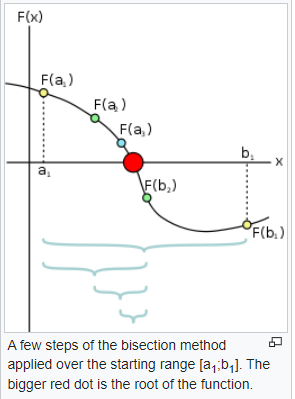
In [mathematics](https://en.wikipedia.org/wiki/Mathematics), the **bisection method** is a [root-finding method](https://en.wikipedia.org/wiki/Root-finding_algorithm) that applies to any [continuous functions](https://en.wikipedia.org/wiki/Continuous_function) for which one knows two values with opposite signs. The method consists of repeatedly [bisecting](https://en.wikipedia.org/wiki/Bisection) the [interval](https://en.wikipedia.org/wiki/Interval_(mathematics)) defined by these values and then selecting the subinterval in which the function changes sign, and therefore must contain a [root](https://en.wikipedia.org/wiki/Root_of_a_function). It is a very simple and robust method, but it is also relatively slow. Because of this, it is often used to obtain a rough approximation to a solution which is then used as a starting point for more rapidly converging methods. The method is also called the **interval halving** method, the [**binary search method**](https://en.wikipedia.org/wiki/Binary_search_algorithm), or the **dichotomy method**.



The method is applicable for numerically solving the equation *f*(*x*) = 0 for the [real](https://en.wikipedia.org/wiki/Real_number) variable *x*, where *f* is a [continuous function](https://en.wikipedia.org/wiki/Continuous_function) defined on an interval [*a*, *b*] and where *f*(*a*) and *f*(*b*) have opposite signs. In this case *a* and *b* are said to bracket a root since, by the [intermediate value theorem](https://en.wikipedia.org/wiki/Intermediate_value_theorem), the continuous function *f* must have at least one root in the interval (*a*, *b*).

At each step the method divides the interval in two by computing the midpoint *c* = (*a*+*b*) / 2 of the interval and the value of the function *f*(*c*) at that point. Unless *c* is itself a root (which is very unlikely, but possible) there are now only two possibilities: either *f*(*a*) and *f*(*c*) have opposite signs and bracket a root, or *f*(*c*) and *f*(*b*) have opposite signs and bracket a root.[[5]](https://en.wikipedia.org/wiki/Bisection_method#cite_note-5) The method selects the subinterval that is guaranteed to be a bracket as the new interval to be used in the next step. In this way an interval that contains a zero of *f* is reduced in width by 50% at each step. The process is continued until the interval is sufficiently small.

Explicitly, if *f*(*a*) and *f*(*c*) have opposite signs, then the method sets *c* as the new value for *b*, and if *f*(*b*) and *f*(*c*) have opposite signs then the method sets *c* as the new *a*. (If *f*(*c*)=0 then *c* may be taken as the solution and the process stops.) In both cases, the new *f*(*a*) and *f*(*b*) have opposite signs, so the method is applicable to this smaller interval.[[6]](https://en.wikipedia.org/wiki/Bisection_method#cite_note-6)

import random

a=float(input("Enter value of a "))

b=float(input("Enter value of b "))

c=float(input("Enter value of c "))

print()

print("The polynomial is",a,"x^2+",b,"x+",c)

def poly(a,b,c,x):

    v=float((a\*x\*x)+(b\*x)+c)

    return v

def ab(a,b,c):

    for i in range(2):

        count=0

        aint=1.0

        while(aint>0.0):

            #ax=float(input("Enter value for which f(x) is -ve "))

            ax=random.randint(-1000,1000)

            aint=poly(a,b,c,ax)

            count=count+1

        bint=-1.0

        while(bint<0.0):

            #bx=float(input("Enter value for which f(x) is +ve "))

            bx=random.randint(-1000,1000)

            bint=poly(a,b,c,bx)

            count=count+1

        cint=float((ax+bx)/2)

        while(poly(a,b,c,cint)!=0.0):

            if(poly(a,b,c,cint)>0):

                bx=cint

            if(poly(a,b,c,cint)<0):

                ax=cint

            cint=float((ax+bx)/2.0)

            count=count+1

        print("The root is=", cint)

        print("The number of iteration required are",count)

        print()

ab(a,b,c)

