Functional Languages 6th practice

1. Define the factorial function recursively.

```
fact 0 == 1
fact 1 == 1
fact 3 == 6
fact 6 == 720
```

2. Determine the nth Fibonacci number!

```
fib 0 == 0
fib 1 == 1
fib 2 == 1
fib 4 == 3
fib 5 == 5
```

3. *Function `fib` is very slow even for small numbers. This is caused by the fact `fib` computes a Fibonacci number multiple times. Can you find a faster solution to `fib`?

4. Define the power function. In Haskell, this is operator `^`, now lets call our function `pow`. Define recursively, do not use `^`.

```
pow 0 2 == 0
pow 0 0 == 1
pow 2 0 == 1
pow 2 1 == 2
pow 3 2 == 9
```

5. Define a function `range`, which lists all integers between two integers recursively. Do not use expression like `[ .. ]`. We assume the second parameter is not smaller than the first.

```
range 5 9 == [5, 6, 7, 8, 9]
range 5 5 == [5]
range 0 3 == [0, 1, 2, 3]
```

6. Change function `range` so that it can also produce a decreasing sequence when the second parameter is smaller that the first.

```
range 6 8 == [6, 7, 8]
range 6 6 == [6]
range 4 1 == [4, 3, 2, 1]
```

7. Redefine function `length`, which counts the length of a list.

```
length' []      == 0
length' [5]     == 1
length' [8,0,3] == 3
```

8. Redefine function `minimum`, which recursively searches for the least element in a list.

```
minimum' [0] == 0
minimum' [9, 3, 4, 1, 10] == 1
```

9. Define a function which recursively collects every second element in a list.

```
everySecond "Haskell" == "akl"
everySecond "H"       == ""
everySecond "java"    == "aa"
everySecond ""        == ""
```

10. Redefine function `elem`, which recursively checks whether an element is in a list.

```
elem' 'l' "Haskell"
not (elem' 'v' "Lujzi")
not (elem' 'x' "")
```

11. Define a function which returns the corresponding value of a key in a key-value list. When the key is not found, raise an error using the function `error`.

```
value 5 [(0,"c++"),(5,"python"),(4,"rust")] == "python"
value 4 [(0,"c++"),(5,"python"),(4,"rust")] == "rust"
value 4 [(0,"c++"),(5,"python"),(4,"go")]   == "go"
```

12. Modify function `value` so that it takes one more parameter. When `value` does not found the key, return the extra parameter.

```
value 5 "scala" [(0, "c++"), (5, "python")] == "python"
value 3 "scala" [(0, "c++"), (1, "java"), (5, "python"), (4, "rust")] == "scala"
```