Introduction to programming

Homework 3

1. (Math: pentagonal numbers) A pentagonal number is defined as n(3n-1)/2 for and so on. So, the first few numbers are 1, 5, 12, 22, .... Write a function with the following header that returns a pentagonal number:

def getPentagonalNumber(n):

Write a test program that uses this function to display the first 100 pentagonal numbers with 10 numbers on each line.

1. (Palindrome integer) Write the functions with the following headers:

# Return the reversal of an integer, e.g. reverse(456) returns # 654

def reverse(number):

# Return true if number is a palindrome

def isPalindrome(number):

Use the reverse function to implement isPalindrome. A number is a palindrome if its reversal is the same as itself. Write a test program that prompts the user to enter an integer and reports whether the integer is a palindrome.

1. (Math: approximate the square root) There are several techniques for implementing the sqrt function in the math module. One such technique is known as the Babylonian function. It approximates the square root of a number, n, by repeatedly performing a calculation using the following formula:

nextGuess = (lastGuess + (n / lastGuess)) / 2

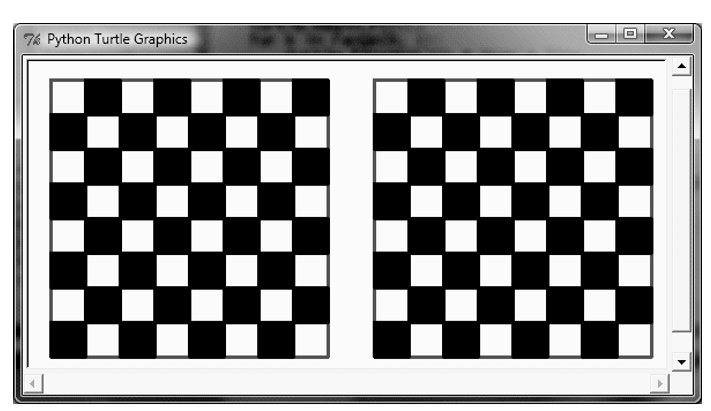
When nextGuess and lastGuess are almost identical, nextGuess is the approximated square root. The initial guess can be any positive value (e.g., 1). This value will be the starting value for lastGuess. If the difference between nextGuess and lastGuess is less than a very small number, such as 0.0001, you can claim that nextGuess is the approximated square root of n. If not, nextGuess becomes lastGuess and the approximation process continues. Implement the following function that returns the square root of n.

def sqrt(n):

1. (Turtle: two chessboards) Write a program that displays two chessboards, as shown in Figure. Your program should define at least the following function:

# Draw one chessboard whose upper-left corner is at # (startx, starty) and bottom-right corner is at (endx, endy)

def drawChessboard(startx, endx, starty, endy):



1. (Financial: credit card number validation) Credit card numbers follow certain patterns: It must have between 13 and 16 digits, and the number must start with:

■ 4 for Visa cards

■ 5 for MasterCard credit cards

■ 37 for American Express cards

■ 6 for Discover cards

In 1954, Hans Luhn of IBM proposed an algorithm for validating credit card numbers. The algorithm is useful to determine whether a card number is entered correctly or whether a credit card is scanned correctly by a scanner. Credit card numbers are generated following this validity check, commonly known as the Luhn check or the Mod 10 check, which can be described as follows (for illustration, consider the card number 4388576018402626):

* 1. Double every second digit from right to left. If doubling of a digit results in a two-digit number, add up the two digits to get a single-digit number.

Machine generated alternative text:
4388576018402626 
4*2- 
6*2- 
4*2: 
4 
4 
8 
2 
12 (1 +22 
10 (1 +0 z 
16 (1 +6 
8 
3) 
1) 
7) 

* 1. Now add all single-digit numbers from Step 1.

4 + 4 + 8 + 2 + 3 + 1 + 7 + 8 = 37

* 1. Add all digits in the odd places from right to left in the card number.

6 + 6 + 0 + 8 + 0 + 7 + 8 + 3 = 38

* 1. Sum the results from Steps 2 and 3.

37 + 38 = 75

* 1. If the result from Step 4 is divisible by 10, the card number is valid; otherwise, it is invalid. For example, the number 4388576018402626 is invalid, but the number 4388576018410707 is valid.

Write a program that prompts the user to enter a credit card number as an integer. Display whether the number is valid or invalid. Design your program to use the following functions:

# Return true if the card number is valid

def isValid(number):

# Get the result from Step 2

def sumOfDoubleEvenPlace(number):

# Return this number if it is a single digit, otherwise, return # the sum of the two digits

def getDigit(number):

# Return sum of odd place digits in number

def sumOfOddPlace(number):