In computer science, recursion is a method of solving a computational problem where the solution depends on solutions to smaller instances of the same problem.Recursion solves such recursive problems by using functions that call themselves from within their own code. The approach can be applied to many types of problems, and recursion is one of the central ideas of computer science.

The power of recursion evidently lies in the possibility of defining an infinite set of objects by a finite statement. In the same manner, an infinite number of computations can be described by a finite recursive program, even if this program contains no explicit repetitions.
— Niklaus Wirth, Algorithms + Data Structures = Programs, 1976

Most computer programming languages support recursion by allowing a function to call itself from within its own code. Some functional programming languages (for instance, Clojure) do not define any looping constructs but rely solely on recursion to repeatedly call code. It is proved in computability theory that these recursive-only languages are Turing complete; this means that they are as powerful (they can be used to solve the same problems) as imperative languages based on control structures such as while and for.

Repeatedly calling a function from within itself may cause the call stack to have a size equal to the sum of the input sizes of all involved calls. It follows that, for problems that can be solved easily by iteration, recursion is generally less efficient, and, for certain problems, algorithmic or compiler-optimization techniques such as tail call optimization may improve computational performance over a naive recursive implementation.

A common algorithm design tactic is to divide a problem into sub-problems of the same type as the original, solve those sub-problems, and combine the results. This is often referred to as the divide-and-conquer method; when combined with a lookup table that stores the results of previously solved sub-problems (to avoid solving them repeatedly and incurring extra computation time), it can be referred to as dynamic programming or memoization.

A recursive function definition has one or more base cases, meaning input(s) for which the function produces a result trivially (without recurring), and one or more recursive cases, meaning input(s) for which the program recurs (calls itself). For example, the factorial function can be defined recursively by the equations $0! = 1$ and, for all $n > 0$, $n! = n(n - 1)!$. Neither equation by itself constitutes a complete definition; the first is the base case, and the second is the recursive case. Because the base case breaks the chain of recursion, it is sometimes also called the "terminating case".

The job of the recursive cases can be seen as breaking down complex inputs into simpler ones. In a properly designed recursive function, with each recursive call, the input problem must be simplified in such a way that eventually the base case must be reached. (Functions that are not intended to terminate under normal circumstances—for example, some system and server processes—are an exception to this.) Neglecting to write a base case, or testing for it incorrectly, can cause an infinite loop.

Many computer programs must process or generate an arbitrarily large quantity of data. Recursion is a technique for representing data whose exact size is unknown to the programmer: the programmer can specify this data with a self-referential definition. There are two types of self-referential definitions: inductive and coinductive definitions.
For some functions (such as one that computes the series for $e = 1/0! + 1/1! + 1/2! + 1/3! + ...$) there is not an obvious base case implied by the input data; for these one may add a parameter (such as the number of terms to be added, in our series example) to provide a 'stopping criterion' that establishes the base case. Such an example is more naturally treated by corecursion,[how?] where successive terms in the output are the partial sums; this can be converted to a recursion by using the indexing parameter to say "compute the nth term (nth partial sum)".