Example result you should get from your function:

```
Prelude> capitalizeParagraph "blah. woot ha."
"Blah. Woot ha."
```

Phone exercise

This exercise by geophf[8] originally for 1HaskellADay.[9] Thank you for letting us use this exercise!

Remember old-fashioned phone inputs for writing text where you had to press a button multiple times to get different letters to come up? You may still have to do this when you try to search for a movie to watch using your television remote control. You're going to write code to translate sequences of button presses into strings and vice versa.

So! Here is the layout of the phone:

```
-----------------------------------------
|   1       |   2 ABC     |  3 DEF      |
_________________________________________
|   4 GHI   |   5 JKL     |  6 MNO      |
-----------------------------------------
|   7 PQRS  |   8 TUV     |  9 WXYZ     |
-----------------------------------------
|   * ^     |   0 + _     |  # .,       |
-----------------------------------------
```

Where star (*) gives you capitalization of the letter you're writing to your friends, and 0 is your space bar. To represent the digit itself, you press that digit once more than the letters it represents. If you press a button one more than is required to type the digit, it wraps around to the first letter. For example,

```
2     -> 'A'
22    -> 'B'
222   -> 'C'
2222  -> '2'
22222 -> 'A'
```

[8] https://twitter.com/geophf
[9] https://twitter.com/1haskelladay

So on and so forth. We're going to kick this around.

1. Create a data structure that captures the phone layout above. The data structure should be able to express enough of how the layout works that you can use it to dictate the behavior of the functions in the following exercises.

```
-- fill in the rest.
data DaPhone = DaPhone
```

2. Convert the following conversations into the keypresses required to express them. We're going to suggest types and functions to fill in order to accomplish the goal, but they're not obligatory. If you want to do it differently...you do you.

```
convo :: [String]
convo =
  ["Wanna play 20 questions",
   "Ya",
   "U 1st haha",
   "Lol ok. Have u ever tasted alcohol lol",
   "Lol ya",
   "Wow ur cool haha. Ur turn",
   "Ok. Do u think I am pretty Lol",
   "Lol ya",
   "Haha thanks just making sure rofl ur turn"]
```

```haskell
-- validButtons = "1234567890*#"
type Digit = Char

-- Valid presses: 1 and up
type Presses = Int

reverseTaps :: DaPhone -> Char -> [(Digit, Presses)]
reverseTaps = undefined
-- assuming the default phone definition
-- 'a' -> [('2', 1)]
-- 'A' -> [('*', 1), ('2', 1)]

cellPhonesDead :: DaPhone
               -> String
               -> [(Digit, Presses)]
cellPhonesDead = undefined
```

3. How many times do digits need to be pressed for each message?

```haskell
fingerTaps :: [(Digit, Presses)] -> Presses
fingerTaps = undefined
```

4. What was the most popular letter for each message? What was its cost? You'll want to combine `reverseTaps` and `fingerTaps` to figure out what it cost in taps. `reverseTaps` is a list because you need to press a different button in order to get capitals.

```haskell
mostPopularLetter :: String -> Char
mostPopularLetter = undefined
```

5. What was the most popular letter overall? What was the most popular word?

```haskell
coolestLtr :: [String] -> Char
coolestLtr = undefined

coolestWord :: [String] -> String
coolestWord = undefined
```