Prime Numbers Calculator:
(https://melyaman.github.io/)

Implementation details:

1. A first approach to the solution was tested using the definition of prime numbers and verifying whether every number less than **n** is prime or not , and then return the number of primes found. This solution proved to be quite slow (runtime of O(n^2)) , even with the optimization to test only the numbers up to $\sqrt{n}$ (because a composite number will be a product of two number **p** and **q ,** one of which must be less than or equal than the square root of **n.**)

2. The approach implemented uses the sieve of Eratosthenes, which gives an efficient way to find the prime numbers less than **n** , with the following optimizations:

    a. If the current number is **k** , we can mark off multiples of **k** starting at k^2 in order not to mark off already marked off numbers.
    b. Terminating the loop condition at the $\sqrt{n}$.

    This method runs in O(nlog(log(n)) time , which is a good improvement , but it has the **limitation** of having O(n) space complexity because we have to declare the array of elements up to n , which can be problematic for very large numbers as the array may not fit in memory.
    In practise , this method is used for numbers up to $10^{\wedge 8}$. For larger numbers , an asymptotic approximation is used (n/log(n))[1].

Further improvements:

The program performance can be improved by including a list of prime numbers in a lookup table in an increasing order which should be faster for large numbers.

This task has the property of being **embarrassingly parallel** , so one could use a parallel approach using a MapReduce programming model , the reduce function would be just a sum . For this , the sieve has to be modified to output the number of primes in a range [a,b] instead of primes up to n.

References:

[1] https://en.wikipedia.org/wiki/Prime-counting_function
[2] **http://www.w3schools.com/bootstrap/** (some elements of the graphical interface were taken from examples of this website)