

원형 연결 리스트

학습목표

- 원형 연결 리스트의 개념을 파악한다.
- 원형 연결 리스트와 단순 연결 리스트의 차이를 이해한다.
- 원형 연결 리스트의 데이터 삽입/삭제 원리를 이해한다.
- 파이썬으로 원형 연결 리스트를 조작하는 코드를 작성한다.

SECTION 00 생활 속 자료구조와 알고리즘

SECTION 01 원형 연결 리스트의 기본

SECTION 02 원형 연결 리스트의 간단 구현

SECTION 03 원형 연결 리스트의 일반 구현

SECTION 04 원형 연결 리스트의 응용

연습문제

응용예제



Section 00 생활 속 자료구조와 알고리즘

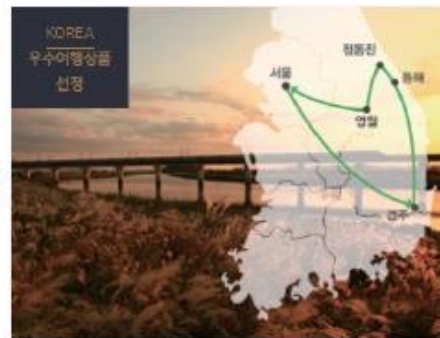
■ 원형 연결 리스트란?

- 시작 위치와 다음 위치가 계속 이어진 후 마지막에 다시 시작으로 돌아오는 형태



전국일주 2박 3일

전국 방방곡곡을 다니는 명품 기차 여행



동부권 1박 2일

동남부권을 다니는 명품 기차 여행



서부권 1박 2일

서남부권을 다니는 명품 기차 여행

Section 01 원형 연결 리스트의 기본

■ 원형 연결 리스트의 개념

- 단순 연결 리스트와 구조와 구현 코드가 상당히 유사
- 리스트 형태가 원(Circle) 형태로 구성(계속 회전하면서 연속 가능)
- 오버헤드가 발생하지 않음

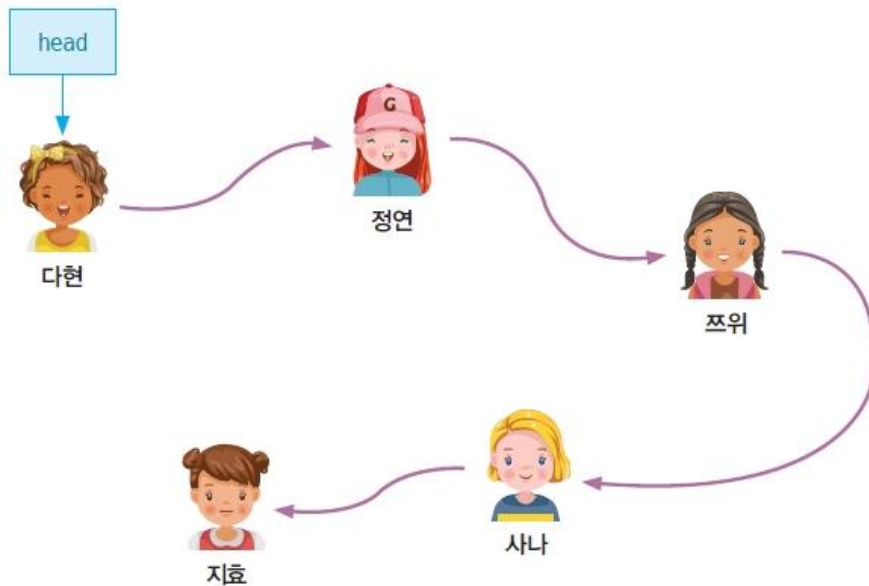


그림 5-1 단순 연결 리스트 예

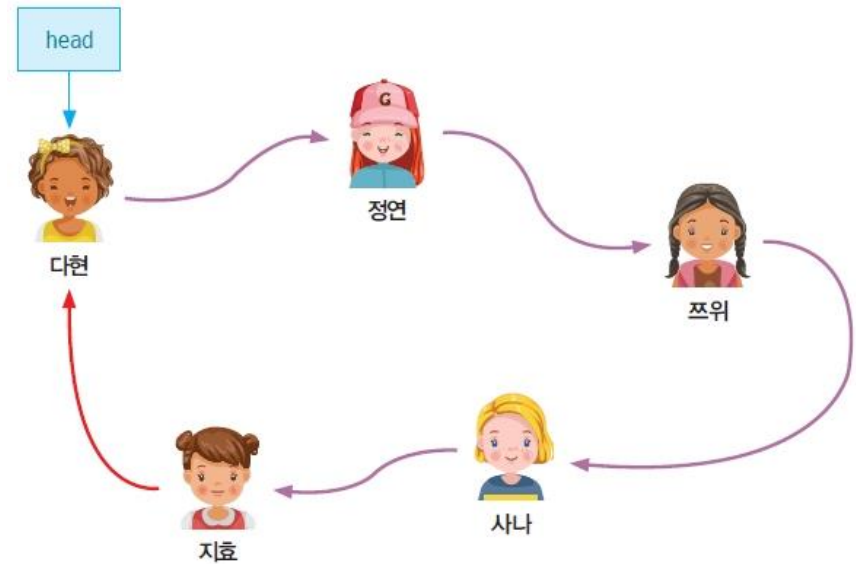


그림 5-2 원형 연결 리스트 예

Section 01 원형 연결 리스트의 기본

■ 원형 연결 리스트의 원리

■ 노드 구조

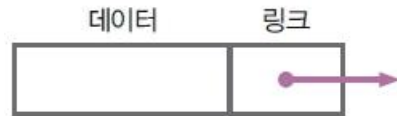


그림 5-3 노드 구조(단순 연결 리스트와 원형 연결 리스트 공통)

노드로 표현

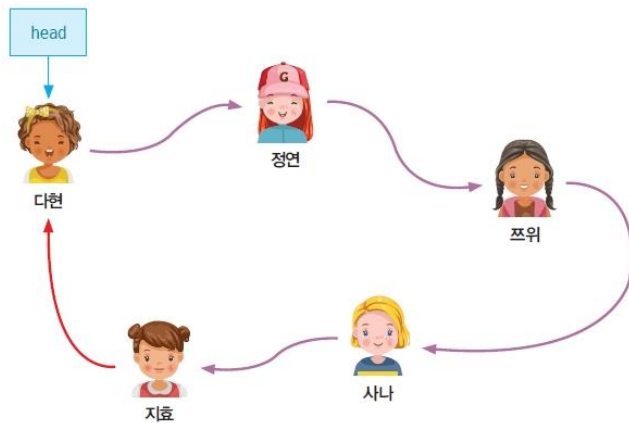


그림 5-2 원형 연결 리스트 예

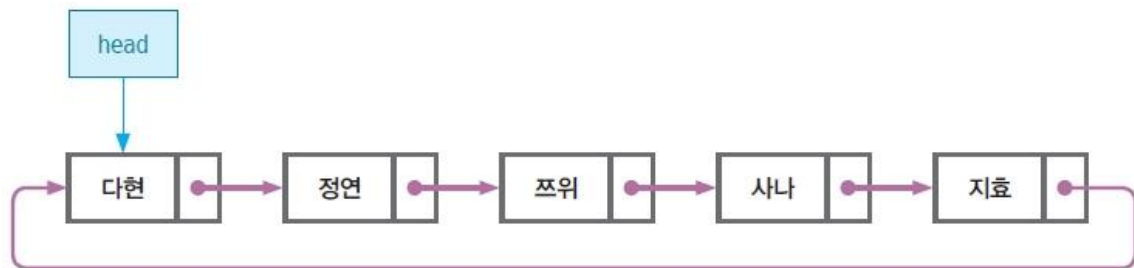


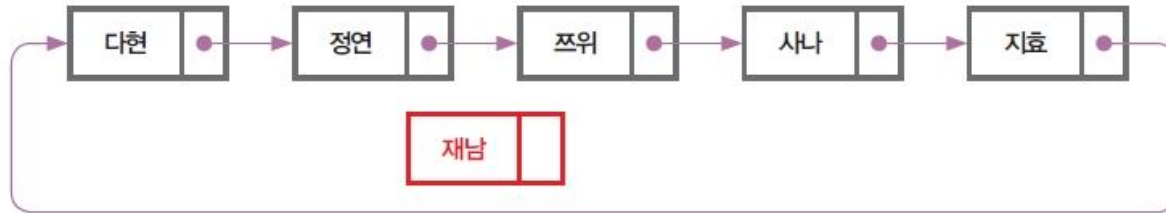
그림 5-4 노드로 구성된 원형 연결 리스트

Section 01 원형 연결 리스트의 기본

- 노드 삽입 : 중간에 노드 삽입

1단계

새 노드 생성



2단계

링크 수정

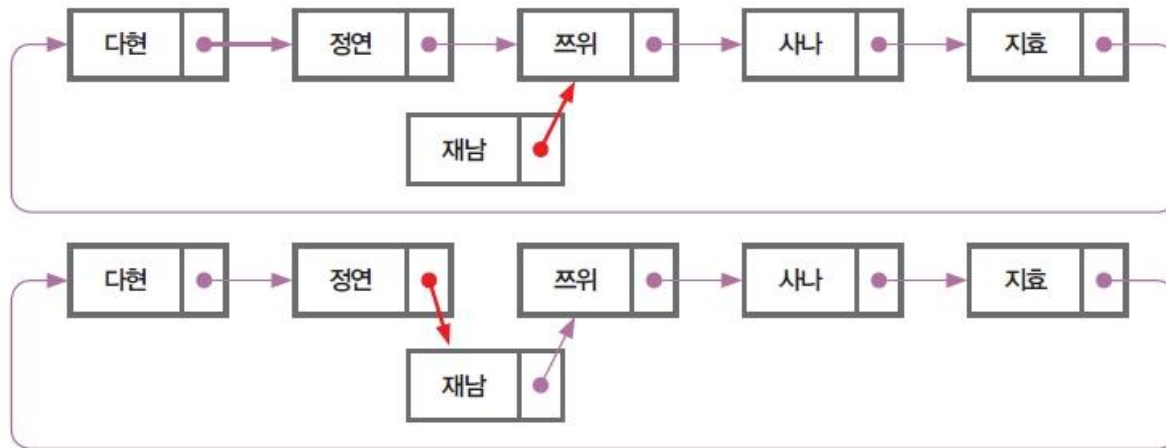


그림 5-5 원형 연결 리스트에서 중간 위치에 노드를 삽입하는 과정 예

Section 01 원형 연결 리스트의 기본

- 노드 삭제 : 중간 노드 삭제

1단계
링크 수정



2단계
노드 삭제



그림 5-6 원형 연결 리스트에서 데이터를 삭제하는 과정 예

Section 02 원형 연결 리스트의 간단 구현

■ 노드의 생성과 연결 구현

■ 노드 생성

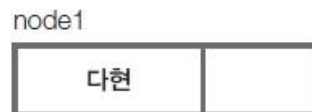
■ 첫 번째 노드 생성

```
1 { class Node() :  
    def __init__(self):  
        self.data = None  
        self.link = None  
2 { node1 = Node()  
   node1.data = "다현"  
3 node1.link = node1
```

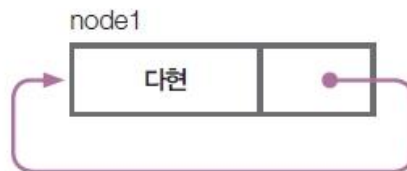
Node라는 데이터형을 만드는 것

데이터형을 생성할 때 자동으로 실행되는 부분

데이터와 링크가 저장되는 부분



(a)



(b)

그림 5-7 첫 번째 노드 생성 결과

Section 02 원형 연결 리스트의 간단 구현

■ 노드 연결

- 두 번째 노드 생성 후 첫 번째 노드의 링크로 연결

```
node2 = Node()  
node2.data = "정연"  
① node1.link = node2  
② node2.link = node1
```

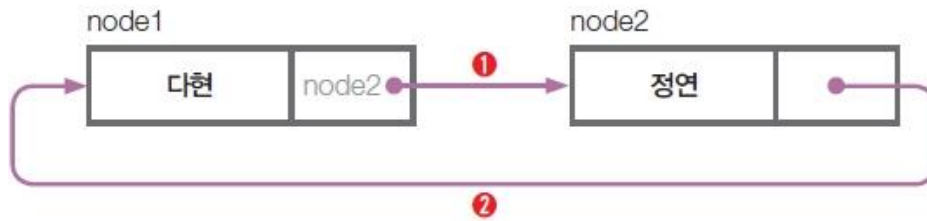


그림 5-8 두 번째 노드 생성 결과

Section 02 원형 연결 리스트의 간단 구현

■ 데이터가 5개인 원형 연결 리스트 생성



그림 5-9 생성할 원형 연결 리스트 예

Code05-01.py 데이터가 5개인 원형 연결 리스트 생성

```
1 class Node() :
2     def __init__(self) :
3         self.data = None
4         self.link = None
5
6 node1 = Node()
7 node1.data = "다현"
8 node1.link = node1
9
10 node2 = Node()
11 node2.data = "정연"
12 node1.link = node2
```

Section 02 원형 연결 리스트의 간단 구현

```
13 node2.link = node1
14
15 node3 = Node()
16 node3.data = "쯔위"
17 node2.link = node3
18 node3.link = node1
19
20 node4 = Node()
21 node4.data = "사나"
22 node3.link = node4
23 node4.link = node1
24
25 node5 = Node()
26 node5.data = "지효"
27 node4.link = node5
28 node5.link = node1
29
30 current = node1
31 print(current.data, end = ' ')
32 while current.link != node1 :
33     current = current.link
34     print(current.data, end = ' ')
```

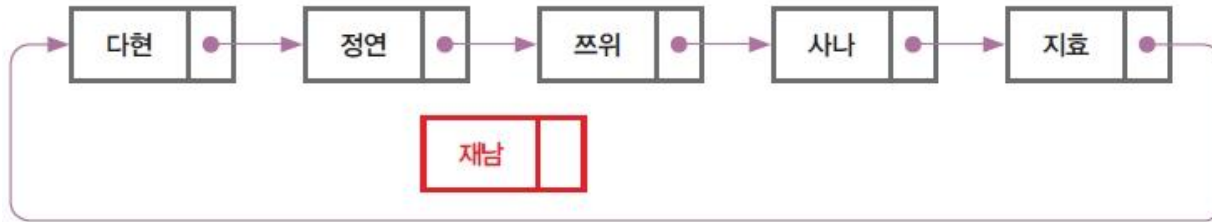
실행 결과

다현 정연 쑤위 사나 지효

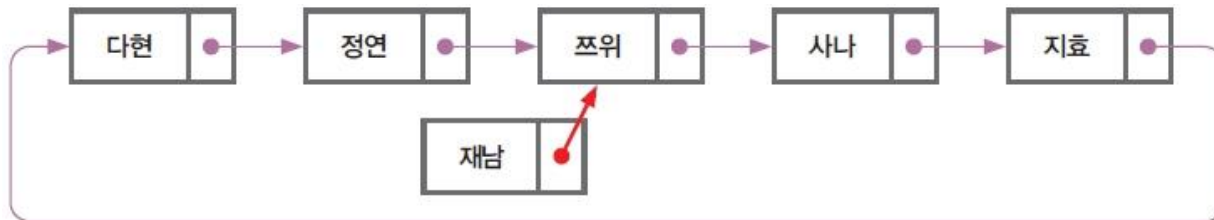
Section 02 원형 연결 리스트의 간단 구현

■ 노드 삽입 : 중간에 데이터 삽입

1 새 노드를 생성하고 데이터에 재남을 입력한다.



2 새 노드(재남 노드)의 링크에 정연 노드의 링크를 복사한다.
그러면 정연 노드와 재남 노드 모두 쫘위 노드를 가리킨다.



Section 02 원형 연결 리스트의 간단 구현

3 정연 노드의 링크에 재남 노드를 지정한다.

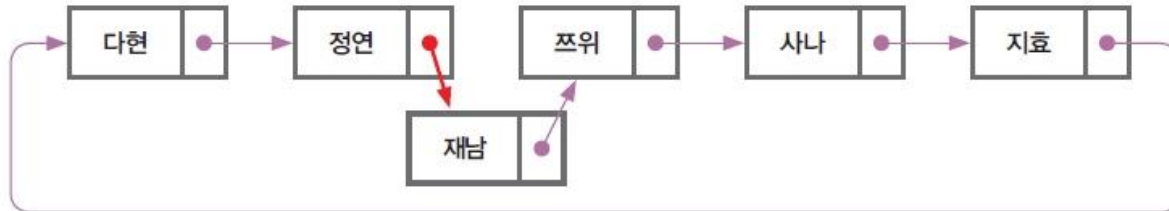


그림 5-10 원형 연결 리스트의 중간 노드 삽입

Code05-02.py 데이터가 5개인 원형 연결 리스트의 중간 노드 삽입

```
... # 생략(Code05-01.py의 1~28행과 동일)
29
30 newNode = Node()
31 newNode.data = "재남"
32 newNode.link = node2.link # 정연 노드의 링크
33 node2.link = newNode
34
... # 생략(Code05-01.py의 30~34행과 동일)
```

실행 결과

다현 정연 재남 쑤위 사나 지효

Section 02 원형 연결 리스트의 간단 구현

■ 노드 삭제 : 중간 데이터 삭제

- 1 삭제할 째워 노드의 링크를 바로 앞 정연 노드의 링크로 복사한다.
그러면 정연 노드가 사나 노드를 가리킨다.



- 2 째워 노드를 삭제한다.



그림 5-11 원형 연결 리스트의 노드 삭제

Section 02 원형 연결 리스트의 간단 구현

Code05-03.py 데이터가 5개인 원형 연결 리스트의 중간 노드 삭제

```
... # 생략(Code05-01.py의 1~28행과 동일)
29
30 node2.link = node3.link
31 del(node3)
32
33 current = node1
34 print(current.data, end = ' ')
35 while current.link != node1 :
36     current = current.link
37     print(current.data, end = ' ')
```

실행 결과

다현 정연 사나 지효

Section 03 원형 연결 리스트의 일반 구현

■ 원형 연결 리스트의 일반 형태

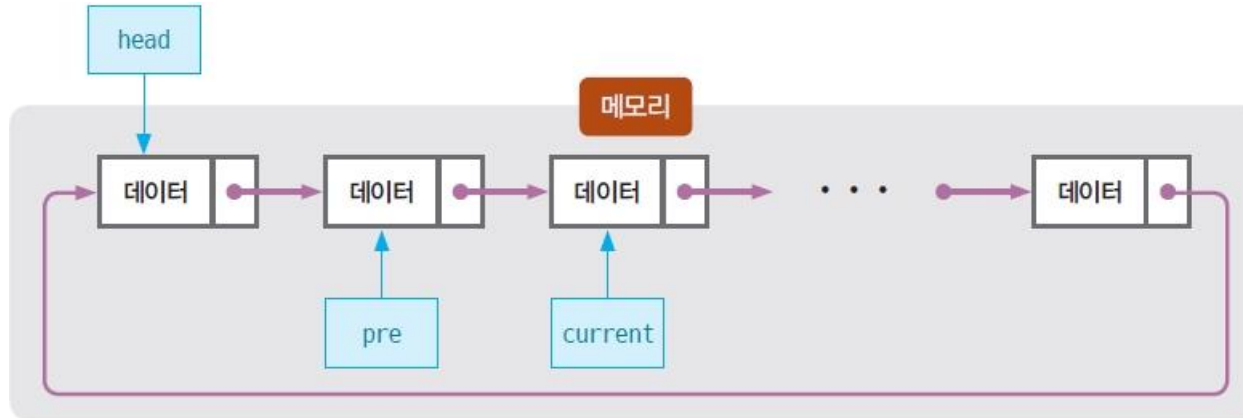


그림 5-12 원형 연결 리스트의 전체 구성 환경

- 헤드(head)는 첫 번째 노드를, 현재(current)는 지금 처리 중인 노드를, 이전(pre)은 현재 처리 중인 노드의 바로 앞 노드를 가리킴
- 처음에는 모두 비어 있으면 되므로 다음과 같이 초기화함

```
memory = []  
head, current, pre = None, None, None
```

Section 03 원형 연결 리스트의 일반 구현

■ 배열에 저장된 데이터 입력 과정

- 데이터 입력 과정 : 첫 번째 데이터 입력

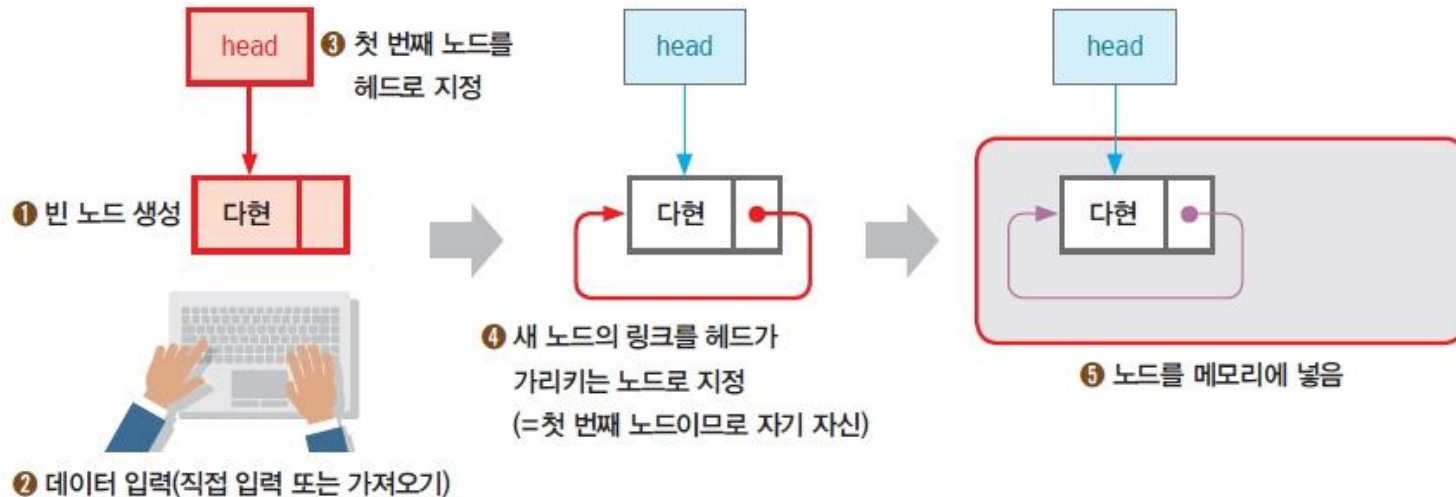


그림 5-13 원형 연결 리스트의 첫 번째 데이터 입력

```
① node = Node()
② node.data = dataArray[0] # 첫 번째 노드
③ head = node
④ node.link = head
⑤ memory.append(node)
```


Section 03 원형 연결 리스트의 일반 구현

- 데이터 입력 과정 : 두 번째 이후 데이터 입력

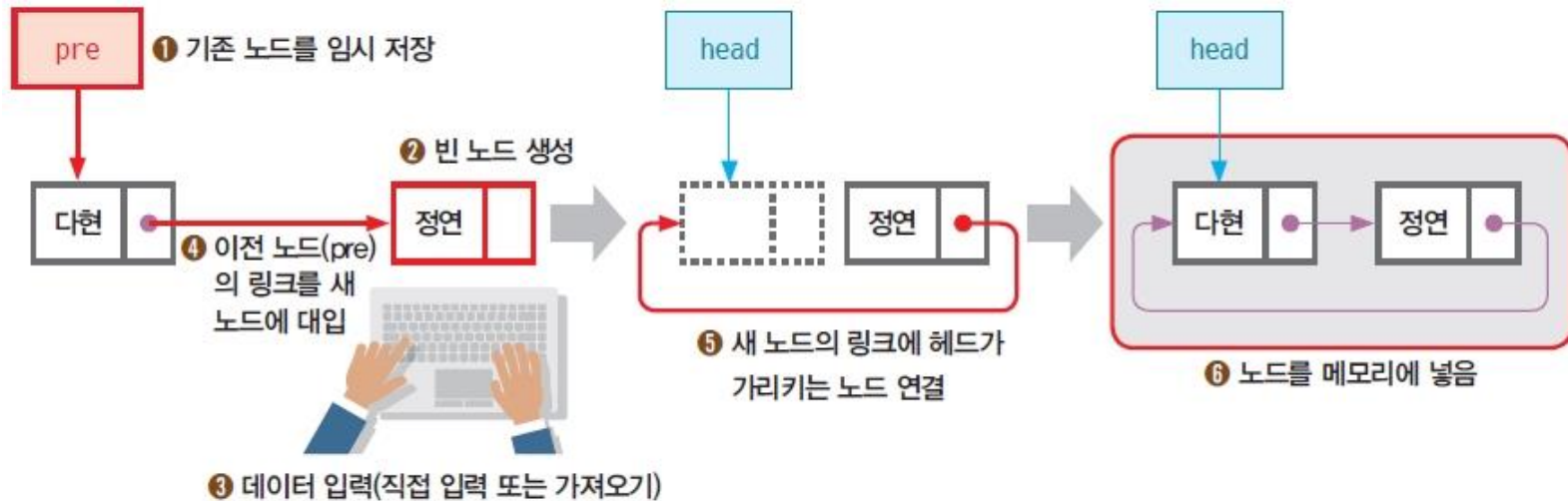


그림 5-14 원형 연결 리스트의 두 번째 이후 데이터 입력

```
1 pre = node
2 node = Node()
3 node.data = data    # 두 번째 이후 노드
4 pre.link = node
5 node.link = head
6 memory.append(node)
```

Section 03 원형 연결 리스트의 일반 구현

- 원형 연결 리스트의 생성 함수 완성

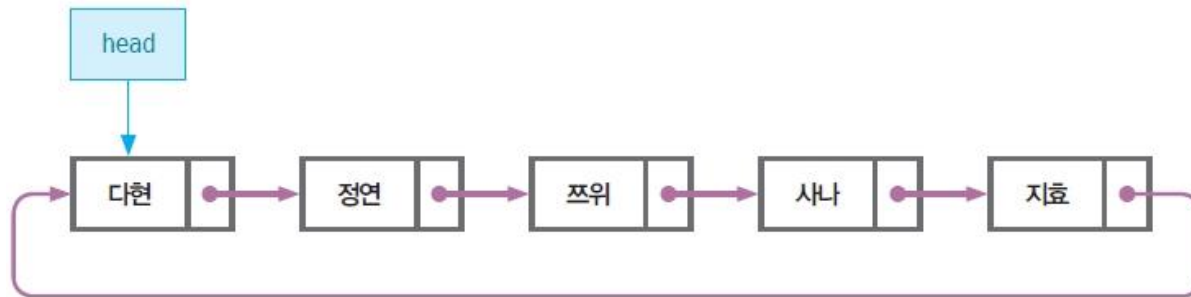


그림 5-15 배열을 이용하여 생성할 원형 연결 리스트 예

Code05-04.py 원형 연결 리스트 생성

```
1  ## 클래스와 함수 선언 부분 ##
2  class Node() :
3      def __init__(self) :
4          self.data = None
5          self.link = None
6
7  def printNodes(start) :
8      current = start
9      if current.link == None :
10         return
11     print(current.data, end = ' ')
12     while current.link != start :
13         current = current.link
14         print(current.data, end = ' ')
```

Section 03 원형 연결 리스트의 일반 구현

```
15     print()
16
17 ## 전역 변수 선언 부분 ##
18 memory = []
19 head, current, pre = None, None, None
20 dataArray = ["다현", "정연", "쯔위", "사나", "지효"]
21
22 ## 메인 코드 부분 ##
23 if __name__ == "__main__" :
24
25     node = Node()
26     node.data = dataArray[0]    # 첫 번째 노드
27     head = node
28     node.link = head
29     memory.append(node)
30
31     for data in dataArray[1:] : # 두 번째 이후 노드
32         pre = node
33         node = Node()
34         node.data = data
35         pre.link = node
36         node.link = head
37         memory.append(node)
38
39     printNodes(head)
```

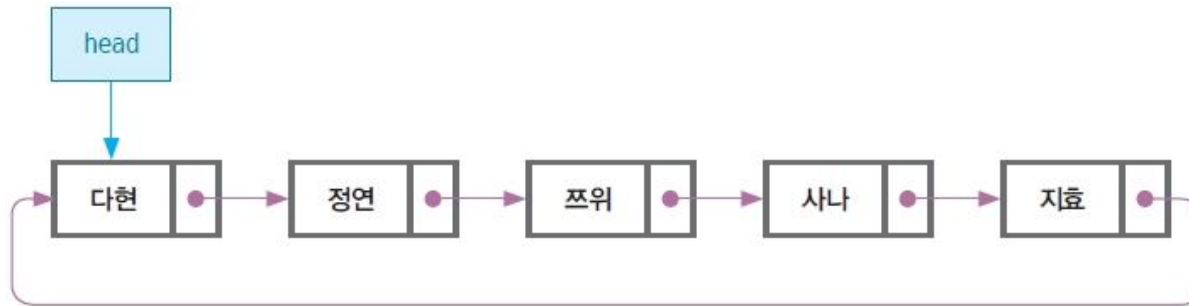
실행 결과

다현 정연 쫘위 사나 지효

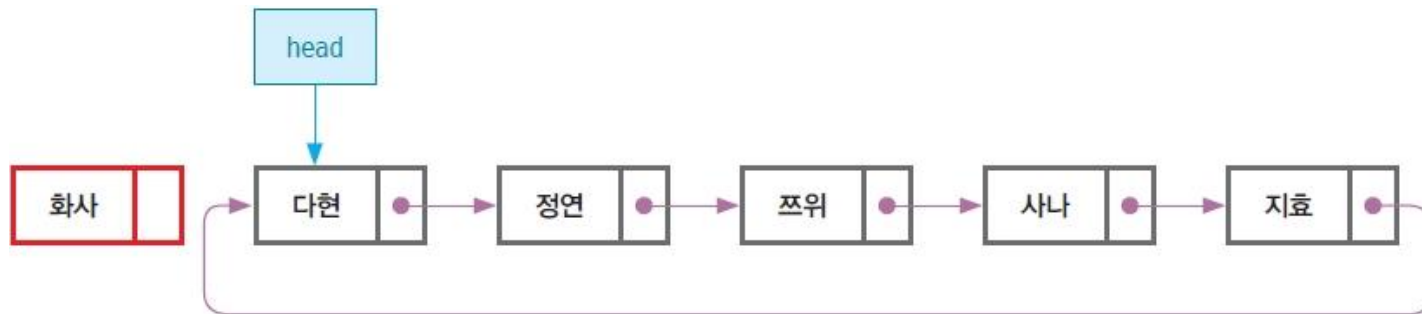
Section 03 원형 연결 리스트의 일반 구현

■ 노드 삽입 : 첫 번째 노드 삽입

0 맨 앞에 노드를 삽입하기 전 초기 상태

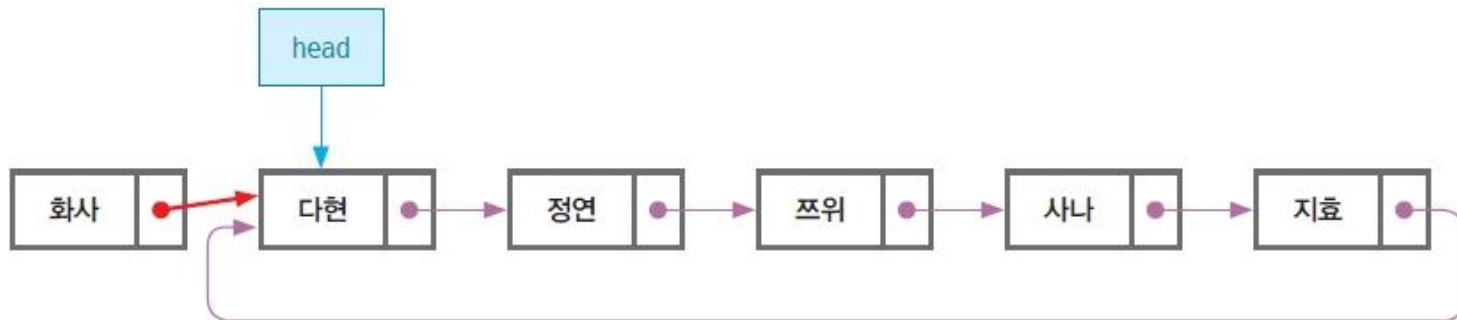


1 새 노드(화사 노드)를 생성한다.

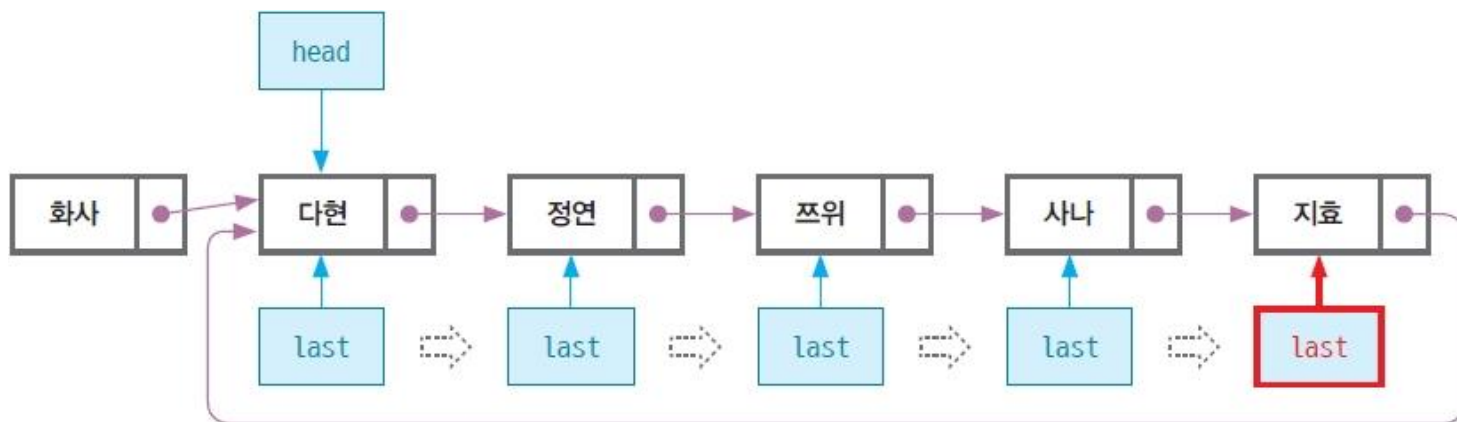


Section 03 원형 연결 리스트의 일반 구현

2 새 노드의 링크로 헤드(head)가 가리키는 노드를 지정한다.

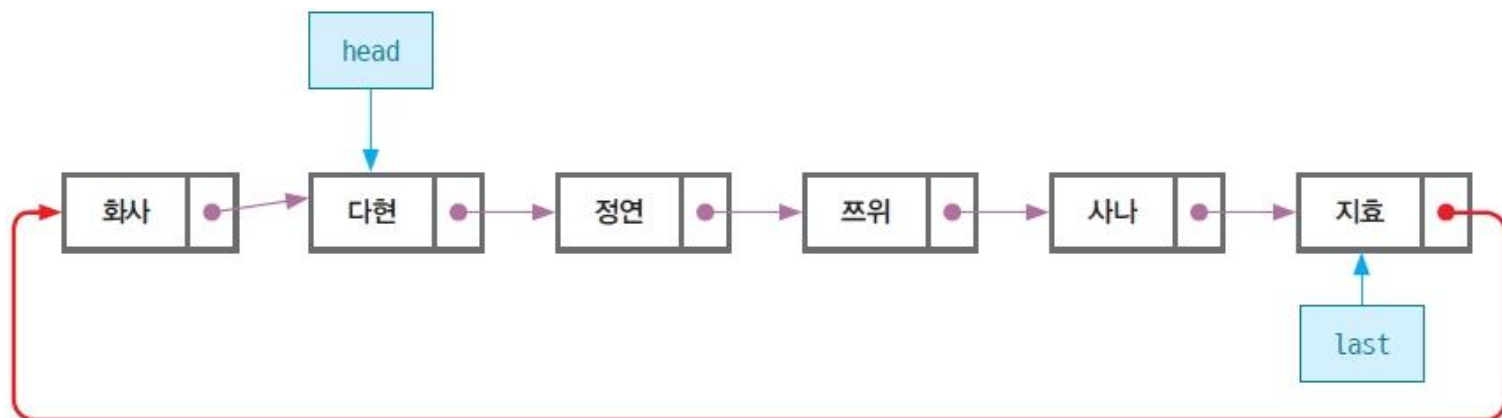


3 헤드가 가리키는 노드에서 시작해서 last를 다시 다음 노드로 변경하며 마지막 노드를 찾는다.

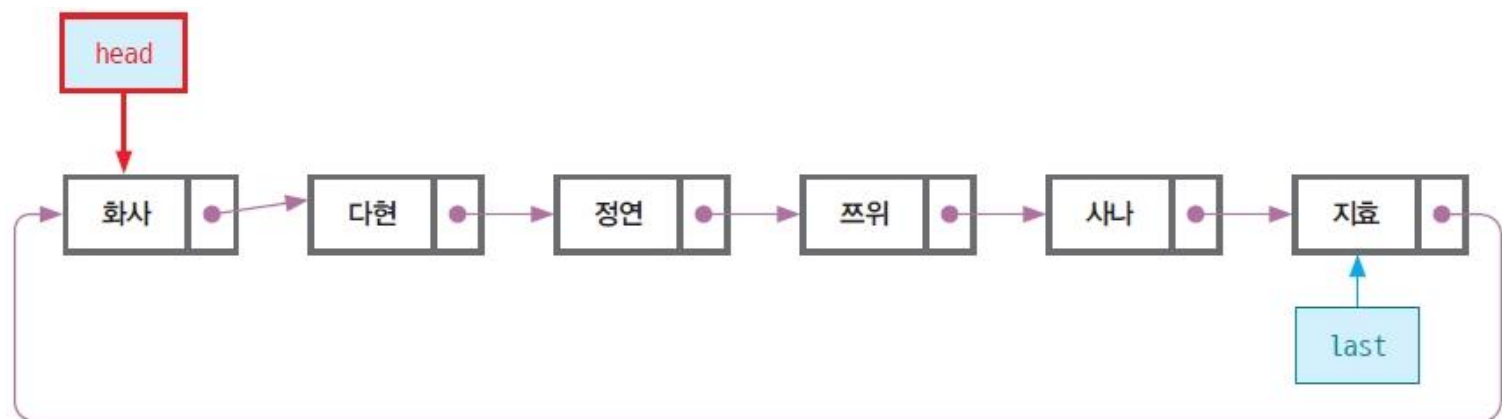


Section 03 원형 연결 리스트의 일반 구현

4 마지막 노드의 링크에 새 노드를 지정한다.



5 헤드 노드를 새 노드로 지정한다.



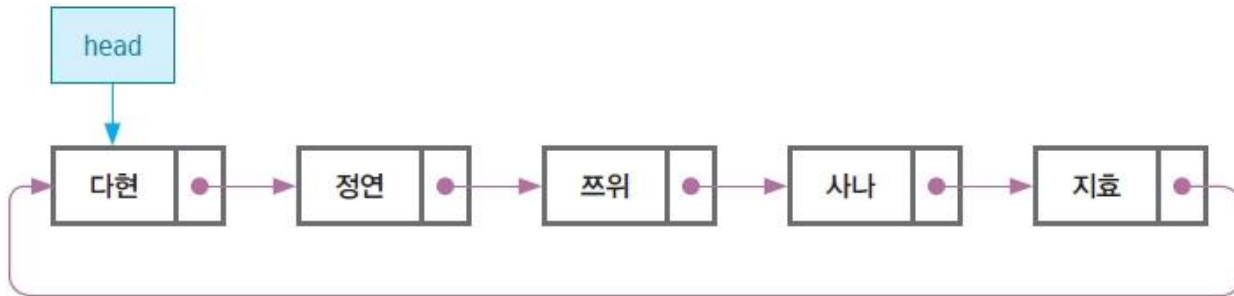
Section 03 원형 연결 리스트의 일반 구현

```
1 { node = Node()  
  node.data = "화사"  
2 { node.link = head  
  last = head          # 마지막 노드를 첫 번째 노드로 우선 지정  
3 { while 마지막 노드까지 : # 마지막 노드를 찾으면 반복 종료  
    last = last.link      # last를 다음 노드로 변경  
4 last.link = node        # 마지막 노드의 링크에 새 노드 지정  
5 head = node
```

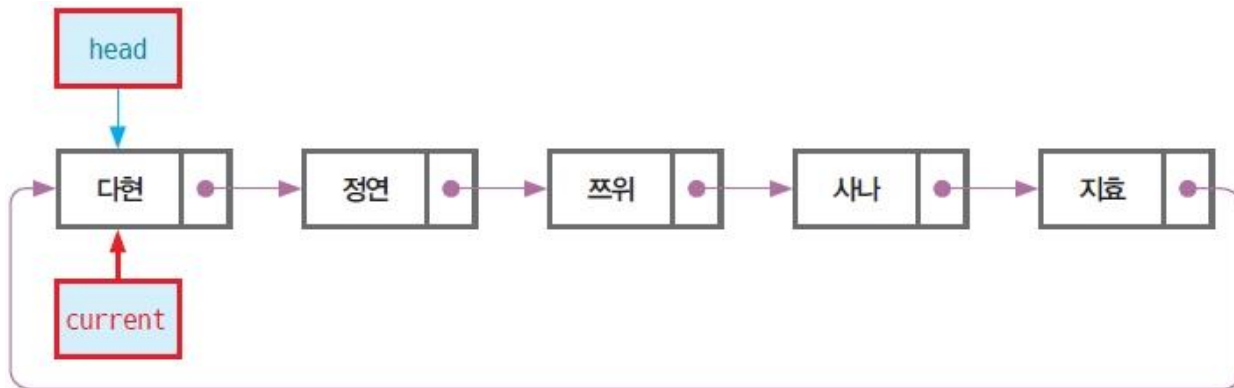
Section 03 원형 연결 리스트의 일반 구현

■ 노드 삽입 : 중간 노드 삽입

0 중간 노드 삽입 전 초기 상태

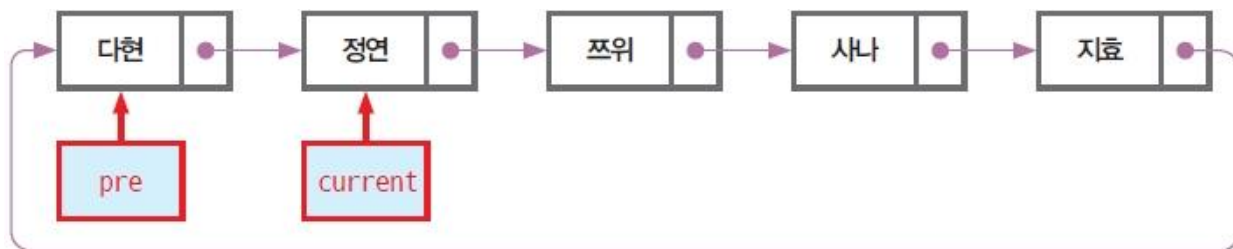


1 헤드(head)에서 시작해서 현재(current) 노드가 사나인지 확인한다.

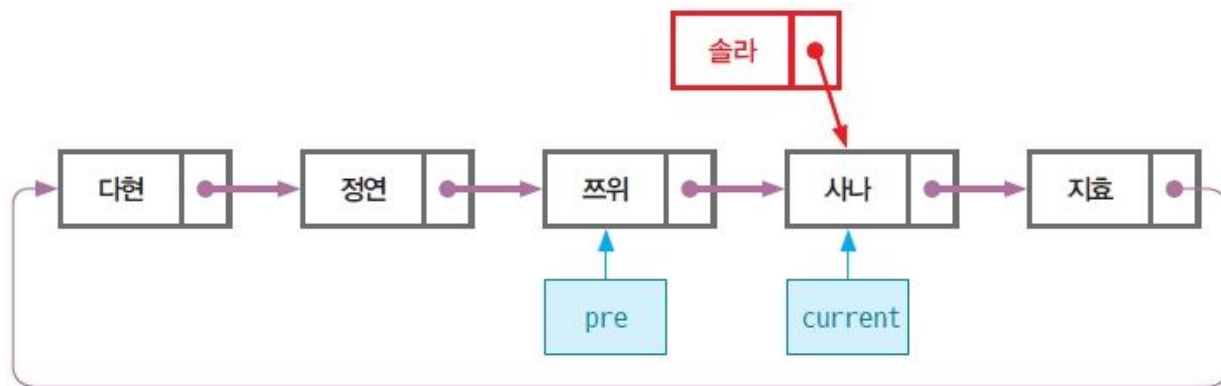


Section 03 원형 연결 리스트의 일반 구현

- 2 현재 노드를 이전(pre) 노드로 저장하고, 현재 노드를 다음 노드로 이동한다.
그리고 현재 노드가 사나인지 확인한다.

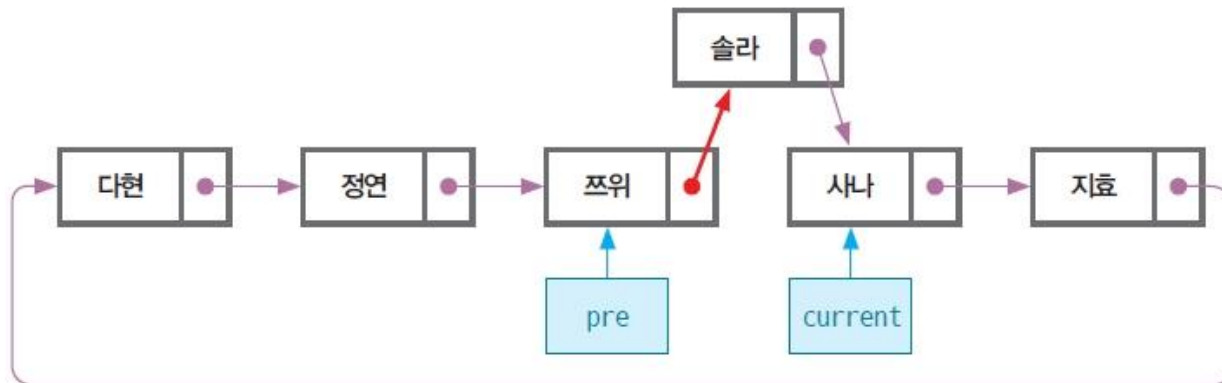


- 3 현재 노드가 사나일 때까지 2 단계를 반복한다.
4 현재 노드가 사나라면 우선 새 노드(솔라 노드)를 생성한 후 이전 노드의 링크를 새 노드의 링크로 지정한다.



Section 03 원형 연결 리스트의 일반 구현

5 이전 노드의 링크를 새 노드로 지정한다.

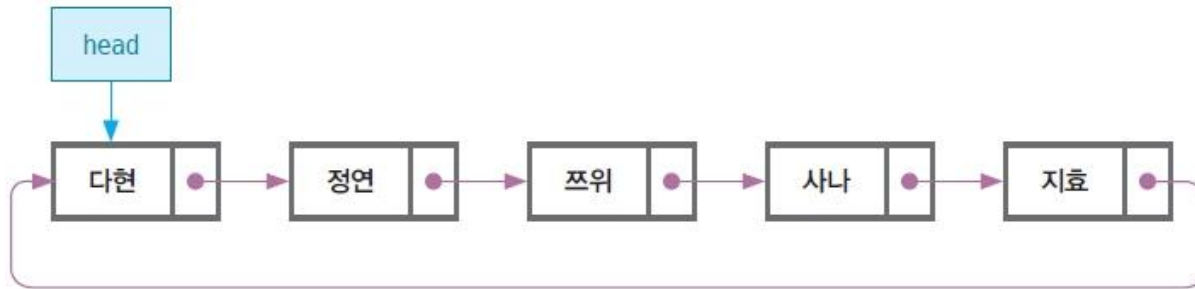


```
1 current = head
  while 마지막 노드까지 :
    pre = current
    2 { current = current.link
    3 { if current.data == "사나" :
      4 { node = Node()
        4 { node.data = "솔라"
          node.link = current
        5 pre.link = node
```

Section 03 원형 연결 리스트의 일반 구현

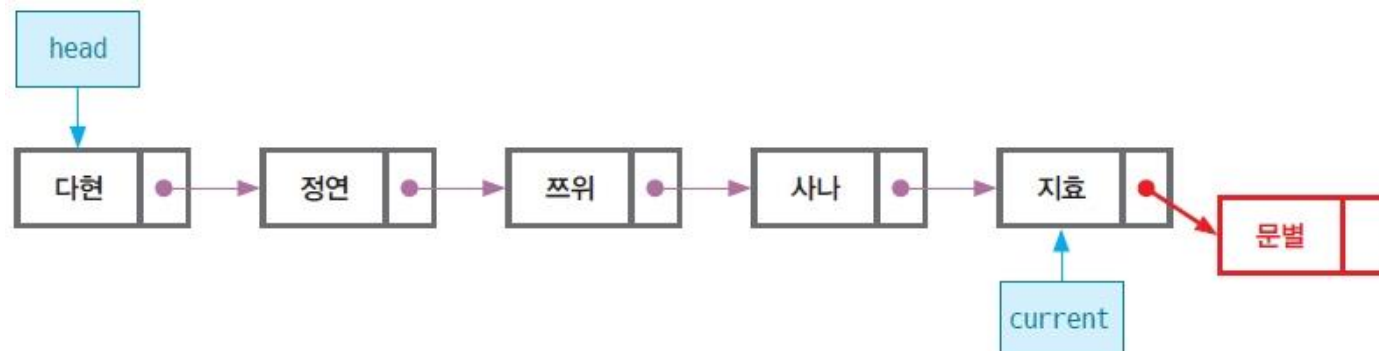
■ 노드 삽입 : 마지막 노드 삽입

0 마지막 노드 삽입 전 초기 상태



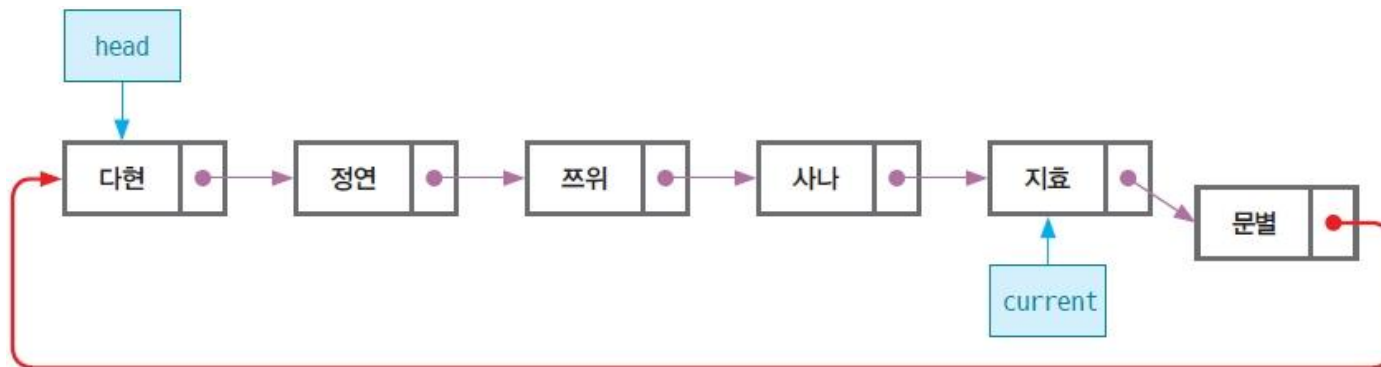
1 ~ 3 중간 노드 삽입 과정과 동일하므로 없는 데이터인 재남을 찾는다.

4 마지막 노드까지 재남을 찾지 못했다면 우선 새 노드(문별 노드)를 생성한 후 현재(current) 노드의 링크를 새 노드로 지정한다.



Section 03 원형 연결 리스트의 일반 구현

5 새 노드의 링크를 헤드가 가리키는 노드로 지정한다.



1~3 # 마지막 노드까지 "재남"을 찾지 못한 후

```
{ node = Node()
4 { node.data = "문별"
  current.link = node
5 node.link = head
```

Section 03 원형 연결 리스트의 일반 구현

- 노드 삽입 함수의 완성
 - 세 가지 경우의 데이터를 입력하는 함수 작성

Code05-05.py 원형 연결 리스트의 노드 삽입 함수

```
1  ## 클래스와 함수 선언 부분 ##
2  class Node() :
3      def __init__ (self) :
4          self.data = None
5          self.link = None
6
7  def printNodes(start) :
8      current = start
9      if current.link == None :
10         return
11     print(current.data, end = ' ')
12     while current.link != start :
13         current = current.link
14         print(current.data, end = ' ')
15     print()
16
17 def insertNode(findData, insertData) :
18     global memory, head, current, pre
19
```

Section 03 원형 연결 리스트의 일반 구현

```
20  if head.data == findData :           # 첫 번째 노드 삽입
21      node = Node()
22      node.data = insertData
23      node.link = head
24      last = head                       # 마지막 노드를 첫 번째 노드로 우선 지정
25      while last.link != head :         # 마지막 노드를 찾으면 반복 종료
26          last = last.link             # last를 다음 노드로 변경
27      last.link = node                  # 마지막 노드의 링크에 새 노드 지정
28      head = node
29      return
30
31  current = head
32  while current.link != head :          # 중간 노드 삽입
33      pre = current
34      current = current.link
35      if current.data == findData :
36          node = Node()
37          node.data = insertData
38          node.link = current
39          pre.link = node
40          return
41
42  node = Node()                          # 마지막 노드 삽입
43  node.data = insertData
44  current.link = node
45  node.link = head
```

Section 03 원형 연결 리스트의 일반 구현

```
46
47 ## 전역 변수 선언 부분 ##
... # 생략(Code05-04.py의 18~20행과 동일)
51
52 ## 메인 코드 부분 ##
53 if __name__ == "__main__" :
54
... # 생략(Code05-04.py의 25~39행과 동일)
70
71     insertNode("다현", "화사")
72     printNodes(head)
73
74     insertNode("사나", "솔라")
75     printNodes(head)
76
77     insertNode("재남", "문별")
78     printNodes(head)
```

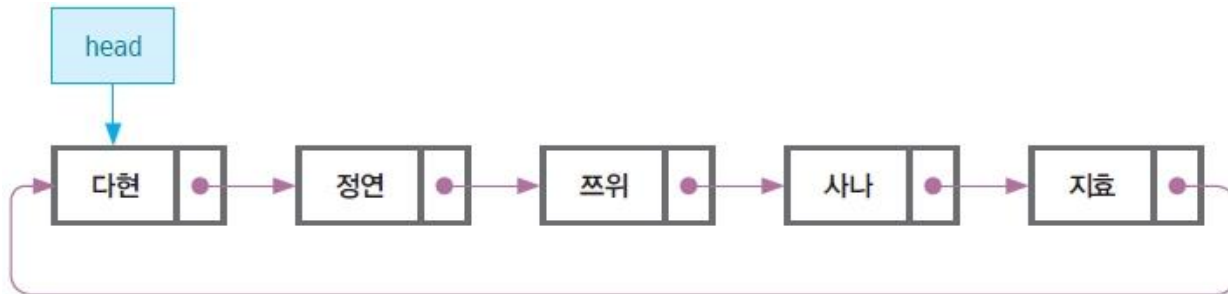
실행 결과

다현 정연 쫘위 사나 지효 → 초기상태
화사 다현 정연 쫘위 사나 지효
화사 다현 정연 쫘위 솔라 사나 지효
화사 다현 정연 쫘위 솔라 사나 지효 문별

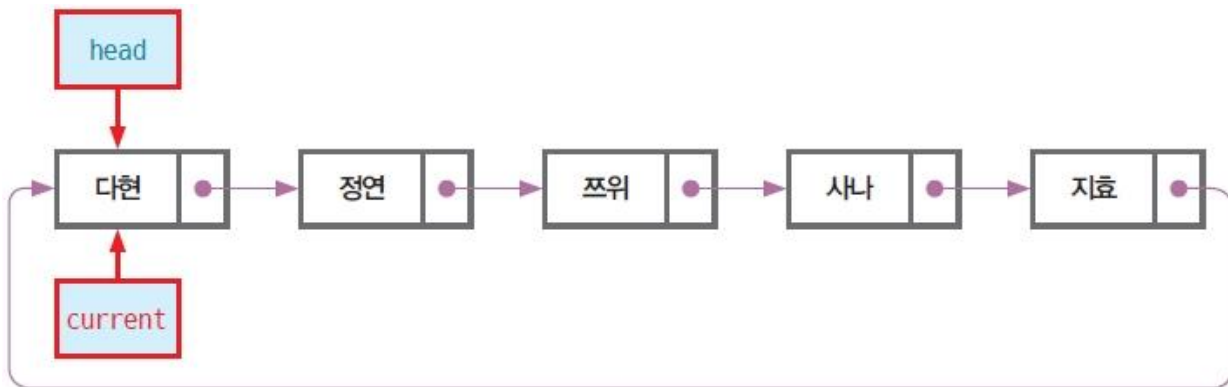
Section 03 원형 연결 리스트의 일반 구현

■ 노드 삭제 : 첫 번째 노드 삭제

0 노드 삭제 전 초기 상태

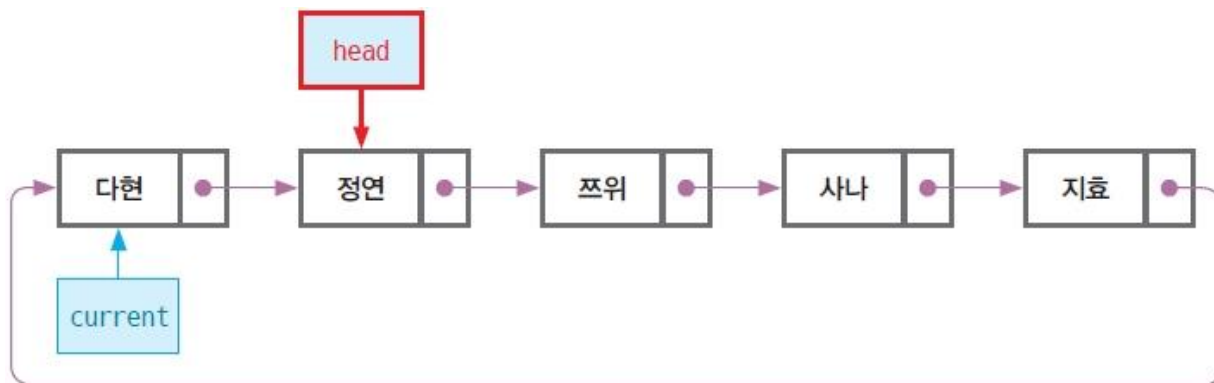


1 현재 노드(current)를 삭제할 노드인 헤드(head)와 동일하게 만든다.

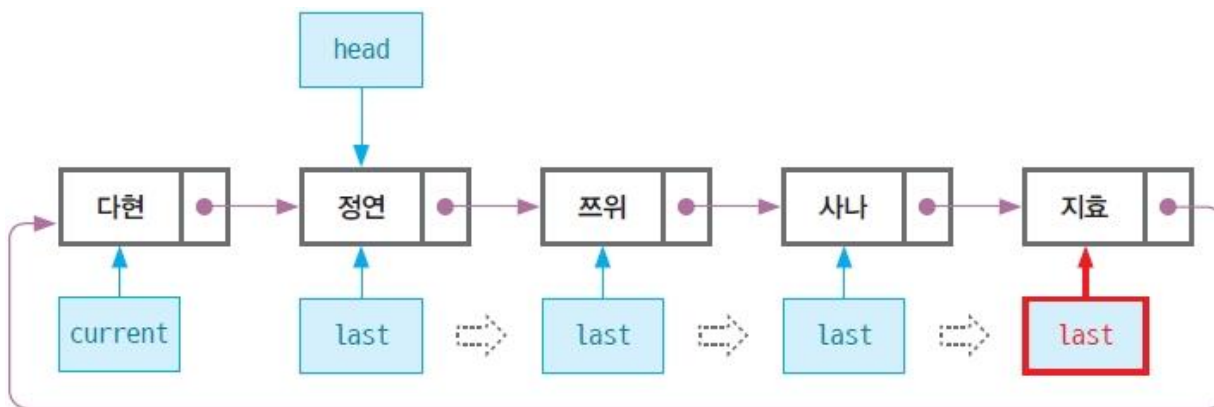


Section 03 원형 연결 리스트의 일반 구현

2 헤드를 삭제할 노드(다현 노드)의 링크가 가리키던 정연 노드로 변경된다.

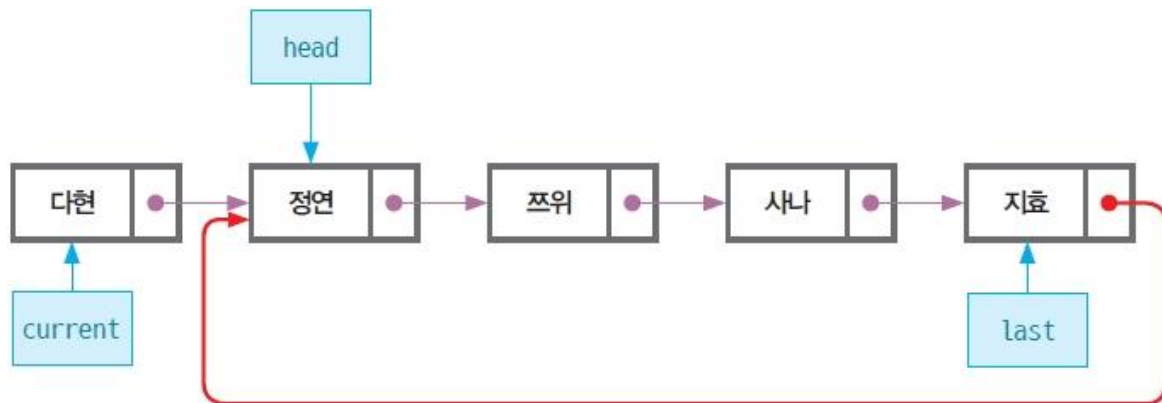


3 헤드에서 시작해서 마지막 노드를 찾는다.

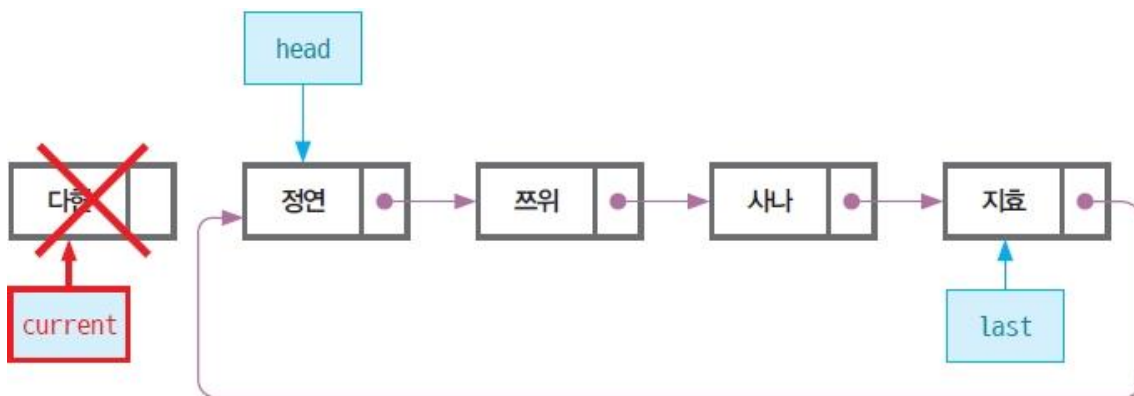


Section 03 원형 연결 리스트의 일반 구현

- 4 마지막 노드의 링크에 헤드 가리키는 노드를 지정한다.



- 5 현재 노드를 메모리에서 제거한다.



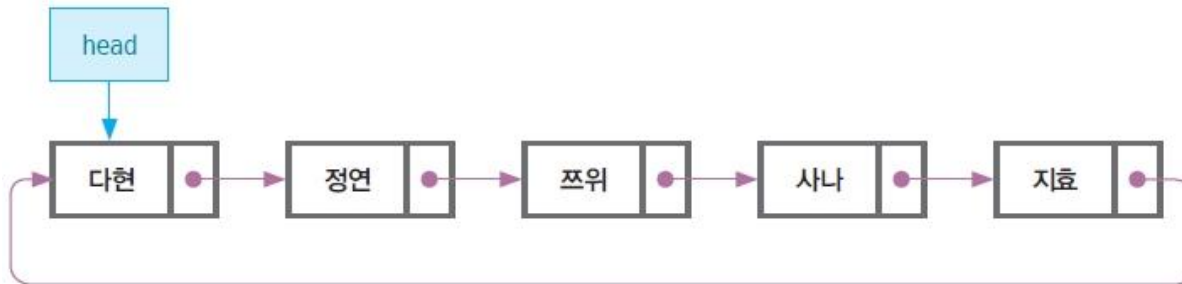
Section 03 원형 연결 리스트의 일반 구현

```
1 current = head
2 { head = head.link
  last = head
3 { while last.link != current : # 마지막 노드를 찾으면 반복 종료
    last = last.link           # last를 다음 노드로 변경
4 last.link = head             # 마지막 노드의 링크에 head가 가리키는 노드 지정
5 del(current)
```

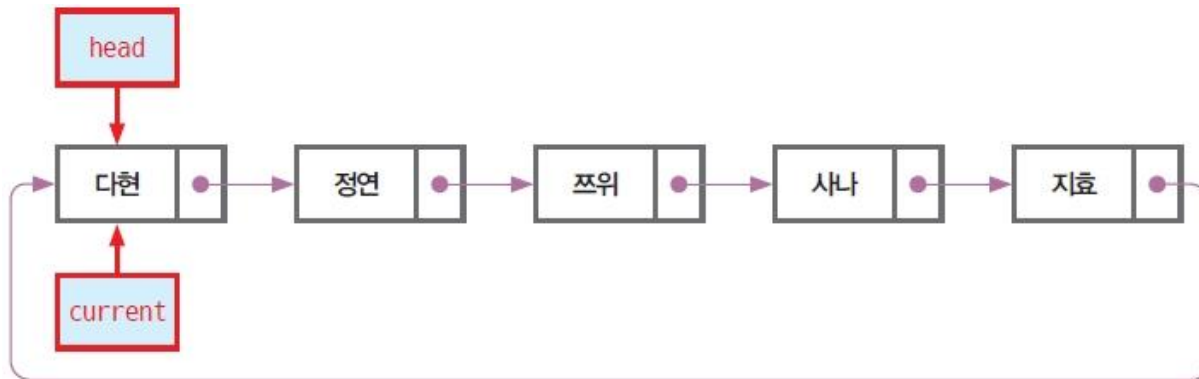
Section 03 원형 연결 리스트의 일반 구현

■ 노드 삭제 : 첫 번째 외 노드 삭제

0 노드 삭제 전 초기 상태

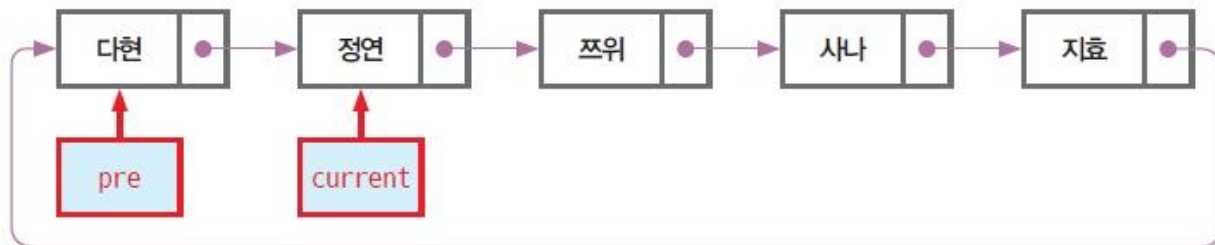


1 헤드(head)에서 시작해서 현재 노드(current)가 썬위인지 확인한다.

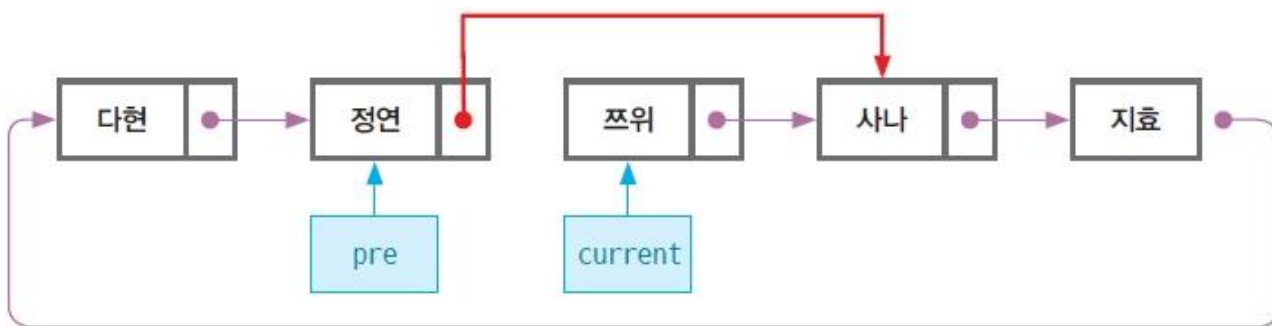


Section 03 원형 연결 리스트의 일반 구현

- 2 현재 노드를 이전 노드(pre)로 저장하고, 현재 노드를 다음 노드로 이동한다.
그리고 현재 노드가 짝수인지 확인한다.

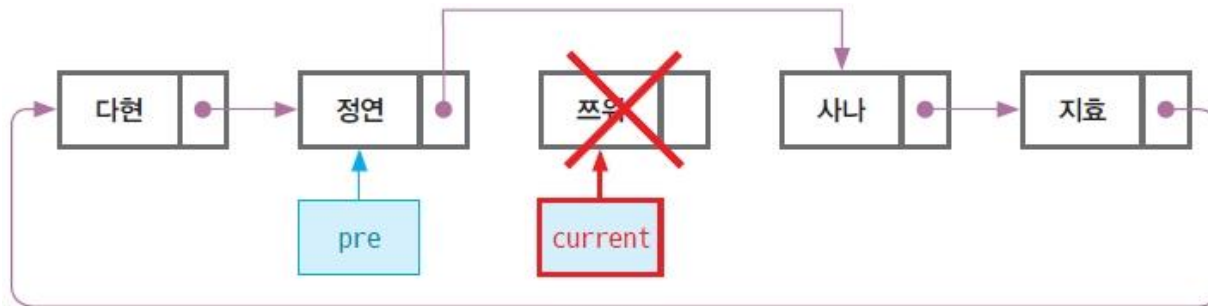


- 3 현재 노드가 짝수일 때까지 2 단계를 반복한다.
4 현재 노드가 짝수라면, 이전 노드의 링크를 현재 노드의 링크로 지정한다.



Section 03 원형 연결 리스트의 일반 구현

5 현재 노드를 메모리에서 삭제한다.



```
1 current = head
  while current.link != head :
    2 { pre = current
      3 { current = current.link
        4 { if current.data == "찰위" :
          5 { pre.link = current.link
            del(current)
```

Section 03 원형 연결 리스트의 일반 구현

- 노드 삭제 함수의 완성
 - 두 가지 경우의 데이터를 삭제하는 함수 작성

Code05-06.py 원형 연결 리스트의 노드 삭제 함수

```
1  ## 클래스와 함수 선언 부분 ##
2  class Node() :
3      def __init__ (self) :
4          self.data = None
5          self.link = None
6
7  def printNodes(start) :
8      current = start
9      if current.link == None :
10         return
11     print(current.data, end = ' ')
12     while current.link != start :
13         current = current.link
14         print(current.data, end = ' ')
15     print()
16
17 def deleteNode(deleteData) :
18     global memory, head, current, pre
19
```

Section 03 원형 연결 리스트의 일반 구현

```
20     if head.data == deleteData :           # 첫 번째 노드 삭제
21         current = head
22         head = head.link
23         last = head
24         while last.link != current :       # 마지막 노드를 찾으면 반복 종료
25             last = last.link               # last를 다음 노드로 변경
26             last.link = head               # 마지막 노드의 링크에 head가 가리키는 노드 지정
27         del(current)
28         return
29
30     current = head                           # 첫 번째 외 노드 삭제
31     while current.link != head :
32         pre = current
33         current = current.link
34         if current.data == deleteData :    # 중간 노드를 찾았을 때
35             pre.link = current.link
36             del(current)
37         return
38
39 ## 전역 변수 선언 부분 ##
... # 생략(Code05-04.py의 18~20행과 동일)
43
44 ## 메인 코드 부분 ##
45 if __name__ == "__main__" :
46
... # 생략(Code05-04.py의 25~39행과 동일)
```


Section 03 원형 연결 리스트의 일반 구현

```
62
63     deleteNode("다현")
64     printNodes(head)
65
66     deleteNode("쯔위")
67     printNodes(head)
68
69     deleteNode("지효")
70     printNodes(head)
71
72     deleteNode("재남")
73     printNodes(head)
```

실행 결과

다현 정연 쑤위 사나 지효 → 초기상태

정연 쑤위 사나 지효

정연 사나 지효

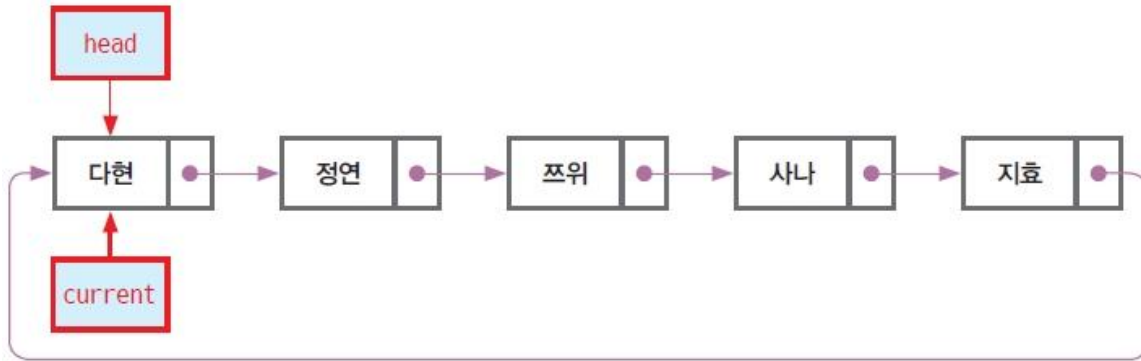
정연 사나

정연 사나

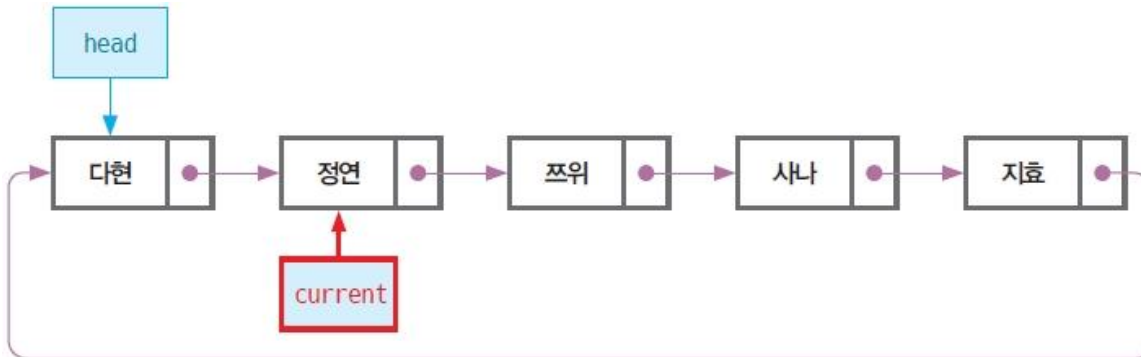
Section 03 원형 연결 리스트의 일반 구현

■ 노드 검색

- 1 현재 노드(current)를 첫 번째 노드인 헤드(head)와 동일하게 만들고, 현재 노드가 검색할 데이터인지 비교한다. 검색할 데이터와 동일하다면 현재 노드를 반환한다.



- 2 현재 노드를 다음 노드로 이동하고, 검색할 데이터와 동일하다면 현재 노드를 반환한다.



Section 03 원형 연결 리스트의 일반 구현

3 앞의 2 단계를 끝까지 진행하고, 검색할 데이터를 찾지 못했다면 None을 반환한다.

Code05-07.py 원형 연결 리스트의 노드 검색 함수

```
1  ## 클래스와 함수 선언 부분 ##
2  class Node() :
3      def __init__ (self) :
4          self.data = None
5          self.link = None
6
7  def printNodes(start) :
8      current = start
9      if current.link == None :
10         return
11     print(current.data, end = ' ')
12     while current.link != start :
13         current = current.link
14         print(current.data, end = ' ')
15     print()
16
17 def findNode(findData) :
18     global memory, head, current, pre
19
```

Section 03 원형 연결 리스트의 일반 구현

```
20     current = head
21     if current.data == findData :
22         return current
23     while current.link != head :
24         current = current.link
25         if current.data == findData :
26             return current
27     return Node()    # 빈 노드 반환
28
29 ## 전역 변수 선언 부분 ##
... # 생략(Code05-04.py의 18~20행과 동일)
33
34 ## 메인 코드 부분 ##
35 if __name__ == "__main__" :
36
... # 생략(Code05-04.py의 25~39행과 동일)
52
53     fNode = findNode("다현")
54     print(fNode.data)
55
56     fNode = findNode("쯔위")
57     print(fNode.data)
58
59     fNode = findNode("재남")
60     print(fNode.data)
```

실행 결과

다현 정연 쑤위 사나 지효 → 초기상태

다현

쑤위

None

Section 03 원형 연결 리스트의 일반 구현

SELF STUDY 5-1

Code05-07.py를 수정해서 데이터를 찾은 경우와 찾지 못한 경우를 출력하도록 한다. 예로 첫 번째 노드가 찾는 데이터이면 “첫 노드에서 찾음”이, 중간 노드가 찾는 데이터이면 “중간 노드에서 찾음”이, 찾는 노드가 없으면 “찾는 노드가 없음”이 출력되도록 한다.

실행 결과

```
다현 정연 쑤위 사나 지효
# 첫 노드에서 찾음 #
다현
# 중간 노드에서 찾음 #
쑤위
# 찾는 노드가 없음 #
None
```

Section 04 원형 연결 리스트의 응용

■ 원형 연결 리스트를 응용하는 프로그램 작성 예

- 1~100 숫자 중 7개 숫자 랜덤으로 뽑은 후 순서대로 원형 연결 리스트 구성
- 홀수, 짝수 중 더 많은 것을 모두 음수로 변경

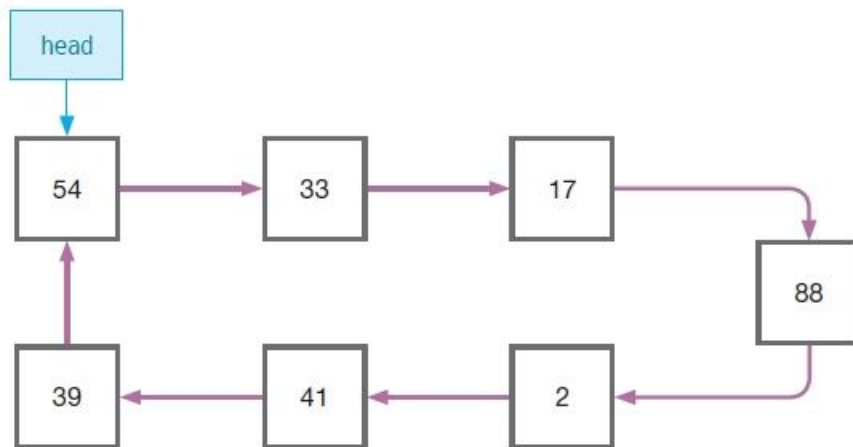


그림 5-16 랜덤 숫자 7개로 구성된 원형 연결 리스트 초기 구성

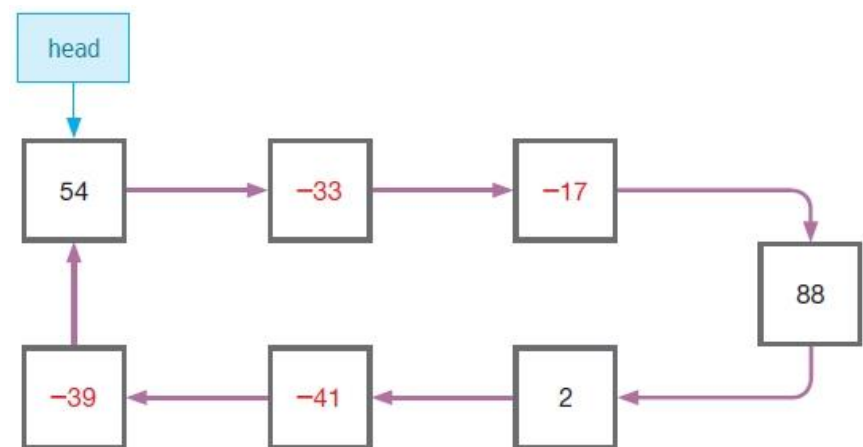


그림 5-17 랜덤 숫자 7개로 구성된 원형 연결 리스트 결과 구성

Section 04 원형 연결 리스트의 응용

- 1 입력할 데이터를 랜덤하게 7개 발생시켜 dataArray 배열에 저장한다.

```
dataArray = []  
for _ in range(7) :  
    dataArray.append(random.randint(1, 100))
```

- 2 데이터를 차례대로 가져와 원형 연결 리스트를 만든다.

```
node = Node()  
node.data = dataArray[0]  
head = node  
node.link = head  
memory.append(node)  
  
for data in dataArray[1:] :  
    pre = node  
    node = Node()  
    node.data = data  
    pre.link = node  
    node.link = head  
    memory.append(node)
```

Section 04 원형 연결 리스트의 응용

- ③ 원형 연결 리스트 전체를 1회 방문하면서 홀수와 짝수의 개수를 센다.

```
current = head
while True :
    if current.data가 짝수면 :
        짝수개수 += 1
    else :
        홀수개수 += 1
    if current.link == head :
        break;
    current = current.link
```

- ④ 짝수 또는 홀수의 개수 중 많은 쪽 숫자를 음수로 만든다.

```
if 홀수개수 > 짝수개수 :
    나머지값 = 1
else :
    나머지값 = 0

current = head
while True :
    if current.data % 2 == 나머지값 :
        current.data *= -1
    if current.link == head :
        break;
    current = current.link
```


Section 04 원형 연결 리스트의 응용

Code05-08.py 원형 연결 리스트를 활용한 홀수와 짝수 구분 프로그램

```
1  import random
2
3  ## 클래스와 함수 선언 부분 ##
4  class Node() :
5      def __init__ (self) :
6          self.data = None
7          self.link = None
8
9  def printNodes(start) :
10     current = start
11     if current.link == None :
12         return
13     print(current.data, end = ' ')
14     while current.link != start :
15         current = current.link
16         print(current.data, end = ' ')
17     print()
18
19 def countOddEven() :
20     global memory, head, current, pre
21
22     odd, even = 0, 0
23     if head == None :
24         return False
25
```

Section 04 원형 연결 리스트의 응용

```
26     current = head
27     while True :
28         if current.data % 2 == 0 :
29             even += 1
30         else :
31             odd += 1
32         if current.link == head :
33             break;
34         current = current.link
35
36     return odd, even
37
38 def makeZeroNumber(odd, even) :
39     if odd > even :
40         reminder = 1
41     else :
42         reminder = 0
43
44     current = head
45     while True :
46         if current.data % 2 == reminder :
47             current.data *= -1
48         if current.link == head :
49             break;
50         current = current.link
51
```

Section 04 원형 연결 리스트의 응용

```
52 ## 전역 변수 선언 부분 ##
53 memory = []
54 head, current, pre = None, None, None
55
56 ## 메인 코드 부분 ##
57 if __name__ == "__main__":
58
59     dataArray = []
60     for _ in range(7):
61         dataArray.append(random.randint(1, 100))
62
63     ... # 생략(Code05-04.py의 25~39행과 동일)
78
79     oddCount, evenCount = countOddEven()
80     print('홀수 -->', oddCount, '\t', '짝수 -->', evenCount)
81
82     makeZeroNumber(oddCount, evenCount)
83     printNodes(head)
```

실행 결과

83 53 48 26 23 99 21 → 초기상태
홀수 --> 5 짝수 --> 2
-83 -53 48 26 -23 -99 -21

Section 04 원형 연결 리스트의 응용

SELF STUDY 5-2

Code05-08.py를 수정해서 랜덤하게 -100~100 숫자 중 7개를 뽑고(중복 허용), 이번에는 양수와 음수의 개수를 센다. 그리고 양수는 음수로, 음수는 양수로 변경한다. 0은 양수도 음수도 아닌 것으로 간주한다.

실행 결과

-54 76 -55 94 14 43 -42

양수 → 4 음수 → 3

54 -76 55 -94 -14 -43 42

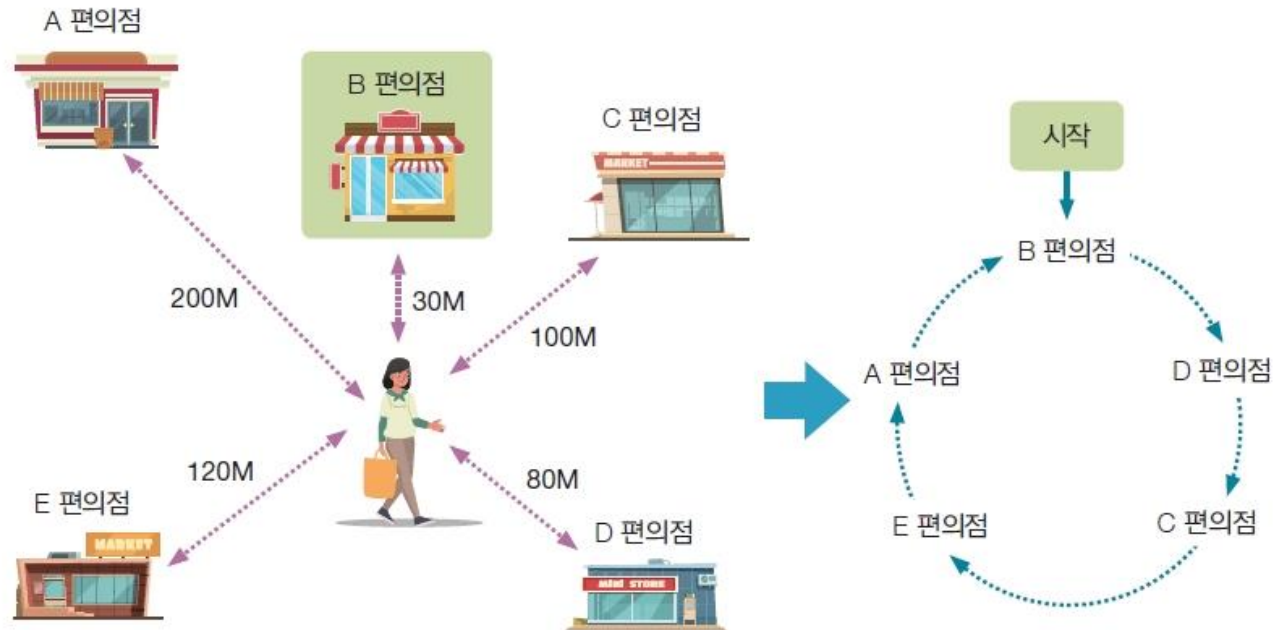
응용예제 01 현재 위치부터 가까운 편의점 관리하기

난이도 ★★★☆☆

예제 설명

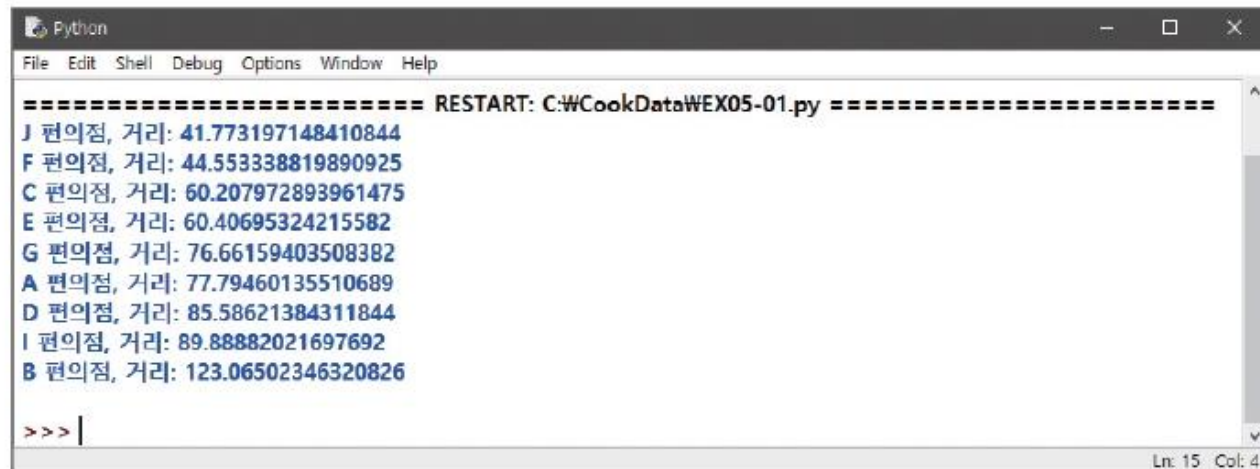
현재 위치를 (0, 0)이라 가정하고, 편의점 위치(x, y)와 거리가 가까운 순서대로 원형 연결 리스트를 생성하는 프로그램을 다음 조건에 맞게 작성한다.

- 편의점 10개를 A, B, C, ... 순서로 이름을 부여한다.
- 편의점 위치 x 와 y 는 1부터 100까지 랜덤하게 좌표가 생성되도록 한다.
- 현재 위치와 편의점 거리는 $(x^2 + y^2)$ 의 제곱근(sqrt)으로 계산한다.
- 편의점 데이터 1개는 (편의점이름, x 좌표, y 좌표) 형식의 튜플로 구성한다.



응용예제 01 현재 위치부터 가까운 편의점 관리하기

실행 결과



A screenshot of a Python Shell window titled 'Python'. The menu bar includes 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area displays the output of a script, starting with a restart message: '==== RESTART: C:\CookData\WEX05-01.py ====='. Below this, a list of convenience stores and their distances is shown, with the store names in blue and distances in black. The list is ordered from top to bottom: J, F, C, E, G, A, D, I, and B. At the bottom of the text area, there is a prompt '>>>' followed by a vertical bar. The status bar at the bottom right indicates 'Ln: 15 Col: 4'.

```
==== RESTART: C:\CookData\WEX05-01.py =====
J 편의점, 거리: 41.773197148410844
F 편의점, 거리: 44.553338819890925
C 편의점, 거리: 60.207972893961475
E 편의점, 거리: 60.40695324215582
G 편의점, 거리: 76.66159403508382
A 편의점, 거리: 77.79460135510689
D 편의점, 거리: 85.58621384311844
I 편의점, 거리: 89.88882021697692
B 편의점, 거리: 123.06502346320826

>>> |
```

응용예제 02 이중 연결 리스트 구현하기

난이도 ★ ★ ☆ ☆ ☆

예제 설명

다음과 같이 양방향으로 링크가 연결되는 이중 연결 리스트를 만든다. 헤드부터 차례대로 출력한 후 이어서 마지막 노드부터 거꾸로 다시 출력해 보자.



실행 결과

```
Python
File Edit Shell Debug Options Window Help
===== RESTART: C:\CookData\WE05-02.py =====
정방향 --> 다현 정연 쑤위 사나 지효
역방향 --> 지효 사나 쑤위 정연 다현
>>> |
```