1. Create a function that receives two numbers: \(a\) and \(b\). The function calculates and returns the multiplication of all the numbers between \(a\) and \(b\). Create three versions of this function. The first version using a \textit{while} loop. The second version using a \textit{for} loop. The third using \textit{recursion}. Define all three versions of the function within the same script. Call them \(\text{multW}(a,b)\), \(\text{multF}(a,b)\), and \(\text{multR}(a,b)\), respectively.

\textit{Tip: This function is very similar to the factorial.}

2. Write a function that takes a string as a parameter and returns \textit{True} if the string is a palindrome, \textit{False} otherwise. Remember that a string is a palindrome if it is spelled the same both forward and backward. Create again three versions of the program. The first version using a \textit{while} loop. The second version using a \textit{for} loop. The third version using \textit{recursion}. Do not use the built-in \texttt{reverse()} method in any of the versions. Same as before, define all three versions of the function within the same script. Call them \(\text{ispalinW}(\text{word})\), \(\text{ispalinF}(\text{word})\), and \(\text{ispalinR}(\text{word})\), respectively.

\textit{Tip for the recursive version: To get the first letter and the last letter of the word you can use the 0 index and the -1 index: \texttt{word[0]}, \texttt{word[-1]}. You can use this to test whether the first and the last letters of the current word are equal or not. Then, to reduce the word, you will probably want to take out the first and last letter. You can use slicing: \texttt{word[1:-1]}. If you have a word \texttt{w = ‘abcde’} then \texttt{w[1:-1]} is ‘bcd’ Once you confirm that the first and last letter of the current word are the same, then you can call the program recursively using the word without the first and last letters.}

3. [Optional] Write a recursive function that solves the \textit{Towers of Hanoi}. There are three rods (A, B, and C) and a number of disks (\(N\)) of different sizes (where 1 is the smallest, topmost and \(N\) is the largest, bottommost) which can slide onto any rod. To begin, the disks are neatly stack in ascending order of size on one rod, the smallest at the top, thus making a conical shape. The objective of the puzzle is to move the entire stack from A to C, obeying the following rules: (1) Only one disk may be moved at a time; (2) Each move consists of taking the upper disk from one of the rods and sliding it onto another rod, on top of the other disks that may already be present on that rod. (3) No disk may be placed on top of a smaller disk. The output of your algorithm should be the series of steps required to move \(N\) discs from peg A to peg C, for any number of disks, \(N\). Use that function to calculate how long it would take our monk friend to move 13 rocks, if each move takes him/her one second.