Programming 3
Functional Programming Challenges

Elliot Purvis
Student ID: 28561716

12^{th} November 2017

1 Composition Law Proof

Given the law, we can translate the left hand side to equal the right.

```
pure (.) <*> u <*> v <*> w = u <*> (v <*> w)
```

Taking only the left hand side, and applying the definition of pure for Maybe:

```
Pure x = Just x
```

The left hand side is left as:

```
(Just (.)) <*> u <*> v <*> w
```

We know that for the Maybe type, $< * >$ is defined as:

1

```
instance Applicative Maybe where
    pure = Just
    Nothing <*> _ = Nothing
    (Just f) <*> something = fmap f something
```

Hence if either argument for $< * >$ is nothing, the entire expression reduces to Nothing, so we can ignore these and presume u, v and w are something. Using this, we can then apply $< * >$ (being careful to keep terms left-applicative) to the first two terms.

```
(Just (.) f) <*> (Just g) <*> (Just x)
```

Repeating this:

```
(Just (.) f g) <*> (Just x)
```

And then applying $< * >$ a final time:

```
(Just (f.g) x)
```

Which expands to:

```
(Just f (g x))
```

From here, we can apply:

```
pure f <*> pure x = pure (f x)
```

We apply the homomorphism law once, and then again to the second generated term.

```
(Just f) <*> Just (g x)

(Just f) <*> ( (Just g) <*> (Just x) )
```

Finally, we can remove the earlier definitions of u, v, w to give the right side of our original equation.

```
u <*> (v <*> w)
```