# PRELIMINARY: RECURSIVE CODE STRUCTURE

```
import sys
sys.setrecursionlimit(10000)   # set recursion-stack depth


def f(arg, ... ):
   if terminating_condition_based_on_arguments’_values :
       return terminal_value   # could be conditional if needed
   else:
       function body that contain recursive calls
       return return_value     # could be conditional if needed


# main code
x = f( initial_arguments )
```

## PROBLEM A: COMBINATION

Most computational problem requires a solution that is a certain combination of state variables, or some quantitative value derived from such a combination.

For an $n$-bit binary number, there are $2^n$ possible combinations, as each digit can be either of the two options; 0 or 1.

Theoretically, a state of every computational problem can be modelled as a sequence of binary digits.

Therefore, generating combinations of options is a core of algorithmic solving technique, when solution to the problem requires selecting an option for every output component.

*Brute force search* for problem state that matches certain characteristics is to generate all possible states i.e. all possible combinations of state variables, **then** test each combination against the specified criteria.

### TASK 1: GENERATING ALL COMBINATIONS

**INPUT:** The number of items to be picked, n.

**OUTPUT:** All the combinations of n items.

**EXAMPLE**

| INPUT | OUTPUT |
|---|---|
| 3 | 0 0 0<br>0 0 1<br>0 1 0<br>0 1 1<br>1 0 0<br>1 0 1<br>1 1 0<br>1 1 1 |

1) Create a **global** list of length n, probably **name it x**. This list will store a partial or complete combination.

2) Develop a recursive function comb(i) which takes an index i, assigns the ith item of the combination based on the possible options, and recursively generates the rest of the combination contents. Print a combination each time that it is completely assigned.

   HINT
   - The initial call is comb(0); 0 is index of the first digit of the combination.
   - Options for selecting an item are “not selected” (0) and “selected” (1)

   NOTE: It is adequate to test with n ≤ 4

## TASK 2: COUNTING NUMBER OF COMBINATIONS

3) Modify comb(i) to return the sum of the two recursive calls that it generates. Then comb(n) returns 1 once it prints a complete combination. Consequently, the value of comb(0) will be the total number of combinations that are generated, which must be $\mathbf{2^n}$.

4) ***Modify the program from step 3*** according to the following description.
   - The program accepts an additional input; an integer k, k ≤ n.
   - The program prints the number of combinations that contain **exactly k 1's**. The output must always be equal to C(n, k) = $\frac{n!}{(n-k)!k!}$
   - IMPORTANT: Keep k as a global variable. Its value never change so there is no point to pass its value to each copy of the recursive function call.

<u>Extra step only for students who finish the first four steps in one hour</u>

By passing an additional argument, s, which is the sum of 1's so far, comb(i, s) can be made to run significantly faster than comb(i) according to the following concept.

1) Whenever s is k, the recursion can terminate right away. The only valid assignment for the rest of the items must be 0 in order to be counted.
2) Whenever the value of n-i + s is k, the only valid assignment for the rest of the items, exactly n-i of them, must be all 1's in order to be counted. So the recursion can terminate as well.
3) Whenever the value of n-i + s is less than k, it becomes impossible to have k 1's from any of further item assignments. Therefore, the recursion can also terminate.

A termination in these cases implies returning an appropriate value.

Terminate recursion when no further recursive call will result in feasible answer is called "pruning".

5) Modify the program **from step 3** to the count all combinations when there are 3 options to be selected for each item. The total number of combinations generated must be $3^n$.

## PROBLEM B: BALANCE SPLIT

You have a number of goods with known values $v_1, \ldots, v_n$. You want to split the goods into two groups such that the difference of the total values between the two groups is minimal. Write a program that compute such a minimal difference.

**INPUT:**

Input contains n values of goods $v_1, \ldots, v_n$ (integers, $1 \le v_i \le 100000$) delimited by white spaces, $1 \le n \le 10$.

**OUTPUT:**

Your program should output a number representing the minimal possible difference between the total values of the two groups.

**EXAMPLE**

| INPUT | OUTPUT |
|---|---|
| 5 8 13 27 14 | 3 |

1) Apply the brute-force recursive search technique to solve this problem. Test your program against the provided test cases (balanceSplit.zip).

   **HINT**

   - n goods can be represented by an n-bit binary number.
   - For each possible solution, a good belongs to either group 0 or group 1.

2) For a student who is fast enough to reach this point before the end of the class, the following enhancement is recommended.
   The sum of values in group 1 can be passed along the recursive call as an additional argument. This way, there will be no need to loop summing values of either group at the end, effectively reduce the running time from $O(n.2^n)$ to $O(2^n)$ … why?