# IN3043 Functional Programming
# Solutions to Exercises 1

1. There are 60 seconds in a minute, 60 minutes in an hourm 24 hours in a day and 7 days in a week. We could use the interpreter to work out the answer and use that, but it is more readable to use the expression:

```
secondsInWeek :: Integer
secondsInWeek = 7*24*60*60
```

2. This is just a matter of translating the expression $\frac{1+\sqrt{5}}{2}$ into Haskell:

```
phi :: Double
phi = (1 + sqrt 5)/2
```

   Note that **phi** is a function with no arguments, i.e. a constant (like **size** in the script in the lecture).

3. A suitable constant definition is

```
mile :: Double
mile = 1.609344
```

   Then the functions may be defined as

```
milesToKm :: Double -> Double
milesToKm x = x*mile

kmToMiles :: Double -> Double
kmToMiles x = x/mile
```

4. This is like **squareOfTriple**, but the other way round:

```
tripleOfSquare :: Integer -> Integer
tripleOfSquare n = triple (square n)
```

5. Again this is like **squareOfDouble**:

```
fourthPower :: Integer -> Integer
fourthPower n = square (square n)
```

6. Generalizing the instances given in the lecture from specific numbers to **n**, we get the definition

```
factorial :: Integer -> Integer
factorial n = product [1..n]
```

7. If we apply the above definition to 21, we get the answer

```
*Week1> factorial 21
51090942171709440000
```

It continues to work for larger numbers, but gets slower.

Switching the definition to **`Int`** (changing the type, not the definition)

```
factorial :: Int -> Int
factorial n = product [1..n]
```

and applying that to 21, the answer is too big for the limited size of **`Int`**, and overflows:

```
*Fact> factorial 21
-4249290049419214848
```

(The value at which it overflows depends on your machine.)

8. This is an example of a function taking more than one argument.

```
norm :: Double -> Double -> Double
norm x y = sqrt (x*x + y*y)
```

An alternative definition:

```
norm x y = sqrt (x^2 + y^2)
```