Software Engineering 2018

Laboratory Exercise No 2B

Zakiya Safi 1319070

Ahmed Ali Karani 1036074

Zubair Ahmed Bulbulia 1249593

Kyle Morris 1112649

9 September 2018

Contents

1 Task Description:

1.1 Program:

We are tasked with creating a prime number calculator. We are going to perform this in python. Our specific requirement is to write a program that will take in an integer, and will calculate all prime numbers less than this number and will include this number if it is a prime, thus showing all primes, as well as the total number of primes- i.e. if we input the number 12, we want to calculate all primes less than 12. This means our program should return [2.3.5.7.11], and 5, to indicate a total of 5 prime numbers less than 12 exist.

1.2 Testing:

We are then required to write a test-driver which will evaluate a variety of scenarios, showing where the program runs correctly, and incorrectly - and why this is the case. This testing will include:

- A comprehensive coverage of the program, and will involve exploring different types of input, their associated outputs, and how to interpret such results.
- Unit tests, integration tests, and system tests to provide a functional and structural overview of different test results.

2 Test file:

The below represents examples of test input and the output associated with it.

2.1 Predicted correct data:

Input	X
Predicted Output	N, Prime[i,j,...,n](where i to n are prime and less than or equal to X)

Input	1
Predicted Output	0, []

Input	2
Predicted Output	1, [2]

Input	18
Predicted Output	7, [2, 3, 5, 7, 11, 13, 17]

Input	107
Predicted Output	28, [2, 3, 5, 7,..., 71, 73, 79, 83, 89, 97, 101, 103, 107]

2.2 Exceptions:

2.2.1 Incorrect inputs:

Input	-1
Predicted Output	1 - input is not a positive integer, []

Input	assdfg
Predicted Output	1 - input is not a positive integer, []

Input	2.7
Predicted Output	1 - input is not a positive integer, []

2.2.2 Incorrect outputs:

2.2.2.1 Non-prime numbers are returned:

Input	6
Predicted Output	5, [2, 3, 4, 5, 6]

2.2.2.2 Incorrect n calculated:

Input	9
Predicted Output	7, [2, 3, 5, 7]

2.2.2.3 Numbers out of range returned:

Input	9
Predicted Output	5, [2, 3, 5, 7, 11]

3 Scenarios tested and results obtained:

3.1 Input:

- Test whether input is a positive integer - not long, float, decimal.
- Input should only contain integer numbers in the range 0,1,...,9.
- Input must be of type integer.
- Input must not have any letters, or special characters contained within it.
- Number MUST be divisible by 1.
- Numbers 0 and 1 are not considered prime.

3.2 Prime number check:

Within the program structure itself, we are required to iterate from 2 up until input n, and determine if each integer along the way is a prime number or not. In order to do so, we need to implement the following checks:

- If current number /1 has no remainder, then it passes that test case, next:
- If current number /itself had no remainder, it passes this test, next:
- If current number is not divisible by any of the integer numbers from 2 until input n, it is a prime number.

Each number will only pass the condition of being prime, if it passes all 3 test cases above. If it fails at any of these tests, we do not need to continue checking the number, and can immediately move on to the next integer number.
If an integer has passed test 1,2, and 3, as we have said this implies it is prime, it is then added to an array.
This array contains all the prime numbers less than or equal to the given input, n.

3.3 Results:

4 System tests:

System testing will be the final, overview-based testing that is performed to check the correctness of the program. System testing examines the overall program. This means that all steps and procedures need to happen in a logical order, and need to all be carried out correctly.
This process could be:

1. A correct input integer is given.
2. The method correctly calculates the number of primes in the specified interval.

3. The program outputs the total number of primes, as well as the array containing them.

There needs to be no errors occurring throughout completion of steps 1-2-3, in order for the program to execute the system test, successfully.

4.1 Unit tests:

4.1.1 Input:

We want to ensure a valid input is given. If not, we want to be able to handle it appropriately. Different possibilities:

1. User inputs a positive integer of appropriate size (not a long). This is expected, no error will be thrown.

2. User inputs a character that is not of numeric value. The program must throw an error before attempting to calculate an output - early detection.

3. A user enters a decimal number. The program must break immediately, and not attempt to calculate a result.

The 3 possibilities above can be simply checked by implementing defensive coding which makes use of an if-statement to check if the given input is a positive integer. Within this, we would simply check that each character of the input contains only integer numbers between 0,1,...,9. To ensure input integrity, we will create a string input, and then attempt to convert this input to an integer, if an error is thrown, we know that the input is not valid, and can either prompt the user to re-enter their input, or simply indicate the input is invalid, and terminate the program. The unit tests for various inputs will assert that the program runs and returns or outputs an error/prompt based on the input received.

4.1.2 Processing:

Our program method involves the process of iterating through all integer numbers from 2 up until input n, and determining if each one is a prime number or not. If an integer is a prime number, it must add it to an array (containing all primes in this interval), and increment a counter.
In order to obtain a valid result, we need to implement relevant and correct checks. These were defined above and are as follows:

1. If current number /1 has NO remainder, then it passes that test case, next:
2. If current number /itself had no remainder, it passes this test, next:
3. If current number is not divisible by any n=2 up until input n, it is a prime number.

ALL 3 of these conditions need to be met in order for a number to be prime, and subsequently added to our array. The processing and testing here is relatively straightforward. As each integer we will be looking at is already valid input (as defined by the input), we simply check if the number passes all the cases. If not, it is not prime, and is not added. An implementation to make use of would be to use a nested if-statement, thus only entering the innermost loop if the previous conditions are met.

4.1.3 Output:

To ensure that either a valid output is returned or a suitable error message is shown, it is necessary to test the output of the function.
Different possibilities:

1. The function returns a non-negative number (of primes) and the array of primes less than or equal to the input.
2. The function returns an error code based on an invalid input

The testing for the above cases can be done by writing unit tests, which will assert that the correct outputs are returned for the correct input cases. Just as importantly, tests will be written which also ensure that for some fringe cases (eg, invalid inputs), the correct error messages are outputted by the function. If the function returns a number and array as an output (as expected), testing can be done to assert that the output number is equal to the length of the array that is outputted to show it is valid. The output can also be tested to ensure that numbers in the array are actually primes and are in the range specified by the input.

4.2 Test results:

4.2.1 Error:

Possible errors that could be made are:

1. We do not ensure an integer is given as input and this erroneous data type attempts to enter the processing method and causes an error.
2. A misinterpretation of the requirement could mean at the processing stage, we add all primes in the array, instead of adding how many elements exist in the required range.
3. We do not ensure that the number provided is positive.

4.2.2 Failure:

Possible failures that could occur:

1. Numbers that are not prime are returned
2. Numbers that are prime in the range are excluded
3. Primes that are out of the stipulated range are returned
4. Incorrect number of total primes are calculated.

4.2.3 Debugging:

Debugging steps for returning numbers that are not prime:

1. Ensure that the math is correct from an overview.
2. Add prints after each operation.
3. Compare results after each operation to expected results when compared to pseudo code.
4. Identify which lines are incorrect (syntax or logical errors).
5. Repair incorrect lines.
6. Rerun steps again to find further errors.