# Dynamic Programming Patterns with Code Snippets & Example Problems

## Fibonacci / Climbing Stairs

Problem: Given n stairs, you can climb 1 or 2 at a time. Find number of ways.
Code:

```
int climbStairs(int n){
    vector<int> dp(n+1,0);
    dp[0]=1; dp[1]=1;
    for(int i=2;i<=n;i++) dp[i]=dp[i-1]+dp[i-2];
    return dp[n];
}
```

## House Robber (1D DP)

Problem: Max sum of non-adjacent houses.
Code:

```
int rob(vector<int>& nums){
    int n=nums.size();
    if(n==0) return 0;
    if(n==1) return nums[0];
    vector<int> dp(n,0);
    dp[0]=nums[0]; dp[1]=max(nums[0],nums[1]);
    for(int i=2;i<n;i++) dp[i]=max(dp[i-1],dp[i-2]+nums[i]);
    return dp[n-1];
}
```

## Minimum Path Sum in Grid (2D DP)

Problem: Find min path sum from top-left to bottom-right in a grid.
Code:

```
int minPathSum(vector<vector<int>>& grid){
    int m=grid.size(),n=grid[0].size();
    vector<vector<int>> dp(m,vector<int>(n,0));
    dp[0][0]=grid[0][0];
    for(int i=1;i<m;i++) dp[i][0]=dp[i-1][0]+grid[i][0];
    for(int j=1;j<n;j++) dp[0][j]=dp[0][j-1]+grid[0][j];
    for(int i=1;i<m;i++){
        for(int j=1;j<n;j++){
            dp[i][j]=grid[i][j]+min(dp[i-1][j],dp[i][j-1]);
        }
    }
    return dp[m-1][n-1];
}
```

## 0/1 Knapsack

Problem: Max value with weight constraint.
Code:

```
int knapsack(vector<int>& wt, vector<int>& val, int W){
    int n=wt.size();
    vector<int> dp(W+1,0);
    for(int i=0;i<n;i++){
        for(int w=W;w>=wt[i];w--){
            dp[w]=max(dp[w],dp[w-wt[i]]+val[i]);
        }
    }
```

```
    return dp[W];
}
```

## Longest Increasing Subsequence (LIS)

```
Problem: Find length of LIS.
Code:
int lengthOfLIS(vector<int>& nums){
    int n=nums.size();
    vector<int> dp(n,1);
    int ans=1;
    for(int i=0;i<n;i++){
        for(int j=0;j<i;j++){
            if(nums[i]>nums[j]) dp[i]=max(dp[i],dp[j]+1);
        }
        ans=max(ans,dp[i]);
    }
    return ans;
}
```

## Longest Common Subsequence (LCS)

```
Problem: Find length of LCS of two strings.
Code:
int LCS(string a,string b){
    int n=a.size(),m=b.size();
    vector<vector<int>> dp(n+1,vector<int>(m+1,0));
    for(int i=1;i<=n;i++){
        for(int j=1;j<=m;j++){
            if(a[i-1]==b[j-1]) dp[i][j]=1+dp[i-1][j-1];
            else dp[i][j]=max(dp[i-1][j],dp[i][j-1]);
        }
    }
    return dp[n][m];
}
```

## Coin Change (Counting Ways)

```
Problem: Number of ways to make sum.
Code:
int change(int amount, vector<int>& coins){
    vector<int> dp(amount+1,0);
    dp[0]=1;
    for(int c:coins){
        for(int j=c;j<=amount;j++) dp[j]+=dp[j-c];
    }
    return dp[amount];
}
```

## Longest Palindromic Substring

```
Problem: Find longest palindromic substring.
Code:
string longestPalindrome(string s){
    int n=s.size();
    vector<vector<bool>> dp(n,vector<bool>(n,false));
    int start=0,maxLen=1;
    for(int i=0;i<n;i++) dp[i][i]=true;
    for(int len=2;len<=n;len++){
        for(int i=0;i+len-1<n;i++){
            int j=i+len-1;
```

```
            if(s[i]==s[j] && (len==2 || dp[i+1][j-1])){
                dp[i][j]=true;
                if(len>maxLen){start=i;maxLen=len;}
            }
        }
    }
    return s.substr(start,maxLen);
}
```