Chapter 7, "More functional patterns"

Syntactic considerations

- *lambda syntax*:

  ```
  \x -> x + 1
  ```

  is equivalent to

  ```
  f x = x + 1
  ```

- *point-free style*:

  ```
  f x = x + 1
  ```

  is equivalent to

  ```
  f = (+1)
  ```

Function definition patterns

- pattern matching
- case expressions
- guards

Pattern matching

```
foo :: Bool -> Int
foo True = 1
foo False = 0
```

Case expression

```
foo :: Bool -> Int
foo b = case b of
    True -> 1
    False -> 0
```

Guards (v1)

```
foo :: Bool -> Int
foo b
    | b == True  = 1
    | b == False = 0
```

Guards (v2)

```
foo :: Bool -> Int
foo b
    | b         = 1
    | otherwise = 0
```

(BTW: `otherwise` isn't a keyword, it's just a synonym for `True`)

For next week: "Chapter 8: Recursion"

- See exercise template here: https://gist.github.com/twopoint718/875626818ea55cfa5ced3e81e1e12180