Lexical Scope

In this lesson, you will learn about lexical scoping and how it can be used to remove redundancies in your code.

In our program for computing the square root of a number, we are using the variable `x` over and over again. We are defining it in each function even though it's the same variable throughout the program.

In a previous lesson, we learned that definitions outside a block are visible to definitions inside a block unless they are being shadowed. The process of setting the scope of a variable is known as **Lexical scoping** and we can use this convention to remove the redundant `x` variables in our square root program.

Let's first take a look at the original program.

This code requires the following environment variables to execute:

LANG	C.UTF-8

```
def abs(x: Double) =
  if (x < 0) -x else x

def sqrt(x: Double) ={
  def sqrtIter(guess: Double, x: Double): Double =
    if (isGoodEnough(guess, x)) guess
    else sqrtIter(improve(guess, x), x)

  def isGoodEnough(guess: Double, x: Double) =
    abs(guess * guess - x) / x < 0.0001

  def improve(guess: Double, x: Double) =
    (guess + x / guess) / 2

  sqrtIter(1.0, x)
}

//Driver Code
println(sqrt(4))
```

We can see that the `x` parameter is being duplicated in the `sqrtIter` function, `isGoodEnough` function and `improve` function, but is never being modified in any of the functions. We can simply remove it from these occurrences and also eliminate

the corresponding parameters.

```
def abs(x: Double) =
  if (x < 0) -x else x

def sqrt(x: Double) ={
  def sqrtIter(guess: Double): Double =
    if (isGoodEnough(guess)) guess
    else sqrtIter(improve(guess))

  def isGoodEnough(guess: Double) =
    abs(guess * guess - x) / x < 0.0001

  def improve(guess: Double) =
    (guess + x / guess) / 2

  sqrtIter(1.0)
}
```

Let's call the `sqrt` function and see if we get the expected output.

This code requires the following environment variables to execute:

LANG	C.UTF-8

```
def abs(x: Double) =
  if (x < 0) -x else x

def sqrt(x: Double) ={
  def sqrtIter(guess: Double): Double =
    if (isGoodEnough(guess)) guess
    else sqrtIter(improve(guess))

  def isGoodEnough(guess: Double) =
    abs(guess * guess - x) / x < 0.0001

  def improve(guess: Double) =
    (guess + x / guess) / 2

  sqrtIter(1.0)
}

val result = sqrt(4)
println(result)
```

It works! We were able to remove all the redundancies of our code without it affecting the output.

In the next lesson, we will change direction a bit and look at tail recursion.