Computer Programming Paradigms Lab – Lab 2

1. Define a user-defined function `primes` that produces the list of all primes up to a given limit.

```
factors :: Int → [Int]
factors n = [x | x <- [1..n], n `mod` x == 0]

prime :: Int → Bool
prime n = factors n == [1,n]

primes :: Int → [Int]
primes n = [ x | x <- [2..n], prime x]
```

2. Using a list comprehension, give an expression that calculates the sum $1^2 + 2^2 + \cdots + 100^2$.

```
sum [ x^2 | x <- [1..100]
```

3. Define a user-defined function `replicate_` that produces a list of identical elements.

```
replicate n x = [x | _ <- [1..n]]
```

4. A triple (x, y, z) of positive integers is Pythagorean if it satisfies the equation $x^2 + y^2 = z^2$. Using a list comprehension, define a function that returns the list of all such triples whose components are at most a given limit. For example,

- `pyths 10`
- `[(3 ,4, 5), (4, 3, 5), (6, 8, 10), (8, 6, 10)]`

```
pyths :: Int → a → [a]
pyths n = [(x, y, z) | x <- [1..n], y <- [1..n], z <- [1..n], x^2 + y^2 ==
z^2]
```

5. A positive integer is perfect if it equals the sum of all its factors, excluding the number itself. Using a list comprehension, and the function factors, define a function that returns the list of all perfect numbers up to a given limit.

- `perfects 500`
- `[6, 28, 496]`

```
perfects :: Int → [Int]
perfects n = [x | x <- [1..n], sum( init( factors x)) == x]
```

6. The scalar product of two lists of integers xs and ys of length n is given by the sum of the products of corresponding integers:

$$\sum_{i=0}^{n-1} (xs_i * ys_i)$$

Define a function that returns the scalar product of two lists.

- `scalarproduct [1,2,3] [4,5,6]`
- `32`

```
scalarproduct :: [Int] → [Int] → Int
scalarproduct xs ys = sum [x*y | (x, y) <- zip xs ys]
```

7. Define a recursive function factorial.

```
fac :: Int → Int
fac 0 = 1
fac n = n * fac (n – 1)
```

8. Define a recursive function for Fibonacci numbers.

```
fib :: Int → Int
fib 0 = 0
fib 1 = 1
fib n = fib (n − 2) + fib (n − 1)
```

9. Define a recursive function that implements quick sort.

```
qsort :: Ord a => [a] → [a]
qsort [] = []
qsort (x:xs) = qsort smaller ++ [x] ++ qsort larger
               where
                   smaller = [ a | a <-xs , a<=x]
                   larger = [b | b <- xs, b>x]
```

10. Define a recursive function that returns the sum of the non-negative integers from a given value down to zero.

```
sumdown :: Int → Int
sumdown n = n + sumdown (n − 1)
```

11. Define a recursive function that implements the Euclid's algorithm for calculating the greatest common divisor of two non-negative integers: if the two numbers are equal, this number is the result; otherwise, the smaller number is subtracted from the larger, and the same process is then repeated. For example,

```
➢ euclid 6 27
➢ 3
```

```
euclid x y | x == y = x
           | x < y = euclid x (y − x)
           | y < x = euclid (x − y) y
```

12. Define a recursive function that merges two sorted lists to give a single sorted list. For example,

```
➢ merge [2,5,6] [1,3,4]
➢ [1,2,3,4,5,6]
```

```
merge :: Ord a => [a] → [a]
merge [] ys = ys
merge xs [] = xs
merge (x:xs) (y:ys) = if x <= y then
                          x : merge xs (y:ys)
                      else
                          y : merge (x:xs) ys
```