Recursion is a topic that can be quite confusing for new programmers. For the majority of us we began learning about and solving problems in computer science by using loops, so when you first encountered a function calling itself your world might’ve been turned upside down. How could this problem possibly be solved by calling the same function over and over again?

Recursion works by reducing the size of the problem on every subsequent call of the function. For example: during the merge sort algorithm the sort function is called within itself twice. The first call is passed the left half of the array, and the second call is passed the right. Each instance now has half as much work to do. This chain of halving the required work continues until an easily solvable subset of the problem is reached (in this case, sorting just two numbers). This subset of the initial problem is the base case, and when it is reached, we return instead of continuing to recurse. Without the base case the program would run infinitely, run out of memory, or crash, similar to when you forget to add a termination condition in a ‘while’ loop. For each call, the state of the program and its variables at the time of calling are stored on the call stack. Once the base case is reached, solutions start returning down the stack to the instance that called it, giving us the solution to the original problem.

The advantages of recursion are that it can possibly shave off runtime at the cost of memory, and it can require less lines of code than its equivalent iterative solution. Memory overhead can be minimized by adding a separate structure to keep track of already found solutions. The next time you come across a problem with a loop, see if you can solve it with recursion instead!