Question

We are going to incrementally build a set of functions that will be used to solve quadratic equations, but only those with real roots. Remember that a quadratic equation has the general form of,

$$ax^2+bx+c=0 \qquad (1)$$

where x is a variable and a, b and c are constants (a and b are also called coefficients). To solve the equation we wish to find values of x, given a, b and c, that would satisfy equation *(1)* i.e. the left hand side of *(1)* would in fact evaluate to zero. There are many ways to solve quadratic equations, but we will implement the quadratic formula in this solution. We will use the constants as inputs and return the real root(s), if there are indeed any. Now the roots (solutions) are given by the equation

$$x=\frac{-b\pm\sqrt{b^2-4ac}}{2a} \qquad (2)$$

where the term

$$b^2-4ac \qquad (3)$$

is called the discriminant. Before we can use equation *(2)* let us create a procedure for finding square roots using Newton's method of successive approximations. This method finds the square root of a number (the radicand) by making a series of approximations or guesses. We discussed this in lecture. If we have a number n, and a guess for its square root, `sqr`, then we can make a better guess for `sqr` by finding the average of `sqr` and `n/sqr`. For example, if the number we wish to find the square root of is 2 (i.e. n=2) and our intial guess is 1 (i.e. `sqr`=1) then a better guess would be

$$(1 + (2\div1)) \div 2 = 1.5$$

repeating the procedure we get

$$(1.5 + (2\div1.5)) \div 2 = (1.5 + 1.3333) \div 2 = 1.4167 \qquad (4)$$

We can see that this number is approaching our expected value of 1.4142 (to 4 decimal places). You may have realized that we also need a way to determine when to stop the approximation procedure. We can do this by squaring the guess and finding the result of the difference of the square with the radicand. If the absolute value of the difference is within a specified range, then we stop the approximations. If we say the difference must be less than 0.0001 (our range) then using the result from *(4)*, the square of 1.4167 is 2.0070 and the difference is 0.007. This is greater than our range and therefore we should continue approximating.

(a) Write a function called `close()` that returns a boolean value. The value is determined by whether or not the result of the difference between two numbers is less than 0.0001 (therefore `close()` takes 2 arguments). You will need to use Python's inbuilt abs function.

[3 marks]

(b) Write a function called `sqroot()` which accepts two numbers, a radicand and a guess, and returns the approximated square root.

[4 marks]

(c) Write a function called `discriminant()` that takes three arguments i.e. the constants a,b and c of a quadratic equation, and returns the value of the discriminant (see statement *(3)* above).

[2 marks]

(d) Write a function called `getRt1()` that accepts three (3) arguments. These arguments are the constant *b*, the discriminant and a constant, `divConst`. The function should return the result (-*b* + √`discriminant`) ÷ `divConst`. Use the `sqroot()` function you wrote earlier and let your initial guess be 1.

[2 marks]

(e) Write a function called `getRt2()` that accepts three (3) arguments. These arguments are the constant *b*, the discriminant and `divConst`. The function should return the result of (-*b* - √discriminant) ÷ `divConst`. Again, use the `sqroot()` function you wrote earlier and let your initial guess be 1.

[2 marks]

(f) Write a function called `solveQuad()` that accepts three arguments, the constants *a*, *b* and *c*. It returns a result based on the value of the discriminant. If the discriminant is greater than zero it displays the roots given by `getRt1()` and `getRt2()`, if it is equal to zero it displays the root given by the formula,

`-(b ÷ divConst)`

If it is less than zero it displays a message indicating there are no real roots to the equation. This function must also set the value of `divConst` to (2 x *a*) (i.e. the denominator in equation *(2)*).

[7 marks]

Question

a)

Write a function called `myZip` that accepts two lists and returns a new list. The new list is comprised of tuples of respective elements in each of the provided lists.

```
e.g. >>> myZip([1,2,3],[4,5,6])
     [(1, 4), (2, 5), (3, 6)]
     >>>
```

[6]

b) Write a function called `isOdd` that accepts a positive integer and returns `True` if the integer is odd, and `False` otherwise.

```
e.g. >>> isOdd(4)
     False
     >>>
```

[2]

c) Using `isOdd` write an *iterative* function called `split` that takes a list of positive integers and returns a tuple of two lists; one a list of the odd integers, the other a list of the even integers.

```
e.g. >>> split([1,2,3,4,5,6])
     ([1, 3, 5], [2, 4, 6])
     >>>
```

[5]

d) Complete the following function `power` to calculate the power of a given number. A helper function `pow_helper` maintains the state of the computation. The power of a number x raised to y, is calculated as follows:

$$x^{y} = x * x^{y-1}$$

Below are step and result values for calculating 2^{3}

Step	result
0	1
1	2
2	4
3	8

[5]