1. [CS106B Introduction to Recursion - Stanford University](https://web.stanford.edu/class/archive/cs/cs106b/cs106b.1206/lectures/intro-to-recursion/?form=MG0AV3" \t "_blank)

The problem of solving the Towers of Hanoi puzzle is a classic example often used to illustrate the power of recursion. The objective is to move a stack of disks from one pole to another, adhering to specific rules such as only moving one disk at a time and never placing a larger disk on top of a smaller one. Recursion simplifies this problem by breaking it down into smaller, more manageable subproblems. Essentially, it involves moving the top n-1 disks to an auxiliary pole, then moving the nth disk to the target pole, and finally moving the n-1 disks from the auxiliary pole to the target pole.

An iterative solution to the Towers of Hanoi puzzle would involve using a stack data structure to keep track of the moves. This approach simulates the recursive process manually by continuously pushing and popping the state of the disks and the poles onto the stack, ensuring the rules are followed without directly invoking recursion. This method can be more complex to implement but offers a different perspective on solving the problem.

1. [How Recursion Works - Explained with Flowcharts and a Video](https://www.freecodecamp.org/news/how-recursion-works-explained-with-flowcharts-and-a-video-de61f40cb7f9/?form=MG0AV3)

The problem of finding a key hidden in nested boxes is another excellent example of recursion's utility. The task requires searching through multiple layers of boxes until the key is found. Recursion naturally fits this scenario, as it allows a function to call itself to continue the search in each nested box. This self-referential approach simplifies the problem by breaking it down into smaller, more manageable pieces, enabling a straightforward and elegant solution.

An iterative solution, on the other hand, would involve using a loop to examine each box sequentially. This method would likely use a queue or stack to keep track of the boxes that need to be searched. By managing this stack or queue, the algorithm can systematically explore each nested level without the need for recursive function calls. While this approach may be less intuitive for problems naturally suited to recursion, it provides a viable alternative by manually controlling the flow of the search process.

1. [Recursion in Programming - Course](https://www.youtube.com/watch?v=IJDJ0kBx2LM&form=MG0AV3)

The problem of calculating the Fibonacci sequence is another classic example where recursion shines. The Fibonacci sequence is a series of numbers where each number is the sum of the two preceding ones, typically starting with 0 and 1. The sequence progresses as 0, 1, 1, 2, 3, 5, 8, 13, and so forth. Recursion is particularly suitable for this problem because each term in the sequence can be naturally defined in terms of the two preceding terms. A recursive function can be employed to call itself to compute the next term in the sequence, continuing this process until the base cases (the first two terms) are reached.

An iterative solution to the Fibonacci sequence would use a loop to calculate the Fibonacci numbers. Starting with the first two terms, the loop would iteratively compute each subsequent term by adding the two previous terms together until the desired position in the sequence is achieved. While recursion offers a more elegant and intuitive solution for this type of problem, iteration can often be more efficient in terms of memory and execution time.