Functional Languages 4th practice

1. Redefine function `head` on list of integers.

```
headInt [5, 6, 7] == 5
headInt [10]      == 10
```

2. Redefine function `tail` on list of integers.

```
tailInt [5, 6, 7] == [6, 7]
tailInt [10]      == []
```

3. Redefine function `null` on list of integers.

```
nullInt []
not (nullInt [1, 2, 3])
not (nullInt [1..])
```

4. Define function `isSingletonInt`, which checks that a list of integers has exactly one element.

```
not (isSingletonInt [])
isSingletonInt [5]
not (isSingletonInt [6, 8])
not (isSingletonInt [5..])
```

5. Define function `toUpperFirst` which converts the first letter of a string into an upper case letter. It leaves the empty string unchanged.

```
toUpperFirst ""                == ""
toUpperFirst "finn the human" == "Finn the human"
toUpperFirst "jake"            == "Jake"
```

Hint: you can use function `toUpper` from module `Data.Char`.

6. Redefine function `isLetter` which recognises upper and lower case letters of the English alphabet.

```
isLetter 'a'
isLetter 'A'
isLetter 'b'
isLetter 'X'
not (isLetter '?')
```

Hint: you can use function `elem` which searches for an element in a list:

```
elem 5 [1,2,3,4,5]
not (elem 0 [1,2,3,4,5])
```

7. Redefine function `isDigit` which returns `True` for decimal integers in form of characters.

```
isDigit '0'
isDigit '5'
isDigit '9'
not (isDigit 'a')
not (isDigit ' ')
```

8. Define a function that returns an increasing and decreasing sequence of numbers.

```
mountain 3 == [1, 2, 3, 2, 1]
mountain 5 == [1, 2, 3, 4, 5, 4, 3, 2, 1]
```

9. Define a function that lists the divisors of an integer. All positive integers are divisors of 0.

```
divisors 10 == [1, 2, 5, 10]
divisors 16 == [1, 2, 4, 8, 16]
divisors 3  == [1, 3]
take 5 (divisors 0) == [1, 2, 3, 4, 5]
```

10. Define a constant for list of powers of two.

```
take 5 powersOfTwo  == [1, 2, 4, 8, 16]
take 10 powersOfTwo ==
  [1, 2, 4, 8, 16, 32, 64, 128, 256, 512]
```

11. *Define a constant for approximate value of π using Leibniz formula:

$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} \dots$

Now we only approximate and four times sum of the first thousand elements suffices.

Hint: first produce the infinite list `[1, -3, 5, -7, 9, -11, ... ]` then take the reciprocal of the elements and sum them up.

Hint: `pi` already exists in Haskell.

12. Produce a list which consists of all hour-minute pairs.

```
length time == 1440
```