Lab 1. Introduction to Scala

Objectives:

- get yourself familiar with Scala syntax basics
- practice writing **tail-recursive** functions as an alternative to imperative **loops**
- keep your code clean and well-structured.

Create a new Scala worksheet to write your solutions

1.1. Recursion

1.1.1. Write a tail-recursive function that computes the factorial of a natural number. Start from the code stub below:

```
def fact (n: Int): Int = {
   def aux_fact(n: Int, acc: Int): Int =
       if (???) acc
       else ???
   ???
}
```

1.1.2. Implement a tail-recursive function that computes the greatest common divisor of two natural number:

```
def gcd(a: Int, b: Int): Int = ???
```

1.1.3. Write a tail-recursive function takes an integer n and computes the value $1+2^2+3^2+\ldots+(n-1)^2+n^2$. (Hint: use inner functions).

```
def sumSquares(n: Int): Int = ???
```

1.1.4. Write a function which computes the sum of all natural numbers within a range. Use **two styles** to write this function: direct recursion, and tail recursion.

```
def sumNats(start: Int, stop: Int): Int = ???
def tailSumNats(start: Int, stop: Int): Int = ???
```

1.1.5. Write a function which computes the sum of all prime numbers within a range.

```scala
def sumPrimes(start: Int, stop: Int): Int = ???
```

1.1.6. (!) Write a function which takes an initial value x and a range of values $x_0, x_1, \ldots, x_n$ and computes $(\ldots((x - x_0) - x_1) - \ldots x_n)$. Use the most appropriate type of recursion for this task.

```scala
def subtractRange(x: Int, start: Int, stop: Int): Int = ???
```

1.1.7. (!) Write a function which takes an initial value x and a range of values $x_0, x_1, \ldots, x_n$ and computes $x_0 - (x_1 - \ldots - (x_n - x)\ldots)$. Use the most appropriate type of recursion for this task.

1.2. Newton's Square Root method

A very fast way to numerically compute $\sqrt{a}$, often used as a standard *sqrt(.)* implementation, relies on Newton's Square Root approximation. The main idea relies on starting with an estimate (often 1), and incrementally improving the estimate. More precisely:

- Start with $x_0 = 1$.
- Compute $x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right)$

1.2.1. Implement the function `improve` which takes an estimate x_n of $\sqrt{a}$ and improves it (computes x_{n+1}).

```scala
def improve(xn: Double, a: Double): Double = ???
```

1.2.2. Implement the function `nthGuess` which starts with $x_0 = 1$ and computes the nth estimate x_n of $\sqrt{a}$:

```scala
def nth_guess(n: Int, a: Double): Double = ???
```

Note that:

- for smaller a, there is no need to compute n estimations as $(x_n)_n$ converges quite fast to $\sqrt{a}$.

1.2.3. Thus, implement the function `acceptable` which returns `true` iff $|x_n^2 - a| \le 0.001$. (Hint, google the `abs` function in Scala. Don't forget to import `scala.math._`).

```scala
  def acceptable(xn: Double, a: Double): Boolean = ???
```

1.2.4. Implement the function `mySqrt` which computes the square root of an integer `a`. Modify the previous implementations to fit the following code structure:

```
def mySqrt(a: Double): Double = {
   def improve(xn: Double): Double = ???
   def acceptable(xn: Double): Boolean = ???

   def tailSqrt(estimate: Double): Double = ???

   ???
}
```

1.2.5. (!) Try out your code for: `2.0e50` (which is $2.0 \cdot 10^{50}$) or `2.0e-50`. The code will likely take a very long time to finish. The reason is that $x_n^2 - a$ will suffer from rounding error which may be larger than 0.001. Can you find a different implementation for the function `acceptable` which takes that into account? (Hint: the code is just as simple as the original one).