|  |  |
| --- | --- |
| **NAME** | **DUE** |
| **TOMOKI KOIKE** | **SEPT. 19 2019** |

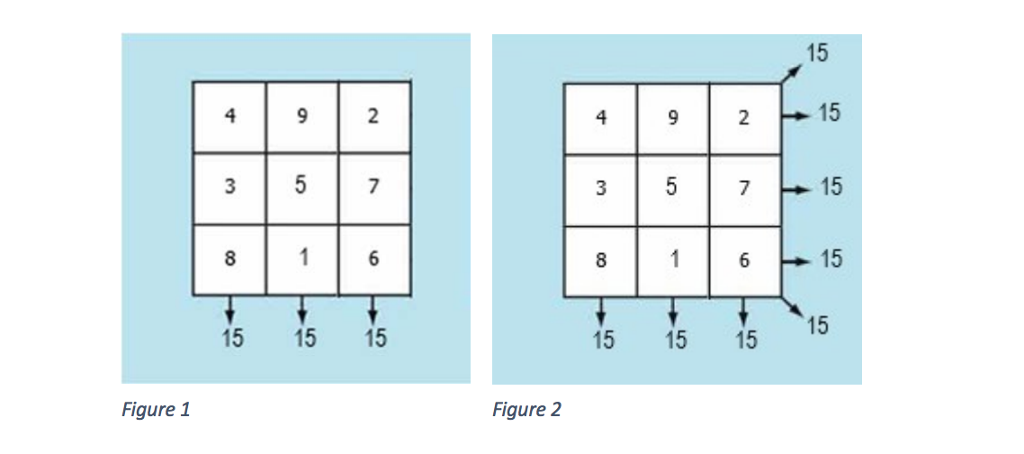
**EBEC – Intermediate Level Programming in Python**

**Week 2 – Programming Exercises**

|  |  |
| --- | --- |
| **QUESTION#** | **POINTS** |
| **1** | **/15** |
| **2** | **/15** |
| **3** | **/15** |
| **4** | **/15** |
| **TOTAL** | **/60** |

1. (15 points) The Lo Shu Magic Square is a grid with 3 rows and 3 columns, shown in the down below figure. The Lo Shu Magic has the following properties:
   1. The grid contains the numbers 1 through 9 exactly.
   2. The sum of each row, each column, and each diagonal all add up to the same number.

In a program you can simulate a magic square using a two-dimensional list. Write a function that accept a two-dimensional list as an argument and determine whether the list is a Lo Shu Magic Square. Test the function with a two-dimensional list which is a Lo Shu Square, and with a two-dimensional list which is not a Lo Shu Square in a program.



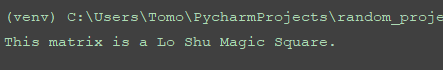
**>>CODE**

**\*The example input is the matrix above in the figure which is a Lo Shu Magic Square. And the matrix [[1,2,3], [4,5,6], [7,8,9]] which is not.**

###  
# AUTHOR: Tomoki Koike  
# DATE: Sept. 19, 2019  
# DESCRIPTION: This program will include a function that analyzes a matrix and determine  
# wether or not the matrix identifies as a Lo Shu Magic Square. That is, whether or not  
# the sum of the rows, cols, and diagonals add up to be all the same.  
###  
  
# The function  
**def is\_loShuMagicSq(**a\_matrix**):** # Initializing a inidcator of whether or not it is a Lo Shu Magic Square  
 sgn **=** 1  
 # Required sum is 15  
 sum\_req **=** 15  
 # Finding the sum in the rows  
 **for** x **in** a\_matrix**:** magic\_sum **=** sum**(**x**)** # Summing the entire rows  
 hold **=** magic\_sum # Placeholder for the summed value  
 **if** hold **!=** sum\_req**:** sgn **=** 0 # If its not a magic square set indicator to zero  
 **break** # Exit loop  
 magic\_sum **=** 0 # Reset this value  
 # Finding the sum in columns  
 **if** sgn **==** 1**:  
 for** x **in** range**(**3**):** magic\_sum **=** 0 # Reset the magic\_sum  
 **for** y **in** range**(**3**):** magic\_sum **+=** a\_matrix**[**y**][**x**]** hold **=** magic\_sum  
 **if** hold **!=** sum\_req**:** sgn **=** 0  
 **break** # Finding the sum in the diagonals  
 **if** sgn **==** 1**:  
 if** a\_matrix**[**0**][**0**]+**a\_matrix**[**1**][**1**]+**a\_matrix**[**2**][**2**] !=** sum\_req**:** sgn **=** 0  
 **if** a\_matrix**[**0**][**2**]+**a\_matrix**[**1**][**1**]+**a\_matrix**[**2**][**0**] !=** sum\_req**:** sgn **=** 0  
 # Print out conclusion  
 **if** sgn **==** 0**:** print**('This matrix is not a Lo Shu Magic Square.')  
 else:** print**('This matrix is a Lo Shu Magic Square.')**

**>>OUTPUT**

**\*Is a Lo Shu Magic Square [[4,9,2],[3,5,7],[8,1,6]]**



**\*Is not a Lo Shu Magic Square [[1,2,3], [4,5,6], [7,8,9]]**



1. (15 points) In the attachments, there is a file named ‘WorldSeriesWinners.txt’. This file contains a chronological list of the World Series’ winning teams from 1903 through 2009. The first line in the file is the name of the team that won in 1903, and the last line is the name of the team that won in 2009. (Note the World Series was not played in 1904 or 1994.There are entries in the file indicating this.)

Write a Python program that reads this file and creates a dictionary in which the keys are the names of the teams, and each key’s associated value is the number of times the team has won the World Series. The program should also create a dictionary in which the keys are the years, and each key’s associated value is the name of the team that won that year.

The program should prompt the user for a year in the range of 1903 through 2009. It should then display the name of the team that won the World Series that year, and the number of times that teams has won the World Series.

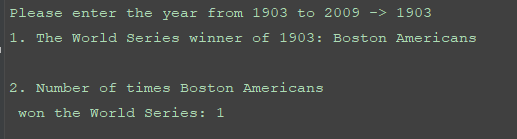
**(Use 1953, 1990, 2000, 2008 to test your program.)**

**>>CODE**

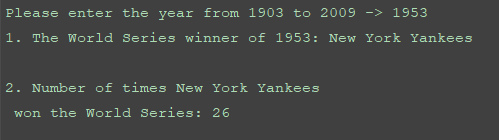
## PROBLEM #2  
# NAME: Tomoki Koike  
# DUE: Sept. 19, 2019  
# DESCRIPTION: This program reads a file with the winners of the World Series  
# into a dictionary and when the user inputs a year from 1904 to 2009 the program will output the  
# winner of that year and the nunmber of times that the team won the World Series.  
# (with input validations)  
##  
  
# Functions  
# Function for input validation  
**def getValid(**prompt**):  
 while True:  
 try:** # trying input with without any conditions at first  
 this **=** int**(**input**(**prompt**))  
 except** ValueError**:** # Prints the user to input again since the input was not valid  
 print**('Sorry, could not understand. Please enter again.')  
 continue** # For when the input is a number  
 **if** this **>=** 1903 **and** this **<=** 2009**:  
 if** this **==** 1904 **or** this **==** 1994**:** print**('There was no world series in this year please select another year -> ')  
 continue  
 else:** # Valid input  
 **break  
 else:** # Invalid input  
 print**('Error. Please enter a year in between the period of 1903 ~ 2009 -> ')  
 continue  
 return** this  
  
# Function to read a file and create a dictionary with team names and number of times winning the  
# World Series  
**def create\_dict1(**file**):** # Opening the file  
 specimen\_file **=** open**(**file,**'r')** # Reading the first line of the file  
 line **=** specimen\_file.readline**()** # Dictionary to store the team names  
 teamName **= []** numberOfWins **= []** # Looping to create the keys and values for the dict by reading the file  
 **while** line **!= '':  
 if** teamName.count**(**line**) ==** 0**:** # Append to the teamName list if it appears for the first time  
 teamName.append**(**line**)** # Add one win for this team  
 numberOfWins.append**(**1**)  
 else:** # Add the number of wins for this team  
 name\_idx **=** teamName.index**(**line**)** numberOfWins**[**name\_idx**] +=** 1  
 # Read the next line in the file  
 line **=** specimen\_file.readline**()** # Close the open file  
 specimen\_file.close**()** # Now create the dictionary  
 WorldSeriesWinner\_dict **=** dict**(**zip**(**teamName, numberOfWins**))  
 return** WorldSeriesWinner\_dict  
  
# Function that creates a dictionary with the team names and the year that they won the World Series  
**def create\_dict2(**file**):** # Opening the file  
 specimen\_file **=** open**(**file, **'r')** # Reading the file  
 line **=** specimen\_file.readline**()** # List of year  
 years **= []** starting\_year **=** 1903  
 winningTeam **= []** # Loop  
 **while** line **!= '':  
 if** starting\_year **!=** 1904 **and** starting\_year **!=** 1994**:** winningTeam.append**(**line**)** years.append**(**starting\_year**)** line **=** specimen\_file.readline**()** starting\_year **+=** 1  
 **else:** starting\_year **+=** 1  
 # Close the open file  
 specimen\_file.close**()** # Create the dictionary  
 winnerTeam\_and\_year **=** dict**(**zip**(**years, winningTeam**))  
 return** winnerTeam\_and\_year  
  
# Function to create the output based on what year the user selects  
**def output(**dict1, dict2, user\_input**):** # Printing out the winner of the year  
 team **=** dict2**[**user\_input**]** print**('1. The World Series winner of {0}: {1}'**.format**(**user\_input, team**))** # Printing out the number of times the team won the world series  
 print**('2. Number of times {0} won the World Series: {1}'**.format**(**team, dict1**[**team**]))  
 return**# Main  
**def main():** # creating the dictionaries  
 WorldSeriesWinner\_dict **=** create\_dict1**('WorldSeriesWinners.txt')** WinnerTeam\_and\_year **=** create\_dict2**('WorldSeriesWinners.txt')** # User input of the year to see the winner  
 user\_input **=** getValid**('Please enter the year from 1903 to 2009 -> ')** # output  
 output**(**WorldSeriesWinner\_dict, WinnerTeam\_and\_year, user\_input**)  
 return  
if** \_\_name\_\_ **== '\_\_main\_\_':** main**()**

**>>OUTPUT**

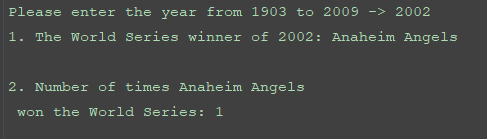
\*1903



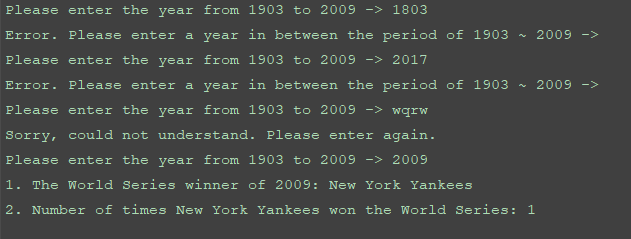
\*1953



\*2002



\*Typo of 1803, 2017, and “wqrw” then correctly entering 2009



1. (15 points) Fundamental theorem of number theory states that every natural number n can be expressed as a product of prime numbers, called its prime factorization. E.g. 15 = 3 x 5, 20 = 2 x 2 x 5.

You are required to write a Python function prime factors(n) which accepts a natural number as the input argument and returns a list of all the prime factors of n in ascending order.

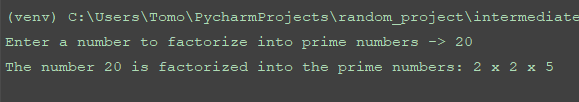
**(Use 20, 666, 4020 to test your program.)**

**>>CODE**

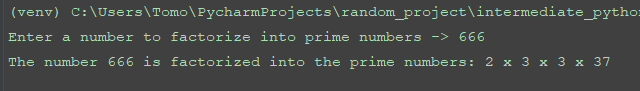
## PROBLEM #3  
# NAME: Tomoki Koike  
# DUE: Sept. 19, 2019  
# DESCRIPTION: This program has a function that conducts prime factorization  
# for any number and outputs all of the factors in ascending order.  
# (with input validation)  
##  
  
# Functions  
# Function for input validation  
**def getValid(**prompt**):  
 while True:  
 try:** # trying input with without any conditions at first  
 this **=** int**(**input**(**prompt**))  
 except** ValueError**:** # Prints the user to input again since the input was not valid  
 print**('Sorry, could not understand. Please enter again.')  
 continue** # For when the input is a number  
 **if** type**(**this**) !=** int **or** this **<=** 0**:** print**('Please enter a proper integer value -> ')  
 continue  
 else:** # Valid input  
 **break  
 return** this  
  
# Determining all the prime numbers up to the number we want to factorize  
**def prime\_nums(**high**):** low **=** 1 # Defining the lower range for the list  
 primes **= []** # Preallocate the list containing all the prime numbers up to the high  
 # The for loop  
 **for** x **in** range**(**low, high**+**1**):** ct **=** 0 # Counter to determine the numbers of divisors  
 **for** y **in** range**(**1,x**+**1**):  
 if** x **%** y **==** 0**:** # If y is a divisor of x increment the counter by 1  
 ct **+=** 1  
 **if** ct **==** 1 **or** ct **==** 2**:** # If the counter indicates 1 or 2 print the prime number  
 primes.append**(**x**)** # Appending the prime number to the list  
 **else:** # If there are more than two divisors, the number x is not a prime number  
 **pass  
 return** primes  
  
# Function that conducts prime factorization  
**def prime\_factor(**anum, primes**):** factors **= []** # Preallocating a list with all the factors  
 idx **=** 1 # Initialize an index counter  
 **while** idx **<** len**(**primes**):** temp **=** anum **//** primes**[**idx**]** # Temporary number  
 **if** anum **%** primes**[**idx**] ==** 0**:** factors.append**(**primes**[**idx**])** anum **=** temp # Swap anum with prime number  
 **else:** idx **+=** 1  
 **return** factors  
  
**def main():** num **=** getValid**('Enter a number to factorize into prime numbers -> ')** factors **=** prime\_factor**(**num, prime\_nums**(**num**))** # Printing out the results  
 print**('The number {0} is factorized into the prime numbers: '**.format**(**num**)**, end**='')  
 for** x **in** range**(**len**(**factors**)):  
 if** x **!=** len**(**factors**)-**1**:** print**(**factors**[**x**]**, **'x '**, end**='')  
 else:** print**(**factors**[**x**])  
  
if** \_\_name\_\_ **== '\_\_main\_\_':** main**()**

**>>OUTPUT**

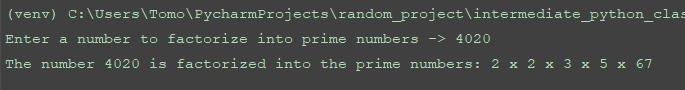
\*20



\*666



\*4020



1. (15 points) Two natural number p,q are called coprime if there are no common prime factors in their prime factorization. E.g. 15 and 20 are not coprime because 5 is a common prime number in their prime factorization, while 20 and 9 are coprime. The Euler’s phi function, φ(n), counts the number of all natural numbers ≤ n which are coprime to n. E.g. φ(9) = 6, as all natural numbers ≤ 9 are 1,2,3,4,5,6,7,8,9. And out of these, the numbers coprime to 9 are 1,2,4,5,7 and 8, a total of 6 natural numbers.

You are required to write a Python function totient(n) that accepts a natural number as input argument and returns the following:

1. a list of all the natural numbers less than or equal to n which are coprime to n; and

2. the result of the Euler’s phi function when run on n.

E.g. totient(9) should return [1,2,4,5,7,8], 6

You can use your function prime factors from Problem 2. to determine if two numbers are coprime or not.

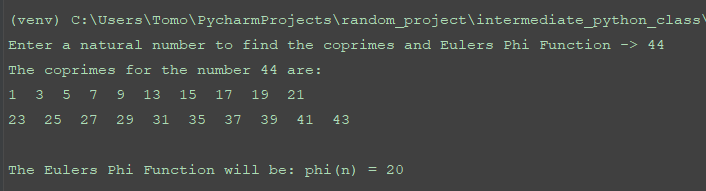
**(Use 44, 100, and 1000 to test your program)**

**>>CODE**

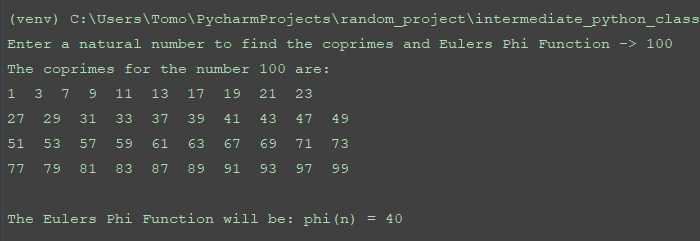
## PROBLEM #4  
# NAME: Tomoki Koike  
# DUE: Sept. 19, 2019  
# DESCRIPTION: This program will take in an input of a natural number and print out all of the coprimes that are  
# smaller than the natural number and bigger than 1. And then solve the Euler's Phi Function for that natural  
# number. (with input validation)  
##  
  
# Functions  
# Function for input validation  
**def getValid(**prompt**):  
 while True:  
 try:** # trying input with without any conditions at first  
 this **=** int**(**input**(**prompt**))  
 except** ValueError**:** # Prints the user to input again since the input was not valid  
 print**('Sorry, could not understand. Please enter again.')  
 continue** # For when the input is a number  
 **if** type**(**this**) !=** int **or** this **<=** 0**:** print**('Please enter a proper integer value -> ')  
 continue  
 else:** # Valid input  
 **break  
 return** this  
  
# Determining all the prime numbers up to the number we want to factorize  
**def prime\_nums(**high**):** low **=** 1 # Defining the lower range for the list  
 primes **= []** # Preallocate the list containing all the prime numbers up to the high  
 # The for loop  
 **for** x **in** range**(**low, high**+**1**):** ct **=** 0 # Counter to determine the numbers of divisors  
 **for** y **in** range**(**1,x**+**1**):  
 if** x **%** y **==** 0**:** # If y is a divisor of x increment the counter by 1  
 ct **+=** 1  
 **if** ct **==** 1 **or** ct **==** 2**:** # If the counter indicates 1 or 2 print the prime number  
 primes.append**(**x**)** # Appending the prime number to the list  
 **else:** # If there are more than two divisors, the number x is not a prime number  
 **pass  
 return** primes  
  
# Function that conducts prime factorization  
**def prime\_factor(**anum, primes**):** factors **= []** # Preallocating a list with all the factors  
 idx **=** 1 # Initialize an index counter  
 **while** idx **<** len**(**primes**):** temp **=** anum **//** primes**[**idx**]** # Temporary number  
 **if** anum **%** primes**[**idx**] ==** 0**:** factors.append**(**primes**[**idx**])** anum **=** temp # Swap anum with prime number  
 **else:** idx **+=** 1  
 **return** factors  
  
# Function that creates a list for the coprimes of a certain natural number  
**def totient(**n**):** coprimes **= []** # Preallocating a list to store all the coprimes  
 coprimes.append**(**1**)** # Always 1 is inlcuded  
 f1 **=** prime\_factor**(**n, prime\_nums**(**n**))** f1 **=** set**(**f1**)** # Convert list into set  
 **for** x **in** range**(**2, n**):** f2 **=** prime\_factor**(**x, prime\_nums**(**x**))** f2 **=** set**(**f2**)** f\_intersect **=** f1 **&** f2  
 **if not** f\_intersect**:** # If there are no intersections between the sets it is a coprime  
 coprimes.append**(**x**)** # Append the number for the Euler's Phi Function at the end of the list  
 coprimes.append**(**len**(**coprimes**))  
 return** coprimes  
  
**def main():** # Accept input  
 num **=** getValid**('Enter a natural number to find the coprimes and Euler''s Phi Function -> ')** # Call the function totient(n)  
 coprimes **=** totient**(**num**)** # Print out the results  
 print**('The coprimes of the number {0} are: '**.format**(**num**)**, end**='')  
 for** x **in** range**(**len**(**coprimes**)-**1**):** print**(**coprimes**[**x**]**, **' '**, end**='')** print**()** print**('The Euler''s Phi Function will be: phi(n) = {0}'**.format**(**coprimes**[-**1**]))  
  
if** \_\_name\_\_ **== '\_\_main\_\_':** main**()**

**>>OUTPUT**

**\*44**



**\*100**



**\*1000**

