
1 inc x = x >>= (\a -> return $ a + 1)

2 add x y = do
3     a <- x
4     b <- y
5     return $ a + b

6 -- alternative notation
7 add' x y = x >>= (\a ->
8               y >>= (\b -> return $ a + b))

9 t1 = Just 10  t4 = []
10 t2 = Nothing  t5 = [2]
11 t3 = Just 20  t6 = [5,3,8]
12 t7 = [9,3]
```

**Problem 4)** For each of these examples, decide as a team which one you think is the best answer.

<table>
<thead>
<tr>
<th>Example</th>
<th>Candidate A</th>
<th>Candidate B</th>
</tr>
</thead>
<tbody>
<tr>
<td>add t1 t3</td>
<td>Just 30</td>
<td>Just (Just 30)</td>
</tr>
<tr>
<td>add t1 t2</td>
<td>Just 10</td>
<td>Nothing</td>
</tr>
<tr>
<td>inc t4</td>
<td>[]</td>
<td>[1]</td>
</tr>
<tr>
<td>inc t5</td>
<td>[3]</td>
<td>[2]</td>
</tr>
</tbody>
</table>

**Problem 5)** Give the output to these examples. Be sure everyone on your team agrees with the answers.

Example   Output
---       ---
inc t6     
add t4 t5  
add t5 t7  
add t6 t7
The Either Monad

Here is the code for Either. The Left constructor is meant to contain an "error message" or failure, and the Right constructor is meant to contain the actual data.

```haskell
1  data Either a b = Left a |
2              | Right b
3
4  instance Functor (Either e) where
5     fmap _ (Left x) = Left x
6     fmap f (Right x) = Right (f x)
7
8  instance Applicative (Either e) where
9     pure = Right
10    (Right f) <*> (Right x) = Right (f x)
11    (Left x) <*> _ = Left x
12    _ <*> (Left x) = Left x
```

Problem 6)

Give the output to these examples. Be sure everyone on your team agrees with the answers.

Example | Output
-------|--------

```haskell
fmap (+1) (Left 10)

fmap (+1) (Right 10)

(+) <$> (Right 10) <*> (Right 32)

(*) <$> (Left "FOO") <*> (Right 99)
```

Problem 7)

Give the implementations of join and bind for Either. The first line will be instance Monad (Either e) where
Monads Activity--- Reflector’s Report

<table>
<thead>
<tr>
<th>Role</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manager</td>
<td>Keeps team on track</td>
</tr>
<tr>
<td>Recorder</td>
<td>Records decisions</td>
</tr>
<tr>
<td>Reporter</td>
<td>Reports to Class</td>
</tr>
<tr>
<td>Reflector</td>
<td>Assesses team performance</td>
</tr>
</tbody>
</table>

1. What was a strength of your team's performance for this activity?

2. What could you do next time to increase your team's performance?

3. What insights did you have about the activity or your team's interaction today?