16-124-836
Marcel ZAUDER

4.1 Monomorphic or Polymorphic Functions

```
mod :: Int -> Int -> Int
factors n = [x \mid x <- [1..n-1], mod n x == 0 ]
```

The function factors is monomorphic because it only accepts inputs of type `Int` and always will return a result of type `[Int]`.
`factors :: Int -> [Int]`

```
isPerfect = sum (factors n) == n
```

The function isPerfect is monomorphic because it only accepts inputs of type `Int` and always will return a result of type `Boolean`.
`isPerfect :: Int -> Boolean`

```
insert _ n [] = [n]
insert 0 n l = n:l
insert i n (x:xs) = x : insert (i-1) n xs
```

The function insert n is polymorphic because the type of input for n could be either one element or a list.
`insert :: Int -> a -> [a] -> [a]`

```
mH (a, b, c) = c
```

The function mH is polymorphic because the type of input and output can vary.
`mH :: (a, b, c) -> c`

4.2 Square Function

```
square :: Int -> Int
square :: Float -> Float
square :: Char -> Char
square :: Double -> Double
```

Because a square of a char is not defined in a sensible way, this type of input and output would be invalid.

4.3 Function to calculate Circumference of a Circle and Area of a Rectangle

```
data Shape = Circle Float | Rectangle Float Float

--exercise :: Shape -> Float
exercise (Circle a) =  2 * 3.14 * a
exercise (Rectangle a b) = a * b
```