Exercises
Types and Classes

Exercise 1

What are the types of the following values?

```
['a', 'b', 'c']
('a', 'b', 'c')
[(False, '0'), (True, '1')]
(['1', '0'], ['0', '1'])
[tail, init, reverse]
```

Use GHCi (:t) to check your answers.

Solution to Exercise 1

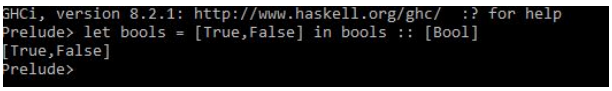
```
['a', 'b', 'c'] ----> type ----> [Char]
('a', 'b', 'c') ----> type ----> (Char,Char, Char)
[(False, '0'), (True, '1')] ----> type ----> [(Bool. Char)]
(['1', '0'], ['0', '1']) ----> type ----> ([Char], [Char])
[tail, init, reverse] ----> type ----> [ [a]-> [a] ]
```

Exercise 2

Write down definitions that have the following types. It does not matter that the definitions actually do as long as they are type correct:

```
bools :: [Bool]
nums :: [[Int]]
add :: Int -> Int -> Int-> Int
copy :: a -> (a,a)
apply :: (a -> b) -> a -> b
```

Check your answers using GHCi. You can do this using a script or by using the let construct:



Solution to Exercise 2

Using script

```
bools :: [Bool]
bools = [True, False]
nums :: [[Int]]
nums = [ [1,2,3], [2,3,4]]
add :: Int -> Int -> Int-> Int
add x y z= x + y + z
copy :: a -> (a,a)
copy x = (x, x)
apply :: (a->b) -> a -> b
apply double x = double x
```

Exercise 3

What are the types of the following functions?

```
second xs = head (tail xs)
swap (x,y) = (y,x)
pair x y = (x,y)
double x = x*2
pallindrome xs = reverse xs == xs
twice f x = f (f x)
```

The easiest way to check this is to use the *:t* at the console. You can also check this by putting these in a script with the type. If they are not consistent, you will get an error when you run/load the script.
Also, take care to include the necessary class constraints (e.g. Eq a => when you are testing for equality)if the functions are defined using overloaded operators.

Solution to Exercise 3

```
copy :: a -> (a,a)
copy x = (x, x)
second :: [a] -> a
second xs = head (tail xs)
swap :: (a,b) -> (b,a)
swap (x,y) = (y,x)
pair :: a -> b-> (a,b)
pair x y = (x,y)
double ::Num a => a -> a
double x = x +x
pallindrome :: Eq a => [a] -> Bool
pallindrome xs = reverse xs == xs
twice :: (a->a) -> a -> a
twice f x = f (f x)
```