// Description::

// To obtain the sqrt of the number in O(logn) complexity we can use binary search to approximate the square root. Initially the start index is set to 0 and end index to given number.Middle value initially set to 0 is then taking the middle of start and end index and checked if its square gives the given number with precision of 0.01. For numbers between 0 and 1 we take the start = given number and end = 1; output the middle value when precision < 0.01 which is the square root.

// Time Complexity::

//Since binary search is used to obtain the square root each time the size of the problem decreases by half. The Complexity thus obtained is O(logn). Other operation take constant time. So the whole algorithm takes complexity O(logn) where n is the given input

// Space Complexity

//Binary search takes space complexity O(1) in iterative approach so the whole algorithm takes O(1) space complexity

/\*

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*

Submission Date : 11/10/2017

Programmer Name : Prasad Kisan Tajane

Version : Initial

Logic Used : Divide and conquer technique is used to solve the algorithm.  
 Function starts with a pre-calculated maximum & minimum value and calculate mid

iteratively till the time absolute difference between mid-square and given number is

smaller than given threshold value.

Time Complexity : Since in every iteration we are calculating middle value (dividing the maximum range

by two) run time complexity of the program would be O(log n).

Space Complexity : As the program is not making use of any pre/custom defined data structure to save any

data, the space complexity of the program would be a constant. Thus, we can

represent it as O(1).

\*/

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*

import java.io.\*;

import java.util.\*;

import java.text.\*;

import java.math.\*;

import java.util.regex.\*;

public class Solution {

public static void main(String[] args) {

/\* Enter your code here. Read input from STDIN. Print output to STDOUT. Your class should be named Solution. \*/

// Took input through scanner class

Scanner scan = new Scanner(System.in);

double square = scan.nextDouble(); // Input number whose square root needs to be calculated

double sqrt; // output square root to be calculated

// start,middle and end are the values set during binary search

// precision is set to 0.01d

// accuracy is value of expression abs(q2 - n)

double start=0.0d, end = square,middle,precision = 0.01d,accuracy;

// if given number is 0 or 1 then this is the same value for sqrt

if (square == 0.0d || square == 1.0d){

sqrt = square;

}

else{

middle=0.0d; // initial value of middle set to 0

accuracy = precisionvalue(middle, square); // calls function which calculates abs(middle2 - square)

// if given number is between 1 and 0 then start is set to number and end to 1

if(square < 1 && square > 0) {

start = square;

end = 1;

}

// check accuracy less than 0.01

while(precision <= accuracy){

// set middle value as (start + end)/2

middle = (start + end)/2.0d;

// checked if given number is less than square of middle

if (square < (middle\* middle)){

end = middle;

}else{

start = middle;

}

accuracy = precisionvalue(middle, square); // calls function which calculates abs(middle2 - square)

}

sqrt = middle; // set the square root of the given number as output

}

System.out.println(sqrt);

}

// calculates the absolute value of a given number

public static double absolutevalue(double number){

double zero = 0.0d;

if (number<zero){

return -number;

}else{

return number;

}

}

// calculates expression abs(num1 - num2)

public static double precisionvalue(double num1, double num2){

double powervalue = num1 \* num1;

double accuracy;

accuracy = absolutevalue(powervalue-num2);

return accuracy;

}

}