Lab6. String, Tuple and Dictionary

CSED101 LAB

2차원 리스트

2차원 리스트

- 1차원 리스트를 원소로 가지는 리스트

```
s1 = [202301, 89, 79]
s2 = [202307, 76, 84]
s3 = [202311, 78, 34]
s4 = [202314, 67, 42]
slist = [s1, s2, s3, s4]

print(slist[1])    ## ??
print(slist[1][2]) ## ??

slist[1][2] = 99
print(slist)       ## ??
```

학번	중간고사	기말고사
202301	89	79
202307	76	84
202311	78	34
202314	67	42

실습 – 2차원 리스트

- 정사각 행렬의 왼쪽에서 오른쪽 대각영역을 0으로 채우고 왼쪽 아래 삼각형은 -1, 오른쪽 위의 삼각형은 1로 채우는 프로그램을 작성하시오. 5 x 5 행렬이라면 프로그램 출력은 아래와 같다.

```
Enter the size of square: 5
 0  1  1  1  1
-1  0  1  1  1
-1 -1  0  1  1
-1 -1 -1  0  1
-1 -1 -1 -1  0
```

00	01	02	03	04
10	11	12	13	14
20	21	22	23	24
30	31	32	33	34
40	41	42	43	44

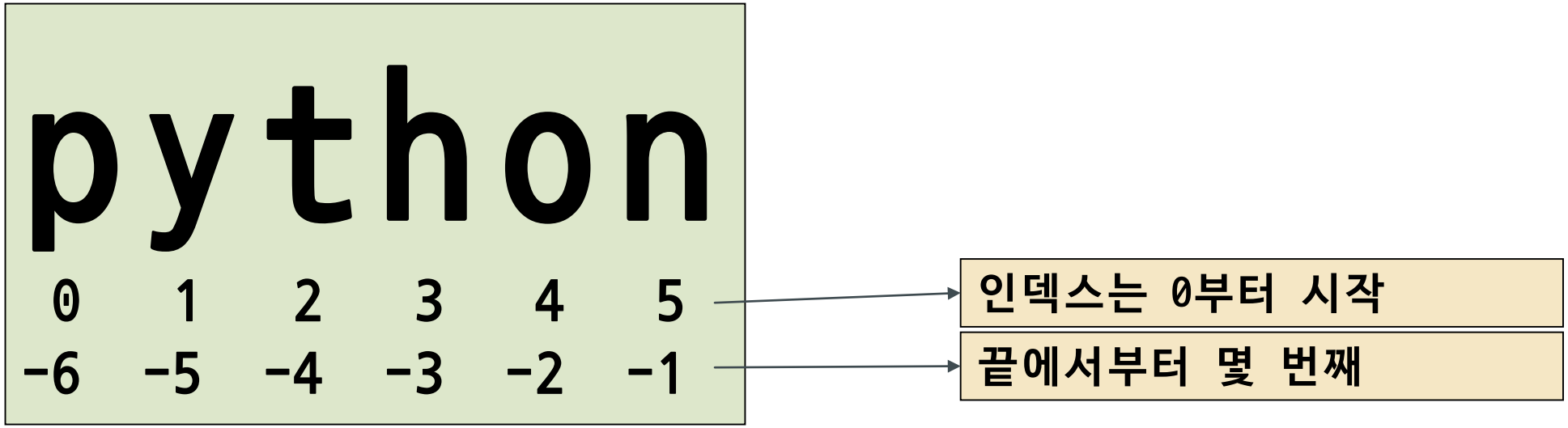
(주의) 반드시 2차원 리스트로 생성 후, 값을 채워서 출력하도록 한다.
2차원 리스트 matrix를 출력하면 아래와 같이 출력 된다

```
>>> print(matrix)
[[0, 1, 1, 1, 1], [-1, 0, 1, 1, 1], [-1, -1, 0, 1, 1], [-1, -1, -1, 0, 1], [-1, -1, -1, -1, 0]]
```

문자열

문자열 인덱싱

- 인덱스를 통해 해당 값에 접근




```
>>> word = "python"
>>> word[0]
'p'
>>> word[-1]
'n'
```

문자열 슬라이싱

- 특정 구간의 값을 취할 수 있음(한 번에 여러 개 추출)

[시작 인덱스:끝 인덱스+1:스텝]

- 시작 인덱스부터 끝 인덱스까지의 자료를 취하라는 의미
- 스텝은 자료를 취하는 간격으로 생략해도 되며 기본값은 1

```
>>> word = "python"
>>> word[0:2]
'py'
>>> word[:2]   # 콜론 앞 생략: 문자열 시작부터 ~
'py'
>>> word[3:]   # 콜론 뒤 생략: ~ 문자열 끝까지
'hon'
```

```
>>> int_str = "12345"
>>> int_str[:]

>>> int_str[::-1]

>>> int_str[::2]
```

문자열 관련함수

```
>>> word = 'Python'
>>> len(word)        # 문자열 길이
6

>>> s = 'Python is fun'
# 문자열 s에서 'n'의 개수 반환
>>> s.count('n')
2

# 문자열 s에서 'n'이 처음 나온 위치 반환
>>> s.index('n')
5

>>> ','.join('1234') #문자열 합해줌
'1,2,3,4'
```

```
>>> s.split()    # 공백으로 분리
['python', 'is', 'fun']

>>> s2 = 'one:two:three'
>>> s2.split(':') # :(콜론)으로 분리
['one', 'two', 'three']
```

실습) 입력 받은 문자열의 마지막 문자를 구해서 출력해 보자.

```
sen = input("문자열 입력: ")

# 음의 인덱스 사용
print(sen[-1])

# len() 함수 이용
??
```

튜플

튜플

- 튜플(tuple)
  - 리스트와 유사하지만 값을 수정할 수 없으며, 읽기만 가능
  - 튜플은 괄호( )로 생성

```
>>> t1 = ()
>>> t2 = (1,)              # 항목이 하나인 경우 주의
>>> t3 = (1, 2, 3)
>>> t4 = 1, 2, 3           # 괄호 생략 가능
```

  - 프로그램이 동작하는 중 변경되지 않는 데이터의 저장

튜플 특성

- 튜플 인덱싱, 슬라이싱

```
>>> t = (1, 2, 3, 4, 5, 6)
>>> t[0]
1
>>> t[:3]
(1, 2, 3)

>>> t[4:6]
(5, 6)

>>> t[0] = 7 ???
```

- 튜플 더하기, 곱하기

```
>>> t1 = (1, 2)
>>> t2 = ('a', 'b')

>>> t3 = t1 + t2
>>> t3
(1, 2, 'a', 'b')

>>> t1 * 3
(1, 2, 1, 2, 1, 2)
```

튜플 패킹, 언패킹

- 튜플 패킹(Tuple Packing)

  : 여러 데이터를 튜플로 묶는 것

```
>>> t = 1, 2, 3
>>> t
(1, 2, 3)
```

- 튜플 언패킹(Tuple Unpacking)

  : 튜플의 각 항목을 여러 개의 변수에 할당하는 것

```
>>> one, two, three = t
>>> print (one, two, three)
1 2 3
```

- 튜플 관련 함수
  - index(), count() 제공

```
# 값 1의 위치
>>> a.index(1)

# 값 1과 일치하는 요소의 개수
>>> a.count(1)
```

딕셔너리

딕셔너리

- 딕셔너리(dictionary)

  - 중괄호{ }로 묶여 있으며, key와 value의 쌍으로 이루어짐

    d = { key1:value1, key2:value2 }

  - key로 value를 관리

    key는 중복 될 수 없음. 예)학번, 주민번호

  - 항목들 사이에 순서가 없음

딕셔너리

```
>>> d = {}
>>> d['kim'] = 1              # 새 항목 추가
>>> d['park'] = 2             # 새 항목 추가
>>> print(d)
{'kim': 1, 'park': 2}

>>> d['park']                 # 키(key)를 사용하여 값에 접근
2

>>> d['kim'] = 3              # 이미 있는 키의 경우 기존 값 변경
>>> print(d)
{'kim': 3, 'park': 2}

>>> d['youn'] = 1             # 새 항목 추가
>>> print(d)
{'kim': 3, 'park': 2, 'youn': 1}
```

딕셔너리

```
>>> d.keys()                                    # 키만 추출
dict_keys(['kim', 'park', 'youn'])

>>> d.values()                                  # 값만 추출
dict_values([3, 2, 1])

>>> d.items()                                   # 키와 값의 항목을 튜플 형태로
dict_items([('kim', 3), ('park', 2), ('youn', 1)])

>>> 'kim' in d                                  # 딕셔너리에 키가 있는 경우
True

>>> del d['kim']                                # 항목 삭제
{'park': 2, 'youn': 1}

>>> d.clear()                                   # 전체 삭제
```

실습 1

- for 문을 사용하여 아래 딕셔너리 d의 모든 value를 출력해 보세요.

```
d = {'kim': 3, 'park': 2, 'youn': 1}
```

실행 예시)

```
3 2 1
```

실습 2 - 1

- 분식집 메뉴 구성
  - 분식점 메뉴를 아래와 같이 딕셔너리를 사용해서 구현합시다.
    총 메뉴 수를 입력 받은 후, 입력한 메뉴 수 만큼 음식명과 가격을 입력 받아 딕셔너리 자료형에 저장하도록 합니다.

```
총 메뉴 수: 3

1번째 추가할 음식명: 김밥
김밥 1인분의 가격: 3500

2번째 추가할 음식명: 떡볶이
떡볶이 1인분의 가격: 2000

3번째 추가할 음식명: 어묵
어묵 1인분의 가격: 1500

{'김밥': 3500, '떡볶이': 2000, '어묵': 1500}
```

실습 2 - 2

- 생성 및 추가한 메뉴를 출력합니다. (for 문 사용)

  : 앞에서 추가한 메뉴를 아래와 같이 출력합니다.

```
--------------
김밥      3500
떡볶이    2000
어묵      1500
--------------
```

실습 2 - 3

- 아래의 실행 예시와 같이 주문을 받습니다.

  : 주문한 내용이 메뉴에 없으면 없다는 메시지를 출력하고 있으면 가격을 출력합니다.

실행 예시1)

```
--------------
김밥      3500
떡볶이    2000
어묵      1500
--------------
메뉴 중 하나를 선택하세요: 김밥
김밥은(는) 3500원입니다.
```

실행 예시2)

```
--------------
김밥      3500
떡볶이    2000
어묵      1500
--------------
메뉴 중 하나를 선택하세요: 순대
순대은(는) 메뉴에 없습니다.
```

Problem

- 문제는 실습시간에 공개합니다.

기타

List Comprehension, zip()

List Comprehension

- 리스트에 for 문을 사용하여 반복적으로 표현식을 실행해서 리스트 원소들을 정의하는 용법

[표현식 for 요소 in 시퀀스자료형 [if 조건문]]

```
>>> L = [ i for i in range(1, 6) ]
>>> L
[1, 2, 3, 4, 5]

>>> [ i ** 2 for i in range(1, 6) ]

>>> [ i for i in range(1, 6) if i % 2 == 0 ]
```

```
# 중첩 for 문
>>> L1 = [1, 2, 3]
>>> L2 = [3, 4, 5]

>>> [ x * y for x in L1 for y in L2 ]
```

- 한 줄 if ~ else : 조건문을 만족하면 A, 아니면 B를 수행

A if 조건문 else B

zip()

- 2개 이상의 리스트를, 각 리스트의 같은 인덱스 원소끼리 묶은 튜플을 원소로 하는 리스트를 만들어 줌

```
>>> list( zip([1,2,3], [4,5,6]) )    # 리스트
[(1, 4), (2, 5), (3, 6)]

>>> list( zip([1,2,3], [4,5,6], ['a','b','c']) )
[(1, 4, 'a'), (2, 5, 'b'), (3, 6, 'c')]

>>> list( zip((1,2,3), (4,5,6)) )    # 튜플
[(1, 4), (2, 5), (3, 6)]

>>> [ sum(x) for x in zip((1,2,3), (4,5,6)) ]
```

실습

- 아래와 같이 주어진 2개의 리스트로 딕셔너리를 만들어 출력하시오. 단, 순서는 다를 수 있음
- zip() 사용할 것

```
L1 = ['one', 'two', 'three', 'four']
L2 = [1, 2, 3, 4]
```

실행 결과)

```
{'one': 1, 'two': 2, 'three': 3, 'four': 4}
```