Exercises
Functions

** Exercise 1

Using library functions, define a function

```
halve :: [a] -> ([a], [a])
```

that splits an even-lengthed list into two halves. For example:

```
*Main> halve [1,2,3,4,5,6]
([1,2,3],[4,5,6])
*Main>
```

** Exercise 2

Define a function ***third***

```
third :: [a] -> a
```

that returns the third element in a list that contains at least this many elements using:

1. ***head*** and ***tail***;
2. list indexing ***!!***;
3. pattern matching.

** Exercise 3

Consider a function ***safetail*** that behaves in the same way as ***tail***, except that ***safetail*** maps the empty list to the empty list, whereas tail gives an error in this case. Define ***safetail*** using:

1. a conditional expression;
2. guarded equations;
3. pattern matching.

Hint: the library function ***null*** :: [a] − > Bool can be used to test if a list is empty.

** Exercise 4

In a similar way to `&&` in this section's slides, show how the disjunction operator || can be defined in three different ways using pattern matching.

* Exercise 5

Given the function with the following type :

```
lucky :: Integral a => a-> String
```

Write the function definition that returns the following strings given the following inputs:

1. When input is 7, the output is the String "Lucky you.. Proceed directly to buy a lottery ticket.".
2. When input is 13, the output is the String "You, sadly are quite unlucky. Do not, under any circumstances, invest money today."
3. For any other input, the output is the String "Mmmm.... Can't really say...."

** Exercise 6

Given the two (Prelude) functions ***fst*** and ***snd*** who return the first and second element of a 2-tuple respectively as in :

```
fst :: (a,b) -> a
fst(x, _) = x
snd :: (a,b) -> b
snd(_, y) = y
```

Write similar functions - first, second and third who return the first, second and third element of a three tuple.

```
second (2,4,'e') = 4
```

**** Exercise 7

The ***Luhn Algorithm*** is used to check bank card numbers for simple errors such as mistyping a digit and proceeds as follows:

- consider each digit as a separate number;
- moving left, double every other number from the second last;
- subtract 9 from each number that is now greater than 9;
- add all the resulting numbers together;
- if the total is divisible by 10, the card number is valid.

Note that the rightmost digit is the check digit. Define a function

luhnDouble :: **Int** -> **Int**

that doubles the a digit and subtracts 9 if the number is greater than 9. For example

```
*Main> luhnDouble 6
3
*Main> luhnDouble 3
6
*Main>
```

Using ***luhnDouble*** and the integer remainder function ***mod***, define a function

luhn :: **Int** -> **Int**-> **Int**-> **Int**-> **Bool**

that decides if a four digit number is valid. For example:

```
*Main> luhn 1 7 8 4
True
*Main> luhn 4 7 8 3
False
*Main>
```

**** Exercise 8

Using the same definition of the luhn algorithm and remembering that the rightmost digit is the check digit, write a function

luhnGetCheck :: **Int** -> **Int**-> **Int**-> **Int**

that, given the leftmost three digits as per the previous example, calculates the check digit. For example

```
*Main> luhnGetCheck 1 7 8
4
*Main> luhnGetCheck 1 7 9
2
```

Solutions

Solution to Exercise 1

```
halve :: [a] -> ([a], [a])
halve xs = (take half xs, drop half xs) where
       half = length xs `div` 2
```

Solution to Exercise 2

```
third :: [a] -> a
third xs = head (tail (tail xs))

third' :: [a] -> a
third' xs = xs !! 3

third'' :: [a] -> a
third'' (_:_:x:_) = x
```

Solution to Exercise 3

```
safetail :: [a] -> [a]
safetail xs  =  if  null xs  then  []  else  tail xs

safetail' :: [a] -> [a]
safetail' xs | null xs  = []
             | otherwise = tail xs

safetail'' ::[a] -> [a]
safetail'' [] = []
safetail'' (_:xs) = xs
```

Solution to Exercise 4

```
myor:: Bool -> Bool -> Bool
False  `myor` False = False
_ `myor` _ = True

myor':: Bool -> Bool -> Bool
False  `myor'` b = b
True `myor'` _ = True
```

Solution to Exercise 5

```
-- (from Learn you a good Haskell)
lucky :: Integral a => a-> String
lucky 7 = "Lucky you.. Proceed directly to buy a lottery ticket."
lucky 13 = "You, sadly are quite unlucky. Do not, under any circumstances, inves
lucky _ = "Mmmm.... Can't really say...."
```

Solution to Exercise 6

```
fst' :: (a,b,c) -> a
fst'(x, _, _) = x
snd' :: (a,b,c) -> b
snd'(_, y,_) = y
thrd' :: (a,b,c) -> c
thrd'(_, _,z) = z
```

Solution to Exercise 7

(From Hutton)

```
luhnDouble :: Int -> Int
luhnDouble x = if (x*2> 9) then x*2-9 else x*2

luhn :: Int -> Int -> Int -> Int -> Bool
luhn a b c d = (a' + b + c'+ d ) `mod` 10 == 0 where
       a' = luhnDouble a
       c' = luhnDouble c
```

Solution to Exercise 8

(From Hutton)

```
luhnGetCheck :: Int-> Int -> Int -> Int
luhnGetCheck a b c = 10 - (luhnSum `mod` 10)  where  --brackets not necessary
       luhnSum = luhnDouble a + b + luhnDouble c
```