There are 90 lines being used in the VSC file “crypts.py”, and 60 lines of actual written code while the rest are either blank lines or mark the sections divided into their common tasks: “Prime Utilities” (referring to the generation of prime numbers and candidate function); “RSA Functions” and “RSA Setup” etc.

The user interface, which was b for testing included firstly generating large prim numbers needed for public and private keys which are to be used later. The user enters a message (the characters used are ASCII) and clicks enter which then prompts the programme to give a public key component “e”. This is used alongside the variable “n”($message^e \ mod \ n$) to encrypt the text. The public key component “e” is then entered by the user to encrypt the text, followed by the private key component “d” which the user then copies and pastes into the prompt which tells them to put in “d” ($cipher^d \ mod \ n$) so that the encrypted message can be decrypted into its original message.

I tested the code several times with as diverse versions of messages as I could think of using every combination of the 26 letters in the English alphabet as well as digits from 0 to 9 and special characters and they all worked out just fine every time.

It’s for the best to do tests repeatedly to ensure the main idea works. The public key component “e” is constantly the same; “n” and “d” are different every time the former a part of both the public and private keys and the latter a part of only the private key. While the programme itself raises value error if the values (e and d) entered are incorrect and thus the user would have to start over.