Functions with conditions

2.1 Write two definitions, one using conditional expressions and another using guards for the function `classify :: Int -> String` that gives a qualititive marks for a grade from 0 to 20 according to the following table.

≤ 9	`"failed"`
10–12	`"passed"`
13–15	`"good"`
16–18	`"very good"`
19–20	`"excellent"`

2.2 The *body mass index* (BMI) is a simple measure for classifying the weight of adult individuals.[1] The BMI is computed from the individual's weight and height (in Kg and meters):

$$\text{BMI} = \textit{weight}/\textit{height}^2$$

For example: an individual with 70Kg and 1.70m height has a BMI of $70/1.70^2 \approx 24.22$. We can classify the result in the following intervals:

	BMI	< 18.5	`"underweight"`
$18.5 \leq$	BMI	< 25	`"normal weight"`
$25 \leq$	BMI	< 30	`"overweight"`
$30 \leq$	BMI		`"obese"`

Write a definition of the functions `classifyBMI :: Float -> Float -> String` to implement the above classification table. The two function arguments are, respectively, the weight and height.

2.3 Consider two definitions of the `max` e `min` from the standard Prelude:

```
max, min :: Ord a => a -> a -> a
max x y = if x>=y then x else y
min x y = if x<=y then x else y
```

(a) Write similar definitions for two functions `max3` e `min3` that compute the maximum and minimum between three values.

(b) Observe that the maximum and minimum operations are *associative*: to compute the maximum of three values we can compute the maximum between two of them and then the maximum between the result and third value. Re-write the function `max3` and `min3` using this idea and the Prelude `max` e `min` functions.

2.4 Write a definition of the *exclusive or* function

[1] `https://www.euro.who.int/en/health-topics/disease-prevention/nutrition/a-healthy-lifestyle/body-mass-index-bmi`

`xor :: Bool -> Bool -> Bool`

using multiple equations with patterns.

2.5 We want to implement a `safetail :: [a] -> [a]` function that behaves like `tail` but gives the empty list when the argument is empty. Write three distinct definitions using conditional expressions, guards and patterns.

2.6 Consider a function `short :: [a] -> Bool` that checks if a list has fewer than three elements.

(a) Write a definition of `short` using the `length` function.

(b) Write another definition using multiple equations and patterns.

2.7 The median of three values is the middle value when we place them in ascending order. For example: `median 2 3 (-1) == 2`.

(a) Write a definition of the `median` function that determines the median of any three values. What is its most general type? Note that we only need comparisons to determine the median.

(b) Instead of defining the median using comparisons you could use the following idea: add all three values e subtract the largest and smallest values. Re-define `median` this way. What the most general type for this new definition?

List comprehensions

2.8 Define a function `propDivs :: Integer -> [Integer]` using a list compreension that computes the list of *proper divisors* of a positive integer (d is a proper divisor of n is d divides n and $d < n$).

Example: `propDivs 10 = [1, 2, 5]`.

2.9 A positive integer n is *perfect* if it equals the sum of it proper divisors. Define a function `perfects :: Integer -> [Integer]` that computes the list of all perfect number up-to a limit given as argument.

Example: `perfects 500 = [6,28,496]`.

Hint: use a list comprehension and together with the function defined in the previous exercise.

2.10 A triple (x, y, z) of positive integers is *pythagorical* if $x^2 + y^2 = z^2$. Define a function `pyths :: Integer -> [(Integer,Integer,Integer)]` that computes all pythogorical triples whose components are limited by the argument.

Example: `pyths 10 = [(3,4,5), (4,3,5), (6,8,10), (8, 6, 10)]`.

2.11 Define a function `isPrime :: Integer -> Bool` that tests primality: n is prime if it has exactly two divisors, namely, 1 and n.

Example: `isPrime 17 = True`, `isPrime 21 = False`.

Hint: use a list comprehension to get the list of divisors.

2.12 Show that you can write alternative definitions of the Prelude functions `concat`, `replicate` and `(!!)` using just list compreehensions (not recursion). Rename your functions to avoid clashes, e.g. `myconcat`, `myreplicate`, etc.

2.13 Recall that the *binomial coefficient* $\binom{n}{k}$ is the number of subsets of size k that can be taken from a set of n elements, for $0 \leq k \leq n$.

(a) Define a function `binom :: Integer -> Integer -> Integer` to compute the binomal coeffient using the following formula

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

where $n! = 1 \times 2 \times \cdots \times n$ is the factorial of n.

Example: $\binom{3}{1} = 3!/(1!(3-1)! = 6/2 = 3$.

Hint: use the product function from the Prelude to compute factorials.

(b) Define another function `pascal :: Integer -> [[Integer]]` that computes the Pascal triangle up-to a given line.

$$\begin{aligned} \texttt{pascal}\ n = [\ & [\texttt{binom}\ 0\ 0], \\ & [\texttt{binom}\ 1\ 0,\ \texttt{binom}\ 1\ 1], \\ & [\texttt{binom}\ 2\ 0,\ \texttt{binom}\ 2\ 1,\ \texttt{binom}\ 2\ 2], \\ & \vdots \\ & [\texttt{binom}\ n\ 0,\ \texttt{binom}\ n\ 1, \ldots, \texttt{binom}\ n\ n]\ ] \end{aligned}$$

Example: `pascal 3 = [[1], [1,1], [1,2,1], [1,3,3,1]]`.
Hint: use the `binom` function and list comprehensions.