CSC281 Project

­Primitive roots generator

# Project supervisor:

# Prof: Aqil Alazmi

# Students

Mohammad Al-Khenizan (443102405)

Musaed Al-Ameel (443102435)

# Contents:

1. Introduction
2. Algorithm
3. Code
4. C code
5. Code analysis
6. Demo
7. Bonus

### Introduction:

In this paper, we will demonstrate an algorithm for computing primitive roots modulo n iteratively using python and c.

Keywords: primitive roots, prime numbers, Python, c language, cycles, iteration.

Zp = {0, 1, 2, … , p-1}

An integer r belonging to Zp is a primitive root modulo p if every non-zero element of Zp is some power of r modulo p.

### Algorithm:

Before we begin, of course, we must check that p is in fact prime. To do this we just iterate over all numbers from 2 to floor(sqrt(p)) and check for divisibility. Nothing fancy.

We will determine primitivity of g modulo p iteratively by raising g to the power of every element in the set {0, 1, 2, … , p-2} and will take notice of two properties:

1. That raising g to consecutive powers in the set is cyclical.
2. Each value appears only once in the cycle

Knowing these two facts helps with determining primitivity of g, since all we need to show is that the cycle produced by g has p-1 unique numbers. Another thing to take note of is each cycle starts with 1 (since g^0 mod p is congruent to 1 for any g, p) and so naturally the cycle terminates when another k belonging to Zp gives 1 for answer. Thus, the cycle *does not* terminate before p-1 then g is a primitive root (modulo p).

First, we need a method to calculate modulo of g for any exponent less than p. We will define this function recursively as such:

g^k mod n congruent (g^k/2 \* g^k/2 mod n) \* g^(k mod 2) mod n

g^0 mod n congruent to 1

g^1 mod n congruent to g

where both g and k are *less than* n

Finally, to find *all* primitive roots modulo p, we just iterate. Simple as.

# Code:

First, we will need the is\_prime() function to check the input number:

def is\_prime(p):

    p\_sqrt = int(pow(p, 0.5)) + 1

    if p <= 1:  # Exclude negative and 1

        return False

    if p == 2:  # Include 2

        return True

    for i in range(2, p\_sqrt):

        if p % i == 0:

            return False

    return True

Also, for :

def powmod(base, exp, mod):

    if exp == 0: return 1

    if exp == 1: return base

    if exp == 2: return (base\*base) % mod

    half\_pow = powmod(powmod(base, exp//2, mod), 2, mod)

    return (powmod(base, exp % 2, mod) \* half\_pow) % mod

And for the primitive roots, we broke it down to 2 functions:

1. is\_primitive\_root(g, p)
2. get\_all\_primitive\_roots(p)

The is\_primitive\_root function implementation is:

def is\_primitive\_root(p, g):

    for i in range(1, p-1):

        if powmod(g, i, p) == 1:

            return False

    return True

And get\_all\_primitive\_roots():

def get\_all\_primitive\_roots(p):

    result = []

    for g in range(1, p):

        if is\_primitive\_root(p, g):

            result.append(g)

    return result

# Demo:

For the demo, we have made simple, yet effective GUI for the functions.

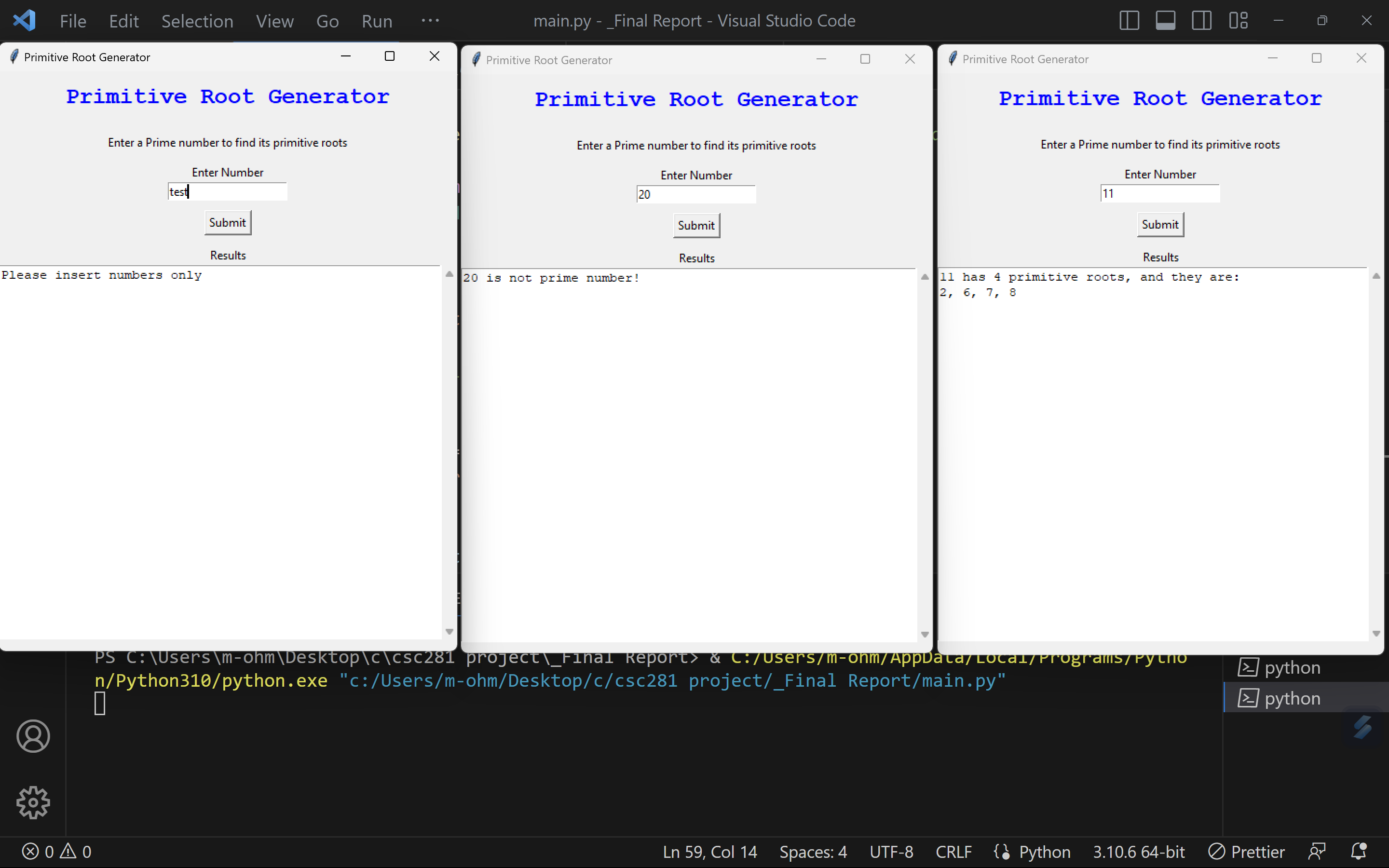
It has a text box for input, then a button to submit the number.

It also handles some exceptions like if you enter composite number, or text instead of numbers.

A screen shot of a computer

Description automatically generated with medium confidence

And for test cases



It also works for very big numbers.

A screenshot of a computer

Description automatically generated with medium confidence