AI Lab 02
Section D

Fibonacci Series:

A Fibonacci sequence is the integer sequence of 0, 1, 1, 2, 3, 5, 8....
The first two terms are 0 and 1. All other terms are obtained by adding the preceding two terms. This means to say the nth term is the sum of (n-1)th and (n-2)th term.

There are multiple ways to display Fibonacci series up to nth number:

1. Using Loop:
2. By Recursion
3. By Dynamic Programming

Just by using while loop, we can easily print the series, but the method is not much efficient.

```
# Program to display the Fibonacci sequence up to n-th term

nterms = int(input("How many terms? "))

# first two terms
n1, n2 = 0, 1
count = 0

# check if the number of terms is valid
if nterms <= 0:
   print("Please enter a positive integer")
# if there is only one term, return n1
elif nterms == 1:
   print("Fibonacci sequence upto",nterms,":")
   print(n1)
# generate fibonacci sequence
else:
   print("Fibonacci sequence:")
   while count < nterms:
       print(n1)
       nth = n1 + n2
       # update values
       n1 = n2
       n2 = nth
       count += 1
```

We can also solve the same problem by Recursion:

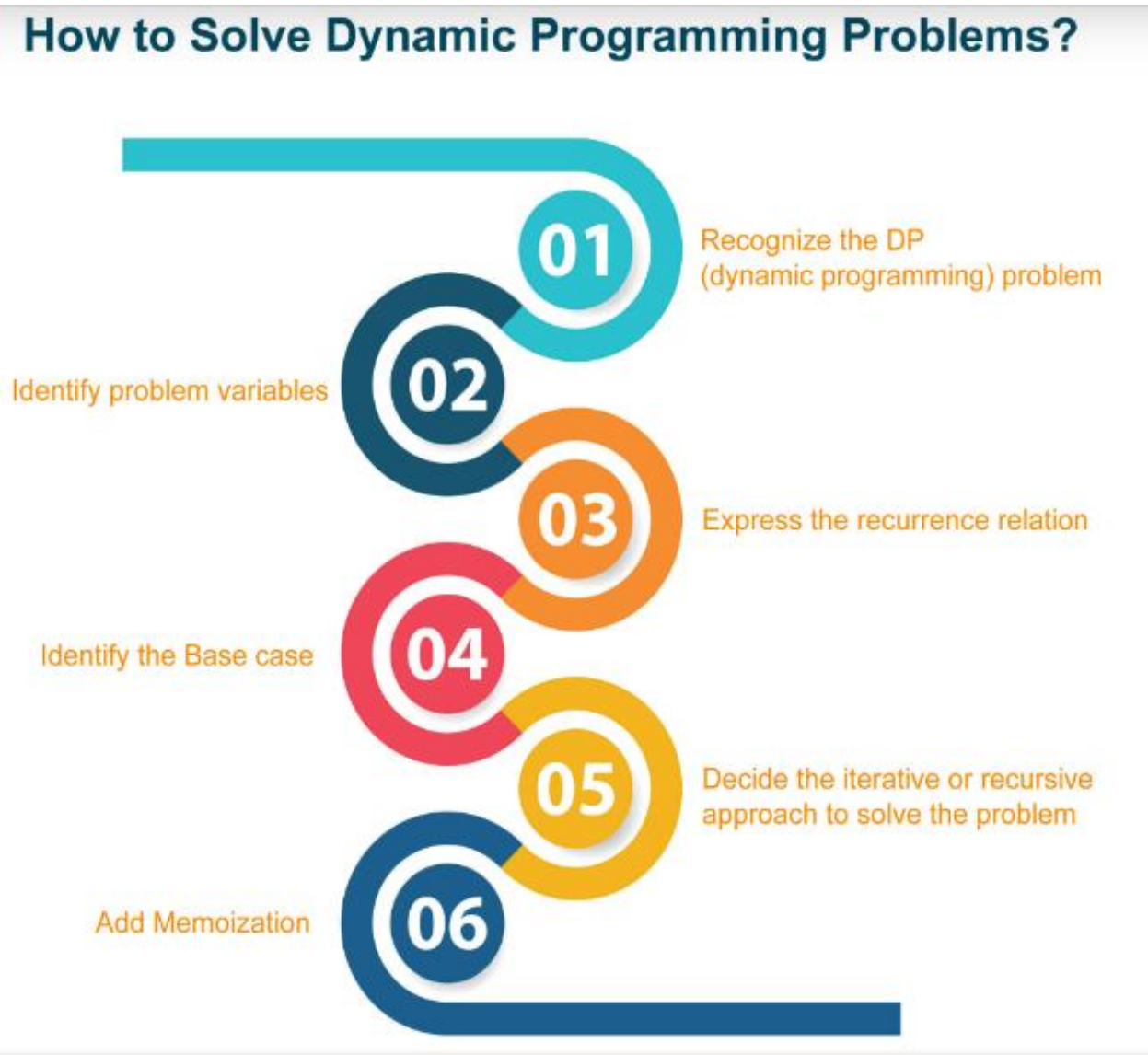
```python
# Python program to display the Fibonacci sequence

def recur_fibo(n):
   if n <= 1:
       return n
   else:
       return(recur_fibo(n-1) + recur_fibo(n-2))

nterms = 10

# check if the number of terms is valid
if nterms <= 0:
   print("Plese enter a positive integer")
else:
   print("Fibonacci sequence:")
   for i in range(nterms):
       print(recur_fibo(i))
```

Dynamic Programming and Recursion are very similar

1. Both recursion and dynamic programming are starting with the base case where we initialize the start.

2. After we wrote the base case, we will try to find any patterns followed by the problem's logic flow. Once we find it, we are basically done.

3. The main difference is that, for recursion, we **do not store any intermediate values** whereas dynamic programming do utilize that.



The Fibonacci problem is solved by dynamic programming by following steps:

1: Check for the base case

2: Create an empty dynamic programming array to store all the intermediate and temporary results in order for faster computing. Since n starts from 0, we will create a list with n+1 length.

3: The next step is to think about the general pattern of how the F[n] will be generated afterward. Luckily, the problem prompt already gave us the pattern:

F[n] = F[n-1] + F[n-2]

And this is actually the major difference separate dynamic programming with recursion. In recursion, we do not store any intermediate results vs in dynamic programming, we do store all intermediate steps.

Task 1: Your task is to write a code for dynamic programming solution of the following problem.

Task 02:

Problem Statement: You are given the strings s1 and s2. You have to find the length of the longest common substring of s1 and s2.

Solution: Here you can start checking all substrings from the first string s1 with the character of the second string s2 and keep a record of the maximum. You can solve this problem using dynamic programming by following the bottom-up manner.

Create a matrix of size len(s1)*len(s2) and store the value of the solutions of the substrings to use later to solve similar sub-problems. The problem is similar to the Longest Common Subsequence problem with the only difference that if the characters do not match then the answer at that instance becomes 0.

Problem Logic:

```
Base Case:
If any string is null then LCS will be 0 else, compare the ith character of
string A with jth character of string B.

Case 1: If both characters are same,
LCS[i][j] = 1 + LCS[i-1][j-1] (adding 1 to the result and remove the last
character from both strings)

Case 2: If both characters are not same
LCS[i][j] = 0

At last, traverse the matrix and find the maximum character in it. The output
will be the length of the Longest Common Substring.
```

		A	B	C	D	A	F
	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0
B	0	0	**1**	0	0	0	0
C	0	0	0	**2**	0	0	0
D	0	0	0	0	**3**	0	0
F	0	0	0	0	0	0	1

Your task is to write code for dynamic programming solution of this problem.

Task 03: Given a rod of length n inches and an array of prices that includes prices of all pieces of size smaller than n. Determine the maximum value obtainable by cutting up the rod and selling the pieces.

For example, if the length of the rod is 8 and the values of different pieces are given as the following, then the maximum obtainable value is 22 (by cutting in two pieces of lengths 2 and 6)

length	1	2	3	4	5	6	7	8
price	1	5	8	9	10	17	17	20

And if the prices are as follows, then the maximum obtainable value is 24 (by cutting in eight pieces of length 1)

length	1	2	3	4	5	6	7	8
price	3	5	8	9	10	17	17	20

$$cutRod(n) = \max(price[i] + cutRod(n-i-1)) \text{ for all } i \text{ in } \{0, 1 .. n-1\}$$

Your task is to code the dynamic problem solution of this problem.