204: Swift Functional Programming, Part 4: Challenge Instructions

Re-implement frequences

For the challenge, reimplement `frequencies` using `reduce`.

Hint: Think first about the types of the input array and the accumulated value, and then the type of the combining function.

Take special care regarding the order of the parameters to the combining function, and make your life easier by giving the parameters descriptive names.

Switch to shorthand closure syntax

So far, we have defined all closures using explicit closure syntax, where we name the arguments, the argument types, and the return types of closures, leading to expression like so:

```
{ (x:Int) -> String in
  return f( x + 1 ) */
  }
```

In typed FP languages, it is common to rely on the language's type inference to fill in this information.

Where it is clarifying, update all these calls to map, filter, and reduce in order to use the shorthand syntax, which allows expressions like so:

```
{ f( $0 + 1 ) }
```

Take special care to avoid *superflous wrapping*, which is where a closure wraps a function that could be passed directly, such as writing

```
{ (x:Int) -> String in return f( x ) }
```

or even

```
{ f( $0 ) }
```

instead of just writing

```
f
```