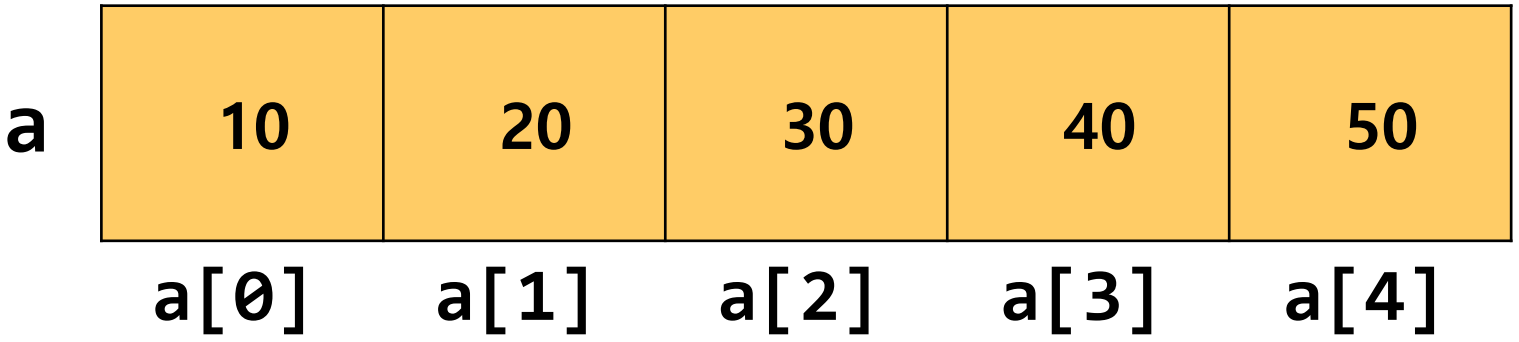
Data Structure
(List, Tuple, Dictionary, String)

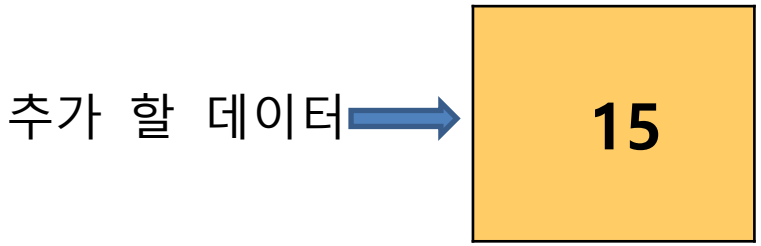
Contents

- List
- Tuple
- Dictionary
- String

문제 해결 방법 생각하기

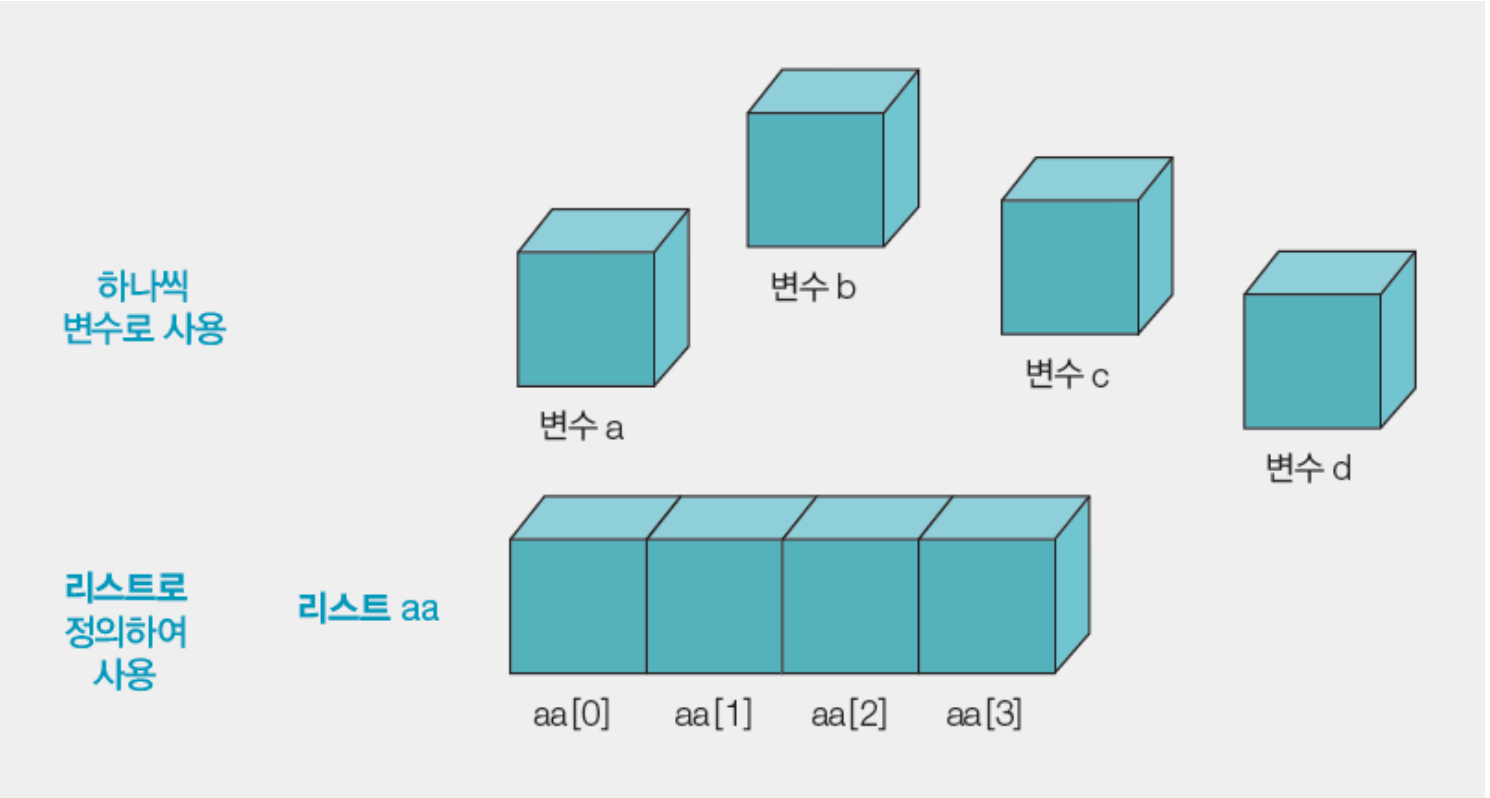
- 데이터를 묶어서 배열로 선언했는데 데이터의 삽입 삭제가 자주 일어나는 경우에 해결 방법은?





리스트(List) : Python에서는 C 언어의 배열, 포인터, 구조체 등을 활용하여 만든 리스트 데이터 구조를 기본으로 제공함

- 리스트의 이해



- 리스트는 박스(변수)를 한 줄로 붙인 뒤에 박스 전체의 이름(aa)을 지정.
- 각각은 aa[0], aa[1], aa[2], aa[3]과 같이 번호(첨자)를 붙여서 사용.

리스트(List)

- 리스트 생성 방법

```
리스트이름=[값1, 값2, 값3, …]
```

```
aa=[ 10, 20, 30, 40 ]
```

- 리스트를 사용하지 않는다면 각각의 변수를 a, b, c, d와 같이 선언
- 하지만 리스트를 사용하면 첨자를 넣어 aa[0], aa[1], aa[2], aa[3]과 같이 선언
- 항목이 4개인 리스트를 생성한다면 첨자는 1~4가 아닌 0~3을 사용함

① 각 변수 사용	② 리스트 사용
a, b, c, d = 10, 20, 30, 40	aa = [10, 20, 30, 40]
a 사용	aa[0] 사용
b 사용	aa[1] 사용
c 사용	aa[2] 사용
d 사용	aa[3] 사용

리스트(List)

- 리스트의 일반적인 사용법
  - 빈 리스트와 리스트의 추가
    - 비어있는 리스트를 만들고 '리스트이름.append(값)' 함수로 리스트에 하나씩 추가

```
aa=[]
aa.append(0)
aa.append(0)
aa.append(0)
aa.append(0)
print(aa)
```

```
[0, 0, 0, 0]
```

리스트(List)

- 100개의 리스트를 만들 경우 append()와 함께 for문을 활용

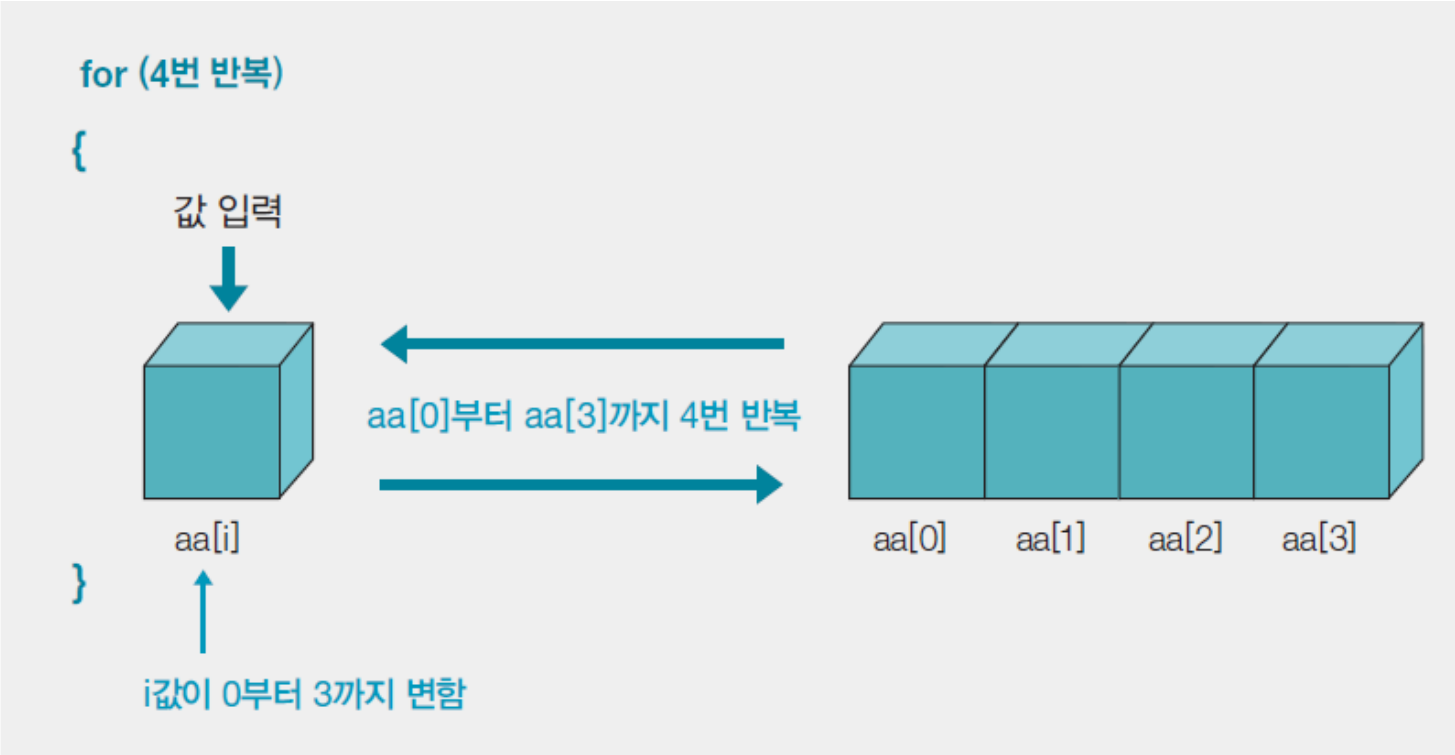
```
aa=[]
for i in range(0, 100) :
    aa.append(0)

len(aa)
```

```
100
```

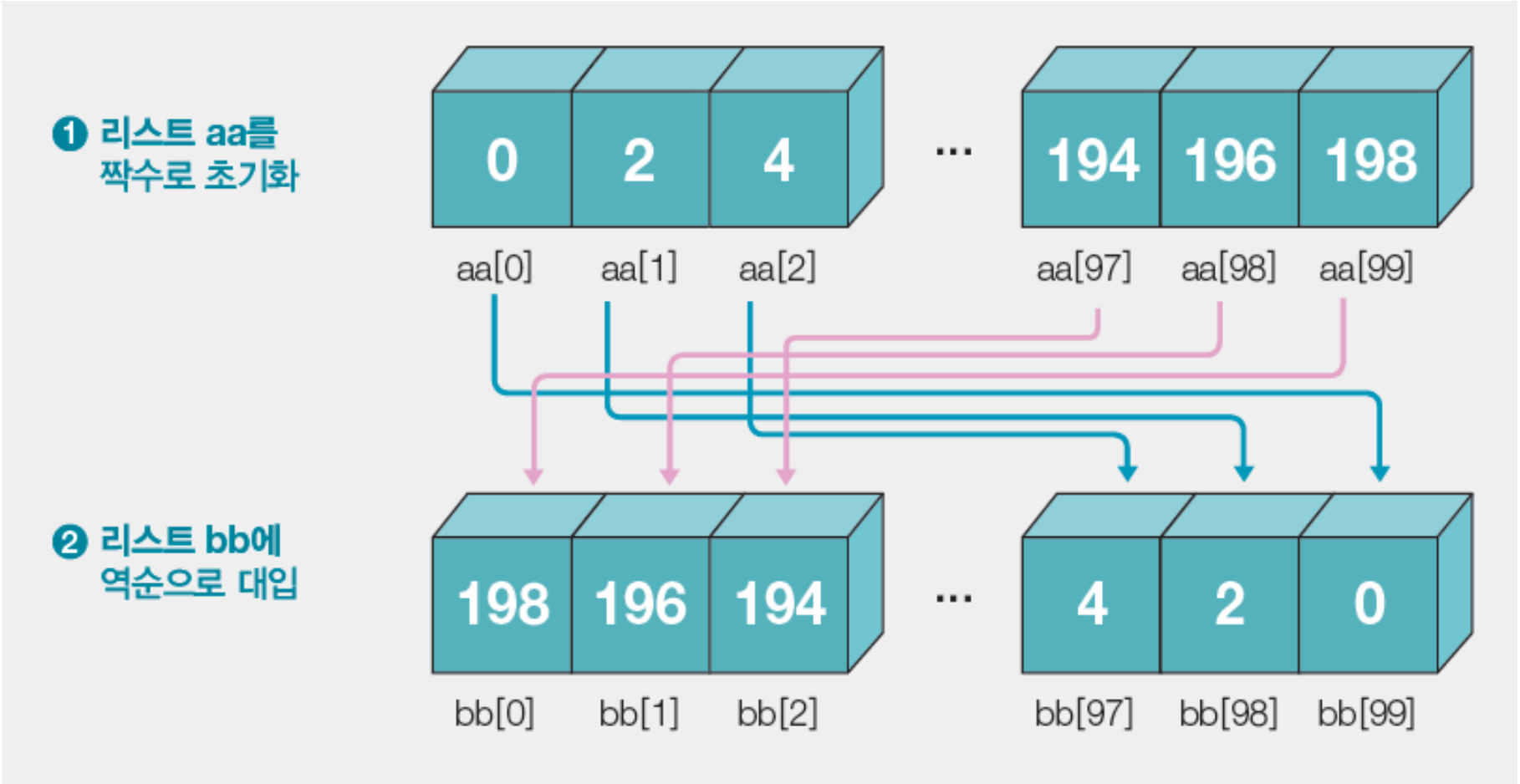
- for문으로 100번(0부터 99까지)을 반복해서 리스트이름.append(0)로 100개 크기의 리스트를 만듦
- len 함수로 리스트의 개수를 확인

리스트(List)

- For문 활용하여 리스트 값 입력



리스트(List) 초기화 및 역순 대입



리스트(List) 초기화 및 역순 대입 예제

```
1  aa=[]
2  bb=[]
3  value=0
4
5  for i in range(0, 100) :
6      aa.append(value)
7      value+=2
8
9  for i in range(0, 100) :
10     bb.append(aa[99-i])
11
12 print(" bb[0]은 %d, bb[99]는 %d 입력됨 " % (bb[0], bb[99] ))
```

```
bb[0]은 198, bb[99]는 0 입력됨
```

리스트(List)

- 여러 개의 리스트 값을 사용하기

```
aa=[10, 20, 30, 40]
print("aa[-1]은 %d, aa[-2]는 %d" % (aa[-1], aa[-2]))
```

```
aa[-1]은 40, aa[-2]는 30
```

리스트(List)-슬라이싱(Slicing)

- 슬라이싱(Slicing) : 리스트 원소를 원하는 시작과 끝을 지정하여 원소 추출
  - 리스트이름[시작 : 끝]로 지정하면 리스트의 모든 값이 나옴

```
aa=[10, 20, 30, 40]
aa[0:3]
aa[2:4]
```

```
[10, 20, 30]
[30, 40]
```

리스트(List)-슬라이싱(Slicing)

- 콜론의 앞이나 뒤 숫자의 생략도 가능

```
aa = [10, 20, 30, 40]
aa[2:]
aa[:2]
```

```
[30, 40]
[10, 20]
```

- 리스트의 더하기, 곱하기 연산

```
aa = [10, 20, 30]
bb = [40, 50, 60]
aa + bb
aa * 3
```

```
[10, 20, 30, 40, 50, 60]
[10, 20, 30, 10, 20, 30, 10, 20, 30]
```

리스트 항목 삭제

- 여러 개의 항목을 삭제하려면 'aa[시작:끝+1]=[ ]' 문장으로 설정

```
aa=[10, 20, 30, 40, 50]
aa[1:4]=[]
aa
```

```
[10, 50]
```

리스트 지원 함수

함수	설명	사용법
append( )	리스트 제일 뒤에 항목을 추가한다.	리스트이름.append(값)
pop( )	리스트 제일 뒤의 항목을 빼내고, 빼낸 항목은 삭제한다.	리스트이름.pop( )
sort( )	리스트의 항목을 정렬한다.	리스트이름.sort( )
reverse( )	리스트 항목의 순서를 역순으로 만든다.	리스트이름.reverse( )
index( )	지정한 값을 찾아서 그 위치를 반환한다.	리스트이름.index(찾을 값)
insert( )	지정된 위치에 값을 삽입한다.	리스트이름.insert(위치, 값)
remove( )	리스트에서 지정한 값을 제거한다. 단 지정한 값이 여러 개일 경우 첫 번째 값만 지운다.	리스트이름.remove(지울 값)
extend( )	리스트 뒤에 리스트를 추가한다. 리스트의 더하기(+) 연산과 동일한 기능을 한다.	리스트이름.extend(리스트)
count( )	리스트에서 찾을 값의 개수를 센다.	리스트이름.count(찾을 값)
del( )	리스트에서 해당 위치의 항목을 삭제한다.	del(리스트이름[위치])
len( )	리스트에 포함된 전체 항목의 개수를 센다.	len(리스트이름)

리스트 지원 함수 예제

```
1  myList=[30, 10, 20]
2  print("현재 리스트 : %s" % myList)
3
4  myList.append(40)
5  print("append(40) 후의 리스트 : %s" % myList)
6
7  print("pop() 으로 추출한 값 : %s" % myList.pop())
8  print("pop() 후의 리스트 : %s" % myList)
9
10 myList.sort()
11 print("sort() 후의 리스트 : %s" % myList)
12
13 myList.reverse()
14 print("reverse() 후의 리스트 : %s" % myList)
15
16 print("20 값의 위치 : %d" % myList.index(20))
17
18 myList.insert(2, 222)
19 print("insert(2, 222) 후의 리스트 : %s" % myList)
```

리스트 지원 함수 예제

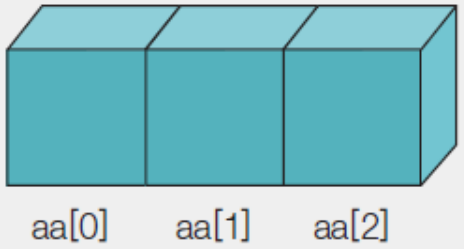
```
20
21  myList.remove(222)
22  print("remove(222) 후의 리스트 : %s" % myList)
23
24  myList.extend( [77 , 88, 77] )
25  print("extend([77, 88, 77]) 후의 리스트 : %s" % myList)
26
27  print("77 값의 개수 : %d" % myList.count(77))
```

```
현재 리스트 : [30, 10, 20]
append(40) 후의  리스트 : [30, 10, 20, 40]
pop() 으로 추출한 값 : 40
pop() 후의  리스트 : [30, 10, 20]
sort() 후의 리스트 : [10, 20, 30]
reverse() 후의 리스트 : [30, 20, 10]
20 값의 위치 : 1
insert(2, 222) 후의 리스트 : [30, 20, 222, 10]
remove(222) 후의 리스트 : [30, 20, 10]
extend([77 , 88]) 후의 리스트 : [30, 20, 10, 77, 88, 77]
77 값의 개수 : 2
```

2차원 리스트

- 2차원 리스트의 기본 개념
  - 1차원 리스트를 여러 개 연결한 것, 두 개의 첨자를 사용하는 리스트

```
aa = [10, 20, 30]
```



1차원 리스트

```
aa = [ [ 1,  2,  3,  4] ,
       [5,  6,  7,  8] ,
       [9, 10,  11, 12] ]
```

2차원 리스트

2차원 리스트

- 첨자가 두 개이므로 중첩 for문을 사용해서 3행 4열 짜리 리스트에 1~12의 숫자를 채움

```
1  list1=[]
2  list2=[]
3  value=1
4  for i in range(0, 3) :
5      for k in range(0, 4) :
6          list1.append(value)
7          value+=1
8      list2.append(list1)
9      list1=[]
10
11 for i in range(0, 3) :
12     for k in range(0, 4) :
13         print("%3d" % list2[i][k], end=" ")
14     print("")
```

```
 1  2  3  4
 5  6  7  8
 9 10 11 12
```

리스트와 튜플(Tuple)

- 튜플의 생성
  - 리스트는 대괄호([ ])로 생성하고 튜플은 괄호( )로 생성.
  - 튜플은 값을 수정할 수 없으며 읽기만 가능하므로 읽기 전용의 자료를 저장할 때 사용

```
tt1=(10, 20, 30)
tt1
tt2=10, 20, 30
tt2
```

```
(10, 20, 30)
(10, 20, 30)
```

튜플(Tuple)

- 튜플은 읽기 전용이므로 다음 코드는 모두 오류 발생.

```
tt1.append(40)
tt1[0]=40
del(tt1[0])
```

- 하지만 튜플 자체는 del( ) 함수로 지울 수 있음

```
del(tt1)
del(tt2)
```

딕셔너리(Dictionary)

- **딕셔너리(Dictionary)는 키-값 쌍으로 구성된 가변적인 데이터 구조**
- **중괄호({})를 사용하여 표현되며, 각 키와 값은 콜론(:)으로 구분**
- **딕셔너리의 주요 특징**
  - 키-값 쌍: 딕셔너리는 각 요소가 키(key)와 값(value)으로 이루어진 키-값 쌍으로 구성
  - 키는 고유하고 변경되지 않는 값으로 사용되며, 값은 키와 연결된 데이터
  - 가변성: 딕셔너리는 가변적인(mutable) 자료형이므로 내부의 요소를 추가, 삭제, 수정 가능
  - 순서 없음: 딕셔너리는 요소들의 순서가 보장되지 않음
  - 순서가 중요한 경우 파이썬 3.7 버전 이상에서는 OrderedDict 클래스를 사용할 수 있음
  - 유연한 구조: 딕셔너리는 다양한 데이터 타입의 키와 값으로 구성
  - 딕셔너리는 데이터를 효율적으로 저장하고 검색하기 위해 사용
  - 예를 들어, 사전에서 단어와 해당하는 뜻을 연결하는 것처럼 딕셔너리에서는 키를 사용하여 값에 접근

딕셔너리(Dictionary) 예제 : 학생 관리

```
students = []
def add_student():
    name = input("학생 이름을 입력하세요: ")
    age = int(input("학생 나이를 입력하세요: "))
    major = input("학생 전공을 입력하세요: ")
    student = {
        'name': name,
        'age': age,
        'major': major    }

    students.append(student)
    print("학생이 추가되었습니다.")

def remove_student():
    name = input("삭제할 학생 이름을 입력하세요: ")
    removed = False
    for student in students:
        if student['name'] == name:
            students.remove(student)
            removed = True
            print("학생이 삭제되었습니다.")
            break
    if not removed:
        print("해당하는 학생을 찾을 수 없습니다.")
```

딕셔너리(Dictionary) 예제 : 학생 관리

```
def print_students():
    print("==== 등록된 학생 목록 ====")
    for student in students:
        print(f"이름: {student['name']}, 나이: {student['age']}, 전공: {student['major']}")
    print("==========================")
while True:
    print("===== 학생 관리 시스템 =====")
    print("1. 학생 추가")
    print("2. 학생 삭제")
    print("3. 학생 목록 출력")
    print("0. 종료")
    choice = input("원하는 작업을 선택하세요: ")

    if choice == '1':
        add_student()
    elif choice == '2':
        remove_student()
    elif choice == '3':
        print_students()
    elif choice == '0':
        break
    else:
        print("잘못된 입력입니다. 다시 선택해주세요.")
```

딕셔너리(Dictionary) 예제 : 학생 관리

```
===== 학생 관리 시스템 =====
1. 학생 추가
2. 학생 삭제
3. 학생 목록 출력
0. 종료
원하는 작업을 선택하세요: 1
학생 이름을 입력하세요: 홍길동
학생 나이를 입력하세요: 20
학생 전공을 입력하세요: 컴퓨터공학
학생이 추가되었습니다.
===== 학생 관리 시스템 =====
1. 학생 추가
2. 학생 삭제
3. 학생 목록 출력
0. 종료
원하는 작업을 선택하세요: 1
학생 이름을 입력하세요: 김아라
학생 나이를 입력하세요: 21
학생 전공을 입력하세요: 경영학
학생이 추가되었습니다.
```

```
===== 학생 관리 시스템 =====
1. 학생 추가
2. 학생 삭제
3. 학생 목록 출력
0. 종료
원하는 작업을 선택하세요: 3
==== 등록된 학생 목록 ====
이름: 홍길동, 나이: 20, 전공: 컴퓨터공학
이름: 김아라, 나이: 21, 전공: 경영학
==========================
===== 학생 관리 시스템 =====
1. 학생 추가
2. 학생 삭제
3. 학생 목록 출력
0. 종료
원하는 작업을 선택하세요:0
```

문자열(String)

- **문자열은 문자들의 시퀀스로 이루어진 데이터 타입**
- **문자열의 주요 특징과 기능**
  - 인덱싱 (Indexing): 인덱스는 0부터 시작하며, 문자열[index] 형식으로 특정 위치의 문자에 접근할 수 있음
  - 슬라이싱 (Slicing): 문자열은 슬라이싱을 통해 부분 문자열을 추출
  - 연결 (Concatenation): + 연산자를 사용하여 문자열을 연결
  - 반복 (Repetition): * 연산자를 사용하여 문자열을 반복
  - 길이 (Length): len() 함수를 사용하여 문자열의 길이를 반환

문자열(String) 예제 : 대소문자 변환 프로그램

```
def convert_case(text):
    converted = ""
    for char in text:
        if char.islower():
            converted += char.upper()
        elif char.isupper():
            converted += char.lower()
        else:
            converted += char
    return converted

user_input = input("문자열을 입력하세요: ")
result = convert_case(user_input)
print("변환된 문자열:", result)
```

```
문자열을 입력하세요: Hello World
변환된 문자열: hELLO wORLD
```