Functional Languages 10th practice

1. Define a function `dropSpaces` which removes spaces from the beginning of a string. Use a higher order function.

```
dropSpaces "   hi  h i "  == "hi  h i "
dropSpaces "apple tree  " == "apple tree  "
dropSpaces ""             == ""
```

2. Define a function `trim` which removes spaces from both ends of a string.

```
trim "  hello! " == "hello!"
trim "Haskell"   == "Haskell"
trim ""          == ""
```

3. Define a function `monogram`. Use `word` and higher order functions.

```
monogram "Jim Carrey"         == "J. C."
monogram "Carl Edward Sagan"  == "C. E. S."
monogram "Paul McCartney" == "P. M."
```

4. Define a function `uniq :: Ord a => [a] -> [a]` which removes duplicates. `sort` combined with `group` can do a lot.

```
uniq "Mississippi" == "Mips"
uniq "parrot"      == "aoprt"
uniq ""            == ""
```

5. Define a function `repeated` which keeps repeated elements only. This is similar to `uniq` except it needs filtering.

```
repeated "Mississippi"       == "ips"
repeated [1,2,3,4,2,5,6,7,1] == [1,2]
repeated ""                  == ""
```

6. Redefine function `zipWith`, which is similar to `zip` except it does not only creates pairs but applies a function on the elements of the list.

```
zipWith' min [1,9,2,5] [5,0,3,8] == [1,0,2,5]
zipWith' min [1,0,3] [5,2,10,1]  == [1,0,3]
zipWith' (*) [2,0,6] [1,5,4,9]   == [2,0,24]
```

7. Define the scalar product of two vectors, which is the sum of elementwise product of the vectors. Use `zipWith`

```
dotProduct [1, 2] [3, 4]       == 11
dotProduct [2, 2, 2] [5, 4, 3] == 24
dotProduct [3] [2]             == 6
dotProduct [1..10] [1..10]     == 385
```

8. Define a function `isPrime` which checks whether a natural number is prime. Use a higher order function.

```
not (isPrime 0)
not (isPrime 1)
isPrime 2
isPrime 3
not (isPrime 4)
```

9. Define a list `primes` using a higher order function.

```
take 5 primes == [2,3,5,7,11]
```

10. *Redefine `iterate :: (a -> a) -> a -> [a]` which constructs an infinite list with successive applications of a function.

```
take 5 (iterate' (\n -> n * 2) 1) == [1,2,4,8,16]
```

11. *Define infinite list `fibonacci` using `iterate` above.

```
take 5 fibonacci == [0,1,1,2,3]
```