자료구조

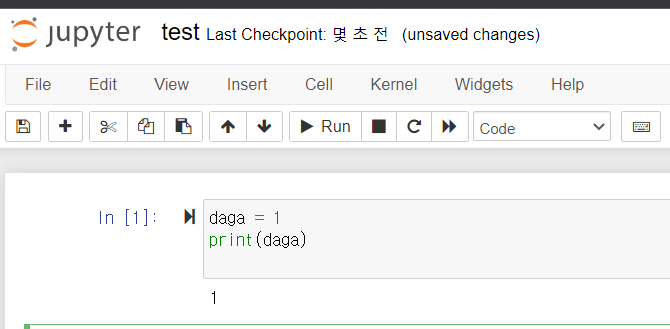
* 자료구조란?
* 대량의 데이터를 효율적으로 관리할 수 있는 데이터의 구조를 의미
* 코드상에서 효율적으로 데이터를 처리하기 위해, 데이터 특성에 따라 체계적으로 데이터를 구조화해야 함
* 어떤 게이터 구조를 사용하느냐에 따라 코드 호율이 달라짐
* 대표적인 자료구조
* 배열, 스택, 큐, 링크드 리스트, 해쉬 테이블, 힙 등
* 알고리즘이란?
* 어떤 문제를 풀기 위한 절차/방법
* 자료구조와 알고리즘이 중요한 이유
* 어떤 자료구조와 알고리즘을 쓰느냐에 따라 성능이 천지차이가 남!

\*파이썬으로 할거임(자료구조와 알고리즘을 쉽게 익히는 데에 용이함!)

\*anaconda: 파이썬 기본(컴파일러)

**\*jupyer notebookf이란 한줄 한줄 코드 실행 결과 확인이 쉽고 효율도 확인 가능**

**주피터 노트북 사용법**

new를 클릭 -> 파이선 -> 코드입력 -> RUN(쉬프트 엔터)  
  
a: 상단부에 셀 만들기, b: 하단부에 셀 만들기, shift+enter:실행

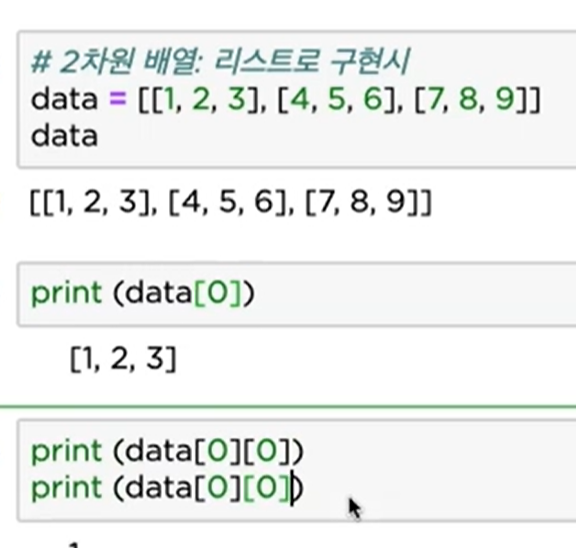
배열

* 꼭 알아둬야 할 자료구조: 배열(Array)
* 데이터를 나열하고, 각 데이터를 인덱스에 대응하도록 구성한 데이터 구조
* 파이썬에서는 리스트 타이빙 배열 기느을 제공하고 있음

1. 배열이 왜 필요할까?  
   -같은 종류의 데이터를 효율적으로 관리하기 위해 사용
2. 같은 종류의 데이터를 순차적으로 저장

* 배열의 장점  
  - 빠른 접근 가능  
  **-첫 데이터의 위치**에서 상대적인 위치로 데이터 접근(인덱스 번호로 접근
* 배열의 단점  
  - 데이터의 추가/삭제가 어려움  
  - 미리 최대 길이를 지정해야 함  
  🡪비가변적

파이썬과 배열

1. 1차원 배열 리스트로 구현시  
   data = [1,2,3,4,5]
2. 2차원 배열 리스트로 구현시  
   data = [[1,2,3], [4,5,6], [7,8,9]]  
   

파이썬 if문

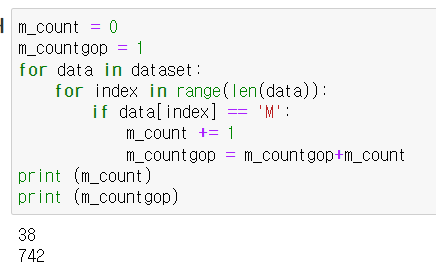
if 조건문:

수행할 문장1  
수행할 문장2

else:

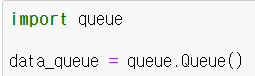
수행할 문장A  
수행할 문장B

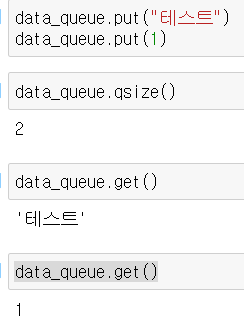
🡪if문을 만들 때는 if 조건문: 바로 아래 문장부터 if문에 속하는 모든 문장에 들여쓰기(indentation)를 해주어야 한다.

파이썬 for문  
  
🡪

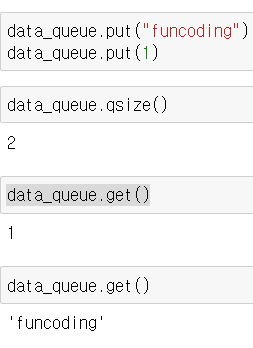
큐

1. 큐 구조  
   - 가장 먼저 넣는 데이터를 가장 먼저 꺼낼 수 있는 구조  
   - FIFO(First-In, First-Out) 또는 LILI(Last-In, Last-Out) 방식으로 스택과 꺼내는 순서가 반대  
   
2. 알아둘 용어  
   **- Enqueue: 큐에 데이터를 넣는 기능  
   - Dequeue: 큐에서 데이터를 꺼내는 기능**
3. 파이썬 queue 라이브러리 활용해서 큐 자료 구조 사용하기  
   - Queue(): 가장 일반적인 큐 자료 구조  
   - LifoQueue(): 나중에 입력된 데이터가 먼저 출력되는 구조(스택 구조)  
   - PriorityQueue(): 데이터마다 우선순위를 넣어서, 우선순위가 높은 순으로 데이터 출력

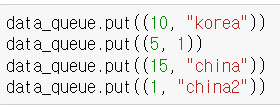
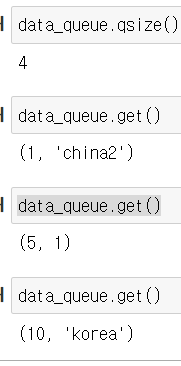
큐 만들기  


큐 넣기: put(), 큐 얻기: get(), 큐 사이즈: qsize()  


LifoQueue()로 큐 만들기 (LIFO(Last-In, First-Out))  


함수는 큐와 마찬가지로 put, get, qsize  
🡪일반적인 큐와는 다르게 마지막에 넣은게 추출됨

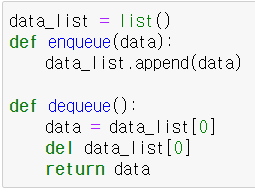
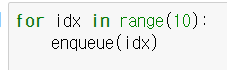
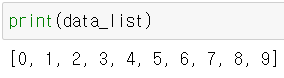
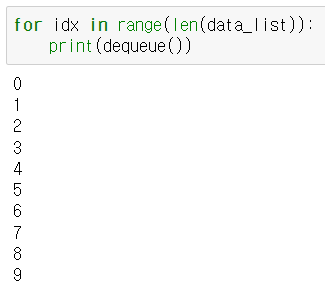
**PriorityQueue()로 큐 만들기(우선순위가 주어지는 큐)🡪중요함**  

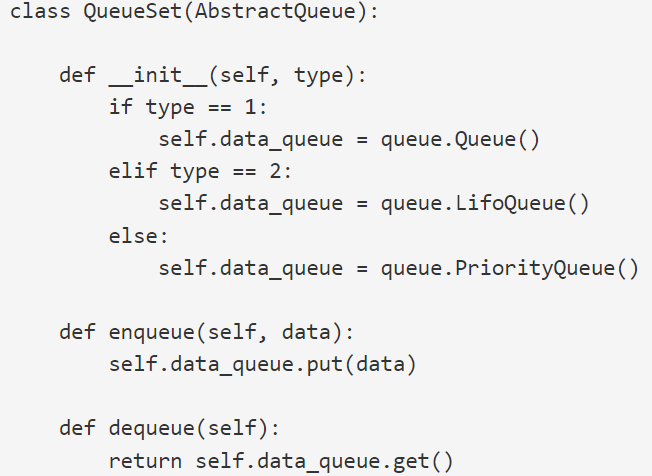

기존의 큐와는 달리 put((우선순위, 데이터))🡪괄호가 2개임  
  
🡪우선순위가 낮은 순서대로 뽑혀져 나옴

* 멀티 태스킹을 위한 프로세스 스케쥴링 방식을 구현하기 위해 많이 사용됨

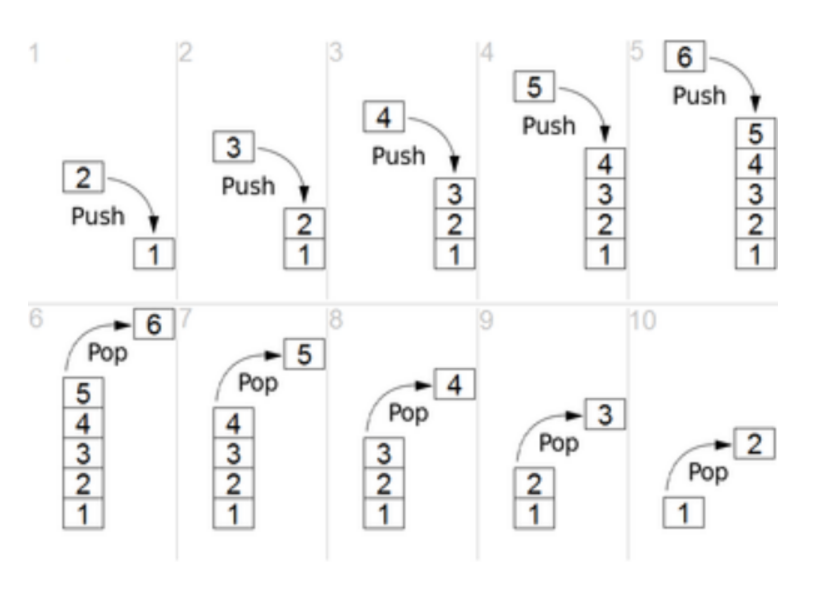
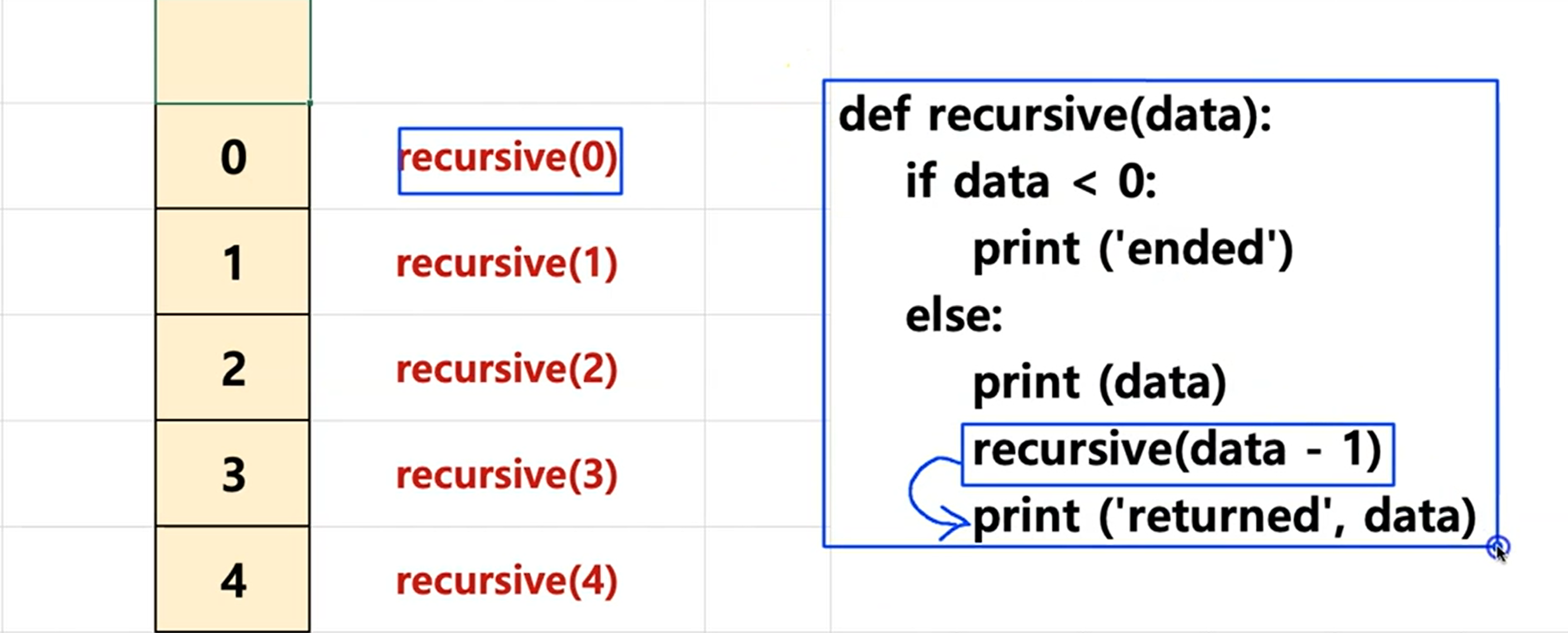
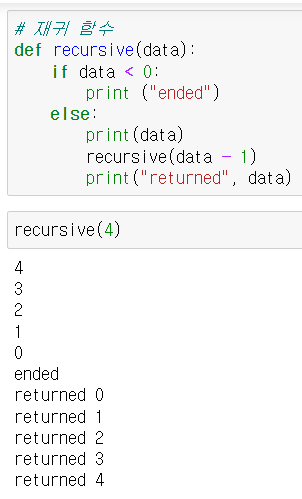
**리스트 변수로 큐를 다루는 enqueue, dequede 기능 구현해보기**

파이썬에서 리스트 선언은 list()  
list의 마지막 값 추가: append()  
list의 값 끄집어내기: pop()(가장최근에 넣는 값을 넣어줌)  
list의 길이: len(list)

파이썬에서 메서드 선언은 def 메서드명(매개변수):  
 🡪  🡪🡪 

**\*파이썬의 기본 생성자:** \_\_init\_\_  


스택(Stack)

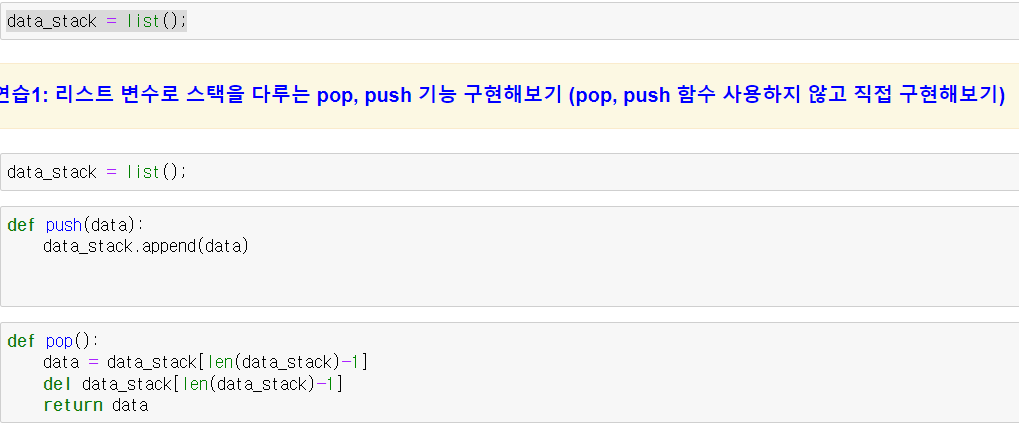
* 데이터를 제한적으로 접근할 수 있는 구조  
  - 한쪽 끝에서만 자료를 넣거라 뺄 수 있는 구조
* 가장 나중에 쌓은 데이터를 가장 먼저 빼낼 수 있는 데이터 구조  
  큐: FIFO 정책  
  스택: LIFO 정책
* 대표적인 스택의 활용  
  - 컴퓨터 내부의 프로세스 구조의 함수 동작 방식
* 주요 기능  
  - push(): 데이터를 스택에 넣기  
  - pop(): 데이터를 스택에서 꺼내기  
  
* **프로세스 스택**🡪재귀함수🡪recursive가 모두 끝나면 다음 코드인 print(‘returned’, data)가 발동  
  🡪 스택처럼 모든 함수가 끝나면 다음 함수로 이동 함
* 스택의 장점  
  - 구조가 단순해서, 구현이 쉽다.  
  - 데이터 저장/읽기 속도가 빠르다.
* 스택의 단점  
  - 데이터 최대 개수를 미리 정해야 한다.  
  (파이썬의 경우 재귀 함수는 1000번까지만 호출이 가능함)  
  - 저장 공간의 낭비가 발생할 수 있음

파이썬 리스트 기능에서 제공하는 메서드로 스택 사용해보기

data\_stack = list();

data\_stack.append(1)

data\_stack.append(2)🡪[1, 2]

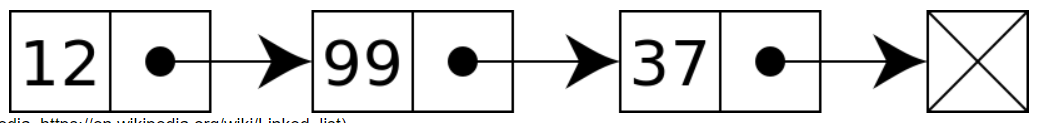
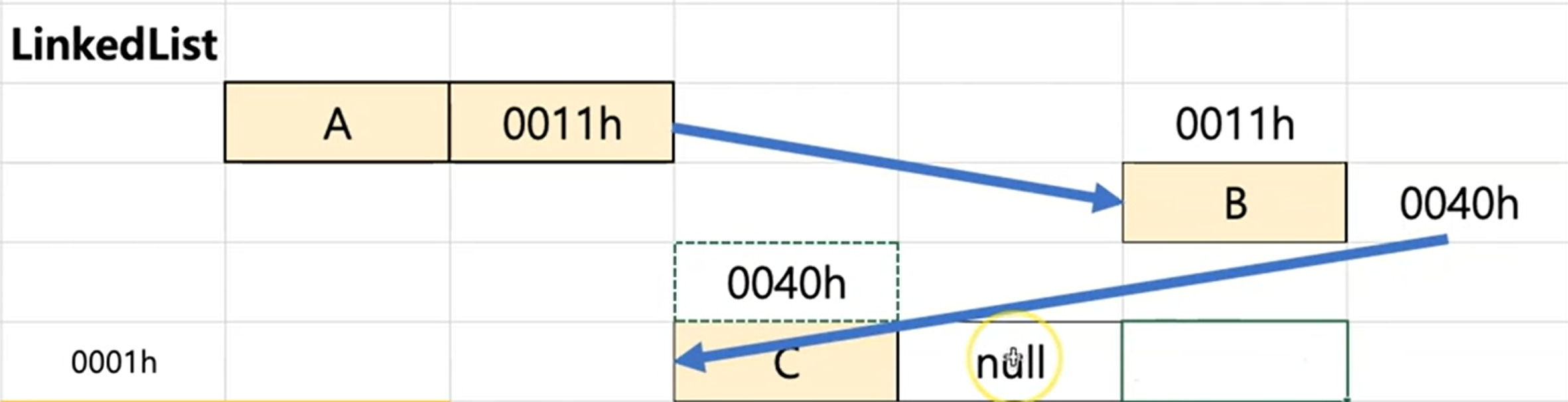


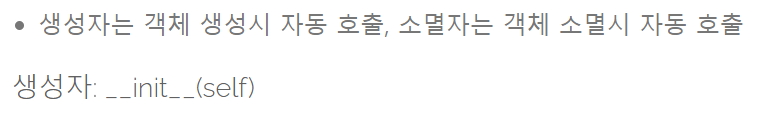
* 이때 stack\_list[-1]은 맨 끝에 있는 값을 가지고 오는 거임(파이썬)

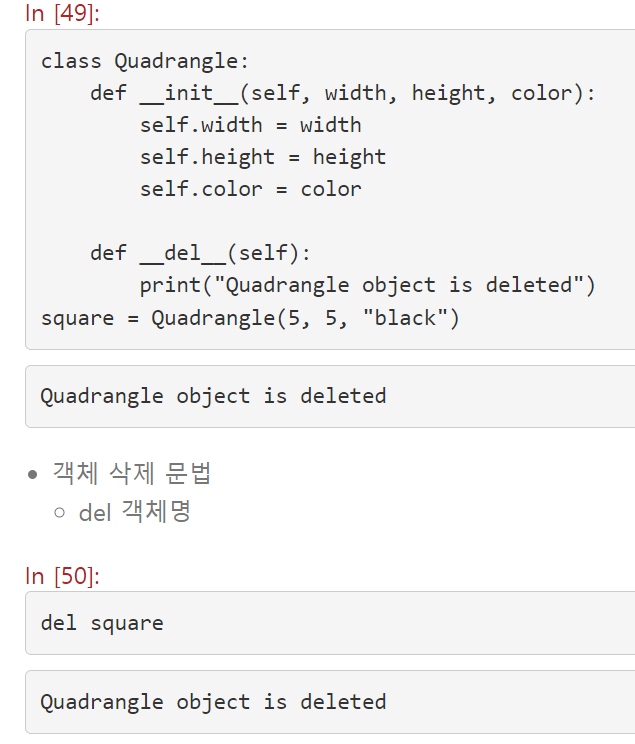
링크드 리스트(Linked List)

* 떨어진 곳에 존재하는 데이터를 화살표로 연결해서 관리하는 데이터 구조
* 배열과 달리 미리 데이터를 예약하지않고 필요할 때마다 추가 가능

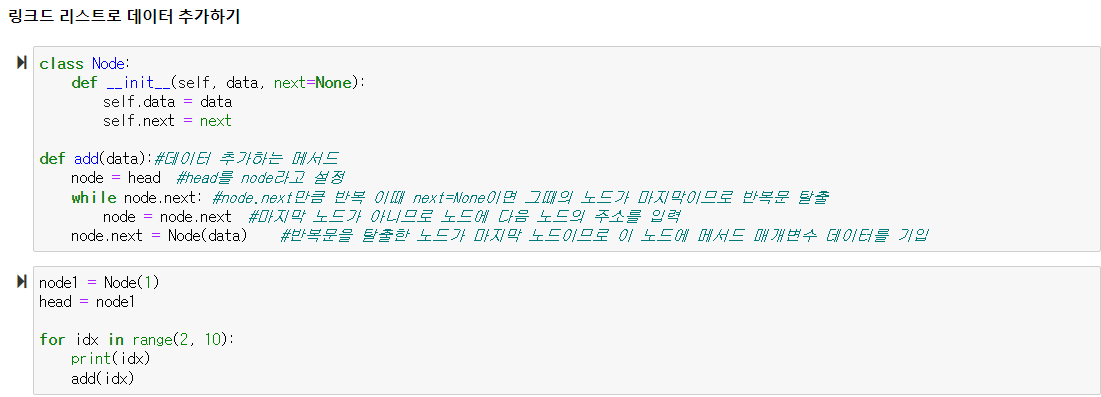
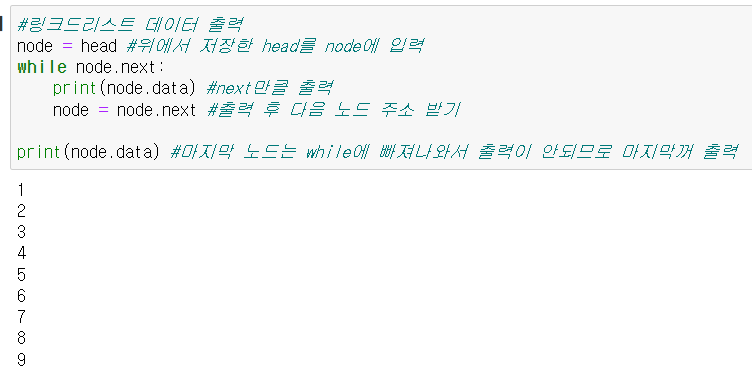
**링크드 리스트 기본 구조와 용어**

* 노드(Node): 데이터 저장 단위(데이터값, 포인터)로 구성
* 포인터(pointer): 각 노드 안에서 다음이나 이전의 노드와의 연결 정보를 가지고 있는 공간  
    
    
  

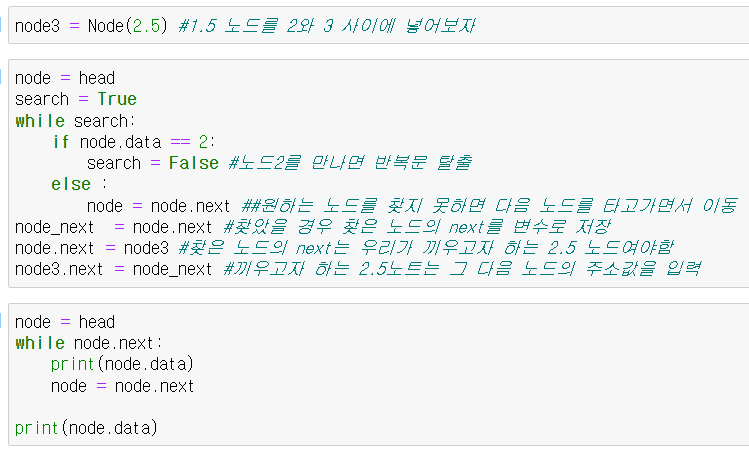


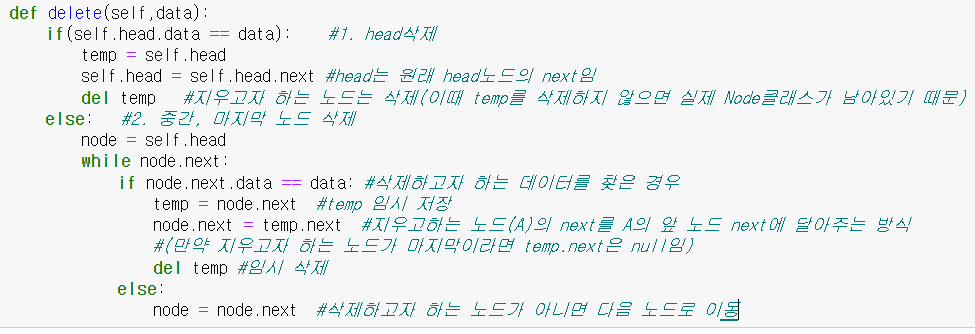
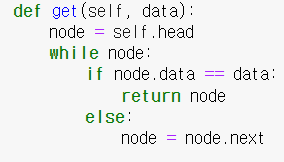
자바에서 this= 파이썬에서 self  


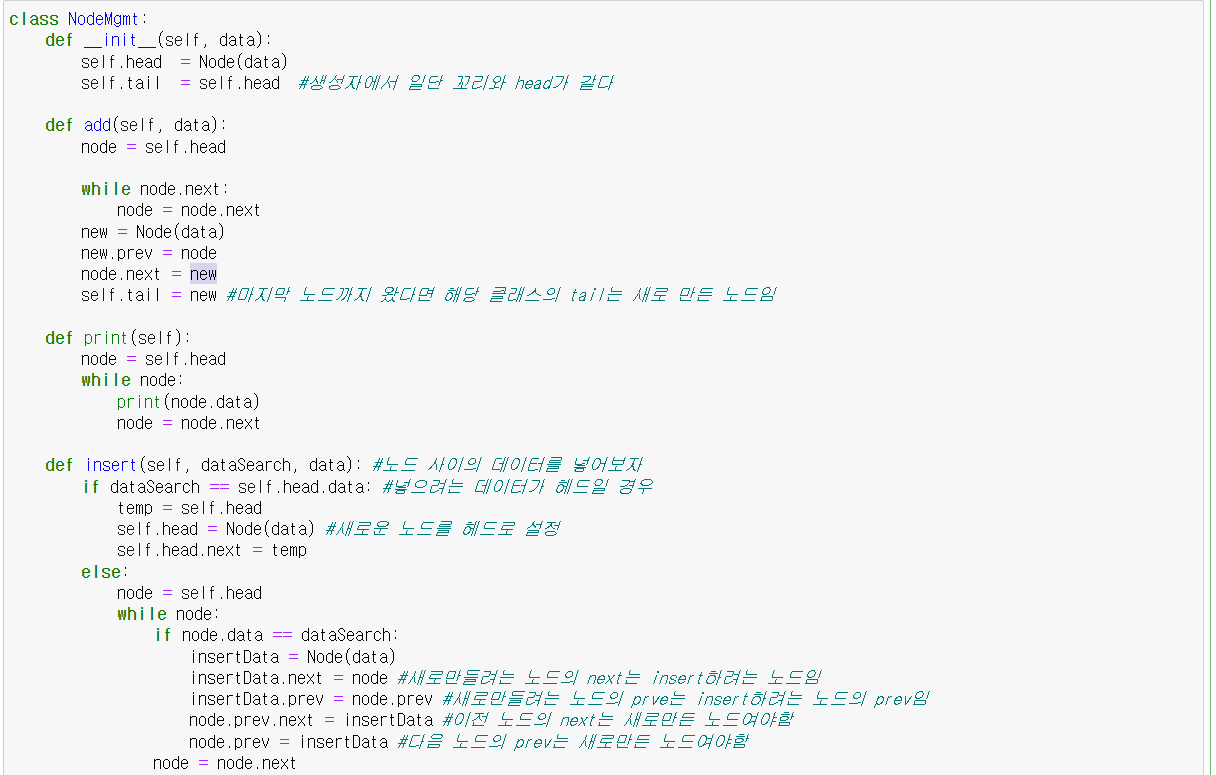
* 자바에서 null이 파이썬에선 None

#링크드 리스트로 데이터 추가  
  
#링크드 리스트로 출력하기  


* 링크드 리스트의 장점  
  -미리 데이터 공간을 미리 할당하지 않아도 됨
* 링크드 리스트의 단점  
  - 연결을 위한 별도 데이터 공간이 필요하므로, 저장공간 호율이 높지 않음  
  - 연결 정보를 찾는 시간이 필요하므로 접근 속도가 느림  
  - 중간 데이터 삭제시, 앞뒤 데이터의 연결을 재구성해야 하는 부가적인 작업 필요

# 링크드 리스트의 복잡한 기능1(데이터 사이에 데이터를 추가)  
  


노드를 지워보기(delete 메서드)  
  
원하는 데이터가 들어있는 노드의 주소값 찾기  


* 다양한 링크드 리스트 구조  
  -더블 링크드 리스트  
    
  이중 연결 리스트라고도함  
  장점: 양방향으로 연결되어 있어서 노트 탐색이 양쪽으로 모두 가능  
  

**시간 복잡도**

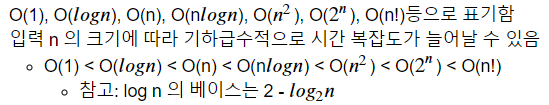
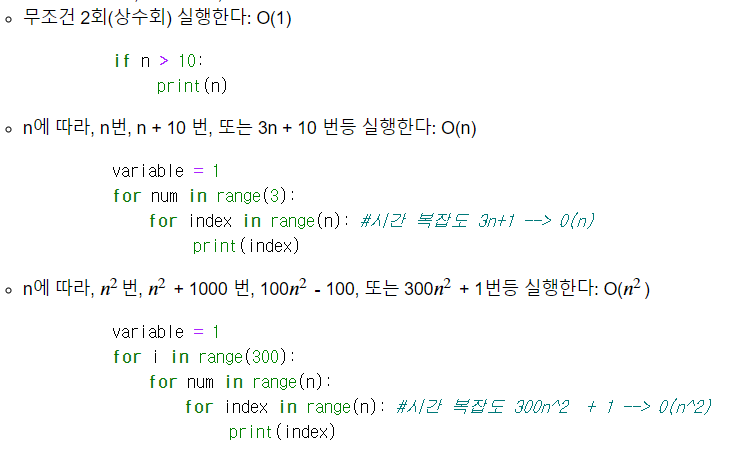
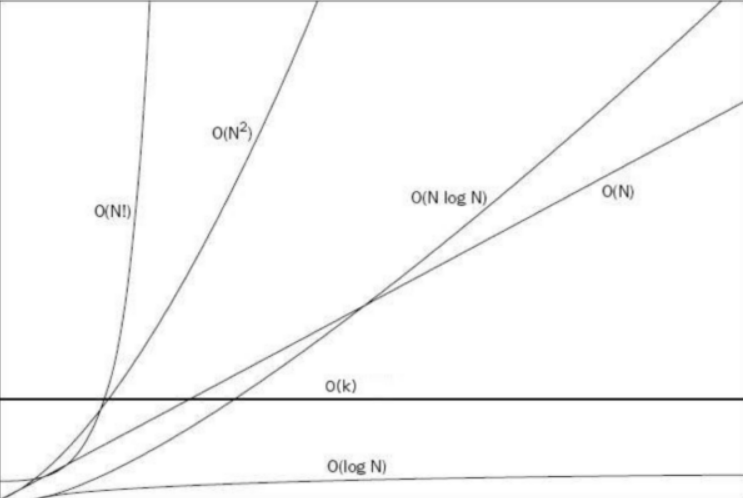
* 알고리즘 복잡도 계산이 필요한 이유  
  - 정수의 절대값 구하기  
  방법1: 정수값에 제곱한 값에 다시 루트를 씌우기  
  방법2: 정수가 음수인지 확인해서, 음수일 때만, -1을 곱하기  
  🡪다양한 알고리즘 중 어느 알고리즘이 더 좋은지를 분석하기 위해, 복잡도를 정의하고 계산함

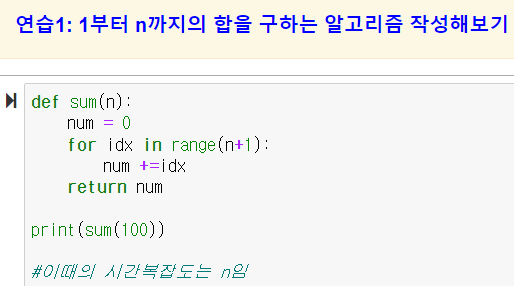
**시간 복잡도: 알고리즘 실행 속도(가장 중요)  
공간 복잡도: 알고리즘이 사용하는 메모리 사이즈**

알고리즘 시간 복잡도의 주요 요소 🡪 반복문이 지배함

* 알고리즘 성능 표기법  
  **-Big O(빅-오) 표기법: O(N) 🡪 가장 자주 씀**  
  **: 알고리즘 최악의 실행 시간을 표기  
  : 가장 많이/일반적으로 사용함**: 아무리 최악의 상황이라도, 이정도의 성능은 보장한다는 의미  
  -Ω (오메가) 표기법: Ω(N)  
  : 최상의 실행 시간을 표기  
  -Θ (세타) 표기법: Θ(N)  
  : 평균 실행 시간을 표기

대문자 O표기법

* 빅 오 표기법, Big-O 표기법 이라고도 부름
* O(입력)  
  -입력 n에 따라 결정되는 시간 복잡도 함수  
  O(1), O(𝑙𝑜𝑔𝑛logn), O(n), O(n𝑙𝑜𝑔𝑛logn), O(𝑛2n2), O(2𝑛2n), O(n!)등으로 표기함  
    
  표현식에 가장 큰 영향을 미치는 n의 단위로 표기  
     
  

만약 시간 복잡도 함수다 2n^2 + 3n이면 빅 오 표기법으로는 O(n^2)  


**해쉬 테이블(Hash Table)**

* 해쉬 구조  
  **-Hash Table: 키(Key)에 데이터(Value)를 저장하는 데이터 구조**-Key를 통해 바로 데이터를 받아올 수 있으므로, 속도가 획기적으로 빨라짐  
  -파이썬 딕셔너리 타입이 해쉬 테이블의 예: Key를 가지고 바로 데이터(Value)를 꺼냄
* 알아둘 용어  
  **-해쉬: 임의 값을 고정 길이로 변환하는 것**  
  **-해쉬 테이블: 키 값의 연산에 의해 직접 접근이 가능한 데이터 구조**  
  **-해싱 함수: Key에 대해 산술 연산을 이용해 데이터 위치를 찾을 수 있는 함수**  
  -해쉬 값, 해쉬 주소: Key를 해싱 함수로 연산해서 해쉬 값을 알아내고 이를 기반으로 해쉬 테이블에서 해당 Key에 대한 데이터 위치를 일관성 있게 찾을 수 있음  
  -별도의 키를 추출할 수 있는 함수가 있을 수 있음  
  