Number Theory Assignment #1

21011712 박준영

Before explaining the assigned problems in this report, I have taken the liberty of making a source code file and running the functions on IDLE, which to my knowledge is no different from running the code directly on IDLE.

1) is_prime(n) function

```
import math

def is_prime(n):
    for i in range(2, int(math.sqrt(n)) + 1):
        if n % i == 0:
            return 0

    return 1
```

the source code

```
========== RESTART: C:/Users/박준영/OneDrive/바탕 화면/정수론 Python/is_prime.py =========
>>> is_prime(11)
1
>>> is_prime(253)
1
>>> is_prime(65537)
1
>>>
```

results of given examples

I've first imported the "math" library in order to use the "sqrt()" function. I then defined the function "is_prime(n)", making clear that variable *n* is an integer. The for loop will continue to check if the given variable *n* is dividable by *i*, which ranges between 2 and the square root of *n*, showing trial division was used in the code. I've made the result of "math.sqrt(n)" into an integer since the range factor can only be integers, then added 1 so the smallest result of "math.sqrt(n)" would be bigger than 2. If *n* is divisible by *i*, the code returns 0, meaning that *n* is not a prime number. Then the return function ends the "is_prime" function

The picture below shows the results of the given examples, and it seems the code is working properly.

2) generate_all_primes(n) function

```
def generate_all_primes(n):
    a = []
    for i in range(2, n + 1):
        if is_prime(i) == 1:
            a.append(i)

    return a
```

the source code

```
========== RESTART: C:/Users/박순영/OneDrive/바탕 화면/정수론 Python/is_prime.py ==========
>>> generate_all_primes(50)
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
>>> generate_all_primes(100)
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]
>>> generate_all_primes(1000)
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 1
37, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277,
281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443
, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 6
17, 619, 631, 641, 643, 647, 653, 659, 661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 797,
809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907, 911, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983
, 991, 997]
```

results of the given examples

I've created a list called "a" that will store all the generated prime numbers. Because the for loop has to check the *n* itself, the end range is set to *n + 1*. Inside the loop I have purposely used the "is_prime" function I have created in the first question to make the calculation process much more clear.

Note that the factor inside the "is_prime" function is set to i which means the code would check every number up to n + 1. If we consider that the "is_prime" function already has a loop function, ultimately the "generate_all_primes" function is using two loops in its code.

If the "is_prime" function returns 1, which means the number is a prime number, the number is appended, or added, to the list a. Then the code returns the list, showing us every prime number from 1 to *n*.

The picture below shows the results of the given examples, and it seems the code is working properly.

3) generate_random_prime(a, b)

```
import random

def generate_random_prime(a, b):
    while(True):
        n = random.randint(a, b)
        if is_prime(n) == 1:
            return n
```

the source code

```
========== RESTART: C:/Users/박준영/OneDrive/바탕 화면/정수론 Python/is_prime.py ==========
>>> generate_random_prime(2, 11)
7
>>> generate_random_prime(100, 200)
173
>>> generate_random_prime(1000, 2000)
1021
>>>
```

results of the given examples

The basic concept behind this function is not to generate a list of prime numbers between factor a and b, but rather generate a random integer first than identify if it's a prime number. It would not only fasten the run time but also keep the code short for the developer's convenience.

The while loop's factor is set to *True*, which indicates the loop will run forever if not stopped by the if function. Inside the while loop *n*, a random integer which positions between *a* and *b*, is generated. If the random integer *n* is a prime number, it returns the number immediately, stopping the loop and the code.

The picture below shows the results of the given examples, and it seems the code is working properly.