# DP

- 정의 : 한번 계산한 문제는 다시 계산하지 않도록 하는 알고리즘
- 메모리 공간을 더 사용하여 연산 속도를 비약적으로 증가 시키는 방법
- 종류는 Top-Down, Bottom-Up 방식이 있음
    1. Top-Down은 재귀함수로 구현 (점화식을 사용)
        - 큰 문제를 해결하기 위해 작은 문제를 호출(재귀)하여 탑 다운 방식이라고 함
    1. Bottom-Up 방식은 for문을 사용
        - 작은 문제부터 차근차근 답을 도출한다고 하여 보텀업방식이라고 함
- DP는 다음 조건을 만족할 때 사용가능
    1. 큰 문제를 작은 문제로 나눌 수 있다.
    2. 작은 문제에서 구한 정답은 그것을 포함하는 큰 문제에서도 동일
- 따라서 작은 문제의 정답을 저장하는 메모제이션 기법을 사용
- 한번 구한 결과를 메모리 공간에 메모해두고 같은 식을 다시 호출하면 메모리 결과를 그대로 가져오는 기법 (캐싱이라고도 부름)
- 대표적인 예시로는 피보나치 수열

## DP를 사용하지 않는 피보나치 수열

```
# 재귀 함수를 사용하면 n이 커질수록 수행 시간이 기하급수적으로 늘어남
def fibo(x):
    if x==1 or x==2:
        return 1
    return fibo(x-1)+fibo(x-2)
```

## DP를 사용한 피보나치 수열 (Top-Down)

```
# 따라서 피보나치 수열을 메모제이션을 이용하여 한번 계산한 값을 저장함
d=[0]*100
# 피보나치 함수를 재귀함수로 구현 (탑다운 다이나믹 프로그래밍)
def fibo_dp(x):
    if x==1 or x==2:
```

```
        return 1

    # 이미 계산한 적 있는 문제라면 그대로 반환
    if d[x]!=0:
        return d[x]

    # 아직 계산하지 않는 문제라면 점화식에 따라 피보나치 결과 반환
    # 이때 메모제이션을 이용해서 한번 구한 값을 기록
    d[x]=fibo(x-1)+fibo(x-2)
    return d[x]
```

# DP를 사용한 피보나치 수열 (Bottom-Up)

```
d=[0]*100
d[1]=1
d[2]=1
def fibo_dp_for(n):
    # 피보나치 함수, 반복문으로 구현 (보텀업 다이나믹 프로그래밍)
    for i in range(3,n+1):
        d[i]=d[i-1]+d[i-2]

    print(d[n])
```

문제 풀이