Homework 2: Control Structures

Please complete this homework assignment in code cells in the iPython notebook. Include comments in your code when necessary. Please rename the notebook as SIS ID_HW02.ipynb (your student ID number) and save the notebook once you have executed it as a PDF (note, that when saving as PDF you don't want to use the option with latex because it crashes, but rather the one to save it directly as a PDF).

Please submit a PDF of the jupyter notebook to Gradescope; keep your version of the notebook.

Problem 1: Prime Number

A prime number is a natural number greater than 1 that is not a product of two smaller natural numbers. In other words, a prime number is a number that cannot be formed by multiplying two smaller natural numbers. For example, under 10, 1, 2, 3, 5, and 7 are prime numbers; while 4, 6, 8, and 9 are not prime numbers.

Part 1: Write a function to determine if a natural number is an integer

Requirements

1. This function (`is_prime`) takes two input arguments, N and verbose. N is the natural number to test, and verbose is a Boolean variable to control if any printout is made while executing the function.

2. The returning variable is Boolean. It indicates if the number under test is a prime number of not. True means it is a prime number, while False means it is not.

3. When the input argument verbose is True. The function should print out the following message(s):

   - If the nubmer is not a prime one, then you must have found at least one way to factorize the number. As such, you can include this line in your code: `print(f"{N} is not prime as it can be expressed as {M} * {K}.")` This line allows you to properly display the values of integers

N, M, and K. For example, `451 is not prime as it can be expressed as 41 * 11.`

- If the number is a prime number, then your function should print something like 3 is a prime number by including a line like `print(f'{N} is a prime number')`

4. This function should not use any modules such as math, numpy, etc.. Simply go over all possible natural numbers smaller than the number to test, e.g., for 5, just check if 5%M == 0, where M could be 1, 2, 3, or 4. This requires you implement a loop structure in the function.

5. Use your function to check if the following numbers are prime:

- 3431, 3533, 3534319, 3534313
- you should turn verbose to True for these tests

8 points in total for Part 1.

Rubric:

Producing the correct output (no need to check their code) [8 pts]

If the student did not produce the correct output, check the following for partial credits (the partial credit items are additive => the grader can check multiple partial credit items)

Partial credits:

having a is_prime function that takes at least two argements (the actual funtion does not to be correct) [3 pts]

calling the function is_prime, but the is_prime function gave incorrect outpots [3 pts]

Solution:

```
In [24]: def is_prime(N, vernpse):
             if N == 1: #special case
                 return False
             for i in range(2,N):
                 if N%i == 0:
                     if vernpse:
                         M = N/i
                         print(f"{N} is not prime as it can be expressed as {M} * {i}")
                     return False
             if vernpse:
                 print(f'{N} is a prime number')
             return True
```

```
# Check if prime, for 3431, 3533, 3534319, 3534313
is_prime(3431, True)
is_prime(3533, True)
is_prime(3534319, True)
is_prime(3534313, True)
```

```
3431 is not prime as it can be expressed as 73.0 * 47.
3533 is a prime number
3534319 is not prime as it can be expressed as 1889.0 * 1871.
3534313 is a prime number
```

Out[25]: True

Part 2

Here, we use the function defined in Part 1 to check the frequency of prime numbers. Specifically, we want to know how many prime numbers are there for every interval of 100, e.g., 1 - 100, 101 - 200, ... 2901 - 3000

Requirements

1. Define a new function (`prime_counter`) that would return the total number of prime numbers within an interval that is specified by start and end, where start and end are the input arguments of this function. `start` and `end` are included in the interval, e.g., for the interval 101 - 200, the `start` is 101, and the `end` is 200, and you would check all the numbers between 101 and 200, as well as 101 and 200.

2. This function should call the function defined in Part-1. In other words, you need exploit the function that you developed and you don't want to reinvent the wheel. Hints - create a loop to go over all the intergers between `start` and `end` , call the `is_prime` function for each integer, count how many times `is_prime` returns True. To avoid lengthy printout, when you can `is_prime` , turn the verbose option to False.

3. Create a list `prime_counts` to store the prime number counts for these intervals. Between 1 and 3000, we have 30 intervals of 100. Your list will then have 30 entries, each of which is the number of prime numbers in the corresponding interval. Print out the list.

8 points in total for Part 2.

Rubric:

Producing the correct output (no need to check their code) [8 pts]

If the student did not produce the correct output, check the following for partial credits (the partial credit items are additive => the grader can check multiple

partial credit items)

Partial credits:

having a `prime_counter` function takes at least two arguments [3 pts]

having some sort of loop (`for` , `while` , etc.) outside of the prime_counter function. They don't need to be calling the `prime_counter` function within the loop. This partial credit is for having any loop outside of their `prime_counter` code [3 pts]

Solution:

```python
In [26]: def prime_counter(start,end):
             prime_count = 0
             for number in range(start,end+1):
                 if is_prime(number, False): #not printing
                     prime_count = prime_count+1
             return prime_count
```

```python
In [28]: prime_counts = []
         for i in range(0,30):
             count = prime_counter(100*i+1,100*i+100)
             prime_counts.append(count)


         print(prime_counts)
```

[25, 21, 16, 16, 17, 14, 16, 14, 15, 14, 16, 12, 15, 11, 17, 12, 15, 12, 12, 13, 14, 10, 15, 15, 10, 11, 15, 14, 12, 11]

Important note for the grader: 1 is mathematically not a prime number. However, we do not take points off for students for including 1 in their prime number list. Therefore, the first element in the list can be either 25 or 26

Part 3: List operations

The list `prime_counts` created in Part-2 has duplicated entries. In this part, we will practice with list operations.

1. Sort the list `prime_counts` so that its entries are ordered in an ascending order. This can be done easily with a python (list) built-in function. You should do your research to figure out how to do it.

2. Create a new list `New_list` that keeps the unique entries in the `prime_counts` (for example, for a list of [2,2,3,4,5,5], the new list of unique entries should be[2, 3, 4, 5]). As we discussed during the lecture, the data structure `set` only has unique entries. So one way to do it is to force

convert the list `prime_counts` to a set, and then convert the set back to a list. You can try these lines

`New_list_1 = list(set(prime_counts)) print(New_list_1)`

however, in this problem, you are not allowed to use set to do. You should write a function that print out this `New_list`

- takes the `prime_counts` as input argument
- return a new list that has all the unique entries in `prime_counts`
- do not use set, or any other python modules
- hints: you can use loops

3. Create a new list `New_list_2` of values in the original list that only appears once. For example, in the list of [2,2,3,4,5,5], the new list should be [3,4] because there are two 2's and two 5's in the original list. You should should write a function that

- takes the `prime_counts` as input argument
- return a new list of all the values that only appears once in `prime_counts`
- do not use set, or any other python modules
- hints: you can use loops. You may want to use the count function of list. Google search that! print out this `New_list_2`

8 points in total for Part 3.

Rubric:

Producing the correct output (no need to check their code) [8 pts]

If the student did not produce the correct output, check the following for partial credits (the partial credit items are additive => the grader can check multiple partial credit items)

Partial credits:

Call the `prime_counts.sort()` function [2 pts]

Wrote some code for sub-part 2. The function does not need to be correct [2 pts]

Wrote some code for sub-part 3. The function does not need to be correct [2 pts]

Solution:

In [30]:
```
# sort
prime_counts.sort()
```

```python
# Part 2, unique list

#taking advantage that the list is in ascending order
New_list = []
for i in range(1,len(prime_counts)):
    if prime_counts[i] != prime_counts[i-1]:
        New_list.append(prime_counts[i])
```

```python
print(New_list)
```

[11, 12, 13, 14, 15, 16, 17, 21, 25]

It is acceptable if there is an 26 in the list

```python
New_list_2 = []
is_unique_value = False
for i in range(len(New_list)):
    counter = 0
    for j in range(len(prime_counts)):
        if New_list[i] == prime_counts[j]:
            counter = counter+1
    if counter == 1:
        New_list_2.append(New_list[i])
```

```python
print(New_list_2)
```

[13, 21, 25]

It is acceptable if there is an 26 in the list

Part 4: Pair of prime numbers

Write a function to count pair of prime numbers that differ by a specific value (distance parameter). **Requirements**

1. Create a list (`prime_list`)of all the prime numbers between 1 and 3000.
2. Define a function that
   - takes two input arguments: `prime_list` and a distance parameter R
   - return the number of pairs of prime numbers that differ by R. For example, if two prime numbers, A and B, meets the condition B - A = 300, then A and B are a pair of prime numbers that differ by 300.
3. Find the most probable difference between prime numbers in the range of 1 to 3000. Since we are looking at prime numbers between 1 and 3000, the possible difference between prime numbers is between 1 and 2999 (or even smaller). We want to know which distance parameter value has the highest count of the prime number pairs.

8 points in total for Part 3.

Rubric:

Producing the correct output (no need to check their code) [6 pts]

If the student did not produce the correct output, check the following for partial credits (the partial credit items are additive => the grader can check multiple partial credit items)

Partial credits:

Wrote some code to create the `prime_list` (item 1) [1 pt]

Wrote some code to calculate the difference between two prime numbers (item 2) [1 pt]

Wrote some code to calculate the maximum value of a list [1 pts]

Wrote some sort of loop (item 2 or 3) [1 pt]

Solution:

In [49]:
```python
prime_list = []

for i in range(1,3001):
    if is_prime(i, False):
        prime_list.append(i)


# function definition
def get_pair_number(list,R):
    pair_number = 0 #number of pairs
    for i in range(len(list)-1):
        for j in range(i+1,len(list)):
            if (list[j]-list[i]) == R:
                pair_number = pair_number+1
    return pair_number


# Find the most probable difference
pair_number_list = [] #list of pair number with r value from 1 to 2999
for r in range(1,3000):
    pair_number_returned = get_pair_number(prime_list,r)
    pair_number_list.append(pair_number_returned)

max_value = max(pair_number_list)
corresponding_difference = [] #in case there are multiple values correspondi
for i in range(len(pair_number_list)):
    if pair_number_list[i]==max_value:
        if len(corresponding_difference) == 0:
            corresponding_difference.append(i+1) #careful with the index shi
        elif (pair_number_list[i] != corresponding_difference[-1]): #value r
            corresponding_difference.append(i+1)
```

```python
print('The max counts is', max(pair_number_list))
```

The max counts is 235

```python
print('The corresponding difference is', corresponding_difference[0])
```

The corresponding difference is 210