CSC 120 (R Section)— Lab Exercise 10

This is a non-credit exercise, which you do not hand in.
You may work on your own or together with another student, as you please.

In this lab, you will warm up with general practice writing a simple function.

Next, you will practice writing a function in two ways, with a loop and using vector operations, and in devising test cases that check whether it works.

Finally, you will modify the set of functions that take an object-oriented approach to drawing things used as an example in the Week 11 lectures, changing the way that boxes are represented, and then adding a new class of object.

**Practice writing a simple function.**

Write a function called `multiply_above_diagonal` that takes as its only argument a square matrix (you don't have to check this), and returns a single number that is the product of all the elements in the matrix that are above the diagonal (ie, all those in positions where the column index is greater than the row index). Note that the product of zero factors is defined to be one.

Here is the correct output for two test cases:

```
> Y <- matrix (1:16, nrow=4, ncol=4)
> Y
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16
> multiply_above_diagonal(Y)
[1] 1228500
> multiply_above_diagonal(matrix(5,nrow=1,ncol=1))
[1] 1
```

**Writing a function in two ways, and testing it.**

Write a function called `is.not.decreasing` that takes a numeric vector as its only argument (you can assume it's a numeric vector without checking) and returns TRUE if this vector is in non-decreasing order and FALSE if at least one element is less than an earlier element. You may assume that none of the elements are NA.

There are several ways of writing such a function. Perhaps the most straightforward way is to use a loop over the elements of the vector, but it is also possible to use vector operations rather than a loop.

You should try using a loop first. A `while` loop may be appropriate, and using a variable called `answer_is_known` holding a logical value may be useful in deciding when the loop should end (or there are other ways to do something similar).

Then see if you can do it without a loop, using comparison operations on vectors.

Finally, can you think of a third, rather different, way to write this function?

You should also come up with good test cases to see if the various versions of your function work. Try some typical cases, and also some peculiar cases, where your function might fail even it if usually works.

**Extending the example from Week 11's lecture.**

Finally, you should extend the example of object-oriented programming in the Week 11 lecture slides, by changing the representation of box objects, and adding a new "plus" object.

The functions from the lecture slides are available at

`http://www.cs.utoronto.ca/~radford/csc120/drawfuns.r`

and the script that was used (in slightly different form) to produce the results in the lecture slides is at

`http://www.cs.utoronto.ca/~radford/csc120/drawscript.r`

As a first exercise, modify the functions for the "box" class to represent a box by a list whose elements are the same as the arguments of the `new_box` function. This may require changing `new_box`, `draw.box`, `rescale.box`, and `translate.box`. But it should not require any change to the R code that uses boxes, in the script file.

As a second exercise, try to add a new "plus" object, which looks like a plus sign. You should write a `new_plus` function that creates an object of class `"plus"`, given the $x$ and $y$ coordinates of the centre of the plus (where the two lines cross) and how far the ends of the plus extend away from the centre (the same for the horizontal and vertical lines). You should also write a `draw.plus` function to provide a `draw` method. You should then be able to try drawing plusses. Once you have this working, you can write `rescale.plus` and `translate.plus` methods for objects of this class, and try them out. You should also be able to call the `smaller` function from the lecture with a `plus` object as its argument.

An extra bonus exercise: Implement a new generic function for these objects, called `rotate90`, that rotates an object by 90 degrees counterclockwise. You may find it convenient to define a default method for this generic function. For which classes of objects could you use the default? For which will you need to write a special method?