## https://python.bakyeono.net/chapter-5-3.html

## 리스트 컴프리헨션

리스트를 초기화하는 방법 중 하나.

대괄호 안에 조건문과 반복문을 적용하여 리스트를 초기화 할 수 있다.

Array = [ i for i in range (10) ]

* [1,2,3,4,5,6,7,8,9]

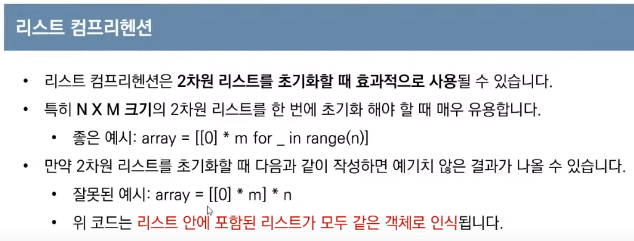
# 0부터 19까지의 수 중에서 홀수만 포함하는 리스트

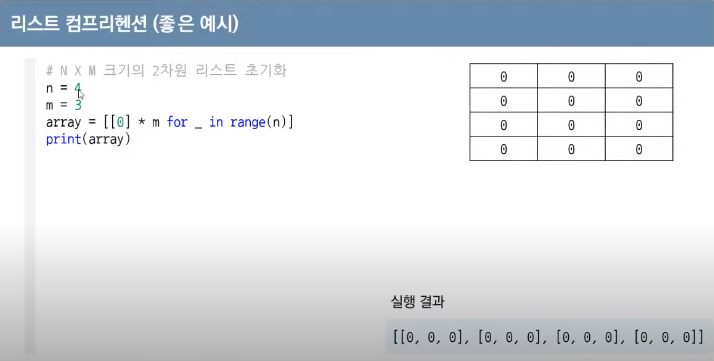
Array = [ i for i in range (20) if i % 2 == 1 ]

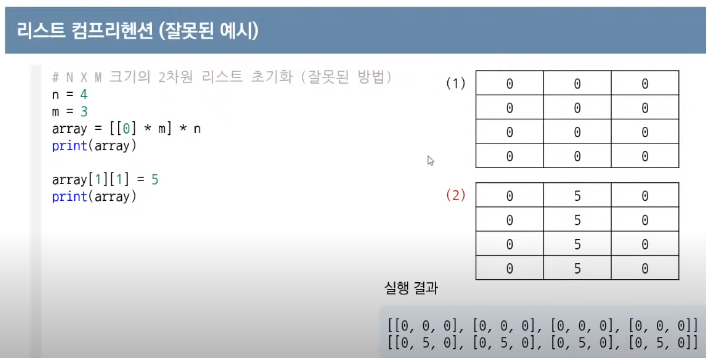
* [1,3,5,7,9,11,13,15,17,19]

# 1부터 9까지의 수들의 제곱 값을 포함하는 리스트

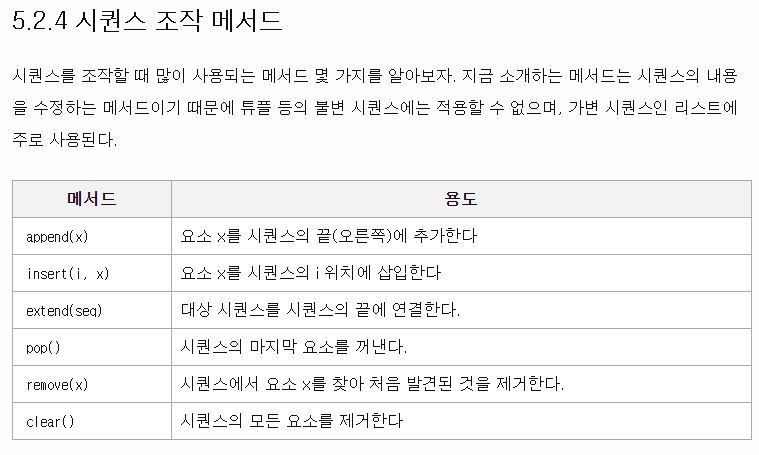
Array = [ i\*i for i in range(1,10) ]

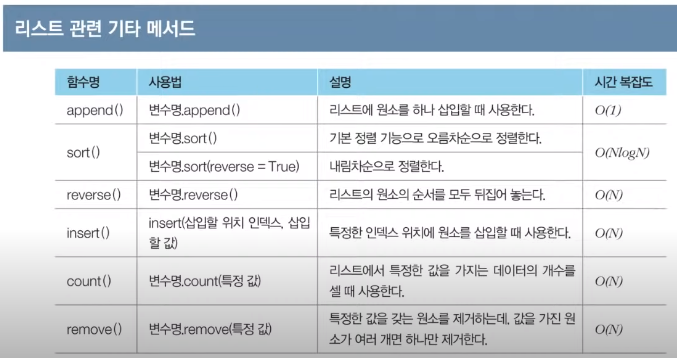




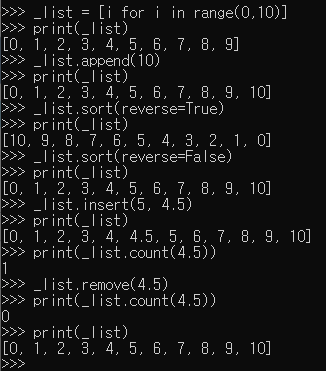


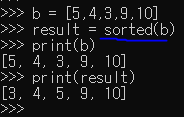
## 리스트 관련 기타 메서드





아래는 위 리스트 메서드 실습 결과

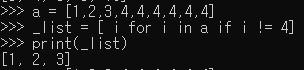


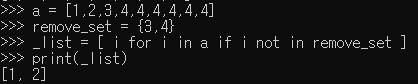


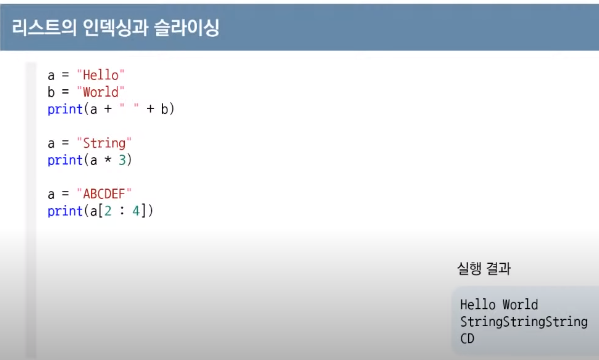
List 클래스의 .sort() 메서드를 쓰면 원래 리스트 값에 바로 적용이 된다.

하지만 파이썬 내장 sorted() 함수를 쓰면 원본 값은 변화시키지 않고 복제된 값을 반환시킨다.

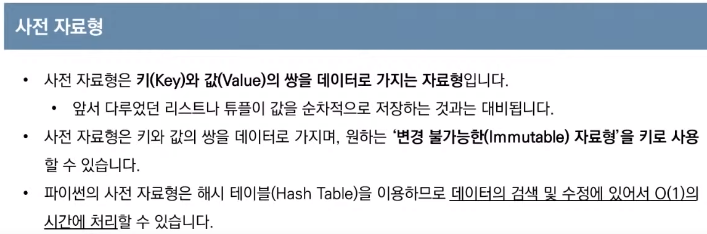
## 리스트에서 특정 값의 원소를 모두 제거하기

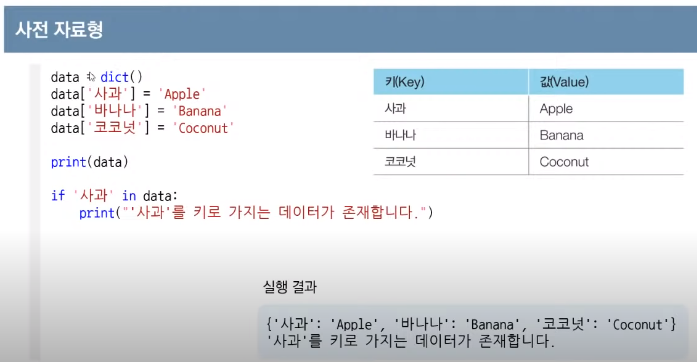


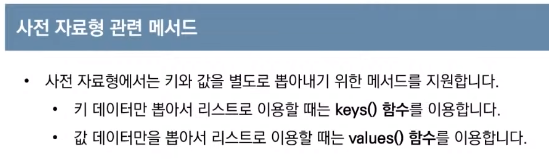


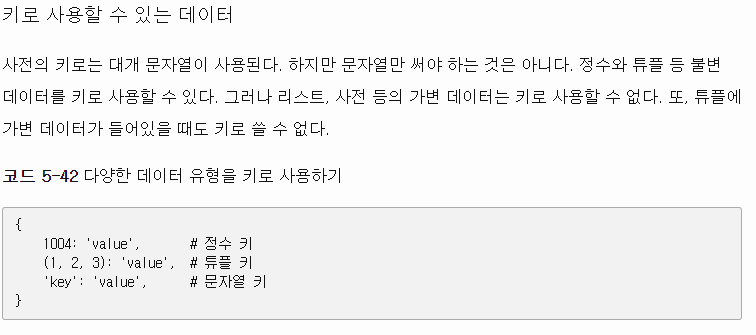


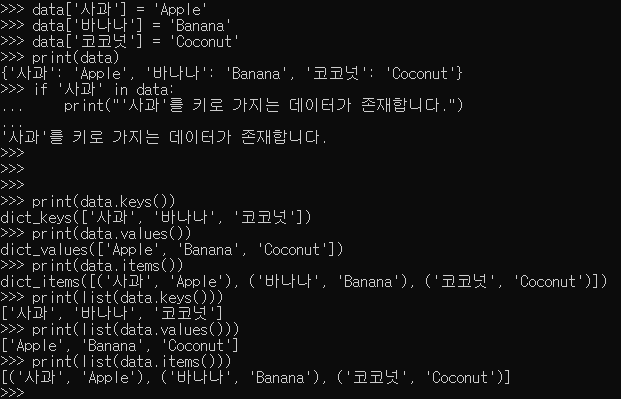
## 사전





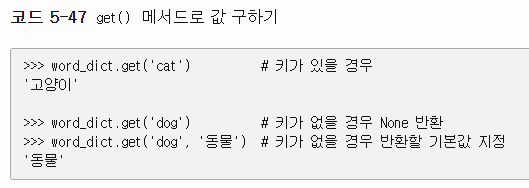




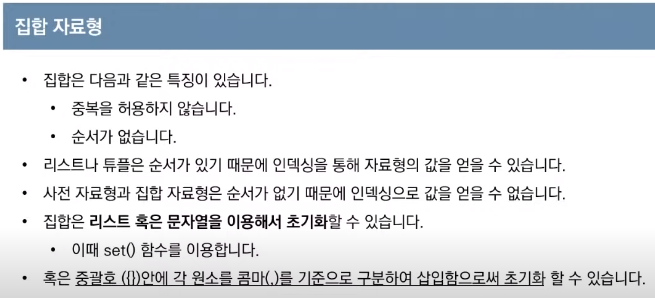


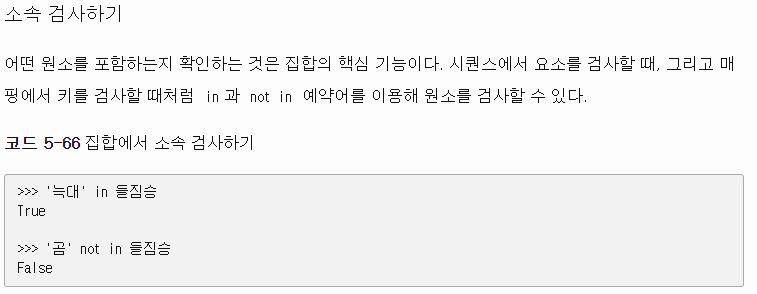
Keys, values, items 메서드에 list() 내장 함수 사용해서 list 형태로 바꿀 수 있다.





## 집합 자료형



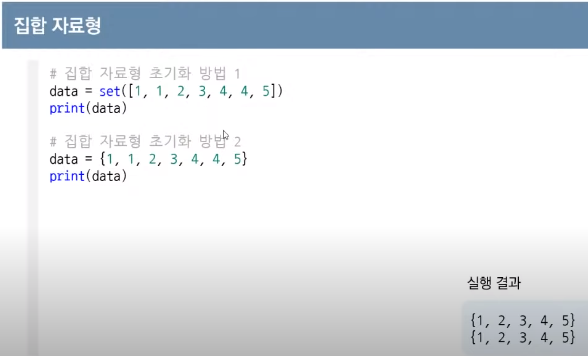
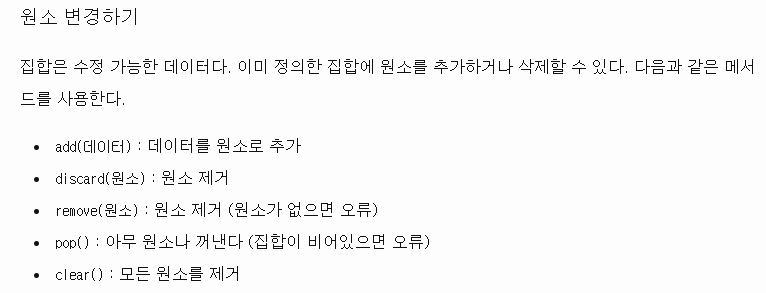


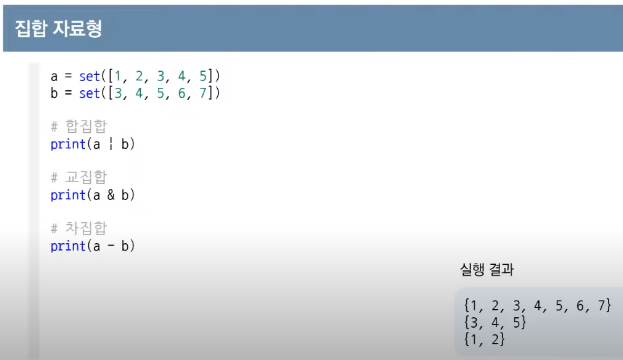
# 소속 검사할 때

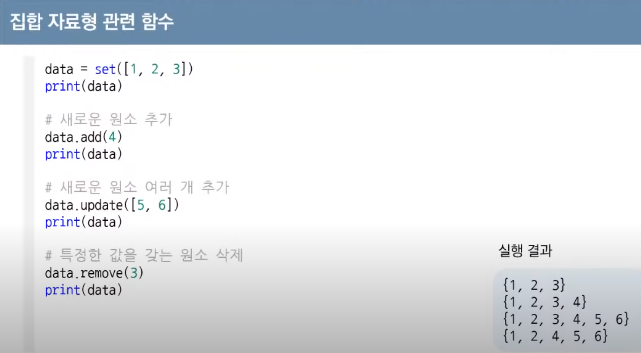
List, tuple = O(n)

Set = O(1)

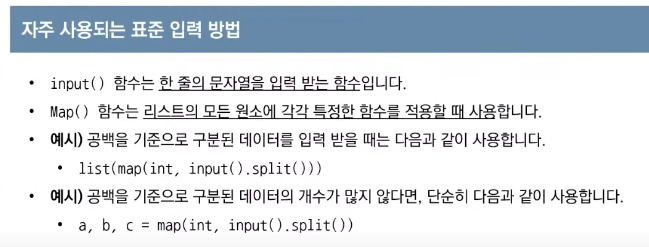
따라서 단순 값이 있는지 없는지 체크할 때는 set을 쓰는게 좋다.





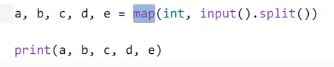


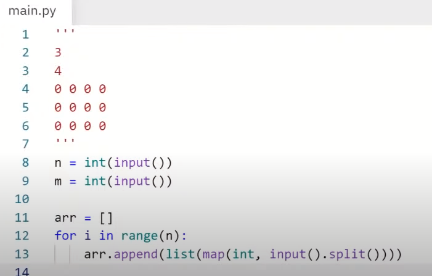
# 기본 입출력



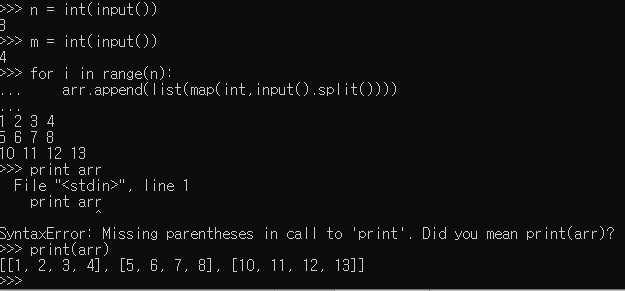


입력 받은 값을 공백으로 구분 한 다음 int 형으로 전환





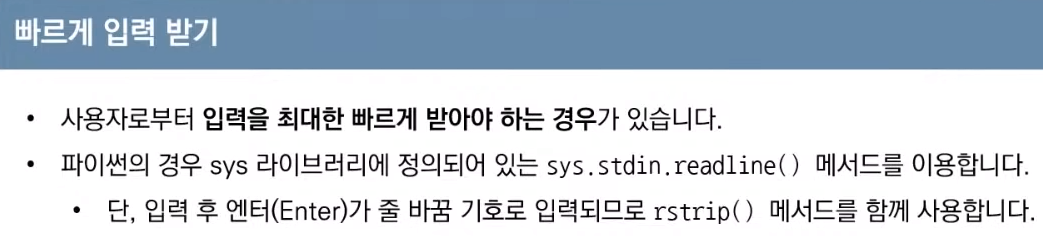
## 2차원 int 배열 초기화 하는 방법

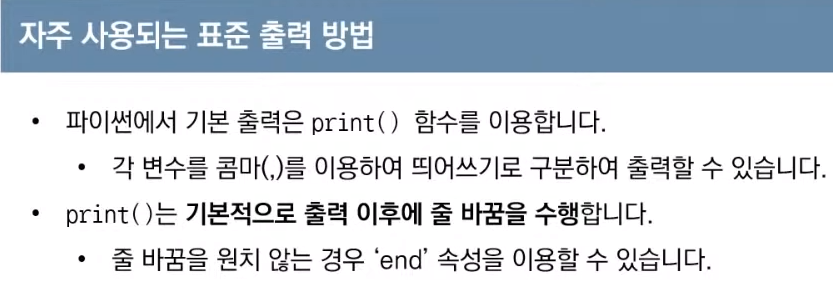


c에서는 arr[3][4] = {0, } 처럼 배열의 크기 먼저 선언하고, for문으로 n,m loop 하면서 scanf로 int값 받으며 2차원 배열 초기화 하는데,

파이썬에서는 1. Map(int, input()) 으로 입력 받은 숫자가 int형 이라는 정보를 주고, list 생성할 때 list size 값을 줘서 초기화 하지 않고, 빈 list를 만들고 list 값들을 append 하여 초기화 한다.

## # 특정 문제의 경우 input이 천만개 이상 되는 경우 input 받는 대만 엄청난 시간이 소요되는 경우가 있다. 그런 경우 아래 함수 사용하여 입력을 빠르게 받을 수 있다.

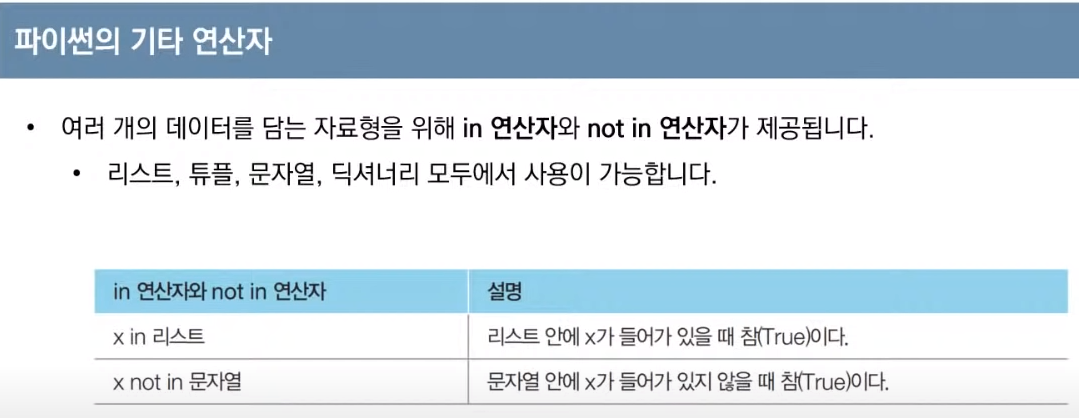




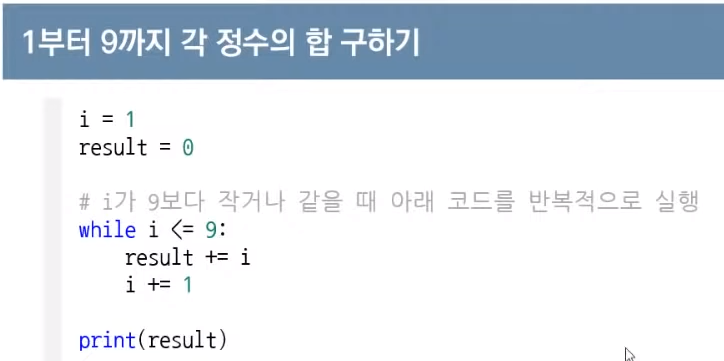
# 조건문

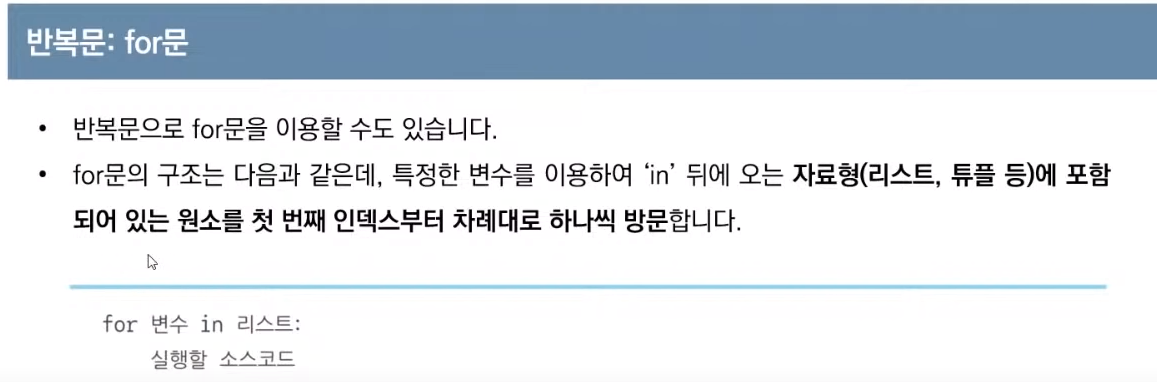


## 여러 개의 데이터를 담는 자료형을 위해 in 연산자와 not in 연산자가 제공된다.



# 반복문

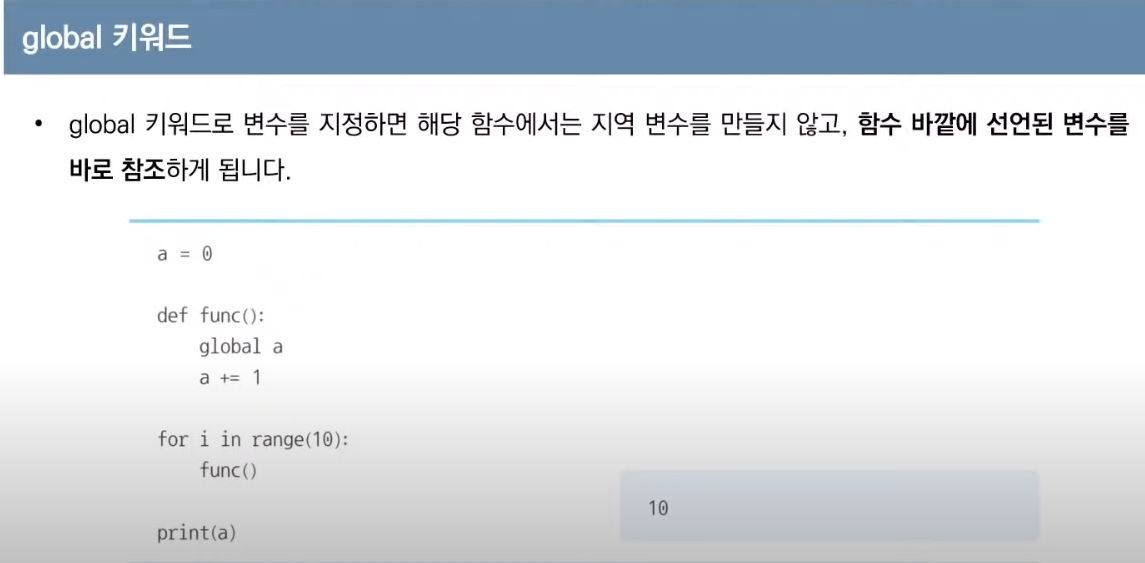




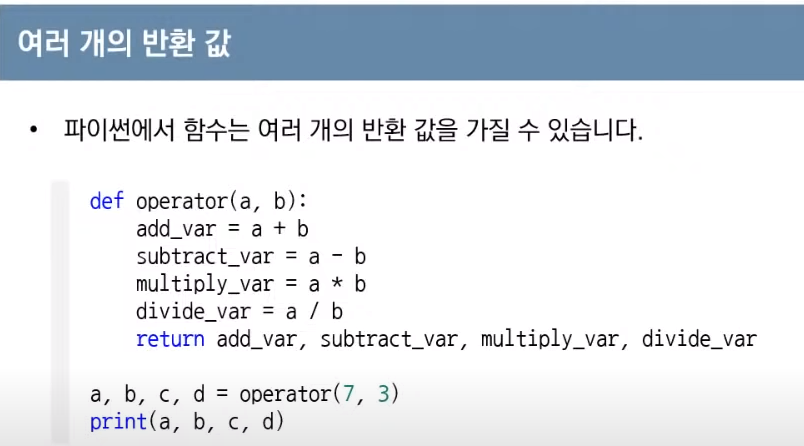


# 함수

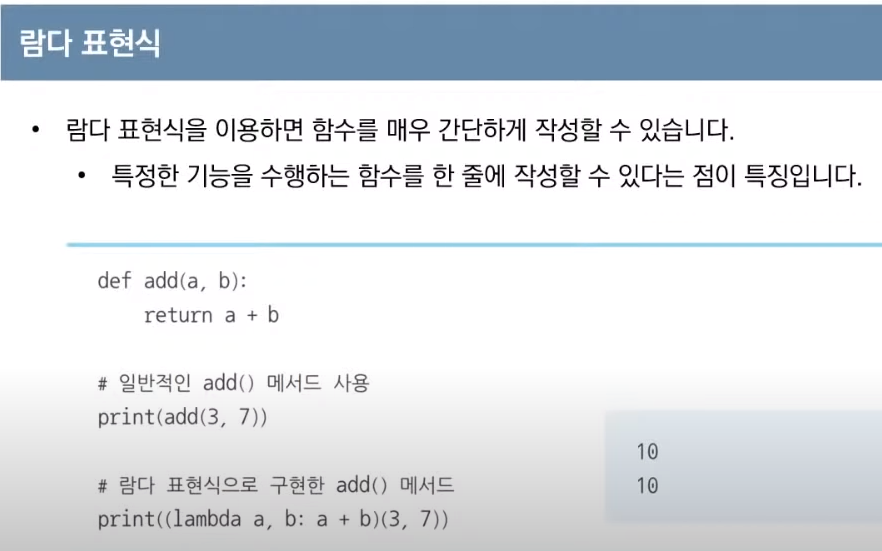
## Global 키워드

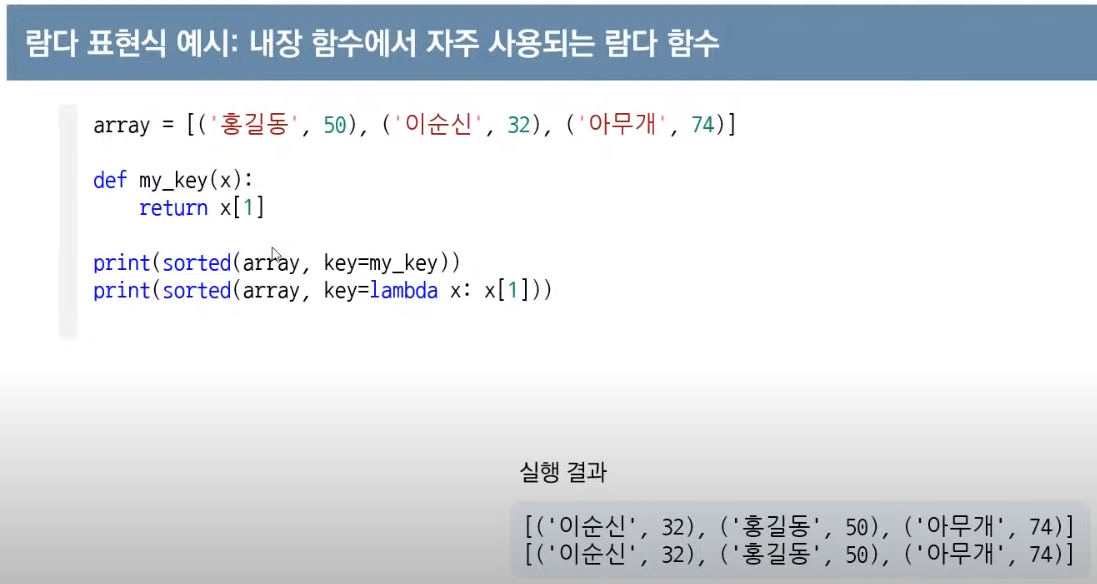


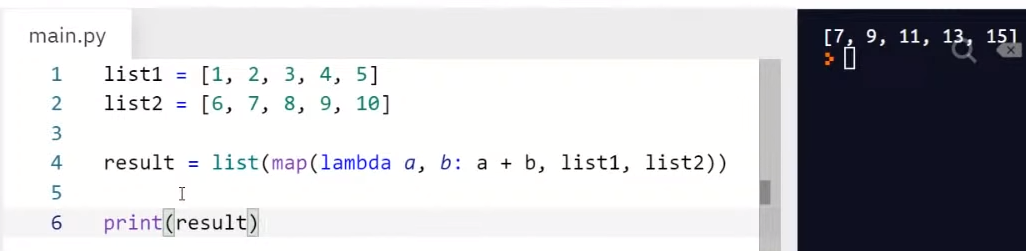
## 여러 개의 반환 값



## 람다 표현식



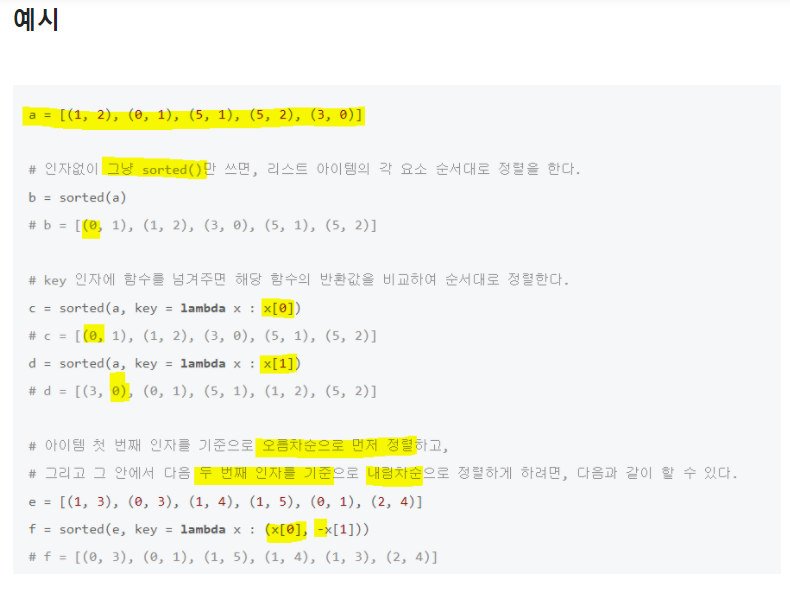




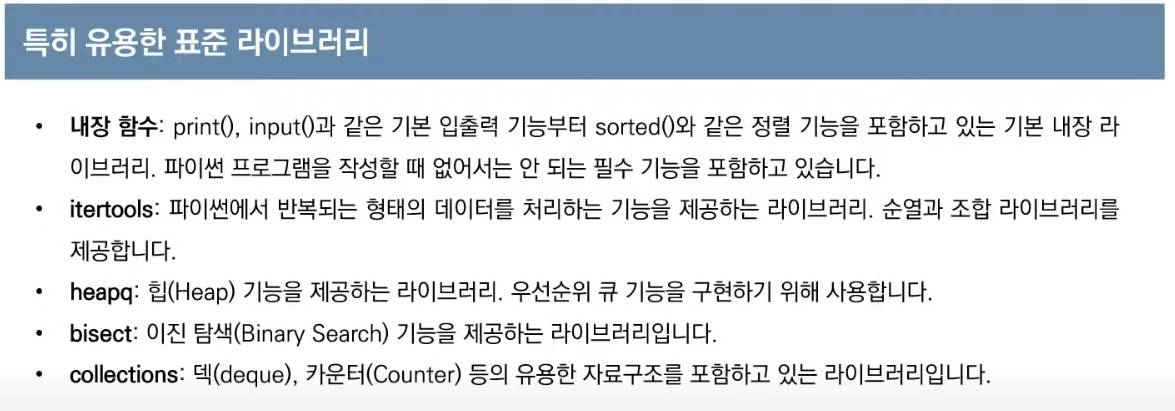
아래는 Sorted 함수 파라메터인 Key 동작에 관한 정리

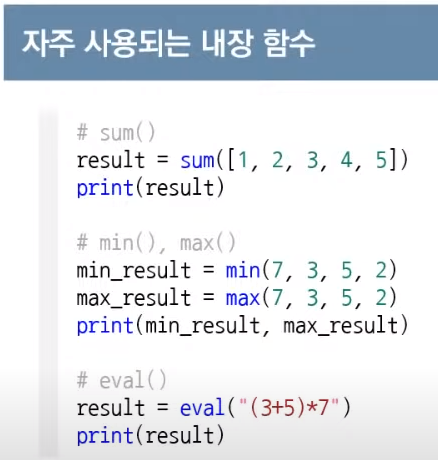
https://velog.io/@aonee/Python-%EC%A0%95%EB%A0%AC-sort-sorted-reverse





# 특히 유용한 표준 라이브러리

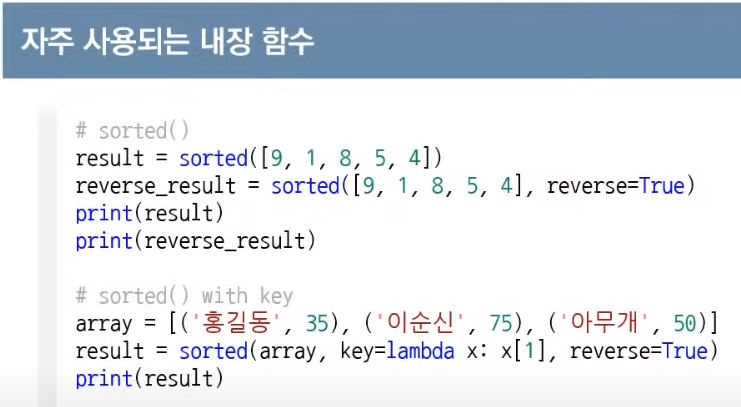




Eval 함수란?

<https://blockdmask.tistory.com/437>

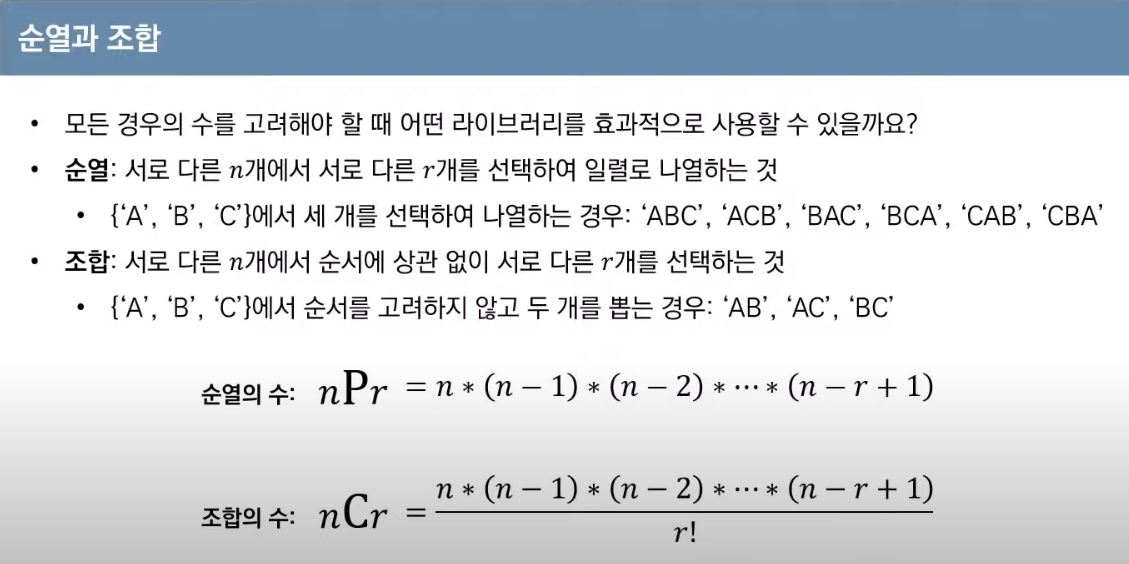
매개변수로 받은 expression(=식)을 문자열로 받아서, 실행하는 함수.



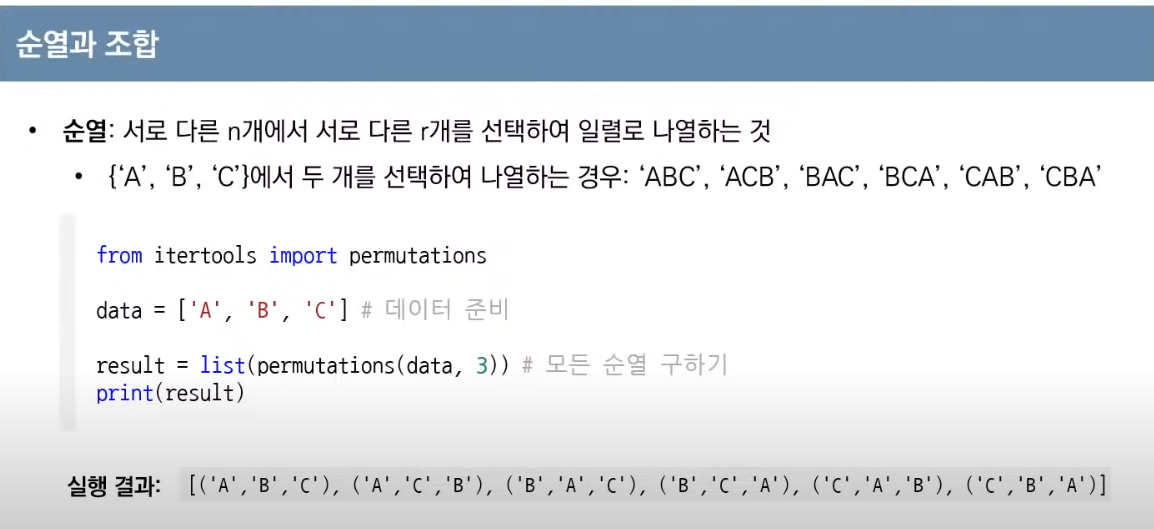
## Sorted( ) 함수는 내부적으로 key를 가지고 있어서 정렬 기준을 선택할 수 있다.

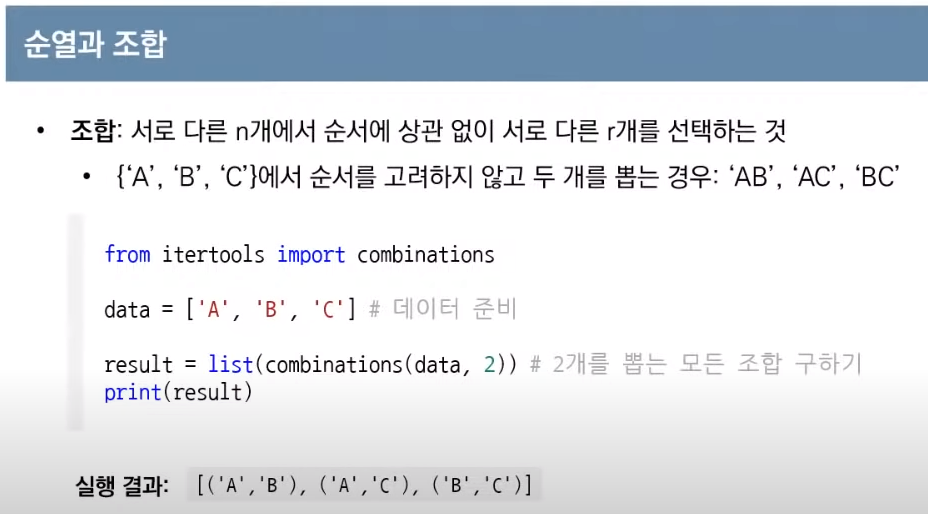
Key = lambda x: x[1] 으로 지정하였는데, ‘홍길동’ 이 아니라 35, 즉 2번째에 있는 숫자를 기준으로 정렬하겠다는 의미이다.

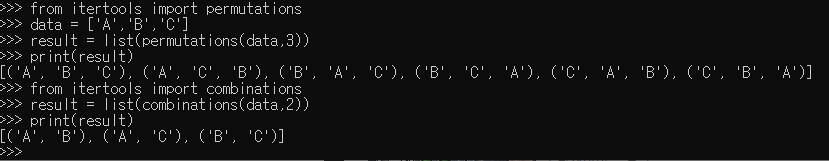
# 순열과 조합

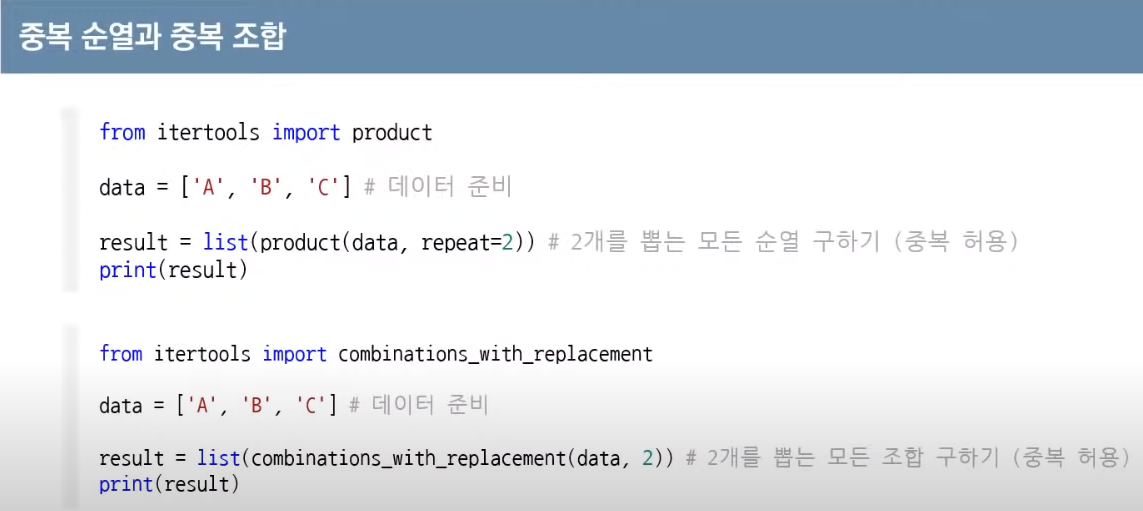


## 순열









# 알고리즘 선택 기준

제한시간 1초인 문제에서 N의 범위에 따라 적용 가능한 시간 복잡도

N < 500 : O(n^3)

N < 2000 : O(n^2)

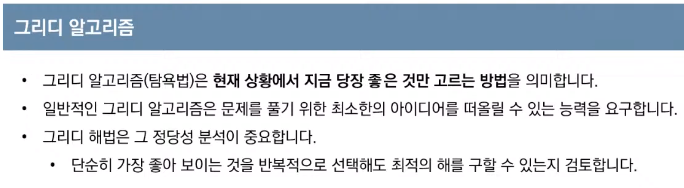
N < 100000 : O(NlogN)

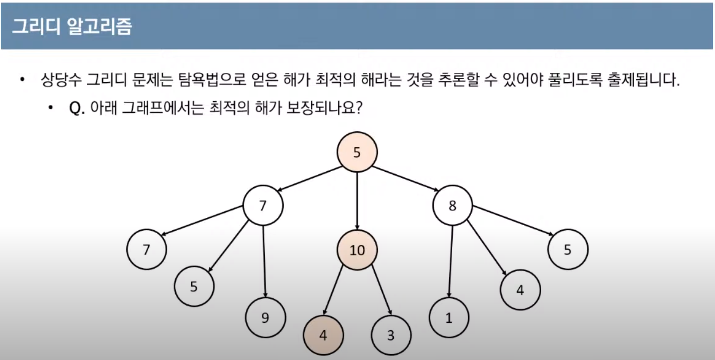
N < 10000000 : O(N)

그리디 알고리즘 – 그리디 알고리즘으로 접근하였을 때 최적의 해를 얻을 수 있다는 것이 확인이 되었을 때 사용한다.

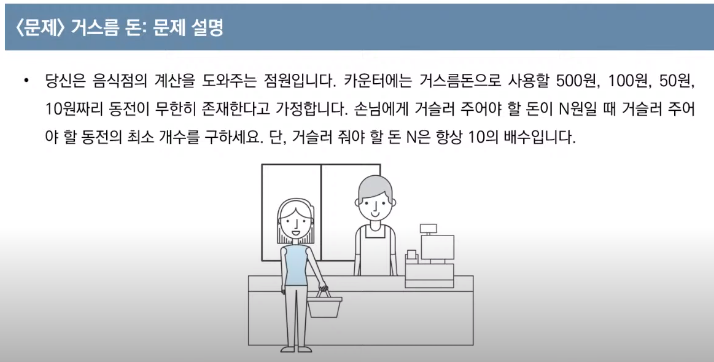
# 그리디 & 구현

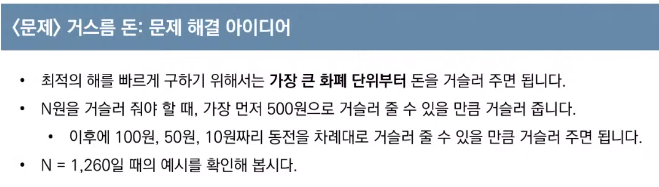
## 그리디





위 그래프는 5->7->9로 가는 것이 5->10->4로 가는 것보다 더 큰 해를 가지므로, 위 문제는 그리디 알고리즘으로 푸는 것이 적합하지 않다.





답 : 500 \* 2, 100 \* 2, 50 \* 1, 10 \* 1

파이썬 코드 :

money = int(input())

obak = money // 500

money = money % 500

bak = money // 100

money = money % 100

osip = money // 50

money = money % 50

sip = money // 10

print("반환해야할 동전 갯수 : {}".format(obak+bak+osip+sip) )

다른 버전 코드

n = 1260

count = 0

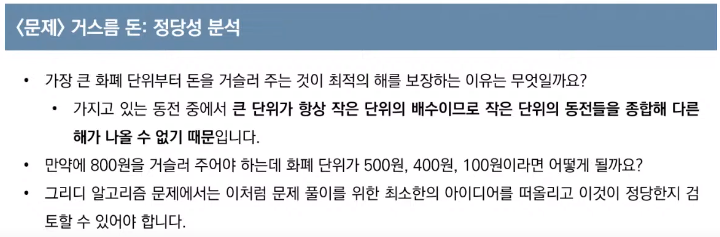
coin\_types = [500,100,50,10]

for coin in coin\_types:

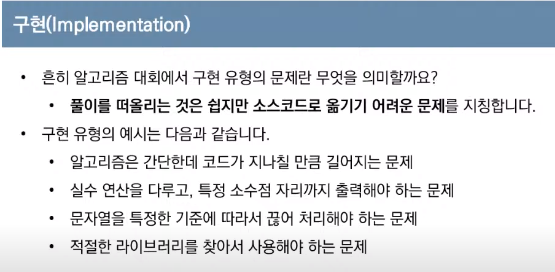
count += n // coin

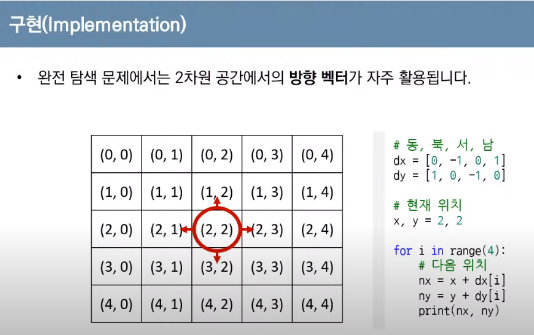
n %= coin

print(count)



## 구현





# DFS/BFS

**Stack은 그냥 List 자료형을 활용**

stack = []

stack.append(1)

stack.append(2)

stack.append(3)

stack.append(4)

stack.pop()

stack.append(5)

stack.append(6)

stack.pop()

print(stack)

print(stack[::-1])

**큐 라이브러리 사용 예제**

from collections import deque

queue = deque()

queue.append(1)

queue.append(2)

queue.append(3)

queue.append(4)

queue.popleft()

queue.append(5)

queue.append(6)

queue.popleft()

print(queue)

queue.pop()

print(queue)

queue.reverse()

print(queue)

print(list(queue))

## DFS

#DFS 동작 과정

1. 탐색 시작 노드를 스택에 삽입하고 방문 처리를 한다.
2. 스택의 최상단 노드에 방문하지 않은 인접 노드가 있으면 그 인접 노드를 스택에 넣고 방문 처리를 한다. 방문하지 않은 인접 노드가 없으면 스택에서 최상단 노드를 꺼낸다.
3. 2)번의 과정을 더 이상 수행할 수 없을 때까지 반복한다.

# DFS 예제

def dfs(graph, v, visited):

# 현재 노드를 방문 처리

visited[v] = True

print(v, end=' ')

# 현재 노드와 연결된 다른 노드를 재귀적으로 방문

for i in graph[v]:

if not visited[i]:

dfs(graph, i, visited)

# 각 노드가 연결된 정보를 리스트 자료형으로 표현(2차원 리스트)

graph = [

[],

[2,3,8],

[1,7],

[1,4,5],

[3,5],

[3,4],

[7],

[2,6,8],

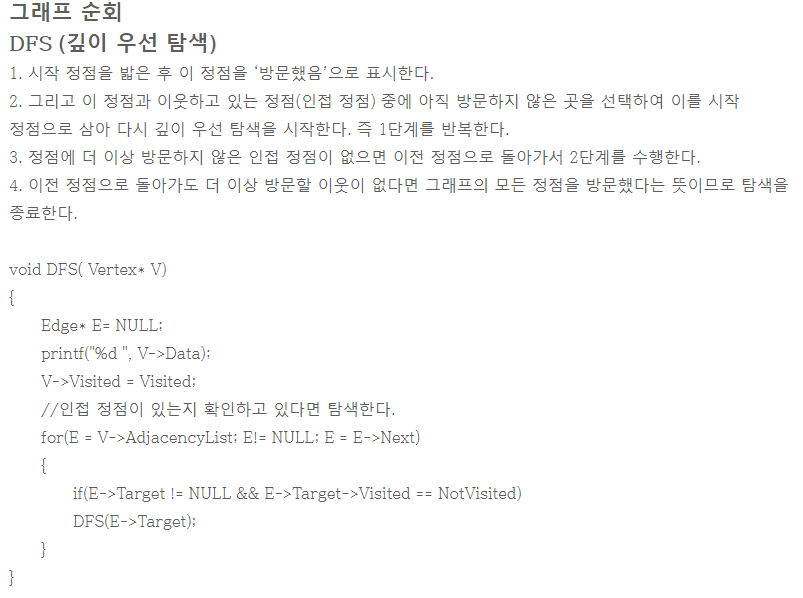
[1,7],

]

# 각 노드가 방문된 정보를 리스트 자료형으로 표현(1차원 리스트)

visited = [False] \* 9

dfs(graph, 1, visited)



## BFS

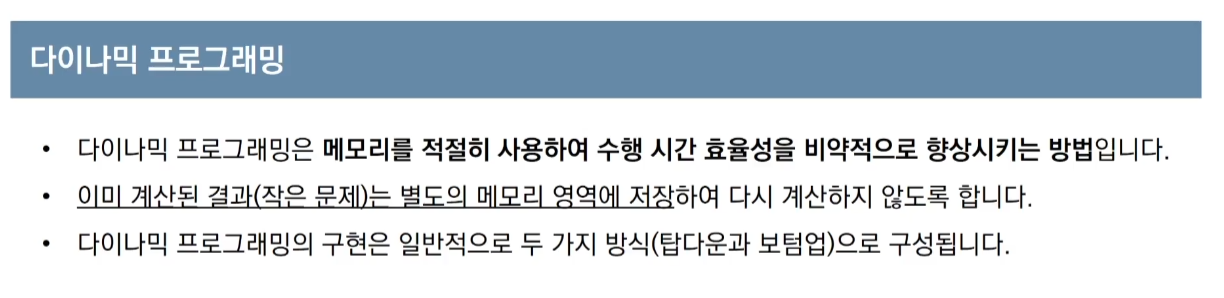
동작방식

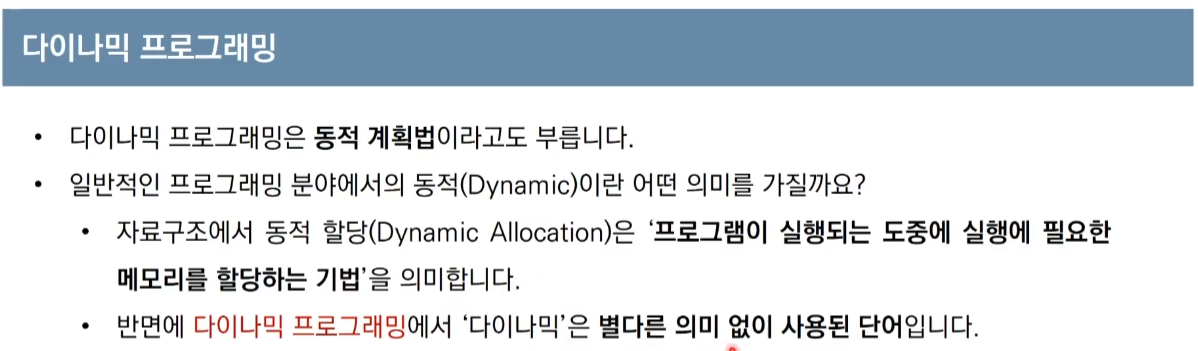
1. 탐색 시작 노드를 큐에 삽입하고 방문 처리를 한다.
2. 큐에서 노드를 꺼내 해당 노드의 인접 노드 중에서 방문하지 않은 노드를 모두 큐에 삽입하고 방문 처리를 한다.
3. 2)번의 과정을 더 이상 수행할 수 없을 때까지 반복한다.

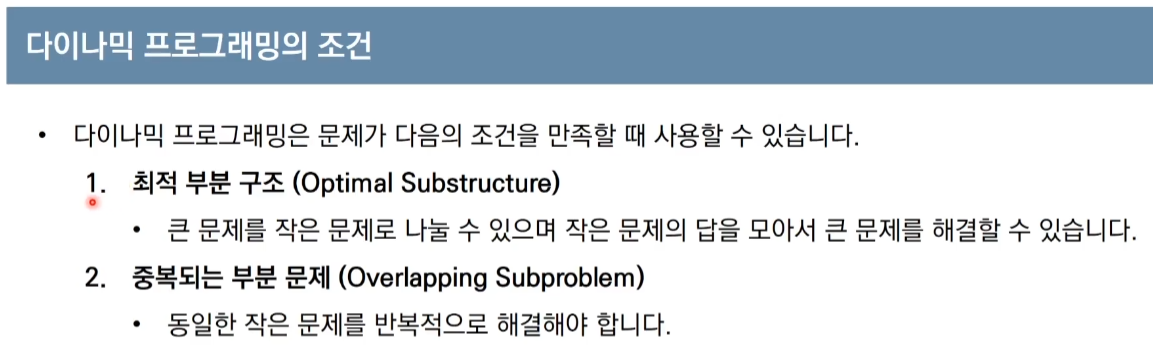
# 다이나믹 프로그래밍

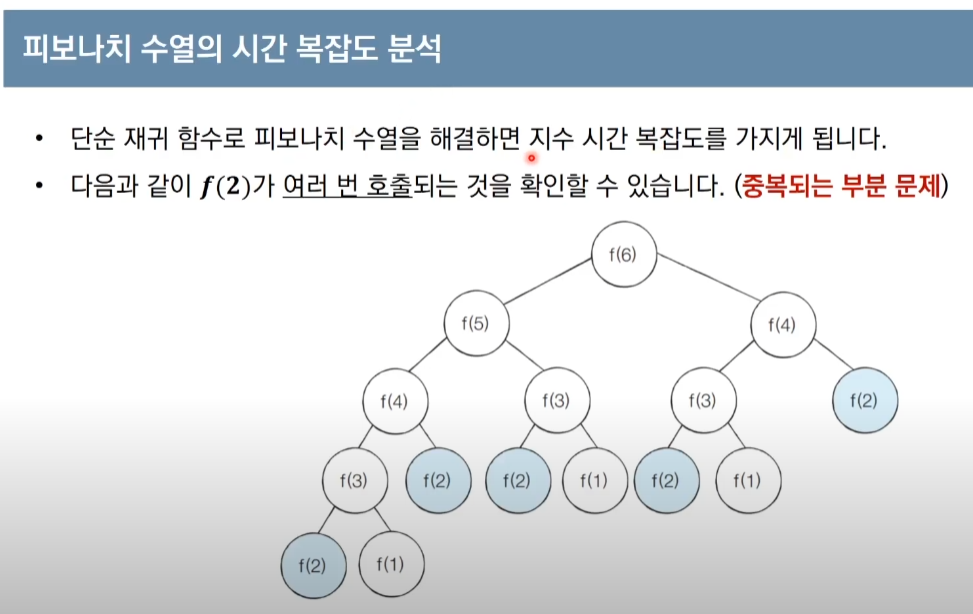
<https://charactermail.tistory.com/163>

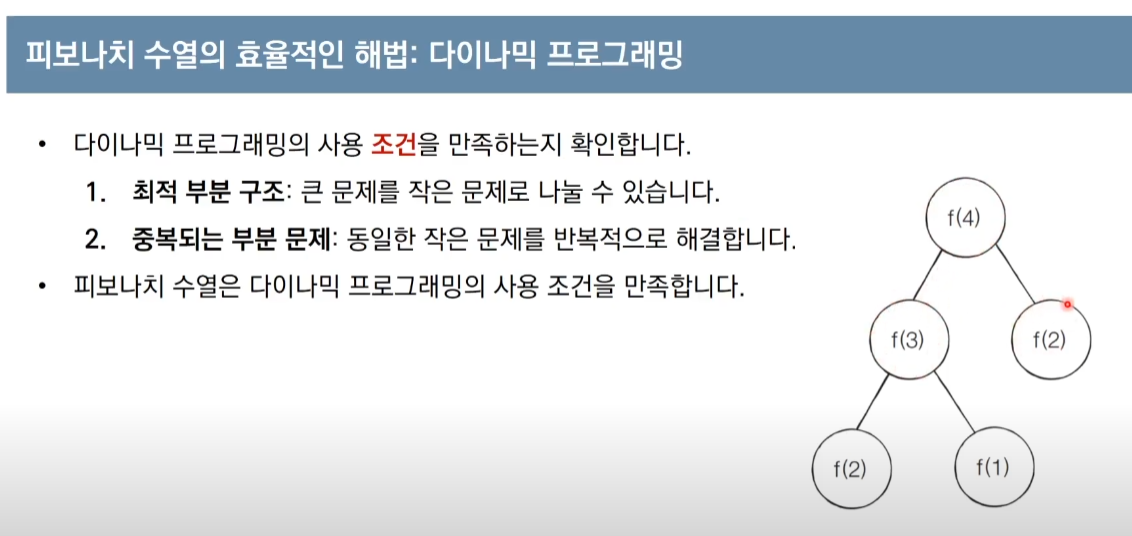
https://www.youtube.com/watch?v=rWbjQphRE9A&list=PLVsNizTWUw7H9\_of5YCB0FmsSc-K44y81&index=28



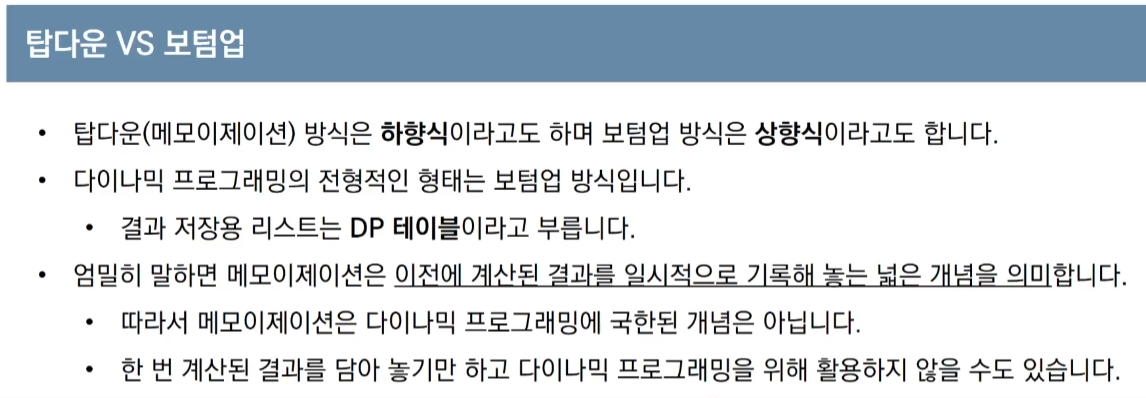












탑다운 : 재귀사용, 바텀업 : 반복문 사용

