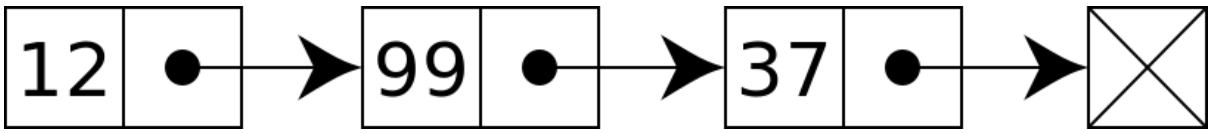
링크드 리스트(Linked List)

1. 링크드 리스트(Linked List) 구조

- 연결 리스트라고도 함
- 배열은 순차적으로 연결된 공간에 데이터를 나열하는 데이터 구조
- 링크드 리스트는 떨어진 곳에 존재하는 데이터를 화살표로 연결해서 관리하는 데이터 구조
- 본래 C언어에서는 주요한 데이터 구조이지만, **파이썬은 리스트 타입이 링크드 리스트의 기능을 모두 지원**

- 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

※ 일반적인 링크드 리스트 형태



wikipedia, https://en.wikipedia.org/wiki/Linked_list

2. 간단한 링크드 리스트 예

Node 구현

- 보통 파이썬에서 링크드 리스트 구현시, 파이썬 클래스를 활용함

```
class Node:
  def __init__(self,data):
    self.data = data
    self.next = None
```

```
class Node:
  def __init__(self,data,next=None):
    self.data = data
    self.next = next
```

Node와 Node연결하기(포인터 활용)

```
node1 = Node(1)
node2 = Node(2)

node1.next = node2
head = node1
```

링크드 리스트로 데이터 추가하기

```
class Node:
  def __init__(self,data,next=None):
    self.data = data
    self.next = next

  def add(data):
    node = head
    while node.next:#node의 next가 있으면
      node = node.next
    node.next = Node(data)
```

```
node1 = Node(1)
head = node1
for i in range(2,10):
  add(i)
```

링크드 리스트 데이터 출력하기(검색하기)

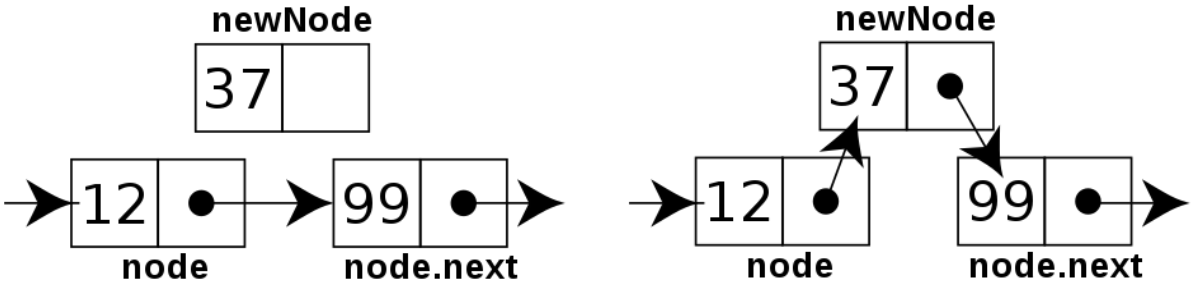
```
node = head
while node.next:
  print(node.data)
  node = node.next
print(node.data)

>>1,2,3,4,5,6,7,8,9
```

3. 링크드 리스트의 장단점(전통적인 C언어에서의 배열과 링크드 리스트)

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

4. 링크드 리스트의 복잡한 기능1(링크드 리스트 데이터 사이에 데이터를 추가)

- 링크드 리스트는 유지 관리에 부가적인 구현이 필요함




```
node = head
while node.next:
    print(node.data)
    node = node.next
print (node.data)

>>1,2,3,4,5,6,7,8,9
```

```
node3 = Node(1.5)
```

```
node = head
search = True

while search:
```

```
  if node.data ==1:
    search=False
  else:
    node = node.next

node_next = node.next
node.next = node3
node3.next = node_next
```

```
node = head
while node.next:
    print(node.data)
    node = node.next
print (node.data)

>>1,1.5,2,3,4,5,6,7,8,9
```

5. 파이썬 객체지향 프로그래밍으로 링크드 리스트 구현하기

```
class Node:
  def __init__(self,data,next=None):
    self.data = data
    self.next = next

class NodeMgmt:
  def __init__(self,data):
    self.head = Node(data)
  def add(self,data):
    if self.head=='':
      self.head = Node(data)
    else:
      node = self.head
      while node.next:
        node = node.next
      node.next = Node(data)
  def desc(self):
    node = self.head
    while node:
      print(node.data)
      node = node.next
```

```
linkedlist1 = NodeMgmt(0)
linkedlist1.desc()
>>0
```

```
for i in range(1,10):
  linkedlist1.add(data)
linkedlist1.desc()
>>0,1,2,3,4,5,6,7,8,9
```

6. 링크드 리스트의 복잡한 기능2(특정 노드를 삭제)

- 다음 코드는 위의 코드에서 delete메서드만 추가한 것임

```
class Node:
  def __init__(self,data,next=None):
    self.data = data
    self.next = next

class NodeMgmt:
  def __init__(self,data):
    self.data = Node(data)
  def add(self,data):
    if self.head=='':
      self.head = Node(data)
    else:
      node = self.head
      while node.next:
        node = node.next
      node.next = Node(data)
  def desc(self):
    node = self.head
    while node:
      print(node.data)
      node = node.next
  def delete(self,data):
    if self.head=='':
      print('해당 값을 가진 노드가 없다.')
      return
    if self.head.data==data: # 1. 헤드 삭제
      temp = self.head
      self.head = self.head.next
      del temp
    else:
      node = self.head
      while node.next:
        if node.next.data == data:
          temp = node.next
          node.next = node.next.next
          del temp
          return
        else:
          node = node.next
```

테스트를 위해 1개 노드를 만들어 봄

```
linkedlist1 = NodeMgmt(0)
linkedlist1.desc()
>>0
```

head가 살아있음을 확인

```
linkedlist1.head
>><__main__.Node at 0x1099fc6a0>
```

head를 지워봄

```
linkedlist1.delete(0)
```

다음 코드 실행시 아무것도 안나온다는 것은 linkedlist1.head가 정상적으로 삭제되었음을 의미

```
linkedlist1.head
```

다시 하나의 노드를 만들어봄

```
linkedlist1 = NodeMgmt(0)
linkedlist1.desc()
>>0
```

이번엔 여러 노드를 더 추가

```
for i in range(1,10):
  linkedlist1.add(i)
linkedlist1.desc()
>>0,1,2,3,4,5,6,7,8.9
```

노드 중에 한개를 삭제함

```
linkedlist1.delete(4)
linkedlist1.desc()
>>0,1,2,3,5,6,7,8,9
```

위 코드에서 노드 데이터가 특정 숫자인 노드를 찾는 함수를 만들고, 테스트 해보기

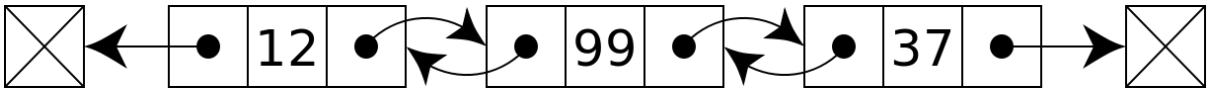
```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class NodeMgmt:
    def __init__(self, data):
        self.head = Node(data)

    def add(self, data):
        if self.head == '':
            self.head = Node(data)
        else:
            node = self.head
            while node.next:
                node = node.next
            node.next = Node(data)

    def desc(self):
        node = self.head
        while node:
            print (node.data)
            node = node.next

    def delete(self, data):
        if self.head == '':
            print ('해당 값을 가진 노드가 없습니다.')
            return
        if self.head.data == data: # 경우의 수1: self.head를 삭제해야할 경우 - self.head를 바꿔줘야 함
            temp = self.head # self.head 객체를 삭제하기 위해, 임시로 temp에 담아서 객체를 삭제했음
            self.head = self.head.next # 만약 self.head 객체를 삭제하면, 이 코드가 실행이 안되기 때문!
            del temp
        else:
            node = self.head
            while node.next: # 경우의 수2: self.head가 아닌 노드를 삭제해야할 경우
                if node.next.data == data:
                    temp = node.next
                    node.next = node.next.next
                    del temp
                    pass
                else:
                    node = node.next
    def search_node(self,data):
      node = self.head
      while node:
        if node.data == data:
          return node
        else:
          node = node.next
```

```
#테스트
node_mgmt = NodeMgmt(0)
```

```
for i in range(1,10):
  node_mgmt.add(i)

node = node_mgmt.search_node(4)
print(node.data)
>>4
```

7. 다양한 링크드 리스트 구조

- 더블 링크드 리스트(Doubly linked list)기본 구조
  - 이중 연결 리스트라고도 함
  - 장점 : 양방향으로 연결되어 있어서 노드 탐색이 양쪽으로 모두 가능




```
class Node:
  def __init__(self,data,prev=None,next=None):
    self.prev=prev
    self.data=data
    self.next=next
class NodeMgmt:
  def __init__(self,data):
    self.head = Node(data)
    self.tail = self.head

  def insert(self,data):
    if self.head==None:
      self.head = Node(data)
      self.tail = self.head
    else:
      node = self.head
      while node.next:
        node = node.next
      new = Node(data)
      node.next = new
      new.prev = node
      self.tail = new

  def desc(self):
    node = self.head
    while node:
      print(node.data)
      node = node.next
```

```
double_linked_list = NodeMgmt(0)
for i in range(1,10):
  double_linked_list.insert(i)
double_linked_list.desc()
>>0,1,2,3,4,5,6,7,8,9
```

연습 : 위 코드에서 노드 데이터가 특정 숫자인 노드 앞에 데이터를 추가하는 함수를 만들고, 테스트 해보기

- 더블 링크드 리스트의 tail에서부터 뒤로 이동하며, 특정 숫자인 노드를 찾는 방식으로 함수를 구현하기
- 테스트 : 임의로 0~9까지 데이터를 링크드 리스트에 넣어보고, 데이터 값이 2인 노드 앞에 1.5 데이터 값을 가진 노드를 추가해보기

```
class Node:
  def __init__(self,data,prev=None,next=None):
    self.prev=prev
    self.next=next
    self.data=data

class NodeMgmt:
  def __init__(self,data):
    self.head = Node(data)
    self.tail = self.head

  def insert(self,data):
    if self.head == None:
      self.head = Node(data)
      self.tail = self.head
    else:
      node = self.head
      while node.next:
        node = node.next
      new = Node(data)
      node.next = new
      new.prev = node
      self.tail = new

  def desc(self):
    node = self.head
    while node:
      print(node.data)
      node = node.next

  def search_from_head(self,data):
    if self.head ==None:
      return False
    node = self.head
    while node:
      if node.data == data:
        return node
      else:
        node = node.next
    return False

  def search_from_tail(self,data):
    if self.head ==None:
      return False
    node = self.tail
```

```
    while node:
      if node.data == data:
        return node
      else:
        node = node.prev
    return False

  def insert_before(self,data,before_data):
    if self.head == None:
      self.head = Node(data)
      return True
    else:
      node = self.tail
      while node.data!=before_data:
        node = node.prev
        if node==None:
          return False
      new = Node(data)
      before_new = node.prev
      before_new.next = new
      new.prev = before_new
      new.next = node
      node.prev = new
      return True
```

```
double_linked_list = NodeMgmt(0)
for data in range(1, 10):
    double_linked_list.insert(data)
double_linked_list.desc()
>>0,1,2,3,4,5,6,7,8,9
```

```
node_3 = double_linked_list.search_from_tail(3)
node_3.data
>>3
```

```
double_linked_list.insert_before(1.5, 2)
double_linked_list.desc()
>>0,1,1,5,2,3,4,5,6,7,8,9
```

```
node_3 = double_linked_list.search_from_tail(1.5)
node_3.data
>>1.5
```