MEMOIZATION TECHNIQUE
(For speeding up a brute-force code)

Start with the brute-force program for rod cutting problem (from the last class)

```
import sys
sys.setrecursionlimit(10000)

p = input().split()
L = len(p)
for i in range(L):
    p[i] = int(p[i])

p.insert(0,0)

def maxRev(l):
    global p,L

    if l == 0:
        return 0
    else:
        mx = 0
        for i in range(1,l+1):
            mx = max(mx, p[i]+maxRev(l-i))
    return mx

print(maxRev(L))
```

1. Create a list named "calls" with size equal to 1 + the largest possible rod length. Initialize all the values in the list to 0. (This requires only basic Python knowledge)

2. Modify the brute-force solution so that the program collects the total number of recursive calls, i.e. increment calls[l] for every call to $\text{maxRev}(l), 1 \leq l \leq L$. Observe how many times each $\text{maxRev}()$ is called by examining the final values in "calls" list.

3. For *each* value of l, does $\text{maxRev}(l)$ return the same or different values ?
   maxRev(l) will always be the same, the max cannot change.
4. Therefore, for length of L, at most how many calls to all $\text{maxRev}()$'s should be adequate ?
   at most the amount of calls equals to l or l+1.
5. What is the reason that there were too many calls to $\text{maxRev}()$ in the brute-force solution ?

6. Suggest a modification to the brute-force solution such that once a value of an $\text{maxRev}(l)$ is already computed, it will never have to be recomputed again. Only idea is needed in this step.

7. Implement the modified solution.

8. Observe how faster the algorithm becomes (comparing the total number of recursive calls).

9. Apply the same concept to speed up the minimum coin change problem. (last class)

```
import sys
sys.setrecursionlimit(10001)

c = input().split()
for i in range(len(c)):
    c[i] = int(c[i])

V = int(input())

def mincoin(v):
    global c

    if v == 0:
        return 0
    else:
        minc = 10000000000
        for x in c:
            if x <= v:
                minc = min(minc, 1 + mincoin(v-x))
        return minc

print(mincoin(V))
```

10. If done correctly, the memoized version (of solution for minimum coin change) will be able to handle the 2nd test case in the example.