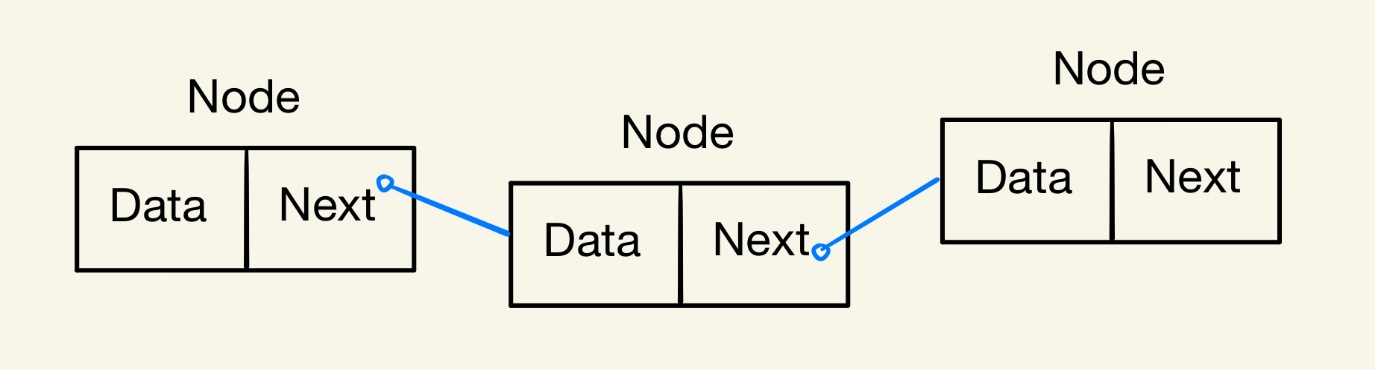
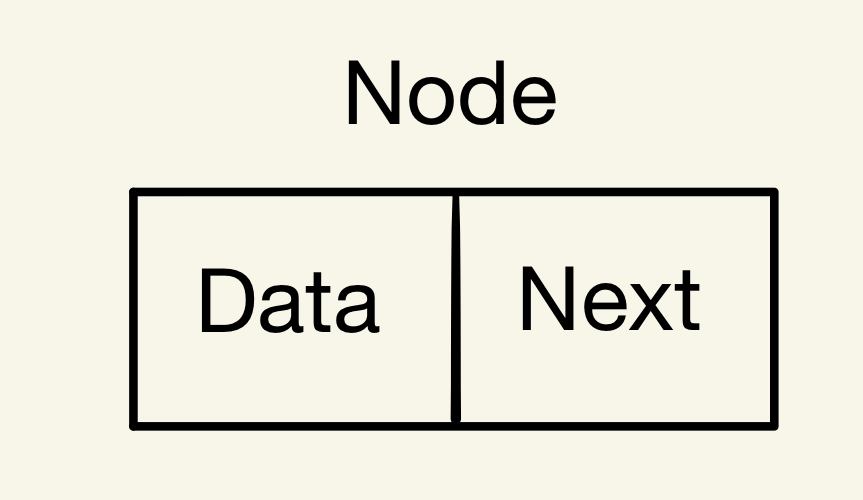
Linked list(연결 리스트)는 (데이터와 다음 노드의 주소를 담고 있는) 노드들이

한 줄로 연결되어 있는 방식의 자료 구조입니다.



**# Linked list란?**

Linked List는 데이터를 노드의 형태로 저장합니다. 노드에는 데이터와 다음 노드를 가르키는 포인터를 담은 구조로 이루어져 있습니다.

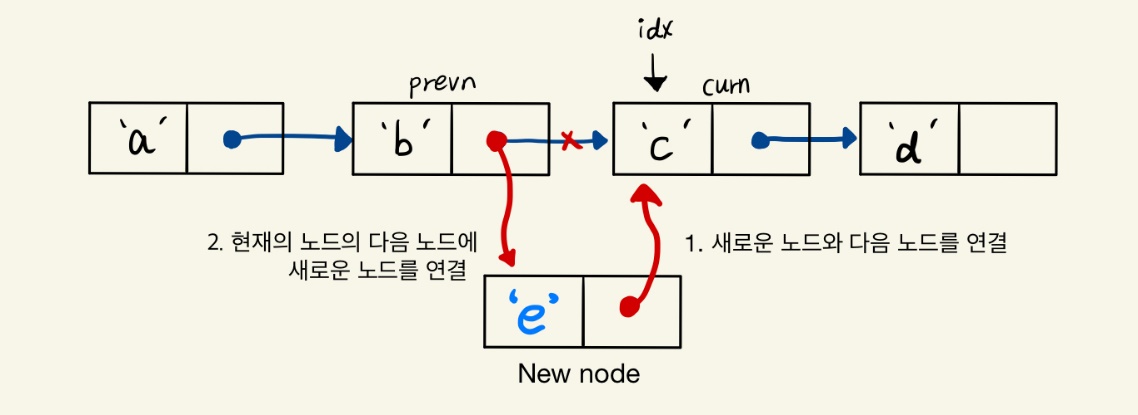
node의 모습

Linked list는 Array 처럼 선형 데이터 자료구조이지만, Array는 물리적인 배치 구조 자체가 연속적으로 저장되어 있고 Linked list는 위의 노드의 Next 부분에 다음 노드의 위치를 저장함으로써 선형적인 데이터 자료구조를 가집니다.

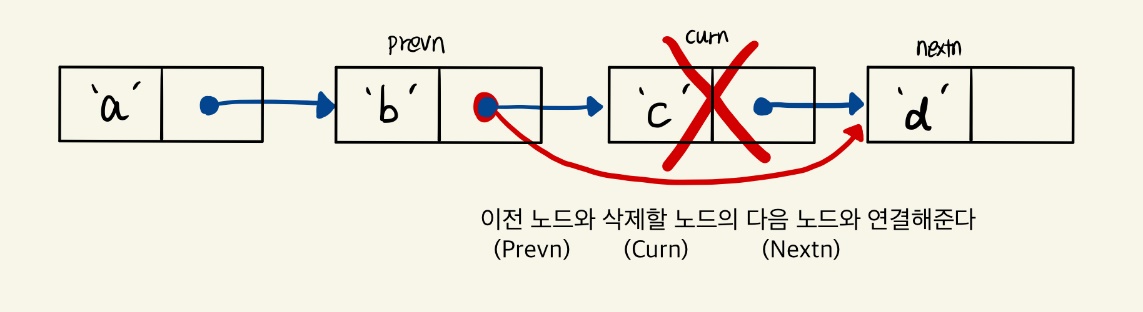
### # Linked list의 장점과 단점은요

Linked lsit의 장점은,   
(1) Linked list의 길이를 동적으로 조절 가능  
(2) 데이터의 삽입과 삭제가 쉬움  
  
Linked list의 단점은,  
1) 임의의 노드에 바로 접근할 수 가 없음  
(2) 다음 노드의 위치를 저장하기 위한 추가 공간이 필요함  
(3) Cache locality를 활용해 근접 데이터를 사전에 캐시에 저장하기가 어려움  
(4) Linked list를 거꾸로 탐색하기가 어려움

#### # 삽입



#### # 삭제



# Linkedlist는 자료 구현의 한방법

# 배열의 연속된 집합인 데이터 구현의 단점(2개)을 보완

#1. 배열은 실행전 크기를 선언하고 프로그램 실행도중에 배열의 크기를 바꿀수 없음

#2. 배열은 연속적으로 인덱싱이 되는 데이터이므로 한칸이 삭제 또는 추가될시 재인덱싱을 해야함(아파트)

# 위의 단점을 보안한게 링크드 리스트

# ------------------

# 코드 설명전중요한것

# 코드에는 번호 별로 주석을 달아놨습니다 번호 순서를 보면서 확인하시면 이해하기 편합니다.

# cur\_node 지금 가르키고있는것(이전의 데이터라고생각해도 좋음) /// new\_node 추가하는것

# Linkedlist 클래스 내부 : 기능(메서드) find(),show(),find\_previous(),remove(), insert() 기능

# Node클래스 내부 : 속성 데이터와 링크

# 변수명은 사이 식별가능하게 하는것인데 밑의 예저로 cur\_node로 하게되면 현재노드라는 의미고 이 기능을 구축 해야함

# 다른 사람이 쉽게 식별하고 이해하기위해 명칭을 짓는것

# ------------------

#Node 클래스 선언

# 사용이유 : Linkedlist의 노드의 데이터와 링크(next)의 속성을를 저장하기 위해서

# next가 None인 이유 : head로 기차 열칸처럼 줄줄이 이어지는 노드들은 삽입 과정이 끝났을때 마지막노드의 Insert()된 노드의 다음값은 없다

# 대신에 처음에 만들어준 노드는 find()라는 과정을 통하여 추가해줄것임

class Node: # 1: 노드정보를 담을노드클래스(데이터)를 선언 해준다(클래스내 데이터,다음데이터)

# 노드클래스초기화 (실행전에 정보를 주는 클래스)

def \_\_init\_\_(self,data): #6: 기본적인 head값 self.data =head ,self.next =none 실행

#11: a.insert('1','head').head==Node(1) a.insert('1','head').data = 1 , a.insert('1','head').next = None 됨

self.data = data

self.next = None

#linkedlist 클래스 선언

#사용이유 : 링크드리스트의 기능적인 방법(메서드)을 정의

class LinkedList: #2.Linkedlist 클래스 선언) (초기화,찾기,삽입,보여주기,이전의 노드찾기,삭제의 메서드만존재)

#링크드리스트클래스초기화

def \_\_init\_\_(self): #4.Linkedlist 클래스 속성(선언)

# 헤드 데이터에 Node('head')를 넣는이유 : 1: 기준 세우기 좋아서(링크드리스트는 처음 노드부터 읽는데 첫번째 노드 삭제또는 수정시 삭제 수정된것을 고려한 find()변수 만들어야함)

# (dummy Node라는 명칭으로 링크드리스트 구현시 첫번째를 의미)

self.head = Node('head') #5,7 Node('head')

# 5: Node('head')클래스 실행 및 'head'값 넣음

# 7: self.head에 6에 실행된정보 넣음

# def find()메서드

# 데이터 값을 입력하기위해 insert()메서드에 종속되는 탐색변수

# insert(new)에 new를 item이라는 매개변수로 사용

def find(self, item): #13: find('head') item='head'

# 비교 기준점인 현재 노드에 헤드를 넣는다

cur\_node = self.head #14: cur\_node = Node(head), 7번의 결과 같음

#첫번째 현재노드(헤드노드)의 데이터와 삽입하고자하는 이전노드 데이터가 같지않으면 같을때까지 맞춰주는 while문

#단순히 말해 현재노드와 내가 넣고자하는 새로운노드의 순서 맞추기위해서

while cur\_node.data != item: #15: 실행안됨 //cur\_node.data = a.head.data와같고(5,7번) 또한 Node('head').data와 같다.Node('head').data 는6번에서 head이니 처음 while문은 그대로 pass한다

cur\_node = cur\_node.next

#현재노드를 리턴해준다.

return cur\_node #16: 15의 while문 실행안되서 Node(head)리턴

# def insert()메서드

#삽입 노드의 데이터(new로들어감)와 삽입 이전의 데이터(current)를 넣는다

#기능 : 1) 노드추가 2) 노드의 next 이어주기

def insert(self, new, current): #9: a.insert('1','head'), new = 1 current= 'head' 입력

# 새로운 노드를 만들자

new\_node = Node(new) #10: Node(1) 실행후 #11 의 값이 저장된 Node(1)을 new\_node에 넣음

# 처음의 head부터 새로운노드 바로 이전의 노드를 찾아줘서 cur\_node라고 명시하자

cur\_node = self.find(current) #12, 17 // 12: a.insert('1','head').find('head') find('head')를 먼저실행

#17: cur\_node= Node(head)

# 새로운 노드에 cur\_node(현재노드)의 next는 None으로 되어 잇으니 그 값을 새로운 노드 next에 넣어주자

new\_node.next = cur\_node.next # 18: Node('head').next = None 을 new\_node.next에 할당

# cur\_node(현재의노드).next에 새로운 노드를 넣으면서 방향을 지시해주자

cur\_node.next = new\_node # 19: new\_node==Node(1)(10번)을 curr\_node.next==Node('head').next==Node('1')을 할당

# def show()

# 현재 구축한 링크드 리스트를 보여주자

def show(self):

# 헤드노드부터 쭉 조회를 시작하자

cur\_node = self.head

# 헤드노드가 None이 되는것은 마지막 노드가 될떄까지 Node클래스의 Data를 보여주고

# Cur\_node.next는 다음노드의클래스(Node(5))의형식이니 넘기면서 보여주자

while cur\_node.next is not None:

print(cur\_node.data, end='->')

cur\_node =cur\_node.next

print(cur\_node.data)

# def find\_previous()

# 이전의 노드를 찾는기능

def find\_previous(self, item):

# 헤드노드부터 출발

cur\_node = self.head

# 마지막 노드가 아니고 and 삭제하고자 하는 노드가 item과 일치할때까지 cur\_node를 만들자

while (cur\_node.next is not None) and ( cur\_node.next.data != item):

cur\_node = cur\_node.next

# cur\_node를 찾고 반환하자.

return cur\_node

# def remove()

# 노드를 지우는 기능, 삽입기능과 비슷

def remove(self, item):

# 이전의 노드를찾는 find\_previous()를 실행하여 cur\_node 를 prev\_node로 할당하자

prev\_node = self.find\_previous(item)

# prev노드의 next가 None이 아니면 .next.next로 건너뛰어서 버려버리자

# 간단하게 1, 2, 3잇으먄 1에 건너뛰는 next를 두번멱여서 head->1->3으로 버리자

# 왜냐하면 show()에서는 .next기준으로 파일을 출력하니까!

if prev\_node.next is not None :

prev\_node.next = prev\_node.next.next

a = LinkedList() #3.Linkedlist 클래스(객체)를 a로 객체의 인스턴스해준다. 3~7 실행

a.insert('1','head') #8. a의 인스턴스를 Linkedlist클래스 insert 메소드 실행 #9~19 실행

a.insert('3','1') # head -> 1 ->3

a.insert('5','3') # head -> 1 ->3 -> 5

a.insert('2','5') # head -> 1 ->3 -> 5 ->2

a.insert('4','2') # head -> 1 ->3 -> 5 ->2 ->4

a.show() #head -> 1 ->3 -> 5 -> 2->4

a.remove('2') # 데이터 노드 2 삭제,

a.show()