|  |  |
| --- | --- |
| **NAME** | **DUE** |
| **TOMOKI KOIKE** | **OCT. 3 2019** |

**Every Boiler Engineering Code – Intermediate Level Programming**

**Week 3 & 4 – Programming Exercises**

|  |  |
| --- | --- |
| **QUESTION#** | **SCORE** |
| **1** | **/10** |
| **2** | **/20** |
| **3** | **/20** |
| **4** | **/25** |
| **5** | **/20** |
| **6** | **/15** |
| **TOTAL** | **/110** |

1. (**10 points**) Design a function that uses **recursion** to find the Greatest Common Divisor (GCD) of two numbers. The function should accept two arguments: the two numbers. Once you’ve designed your function, test it by calling it with two numbers.

Use the following three sets of numbers to test your code:

49, 28

112, 128

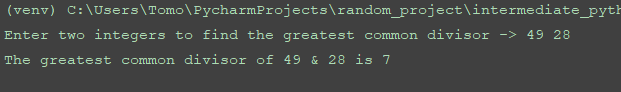
56, 141

**>>CODE**

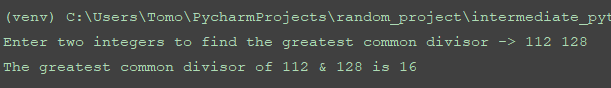
###  
# AUTHOR: Tomoki Koike  
# DUE: Oct. 3, 2019  
# DESCRIPTION: This program includes a function that will calculate the greatest common divisor  
# for the 2 input numbers.  
###  
  
# Main  
**def main():** num1, num2 **= [**int**(**z**) for** z **in** input**('Enter two integers to find the greatest common divisor -> ')**.split**()]** gcd **=** find\_gcd**(**num1, num2**)** print**('The greatest common divisor of {0} & {1} is {2}'**.format**(**num1, num2, gcd**))**# Functions  
# GCD calculator  
**def find\_gcd(**num1, num2**):** # First distinguish which input variable is larger  
 **if** num1 **>=** num2**:** high **=** num1  
 low **=** num2  
 **else:** high **=** num2  
 low **=** num1  
 # Calculating the mod/remainder of high and low division  
 mod **=** high **%** low **if** mod **!=** 0**:** # Recursive use of the find\_gcd function  
 low **=** find\_gcd**(**low, mod**)  
 return** low  
 **else:  
 return** low  
  
**if** \_\_name\_\_ **== "\_\_main\_\_":** main**()**

**>>OUTPUT**

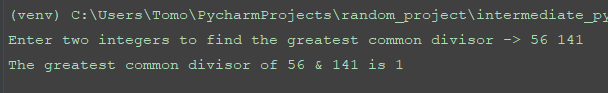
1. **49 and 28**



1. **112 and 128**



1. **56 and 141**



1. (**20 points**) A binary operator is said to be ***infix*** if it is written between is operands, as in the expression x + y. It is said to ***prefix*** if it is written before its operands, as in the expression + x y. Finally, it is said to be ***postfix*** if it is written after its operands as in x y +. An arithmetic expression consisting of numbers, variables, and operators is call i***nfix*** if it uses only ***infix*** operators, ***prefix*** if it uses only ***prefix*** operators, and ***postfix*** if all the operators are ***postfix***. The following table show the examples of infix, prefix and postfix forms of four different expressions:

| **Infix Expression** | **Prefix Expression** | **Postfix Expression** |
| --- | --- | --- |
| A + 23 \* C + D | + + A \* 23 C D | A 23 C \* + D + |
| (A + B) \* (9 + D) | \* + A B + 9 D | A B + 9 D + \* |
| A \* B + C \* 7 | + \* A B \* C 7 | A B \* C 7 \* + |
| A + B + C + D | + + + A B C D | A B + C + D + |

1. Write a function that converts an ***infix*** expression to ***prefix*** expression.
2. Write a function that converts an ***infix*** expression to ***postfix*** expression.

**Use the following expressions to test your functions:**

1. **( A + B ) \* 25 + C / D;**
2. **A + B \*5 – ( C + D )**
3. **( A + B ) \* C – ( D – E ) \* ( F + 10.0 )**

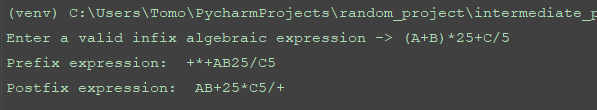
**(Assume the infix expression is a string of tokens delimited by spaces.)**

**>>CODE**

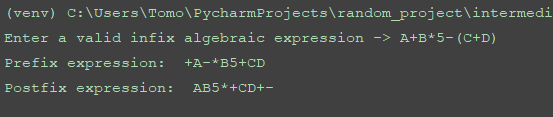
###  
# AUTHOR: Tomoki Koike  
# DUE: Oct. 3, 2019  
# DESCRIPTION: This program is designed to convert algebraic infix expressions into prefix and  
# postfix expressions.  
###  
  
# Modules  
**import** re  
  
# Main function  
**def main():** infix **=** getValid**('Enter a valid infix algebraic expression -> ')** print**('Prefix expression: '**, output**(**infixToPrefix**(**infix**)))** print**('Postfix expression: '**, output**(**infixToPostfix**(**infix**)))  
 return**# Functions  
**def getValid(**prompt**):** base **=** re.compile**('[^a-zA-Z0-9+\+\-\\*/\(\)\.]')  
 while True:  
 try:** # trying input with without any conditions at first  
 this **= (**input**(**prompt**))** this **=** re.sub**(r'\s+'**, **''**, this**)  
 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 there is a mon-digit character in the input  
 **if** base.search**(**this**):** print**('Error! Plesae enter a valid algebraic calculation expression.')  
 continue  
 else:** # Valid input  
 **break** this **=** list**(**this**)** # Convert the string into list  
 this.insert**(**0,**'(')** this.append**(')')  
 return** this  
  
# Dictionary with the operators and the priorities of them  
priority **= {  
 '^':**4,  
 **'\*':**3,  
 **'/':**3,  
 **'+':**2,  
 **'-':**2,  
 **'(':**1,  
 **')':**1  
**}**# Function that converts infix to postfix  
**def infixToPostfix(**infix**):** check **=** re.compile**(r'[\+\-\\*/]')** # Setting RegEx for later conditions  
 base **=** re.compile**(r'\w')** # Setting RegEx for later conditions  
 # Initialize the output postfix as a list  
 postfix **= []** # Initialize the operators stack as a list  
 operators **= []** ct **=** 1 # A counter  
 # Loop to rearrange the infix to postfix  
 **for** x **in** infix**:  
 if** base.search**(**x**) or** x **== '.':** postfix.append**(**x**)  
 elif** x **== '(':** operators.append**(**x**)  
 elif** ct **!=** len**(**infix**) and** x **== ')':  
 while True:** hold **=** operators.pop**()  
 if not** hold **or** hold **== '(':  
 break  
 elif** check.search**(**hold**):** postfix.append**(**hold**)  
 else:  
 pass  
 elif** ct **==** len**(**infix**):  
 while** len**(**operators**) >** 1**:** postfix.append**(**operators.pop**())  
 else:** # For when the index is an operator  
 **if** operators**:** hold **=** operators**[**len**(**operators**)-**1**]  
 while** len**(**operators**) >** 0 **and** priority**[**hold**] >=** priority**[**x**]:** postfix.append**(**operators.pop**())** hold **=** operators**[**len**(**operators**)-**1**]** operators.append**(**x**)** ct **+=** 1 # Increment counter  
 **return** postfix  
  
# Function that converts the infix to a prefix  
**def infixToPrefix(**infix**):** infix **=** infix**[::-**1**]** # Reversing the list  
 # Reversing the positions of ( and )  
 **for** idx, val **in** enumerate**(**infix**):  
 if** val **== '(':** infix**[**idx**] = ')'  
 elif** val **== ')':** infix**[**idx**] = '('  
 else:  
 pass** # Using the postfix converter  
 prefix **=** infixToPostfix**(**infix**)** # Reverse the immediate output to obtain prefixed expression  
 prefix **=** prefix**[::-**1**]  
 return** prefix  
  
# Function for the output  
**def output(**string**):  
 return ''**.join**(**string**)**# Executing main  
**if** \_\_name\_\_ **== '\_\_main\_\_':** main**()**

**>>OUTPUT**

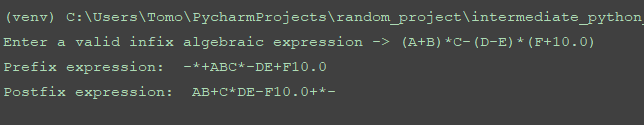
1. **( A + B ) \* 25 + C / D**



1. **A + B \*5 – ( C + D )**



1. **( A + B ) \* C – ( D – E ) \* ( F + 10.0 )**



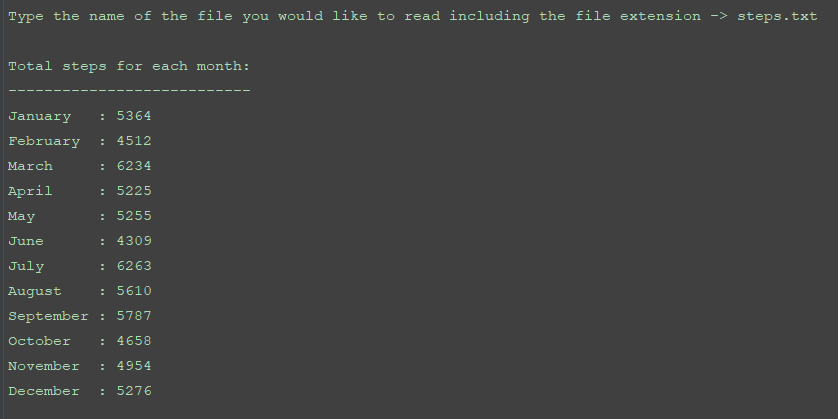
1. (**20 points**) A Personal Fitness Tracker is a wearable device that tracks your physical activity, calories burned, heart rate, sleeping patterns, and so on. One common physical activity that most of these devices track is the number of steps you take each day.

You’ll find a file named steps.txt in the attachment. The steps.txt file contains the number of steps a person has taken each day for a year. There are 365 lines in the file, and each line contains the number of steps taken during a day. (The first line is the number of steps taken on January 1st, the second line is the number of steps taken on January 2nd, and so forth.) Write a Python program that read the file, then displays the average number of steps taken for each month. (The data is from a year that was not a leap year, so February has 28 days.)

**>>CODE**

## PROBLEM #3  
# NAME: Tomoki Koike  
# DUE: Oct. 3, 2019  
# DESCRIPTION: This program will read a file containing the number of steps  
# made by a certain individual for one year. The program will output the average  
# steps for each month  
# (with input validations)  
##  
  
# Modules  
# Importing calendar module  
**import** calendar  
# Importing mean from statistics module  
**from** statistics **import** mean  
  
# FUNCTIONS  
# Function for input validation  
**def getValid(**prompt**):  
 while True:  
 try:** # trying input with without any conditions at first  
 this **=** 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.lstrip**('-')**.replace**('.'**,**''**,1**)**.isdigit**()):** # For when the input is a number  
 **if** this.find**('.') != -**1**:** # Ruling out float inputs as errors  
 print**('Error. Please enter a positive INTEGER value.')  
 continue  
 elif** int**(**this**) >** 0**:** # Valid input (a positive integer)  
 **break  
 else:** # Invalid input  
 print**('Error. Please enter a positive integer value.')  
 continue  
 else:** # For when the input is the file name (string)  
 revStr **=** this**[::-**1**]  
 if** revStr**[**0**:**4**] == 'txt.':** # Valid input for .txt extension for file name  
 **break  
 else:** # Invalid file extension  
 print**('Error. Please enter a valid file name with .txt extension.')  
 return** this  
  
# This function reads the file into one list with all the step per day data  
**def read\_file(**fileName**):** # Open the file to read  
 file **=** open**(**fileName, **'r')** # Read the first line of the file  
 read\_line **=** file.readline**()** # Preallocating list to hold all step values  
 steps\_list **= []** # Loop to read all the values  
 **while** read\_line **!= '':** # Append the read string number into the list  
 steps\_list.append**(**int**(**read\_line**))** # Reading the next line  
 read\_line **=** file.readline**()** # Closing the open file  
 file.close**()  
 return** steps\_list  
  
# This function generates the average steps for each month  
**def avg\_month\_step(**steps\_list**):** # List with the number of days in the months in order from Jan to Dec  
 monthDays **= [**31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31**]** # Initialize Counter  
 counter **=** 1  
 **for** x **in** monthDays**:** sliced\_list **=** steps\_list**[**0**:**x**+**1**]** # Calculate mean for the month  
 step\_month\_mean **=** mean**(**sliced\_list**)** print**('{0:10}: {1:6>d}'**.format**(**calendar.month\_name**[**counter**]**, int**(**step\_month\_mean**)))** # Deleting unnecessary indices from the list with steps  
 **del** steps\_list**[**0**:**x**+**1**]** # Increment counter  
 counter **+=** 1  
 **return**# Main  
**def main():** # Reading the data text file with steps  
 list\_of\_steps **=** read\_file**(**getValid**('Type the name of the file you would like to read including the file extension -> '))** # Calculating and output the average for each month  
 print**()** print**('Total steps for each month:')** print**('---------------------------')** avg\_month\_step**(**list\_of\_steps**)**# Execute main  
**if** \_\_name\_\_ **== '\_\_main\_\_':** main**()**

**>>OUTPUT**



1. (**25 points**) To play the PowerBall lottery, you buy a ticket that has five numbers in the range of 1-69, and a “PowerBall” number in the range of 1-26. (you can pick the numbers yourself, or you can let the ticket machine randomly pick them for you.) Then, on a specified date, a winning set of numbers is randomly selected by a machine. If your first five number match the first five winning numbers in any order, and your PowerBall number matches the winning PowerBall number, then you win the jackpot, which is very large amount of money. If your numbers match only some of the winning numbers, you win a lesser amount, depending on how many of the winning numbers you have matched.

In the attachment, you’ll find a file named pbnumbers.txt, containing the winning PowerBall numbers that were selected between February 3, 2010 and May 11, 2016(The file contains 654 sets of winning numbers.) Each line in the file contains the set of six numbers that were selected on a given date. The number are separated by a space, and the last number in each line is the PowerBall number for that day. For example, the first line in the file shows the numbers for February 3, 2010, which were 17, 22, 36, 37, 52, and the PowerBall number 24.

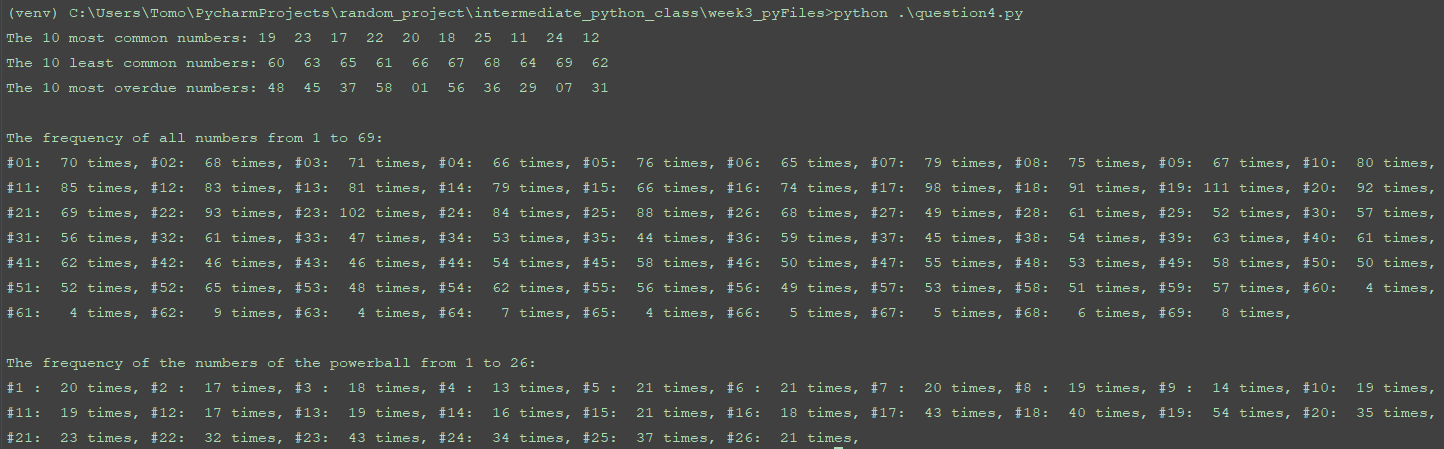
Write one Python program that work with this file to perform the following:

* Display the 10 most common numbers, ordered by frequency
* Display the 10 least common numbers, ordered by frequency
* Display the 10 most overdue numbers (numbers that haven’t been drawn in a long time), ordered from most overdue to least overdue
* Display the frequency of each number 1-69, and the frequency of each Powerball number 1-26

**>>CODE**

## PROBLEM #4  
# NAME: Tomoki Koike  
# DUE: Oct. 3, 2019  
# DESCRIPTION: This program will take in text file with data of the  
# PowerBall Lottery and there winning numbers of a specific period.  
# This program will manipulate the data to output the frequency of the  
# winning numbers and provide the most frequent numbers etc.  
##  
  
# import Modules  
**import** collections  
  
# Functions  
# Function to read the text file  
**def read\_file\_all(**file**):** # Opening the file to read  
 afile **=** open**(**file, **'r')** # Reading the first line  
 aline **=** afile.readline**()** # Initialize a list to contain all the values from the data  
 all\_nums **= []** # Loop to read all the lines  
 **while** aline **!= '':** # stripping the sentence end  
 aline **=** aline.rstrip**('\n')** # Splitting the values into individual strings  
 aline **=** aline.split**()** # Append to the list  
 **for** x **in** aline**:** all\_nums.append**(**x**)** # Read the next line  
 aline **=** afile.readline**()** # Closing the open file  
 afile.close**()  
 return** all\_nums  
  
# Function to read the text file  
**def read\_file\_powerball(**file**):** # Opening the file to read  
 afile **=** open**(**file, **'r')** # Reading the first line  
 aline **=** afile.readline**()** # Initialize a list to contain all the values from the data  
 powerball\_nums **= []** # Loop to read all the lines  
 **while** aline **!= '':** # stripping the sentence end  
 aline **=** aline.rstrip**('\n')** # Splitting the values into individual strings  
 aline **=** aline.split**()** # Append to the list  
 powerball\_nums.append**(**int**(**aline**[**5**]))** # Read the next line  
 aline **=** afile.readline**()** # Closing the open file  
 afile.close**()  
 return** powerball\_nums  
  
# Function to read the text file and create dict to find the overdues  
**def read\_file\_overdue(**file**):** # Opening the file to read  
 afile **=** open**(**file, **'r')** # Reading the first line  
 aline **=** afile.readline**()** # initializing the counter to check for the overdue  
 time **=** 654  
 adict **= {}** # Loop to read all the lines  
 **while** aline **!= '':** # stripping the sentence end  
 aline **=** aline.rstrip**('\n')** # Splitting the values into individual strings  
 aline **=** aline.split**()** # Calling the function to create dict to figure overdues  
 make\_dict\_overdue**(**aline, time, adict**)** # decrement time  
 time **-=** 1  
 # Read the next line  
 aline **=** afile.readline**()** # Closing the file  
 afile.close**()  
 return** adict  
  
# Function to create a dictionary for the frequency of the winning numbers  
**def make\_dict\_freq(**all\_nums**):** # Create a empty dict  
 adict **= {}** # Assigning the value in list as key  
 **for** x **in** range**(**len**(**all\_nums**)):** key **=** all\_nums**[**x**]  
 if** key **in** adict**:** adict**[**key**] +=** 1  
 **else:** adict**[**key**] =** 1  
 **return** adict  
  
# Function to create the overdue dict  
**def make\_dict\_overdue(**aline, time, adict**):  
 for** key **in** aline**:** adict.update**({**key**:** time**})  
 return** adict  
  
# Function to sort dict from small to large  
**def dict\_sort(**adict**):  
 return** collections.OrderedDict**(**sorted**(**adict.items**()**, key**=lambda** kv**:** kv**[**1**]))**# Function to sort dict in reverse large to small  
**def dict\_sort\_rev(**adict**):  
 return** collections.OrderedDict**(**sorted**(**adict.items**()**, key**=lambda** kv**:** kv**[**1**]**, reverse**=True))**# Function to print results  
**def output\_dict(**adict**):** counter **=** 0  
 **for** k **in** adict**:  
 if** counter **==** 10**:  
 break  
 else:** print**(**k, **' '**, end**='')** counter **+=** 1  
 **return**# Function to print out the dictionary with all he numbers as their frequency  
**def print\_freq(**adict**):** counter **=** 0  
 **for** x **in** sorted**(**adict**):** print**('#{0:<2}: {1:3} times, '**.format**(**x, adict**[**x**])**, end**='')** counter **+=** 1  
 **if** counter **%** 10 **==** 0**:** print**()  
 return**# Main function  
**def main():** all\_nums **=** read\_file\_all**('pbnumbers.txt')** overdue\_dict **=** read\_file\_overdue**('pbnumbers.txt')** freq\_dict **=** make\_dict\_freq**(**all\_nums**)** powerball\_nums **=** read\_file\_powerball**('pbnumbers.txt')** powerball\_freq **=** make\_dict\_freq**(**powerball\_nums**)** # Results  
 print**('The 10 most common numbers:'**, end**=' ')** output\_dict**(**dict\_sort\_rev**(**freq\_dict**))** print**()** print**('The 10 least common numbers:'**, end**=' ')** output\_dict**(**dict\_sort**(**freq\_dict**))** print**()** print**('The 10 most overdue numbers:'**, end**=' ')** output\_dict**(**dict\_sort\_rev**(**overdue\_dict**))** print**('\n')** print**('The frequency of all numbers from 1 to 69:')** print\_freq**(**freq\_dict**)** print**('\n')** print**('The frequency of the numbers of the powerball from 1 to 26:')** print\_freq**(**powerball\_freq**)**# Executing the main function  
**if** \_\_name\_\_ **== '\_\_main\_\_':** main**()**

**>>OUTPUT**



1. (**20 points**) Every character in the computer has an associated numeric code with it, called the ASCII code. E.g. the ASCII code for ‘1’ is 49, while for ‘A’ is 65. A table listing the characters and their respective code can be found here: http://www.asciitable.com/. This problem requires you to write a Python function encrypt(). It should do the following:

1. Ask the user to enter some text;

2. Ask the user to enter a natural number;

3. Compute the smallest primitive root of the number entered by the user: If a natural number n has a primitive root modulo n, then the root would be among the numbers 1 to n−1;

4. Divide this primitive root by 128 and take the remainder. This would be your key;

5. Use the computed primitive root to encrypt the entered text by adding the key to the ASCII of each of the characters in the text to get a new character. Return a string whose characters are these new characters.

E.g. For text ’Hi!’, and number 71:

The smallest primitive root of 71 is 7.

The remainder when 7 is divided by 128 is 7 itself, so our key is 7.

The ASCII values for ’H’,’i’ and ’!’ respectively are 72, 105 and 33.

Adding 7 to each of these results in 79, 112 and 40.

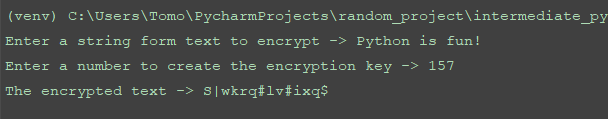
The characters whose ASCII values are 79, 112 and 40 respectively are ’0’, ’p’, and ’(’. The function should return ’0p(’

(**(Use ‘Python is fun!’, and number 157 to test your code)**

**>>CODE**

###  
# AUTHOR: Tomoki Koike  
# DATE: OCT. 3, 2019  
# DESCRIPTION: This program is designed to encrypte a text by finding the primiitivve roots and  
# processing the output with some couple of more steps. (without input validation)  
###  
  
# Main function  
**def main():** txt **=** input**('Enter a string form text to encrypt -> ')** num **=** int**(**input**('Enter a number to create the encryption key -> '))** encrypt**(**txt, int**(**num**))**# Importing modules  
**import** numpy **as** np  
  
# Functions  
# 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**)** phi **=** len**(**coprimes**)  
 return** phi  
  
# Function that computes the primitive root  
**def primitiveRoot(**anum**):** phi **=** totient**(**anum**)** # Obtain the Euler Phi function of the number  
 factors **=** prime\_factor**(**phi, prime\_nums**(**phi**))** # Obtain the prime factors  
 factors **=** set**(**factors**)** # Convert the list to a set to get rid of duplicate values  
 factors **=** list**(**factors**)** # Convert it back to list so it is immutable  
 # Loop to find the primitive root  
 **for** val **in** factors**:** ct **=** 0 # Initiate counter  
 **for** x **in** factors**:** up **=** phi **/** x  
 **if (**val**\*\***up**) %** anum **==** 1**:  
 break  
 else:** ct **+=** 1  
 **if** ct **==** len**(**factors**):** root **=** val  
 **break  
 return** root  
  
**def toAscii(**txt**):** txt\_list **=** list**(**txt**)** # Convert the string to a list  
 # A loop to convert each value in the text list into ascii value  
 **for** idx, val **in** enumerate**(**txt\_list**):** ascii\_val **=** ord**(**val**)** txt\_list**[**idx**] =** ascii\_val # Replacing the value with an ascii value  
 **return** txt\_list  
  
# The function that converts ascii values to characters  
**def asciiToValue(**ascii\_list**):** # A loop to convert ascii values back to values  
 **for** i, x **in** enumerate**(**ascii\_list**):** val **=** chr**(**x**)** ascii\_list**[**i**] =** val  
 **return** ascii\_list  
  
# The encrypt function  
**def encrypt(**txt, num**):** root **=** primitiveRoot**(**num**)** # Retrieving the primitive root of the input number  
 key **=** root **%** 128 # Get the remainder of primitive root modulus of 128  
 ascii\_list **=** toAscii**(**txt**)** # Get the list of txt converted into ascii values  
 new\_ascii\_list **=** list**(**np.array**(**ascii\_list**) +** key**)** new\_list **=** asciiToValue**(**new\_ascii\_list**)** # Converting the new ascii list to character list  
 new\_txt **= ''**.join**(**new\_list**)** # Print output  
 print**('The encrypted text -> {0}'**.format**(**new\_txt**))  
  
if** \_\_name\_\_ **== '\_\_main\_\_':** main**()**

**>>OUTPUT**



1. (**15 points**) In many context (e.g in some web forms), users must enter a phone number; and some of these irritate users by accepting only a specific format. Write a program by using regular expression that reads U.S. phone numbers with the three-digit area and seven-digit local codes accepted as ten digits, or separated into blocks using hyphens or spaces, and with the area code optionally encoded in parentheses. For example, all of these are valid: 555-123-1234, (555)1234567, (555) 123 1234 and 5551234567. Report an error for any that are invalid, or that don’t have exactly ten digits.

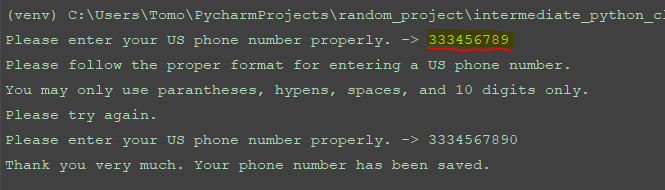
**(Use 333456789, (765)222-1111, (555) 333 2245, 765-123-4567, 8e4-3r4-3333 to test your code)**

**>>CODE**

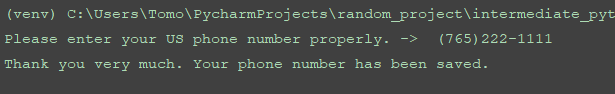
###  
# AUTHOR: Tomoki Koike  
# DATE: OCT. 3, 2019  
# DESCRIPTION: This program is designed to accept several formats of telephone numbers  
# and validate if it is in proper format or not.  
###  
  
# Main function  
**def main():** getValid**('Please enter your US phone number properly. -> ')  
 return**# Import modules  
**import** re  
  
# Functions  
**def getValid(**instruc**):** base **=** re.compile**(r'[^\d\s\(\)\-]')  
 while True:  
 try:** this **=** input**(**instruc**)  
 except** ValueError**:** print**('Cannot understand. Please enter valid input.')  
 continue** # Validating the phone number  
 **if** base.search**(**this**) or not(**10 **<=** len**(**this**) and** len**(**this**) <=** 14**):** print**('Please follow the proper format for entering a US phone number.')** print**('You may only use parantheses, hypens, spaces, and 10 digits.')** print**('Please try again.')  
 continue  
 else:** # Valid input  
 print**('Thank you very much. Your phone number has been saved.')  
 break**# Executing main  
**if** \_\_name\_\_ **== "\_\_main\_\_":** main**()**

**>>OUTPUT**

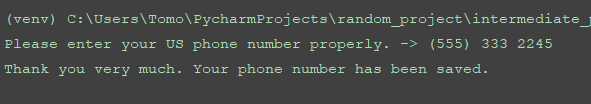
1. **333456789**



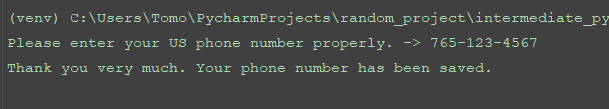
1. **(765)222-1111**



1. **(555) 333 2245**



1. **765-123-4567**



1. **8e4-3r4-3333**

