# Detecting Prime Numbers

**The goal for this exercise** is to give you more opportunities to practice creating methods (functions) with return values, and tricky mathematical operators.

In the starter project, add code to the Detecting\_Prime\_Numbers class, so that the RunExercise method asks the user for a number, then calls the isPrime function (which you will create), and uses the information that it gets back to tell the user whether their number is prime or not.

In order to accomplish this, you will need to write a function, named isPrime(), that has a boolean return type and accepts a single integer parameter. Boolean means ‘true or false’, and is an appropriate data type to use because any given number is either prime, or it’s not prime. The method should return true if the integer passed to it is a prime number, and false otherwise.

**What is a prime number?**

Basically, if you take a number, and list all possible whole-number factors, and you only end up with 1 and the number, then it’s prime. So for the number 4, you can factor it into 2 \* 2, and you can also factor it into 1 \* 4. So the set of all possible factors (of the number 4) is {1, 2, 4}. You’ll notice that 2 is neither one, nor four, so 4 is not prime. On the other hand, you can only factor 7 into 1\*7 (2 \* 3.5 doesn’t count since 3.5 isn’t a whole number), therefore the set of all possible factors of the number 7 is {1, 7}, which meets the definition of a prime number. Therefore 7 is a prime number.

Feel free to discuss the definition of prime numbers with your classmates. If you need a refresher on prime numbers, <http://en.wikipedia.org/wiki/Prime_number> is a good resource (first paragraph & the list is the most useful). Please feel free to further clarify your understanding by posting questions on the Google Group, talking to classmates / co-workers, etc

Here are some numbers, and a listing of whether they’re prime or not:

|  |  |
| --- | --- |
| **Number** | **Is prime?** |
| 1 | No – by definition |
| 2 | Yes – the only even prime! ☺ |
| 3 | Yes |
| 4 | No |
| 5 | Yes |
| 7 | Yes |
| 17 | Yes |
| 18 | No |
| 629 ( = 17 \* 37) | No |

You will have to devise an algorithm to determine whether or not a number is prime, in order to implement the isPrime method. This document will not tell you what algorithm to use, and it’s a bit tricky to find one online. Additionally, the skill of figuring out an algorithm is a valuable skill, and so it’s good for you to practice that skill here.

**What you need to do for this exercise:**

1. In the Detecting\_Prime\_Numbers class, implement the RunExercise method so that it meets the criteria above.
   1. This method will not be tested with NUnit, so you’re free to have the method print whatever you want. The following example provides one possible way of doing this, but it’s not the only correct way (**user input in bold**):  
        
      What is the number you would like to test for prime-ness?  
      **8**  
      8 is NOT prime  
        
      <This is a transcript of another run>  
        
      What is the number you would like to test for prime-ness?  
      **17**  
      17 IS prime
   2. Your RunExercise code is not required to deal with non-integer input (i.e., you don’t have to handle cases like the user typing in their name, instead of an integer)
2. Your isPrime method needs to take a single, integer parameter, and return true if that number is prime, and false if it isn’t.
   1. You function needs to be able to deal with any possible integer input correctly - negative numbers or zero should be considered to be ‘not prime’.