Programming languages - U4

Jan Dietrich - 10-100-436

1

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

This part of the code defines a list comprehension in which each element x is quarded with a comparison of mod and zero

The mod function allows only Int as arguments therefore the types look as follows:

```
factors :: Int -> [Int]
```

Since our function only accepts one type (Int) it's monomorphic

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

Compares the sum of all factors and to the provided number returns a boolean

the sum of factors can only be an Int, because factors accepts only Int (see above)

The comparison `==` returns a `Bool`

```
isPerfect :: Int -> Bool
```

isPerfect also accepts only one type (monomorphic)

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

Inserts a specific type into an array of the same type at the specified index

n doesn't have to be a specific type but the third argument has to be a list of the same type

```
insert :: Int -> b -> [b] -> [b]
```

since b can be of multiple types the function is polymorphic

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

mH has a tuple with three elements of which only the third is relevant

`mH :: (a, b, c) -> c`

since c can be multiple types the function is polymorphic (a,b are ignored anyway)

2

By implementing the square function

```
square n = n*n
```

we can query the type in ghci with

```
:t square
```

the output shows as that only the type `Num` is expected to use with `*`

```
square :: Num a => a -> a
```

If we look at the documentation a `Num` can be `Int, Integer, Float, and Double`

Therefore the `square` function does not accept a `Char`

3

```
data Figure = Circle Double | Rectangular Double Double deriving (Show)
cirea :: Figure -> Double

cirea (Circle r) =  3.1415926 * r * 2
cirea (Rectangular l w)=  l*w

main = do print (cirea (Circle 12))
          print (cirea (Rectangular 3 4))
```