\

Sometimes in Haskell you may want to whip up a function on the fly and not bother naming it. It is like when you are baking and you realise you need a component that you didn't initially plan for and you have to quickly make some. For example, you just need some butter icing to sandwich together your cake. Now are you really going to stop and carefully write a recipe for butter icing and then diligently measure out the butter and the icing sugar? No you're a confident baker you know how to throw together some butter icing! You are also a confident programmer and you can quickly make wee functions that perform simple tasks (obviously only do this for simple functions!!!). Llambdas just allow these little helper functions to take in parameters.

```haskell
type Llama        =  String
type HappyLlama   =  String

feedLlamas :: [Llama] -> [HappyLlama]
feedLlamas = map (\llama -> llama ++ " :-)")
```

feedLlamas [“Adam”,“Alessio”,“Ibrahim”, “Jamie”, “Charlie”] = [“Adam :-)”,“Alessio :-)”,“Ibrahim :-)”,“Jamie :-)”, “Charlie :-)”]

`The above is an example of how you could use llambdas to make a helper function. All this functio`

```haskell
toHappy :: Llama -> HappyLlama
toHappy llama = llama ++ " :-)"
```

The two methods are completely equivalent, I could interchange the lambda function and `toHappy` without changing what `llamaFarm` does. To type a lambda it is just a backslash. The syntax is pretty similar to that of pattern matching except there is an arrow in the place of the equals, and before you the parameters you place a `\`.

This is just a general introduction on how to use lambdas with a silly trivial example. I'm really selling lambdas short since later on you will see that they allow you to do many fun and cool things. (Although personally `llamaFarm` is clearly the pinnacle of Haskell. I've truly found the best use of this programming language :-)