;IntroCS pd4 sec4
;Team name: The Rear End
;Written by Noirit Sami, Preston Chang, Leonid Metlitsky
;lab0- Approaching Greatness
;2022-10-27
;time cost: 5 (hours)

1. The calculator takes the information of the left bound, right bound, and the guess respectively, and uses it to approximate the square root. The order matters here because then the calculator would only have to check values within the set parameters in order to calculate the root.

2. The algorithm brings you kind-of close/closer to the square root of n.
3.

```
(define foo
  (lambda (n g)
     (/(+ g (/ n g)) 2)))

(foo 4 2) "...should be 2"
(foo 4 1) "...should be 2.5"
(foo 4 5) "...should be 2.9"
(foo 9 3) "...should be 3"
(foo 9 1) "...should be 5"
(foo 9 10) "...should be 5.45"
(foo 16 4) "...should be 4"
(foo 16 1) "...should be 8.5"
(foo 16 15) "...should be 8.03"
(foo 16 900) "...should be 450.008"
(foo 10 2) "...should be 3.5"
(foo 12 2) "...should be 4"
```

4. In essence, the above function takes a guess and brings it a bit closer to the true square root of n.
5.

```
(define epsilon 0.1)

(define isGoodEnuf?
```

```
  (lambda (a b)
     (< (abs (- a b)) epsilon)))

(define mySqrtHelper
  (lambda (n g)
     (if (isGoodEnuf? n (* g g))
     g
     (mySqrtHelper n (foo n g) ))))

(define mySqrt
  (lambda (n)
     (mySqrtHelper n (/ n 3))))

(mySqrt 169) "...should be ~13"
(mySqrt 1234) "...should be ~35.1"
(mySqrt 4) "...should be ~2"
(mySqrt 16) "...should be ~4"
(mySqrt 144) "...should be ~12"
```

6.

We understood that when the foo function takes the number and the guess of its square root, it returns a number that gets progressively closer to the actual square root of the number. Recursing the foo function would eventually lead to an approximation of square root. Looking at the mySqrt function, we saw that it only has one input but still returns an approximation of the square root. Thus, we concluded that we just needed to rewrite the guess in terms of the number provided.

The isgoodenuf? Function takes the difference between two inputs and checks if it is less than 0.1 or not. If it is less, then the function will return as true. This means that we can use isgoodenuf? to check if the guess^2 is close to the original number.
Knowing this we created a recursive helper function in order to get an approximation of the square root. If the guess “isgoodenuf?” it will simply return the guess. If not, then it will recurse the function with the starting number and the *closer guess* until the disparity is small enough for the inputs to lead to the base case, which is just the guess.

With this helper function defined, our final job was to find a starting guess and write it in terms of the starting number. We messed around here for a bit, dividing the starting number by numbers 2-10 to create our initial guess, and then inputting those numbers into the helper function which is incorporated into the mySqrt function.

In the end, we just chose a number between 2-10 to divide the initial number by, as all of these numbers still return a value close to the actual square root. As seen above, we chose the number 3 to divide by n. So the helper function, which was designed to take 2 inputs, the number and the guess, takes in the number and the number divided by three as the guess. Ultimately, it returns an approximation of the square root.