

K-NET DEVELOP #4

Hyungyu Kim

K-NET

2024-05-14

지난 시간에 이어서...

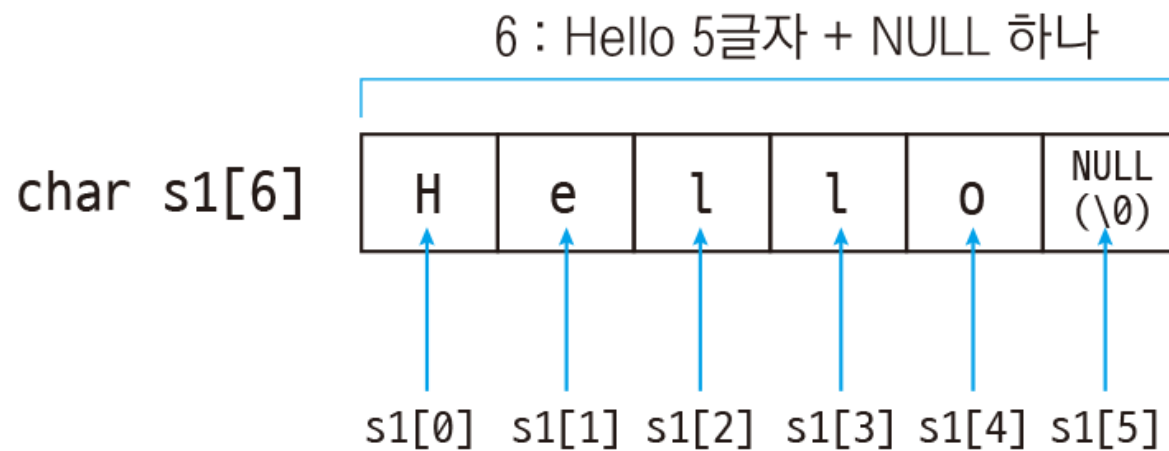
- 변수, 배열, 리스트, 딕셔너리, 조건문, 반복문, 함수, ...
- 이번 회차에서 다룰 주제는 문자열 및 딕셔너리입니다. (튜플은 덤)
- 셋 모두 리스트에서 아주 크게 벗어나지 않는 내용입니다.
(셋의 공통점: 리스트의 인덱싱과 비슷한 것을 지원한다는 특징이 있음)

G	E	E	K	S	F	O	R	G	E	E	K	S
0	1	2	3	4	5	6	7	8	9	10	11	12
-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
>>> a={1:'a'}
>>> a[2]='b'
>>> a
{1: 'a', 2: 'b'}
>>> a['name']='tom'
>>> a
{1: 'a', 2: 'b', 'name': 'tom'}
>>> a[3]=[3,4,5]
>>> a
{1: 'a', 2: 'b', 3: [3, 4, 5], 'name': 'tom'}
>>> del a[1]
>>> a
{2: 'b', 3: [3, 4, 5], 'name': 'tom'}
```

컴퓨터는 숫자만 다루는 것 아니었나요?

- 문자열(string)이란 문자를 나열한 것을 말합니다.
- 숫자만 다루면 별로 재미가 없겠죠? PPT 슬라이드와 chatGPT에서도 문자를 쓰잖아요!
- 중요한 사실: 문자열은 문자의 나열이고, 문자로 이루어진 수열이기 때문에 리스트와 크게 다를 것이 없습니다.



문자열은 따옴표로 감싸서 나타냅니다

- 문자열을 나타내는 방법은 4가지가 있습니다.
 1. 작은 따옴표 한 개씩 써서 양쪽을 감싸기
 2. 큰 따옴표 한 개씩 써서 양쪽을 감싸기
 3. 작은 따옴표 세 개로 양쪽을 감싸기
 4. 큰 따옴표 세 개로 양쪽을 감싸기
- 여기서 3, 4번은 여러 줄을 나타낼 때 유용합니다.

문자열은 따옴표로 감싸서 나타냅니다

- 따옴표의 종류는 자유롭게 구분 없이 섞어서 써도 됩니다.
- 대신, 따옴표 안에 같은 따옴표가 들어가면 안 된다는 것만 주의합니다.

```
print("여러분, 'DEVELOP' 4주차입니다.")
print('여러분, "DEVELOP" 4주차입니다.')
print('''
    여러분, 우리가 드디어...
    "DEVELOP" 4차예요!
    뿌듯하죠?
''')
```

이스케이프 코드

- 참고: 꼭 따옴표 3개짜리가 아니더라도 \n 을 이용하면 줄바꿈(엔터)를 나타낼 수 있습니다. (\t는 탭을 의미)
- C언어 수업을 들으신 분들은 많이 보신 것들입니다.
- 텍스트 파일을 출력할 때 \n이 나오는 건 버그가 아니니까 보시고 놀라면 안 돼요!

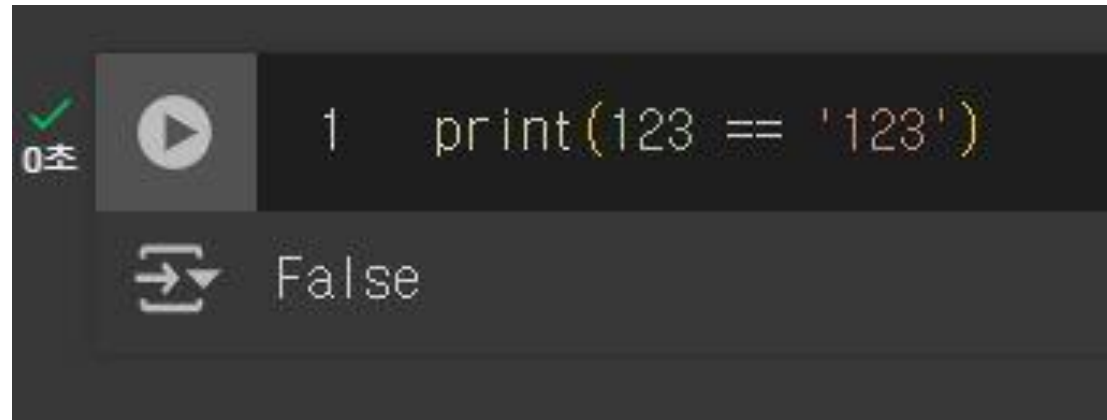
```
[ ] 1 print("여러분, 안녕하세요? 📖이제부터 4주차 내용을 시작하겠습니다.")
```



```
여러분, 안녕하세요?  
이제부터 4주차 내용을 시작하겠습니다.
```

문자열 '123'과 정수 123

- 두 개는 다른 데이터를 뜻하는 것이므로 착오 없길 바랍니다.
- 참고: `input()` 함수로 입력을 받은 것은 모두 문자열이므로 코딩 테스트 시에 헷갈리시면 안 됩니다. – `int(input())`의 형태를 이용해 정수로 변환
- 문자열 '123'은 `int()`로 감싸면 정수 123이 됩니다.
- 정수 123은 `str()`로 감싸면 문자열 '123'이 됩니다.



A screenshot of a code execution environment, likely from a coding platform like LeetCode. It shows a single test case with a green checkmark and a play button icon. The code being executed is `print(123 == '123')`. The result of the execution is `False`, indicating that the test case failed because the integer 123 is not equal to the string '123'.

```
1 print(123 == '123')
```

False

문자열의 연산 및 조작

- 문자열을 더하고 곱하기
- 문자열의 길이 구하기: `len()`
- 문자열 인덱싱
- 문자열 인덱스 슬라이싱 (지난 주차에 시간 상 대충 넘어갔던 것)
- 문자열 포매팅
- 이 외에 자주 쓰는 문자열 관련 함수: `index`, `join`, `split`, `replace`, `upper/lower`

사실 다 몰카였던 것ㅋㅋㅋ

- 리스트에서와 상당히 유사함을 확인할 수 있습니다.
- 문자열을 더하고 곱하기 (띄어쓰기 주의)
- 문자열의 길이 구하기 (리스트와 겹침)
- 문자열 인덱싱 (0부터 시작, 음수 인덱스 가능)

```
1 a = 'nice'
2 b = 'to'
3 c = 'meet'
4 d = 'you'
5 e = a+b+c+d
6 print(e) # 띄어쓰기 주의!
7 print(a, b, c, d) # 띄어쓰기를 하고 싶다면?
```

```
nicetomeetyou
nice to meet you
```

```
[ ] 1 a * 5
```

```
'nicenicenicenice'
```

```
[ ] 1 len(e)
```

```
13
```

```
[ ] 1 e[0], e[1], e[2], e[3]
```

```
('n', 'i', 'c', 'e')
```

```
[ ] 1 e[-1], e[-2], e[-3]
```

```
('u', 'o', 'y')
```

변경 불가능한(immutable) 객체

- 문자열은 인덱스 접근을 통한 수정이 불가능합니다. (곧 나올 튜플도 마찬가지)
- 정수, 실수, 문자열, 튜플 등은 읽기만 가능하고 쓰기는 불가능한 “immutable(변경 불가능한) 객체”입니다.
- 반면 리스트, 딕셔너리는 읽기와 쓰기가 가능한 “mutable(변경 가능한) 객체”입니다.

```
x = "Jython"
x[0] = "P" # 결과는?

print(x)
```

변경 불가능한(immutable) 객체

- 문자열과 튜플은 이뮤터블 객체이므로 **인덱스 접근을 통한 수정이 불가능합니다.**
- 이거 몰라서 코테에서 낭패보는 일 없기 바랍니다!

```
[ ] 1 x = "Jython"
    2 x[0] = "P" # 에러!
    3
    4 print(x)
```



```
-----
TypeError                                Traceback (most recent call last)
<ipython-input-14-02dfb43a3402> in <cell line: 2>()
      1 x = "Jython"
----> 2 x[0] = "P" # 에러!
      3
      4 print(x)

TypeError: 'str' object does not support item assignment
```

인덱스 슬라이싱

- 연속된 여러 문자를 한 번에 잘라내기 위한 방법입니다.
- 리스트, 문자열, 튜플 등에서 사용 가능합니다.
- `str[시작 인덱스 : 끝 인덱스 + 1 : 스텝]`과 같은 식으로 사용 가능합니다.
- 음수 인덱스와 음수 스텝도 넣을 수 있습니다.



```
original = "안녕하세요 여러분, 김현규입니다."
```

```
indexing = original[0] + original[1] + original[2] + original[3] + original[4] + original[5]
```

```
slicing = original[:6] # 0<=x<6번째 인덱스의 문자를 잘라내기(슬라이싱)
```

인덱스 슬라이싱

- 처음 번호, 끝 번호, 스텝이 생략된 경우는 각각 맨 처음, 맨 끝, 1을 나타냅니다.
- 다른 언어에서 보기 힘든 것이라 낯설 수도 있지만, 익숙해지면 아주 편합니다.

```
[ ] 1 original = "안녕하세요 여러분, 김현규입니다."  
    2  
    3 indexing = original[0] + original[1] + original[2] + original[3] + original[4] + original[5]  
    4 slicing = original[:6]  
    5  
    6 print(indexing)  
    7 print(slicing)
```



안녕하세요
안녕하세요

인덱스 슬라이싱 예시

- 사용 예시: 필요 없는 부분 삭제하기
- 아래의 예시는 리스트에서도 모두 똑같이 작용합니다.



```
# 인덱스 슬라이싱이 유용한 경우 - 필요없는 부분 삭제 (또는 필요한 부분만 챙기기)
```


```
with_star = "★택배가 도착했습니다!★"
```

```
without_star = with_star[1:-1]
```

```
print(without_star)
```

인덱스 슬라이싱 예시

- 사용 예시: 문자열 수정하기
- 인덱스 접근을 이용한 수정 대신 인덱스 슬라이싱으로 해결할 수 있습니다.



```
# 인덱스 슬라이싱이 유용한 경우 - 인덱스 접근을 통한 수정

with_typo = "Github"

without_typo = with_typo[:3] + "H" + with_typo[4:]

print(without_typo)
```

인덱스 슬라이싱 예시

- 사용 예시: 회문(palindrome) 확인 – 거꾸로 해도 같은 단어가 되는지?
- 같은 방법으로 리스트 뒤집기도 가능(관련 메서드 `reverse()`도 존재)



```
# 문자열이 회문인지 확인
```

```
example = "우영우"
```

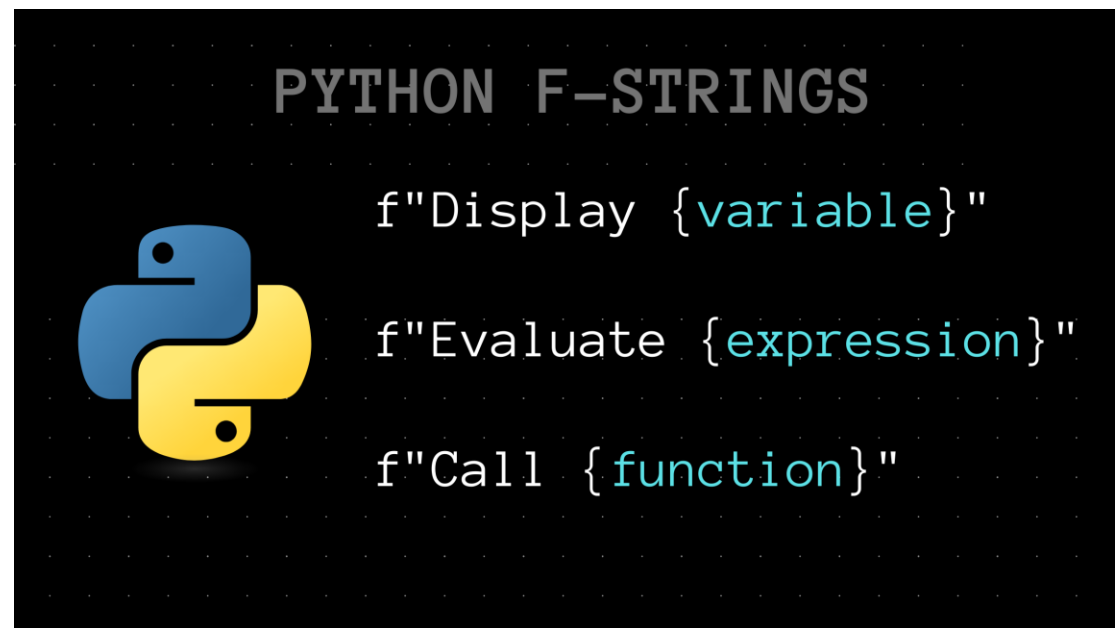
```
reversed = example[::-1] # 거꾸로 한 것
```

```
print(example == reversed)
```


문자열 formatting : f-string

- 문자열 안에 어떤 구체적인 값들을 삽입하는 방법입니다.
- 반복문이나 함수 등에서 문자 안에 변수 값을 섞어 출력해야 할 때 좋습니다.
- 문자열을 포매팅하는 방법이 여럿 있지만 여기서는 **f-string**만 다루겠습니다.
- f-string은 파이썬 3.6 이상 버전에서 제공합니다.

코드	설명
%s	문자열(string)
%c	문자 1개(character)
%d	정수(integer)
%f	부동소수(floating-point)
%o	8진수
%x	16진수
%%	Literal %(문자 % 자체)



문자열 formatting : f-string

- f-string 없이 하라고 해도 할 수는 있습니다. 난잡해질 뿐...

```
# 문자열 formatting 없이 덧셈 연산으로 하려면...
```

```
day = 11  
month = 5  
year = 2024
```

```
print("우리 MT 날짜는 " + str(year) + "년 " + str(month) + "월 " +  
      str(day) + "일입니다.")  
print("여러분, " + str(day+1) + "일 아침까지 놀 준비 됐죠?")
```

문자열 formatting : f-string

- f-string 역시 쓰다 보면 금방 익숙해질 것입니다.
- 문자열 앞의 f를 잊으면 안 됩니다.



f-string을 이용하면 깔끔합니다!

```
print(f"우리 MT 날짜는 {year}년 {month}월 {day}일입니다.")  
print(f"여러분, {day+1}일 아침까지 놀 준비 됐죠?")
```

이외 문자열 관련 함수들

- 자주 쓰는 것만 빠르게 소개하겠습니다.
- 주의: 리스트에서처럼 문자열의 원본을 바꿔주는 것이 아니라, 변경된 새로운 값들을 생성합니다. 코딩 테스트에서 사용 시 주의가 필요합니다.
- `index/find` : 찾는 글자가 몇 번째 인덱스에 있는지 반환
- `join` : 문자열로 이뤄진 리스트 사이사이에 글자를 삽입함
- `split` : 해당 글자를 기준으로 분할하여 리스트로 만들어줌
- `replace` : 문자열 내 해당되는 문자열을 다른 문자열로 교체
- `upper/lower` : 대문자로 바꿈 / 소문자로 바꿈

이외 문자열 관련 함수들

- `index`: 찾는 글자가 몇 번째 인덱스에 있는지 반환. 없다면 `ValueError`
- cf) `find` 메서드: 같은 역할을 하지만, 없다면 `-1` 반환



```
example = "Happy Birthday To You"
```

```
example.index("B")
```

이외 문자열 관련 함수들

- `join` : 문자열로 이뤄진 리스트 사이사이에 글자를 삽입함

```
[ ] 1 query = ["SELECT", "*", "FROM", "my_table", "WHERE", "age > 18", "ORDER BY", "name"]  
    2  
    3 " ".join(query)
```

→ 'SELECT * FROM my_table WHERE age > 18 ORDER BY name'

```
[ ] 1 example = ["승민", "민석", "유찬", "호정"]  
    2  
    3 " 옆에 ".join(example)
```

→ '승민 옆에 민석 옆에 유찬 옆에 호정'

이외 문자열 관련 함수들

- `split` : 해당 글자를 기준으로 분할하여 리스트로 만들어줌
- 괄호 안을 비우면 공백을 기준으로 분할



```
1 example = '승민 옆에 민석 옆에 유찬 옆에 호정'
2
3 example.split(" 옆에 ")
```



```
['승민', '민석', '유찬', '호정']
```



```
[ ] 1 example = "I'm going to buy a computer."
2
3 example.split() # 괄호 안을 비우면 공백(스페이스, \t 등)을 기준으로 split 시행
```



```
["I'm", 'going', 'to', 'buy', 'a', 'computer.']
```

이외 문자열 관련 함수들

- `upper/lower` : 전부 대문자로 바꿈 / 소문자로 바꿈
- 대소문자가 섞인 문자열끼리 내용이 같은지 확인할 때 용이
(대문자 혹은 소문자로 전부 통일시켜서 비교하기)

```
[ ] 1 example = "Life is too short, You need Python"
    2
    3 print(example.upper())
    4 print(example.lower())
```



```
LIFE IS TOO SHORT, YOU NEED PYTHON
life is too short, you need python
```


이외 문자열 관련 함수들

- `replace` : 문자열 내 해당되는 문자열을 다른 문자열로 교체
- 테스트형 가사에서 '테스트형'을 '아버지'로 바꾸기
- '테스트형'이 만약 가사에 없다면 아무 일도 일어나지 않음

```
[ ] 1 lyrics = '''
    2 아! 테스트형 아프다 세상이 눈물 많은 나에게
    3 아! 테스트형 소크라테스트형 세월은 또 왜 저래
    4 먼저 가본 저세상 어떨까요 테스트형
    5 가보니까 천국은 있던가요 테스트형
    6
    7 아! 테스트형 아! 테스트형 아! 테스트형 아! 테스트형
    8 아! 테스트형 아! 테스트형 아! 테스트형 아! 테스트형
    9 '''
   10
   11 modified = lyrics.replace("소크라테스트형", "우리 아버지")
   12 modified = modified.replace("테스트형", "아버지")
   13
   14 print(modified)
```



아! 아버지 아프다 세상이 눈물 많은 나에게
아! 아버지 우리 아버지 세월은 또 왜 저래
먼저 가본 저세상 어떨까요 아버지
가보니까 천국은 있던가요 아버지

아! 아버지 아! 아버지 아! 아버지 아! 아버지
아! 아버지 아! 아버지 아! 아버지 아! 아버지

문자열의 in 연산자

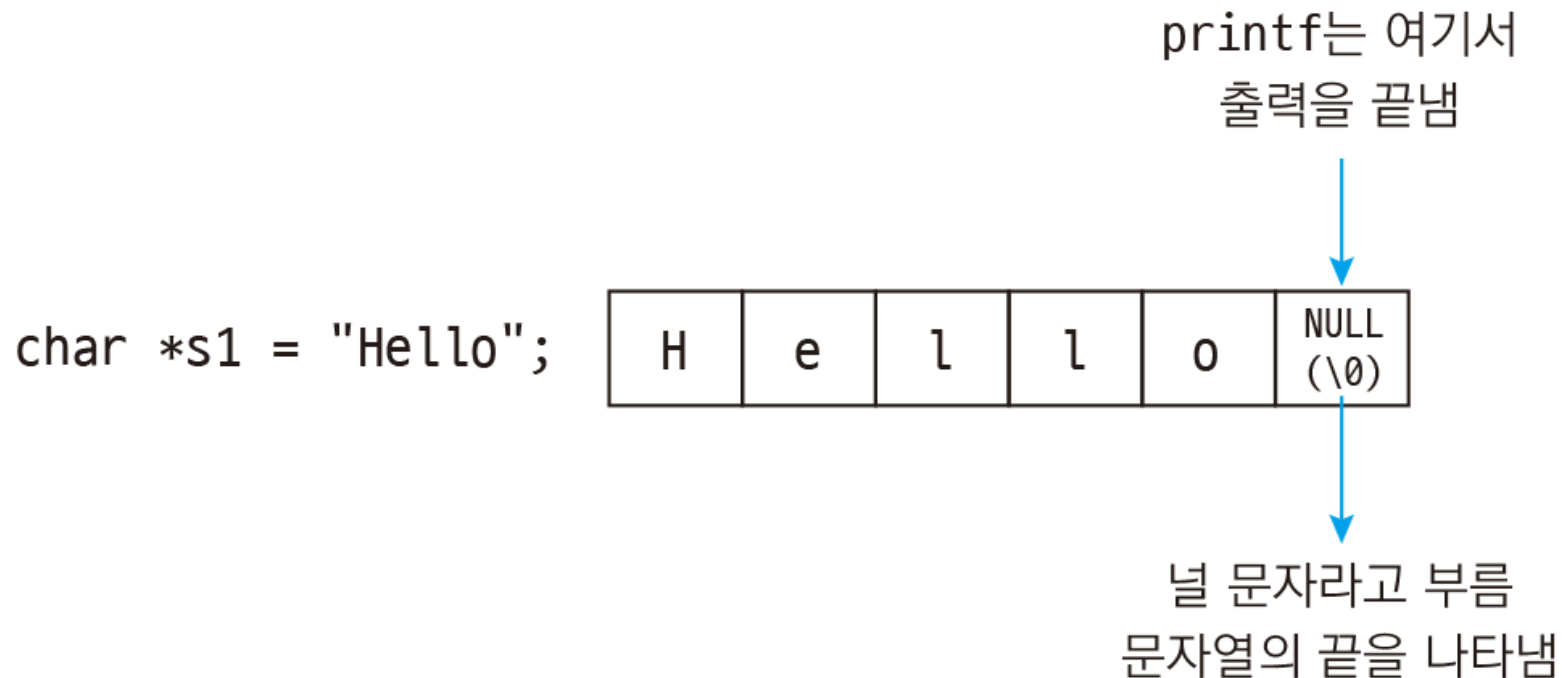
- 해당되는 문자열이 부분으로 존재하는지 찾기
- 리스트, 튜플, 딕셔너리에서도 사용 가능
- 존재하지 않는 것을 검사하려면 `not in` 사용



```
example = "파이썬 잘 모르겠다! 포기하자!"  
  
print("포기" in example) # in 연산자를 이용
```

C언어의 문자열과 파이썬의 문자열

- C언어의 문자열은 배열에 한 글자씩 저장하지만 파이썬에서는 그렇지 않습니다.
- C언어의 문자열은 +를 이용해 더할 수 없으나, 파이썬에서는 +로 더할 수 있습니다.

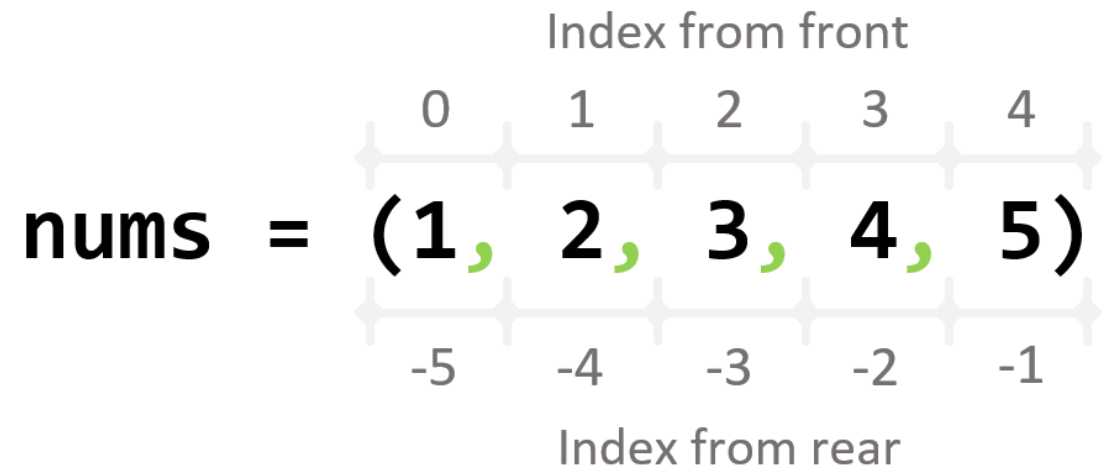


화석 김현규의 끈대질 시간...

- 급체 주의! 아까 나열한 거 지금 다 외우려고 하면 체합니다.
- 혹시 다 삼키신 분? 빨리 토하세요..ㅋㅋㅋ
- 저런 것들은 써보면서 자연스레 익숙해지는 것이 제일 편해요
- 잘 외우는 사람보다는 구글링을 잘 하는 사람이 되면 개꿀이에요
- 이번 한 시간으로 끝내지 말고 이것저것 구현해봐요! 백준 쉬운 문제도 좋고...
- 사실 이거 지난 시간에 적은 것 그대로 복붙한 거임...

간단히만 살펴보는 튜플

- tuple은 기본적으로 순서쌍의 의미를 갖습니다.
- 파이썬의 튜플은 리스트와 거의 비슷하지만, 각 요소(element)에 대한 수정이 불가능합니다. (이뮤터블 객체)
- 리스트를 대괄호로 감싸서 나타내는 반면, 튜플은 소괄호로 감싸서 나타냅니다.



간단히만 살펴보는 튜플

- 코딩 중 바뀌지 않기를 바라는 값, 고유한 값을 저장하고 싶다면 리스트보다 튜플 사용이 유리합니다.
- 다시 말해, 리스트와 튜플을 구분하여 사용하는 것이 유리합니다.
- 튜플에서도 인덱싱, 인덱스 슬라이싱, 덧셈, 곱셈, 길이 등을 리스트나 문자열에서와 똑같이 할 수 있습니다.



```
example_tuple = (1280, 720)

print(len(example_tuple))

print(example_tuple[0], example_tuple[1])
```

간단히만 살펴보는 튜플

- 튜플은 문자열처럼 이뮤터블 객체입니다. 인덱스 접근 수정이 불가능합니다.
- 참고: 튜플이 저장된 변수에 다른 튜플을 할당하는 것은 가능합니다.

```
[ ] 1 example_tuple[0] = 1000
```



```
-----  
TypeError                                Traceback (most recent call last)  
<ipython-input-24-39b8194eaf51> in <cell line: 1>()  
----> 1 example_tuple[0] = 1000  
  
TypeError: 'tuple' object does not support item assignment
```

딕셔너리, 네 정체는 뭐냐?

- 딕셔너리는 기본적으로 **Key-Value** 쌍을 저장합니다.
- 리스트의 **각 요소 값에다가 특정한 의미를 부여**하고 싶을 때 쓰면 좋습니다.
- 예시: 서버에 접속한 클라이언트에게 에러 메시지를 어떻게 주는 것이 더 좋을까요?
1번) [404, 'Not Found', 'Cannot GET /profiles']
2번) {'statusCode': 404,
 'error': 'Not Found',
 'message': 'Cannot GET /profiles'}

딕셔너리, 네 정체는 뭐냐?

- 처음 접할 때는 왜 쓰는지 공감하기 어려울 수 있지만 나중에 이것이 요긴하다는 것이 입증될 것입니다.
- 참고: 곧 알게 되겠지만, 리스트를 쓰는 것은 사실 이전 슬라이드의 예시에서
 {0: 404,
 1: 'Not Found',
 2: 'Cannot GET /profiles'}
 를 쓰는 것과 비슷하게 작동합니다.
- 참고: 딕셔너리는 javascript에서의 객체와 거의 같으며, json(JavaScript Object Notation)과 거의 같습니다.

Key-Value에 대한 이야기

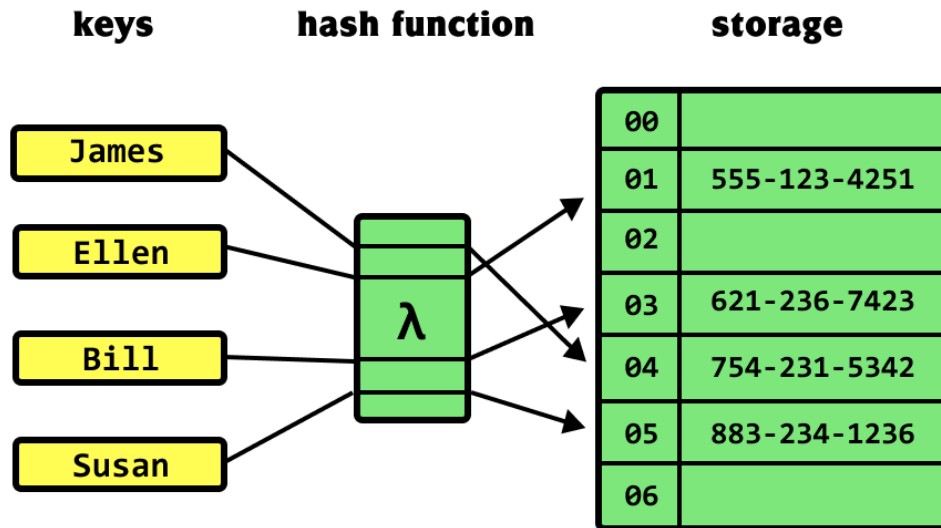
- 딕셔너리의 기본 모습은 이렇습니다.

{key1: value1, key2: value2, key3: value3, ... }

- 빈 딕셔너리는 “new_dict = {}”와 같이 선언합니다.
- 참고: 리스트는 대괄호, 튜플은 소괄호, 딕셔너리는 중괄호를 이용합니다.
- 참고: 위에서 콤마(반점)가 반드시 있어야 합니다. 없다면 에러가 뜹니다.
- 처음 배우시는 분들께서는 그러한 실수나 오타에서 에러가 난 것을 빨리 찾지 못해 두세 시간을 날리곤 합니다… 에러 메시지와 Traceback을 잘 읽어봅시다!

딕셔너리의 내부 구조

- 파이썬의 딕셔너리는 **해시 테이블**로 구현되어 있습니다.
- 관심 있으신 분들은 "해시 테이블"로 검색 바랍니다.
- 코딩 테스트에서 해시를 활용하실 분들은 파이썬의 딕셔너리를 이용하실 수 있습니다.



딕셔너리와 리스트 간 중첩

- 파이썬의 딕셔너리 안에는 정수, 문자열 뿐만 아니라 리스트, 딕셔너리도 들어갈 수 있습니다.
- 지난 시간에 보듯이 리스트 안에도 딕셔너리가 들어갈 수도 있습니다.

```
import pandas as pd

dict1 = {
    'Names': ['Alan', 'Peter', 'John'],
    'Age': [12, 41, 30],
    'Level': [5, 2, 1]
}

df = pd.DataFrame(dict1)

display(df)
```

	Names	Age	Level
0	Alan	12	5
1	Peter	41	2
2	John	30	1

```
my_data = {
    'Python dev': ['Sachin', 'Aman', 'Siya'],
    'C++ dev': ['Rishu', 'Yashwant', 'Rahul'],
    'PHP dev': ['Abhishek', 'Peter', 'Shalini'],
    'detail': {
        'name': ['Sachin', 'Siya'],
        'hobby': ['Coding', 'Reading']
    }
}
```

딕셔너리의 조작

- 키를 통해 값 찾기: `a[key]`
(리스트, 문자열, 튜플의 인덱싱과 비슷)
- 딕셔너리에 쌍 추가 및 수정: `a[new key] = new value`
(리스트에서는 `a[new index]`로 접근하여 추가하면 에러 발생)
- 딕셔너리에서 쌍 제거: `del a[key]`
(리스트와 비슷)

딕셔너리의 조작

- 키를 통해 값 찾기
(리스트, 문자열, 튜플의 인덱싱과 비슷)

```
response = {  
    'statusCode': 404,  
    'error': 'Not Found',  
    'message': 'Cannot GET /profiles'  
}  
  
print(response['statusCode'])
```

딕셔너리의 조작

- 딕셔너리에 Key-Value 쌍 추가 및 수정
(리스트에서는 새 인덱스로 접근하여 추가하면 에러 발생 – 직접 해보세요!)
- 딕셔너리에서 Key-Value 쌍 제거



```
students_result = {"철수": 50, "훈이": 55, "맹구": 60, "유리": 65, "짱구": 70}
```

```
students_result["현등"] = 75 # 새로운 값 추가
```

```
del students_result["훈이"]
```

```
print(students_result)
```

딕셔너리의 출력 순서

- 참고: 파이썬 3.6부터는 Key가 입력된 순서대로 딕셔너리를 출력합니다.
- 아래 사진에서 흰둥이가 딕셔너리의 맨 마지막에 출력됨

```
[ ] 1 students_result = {"철수": 50, "훈이": 55, "맹구": 60, "유리": 65, "짱구": 70}
    2
    3 students_result["흰둥"] = 75 # 새로운 값 추가
    4 del students_result["훈이"]
    5
    6 print(students_result)
```

```
↔ {'철수': 50, '맹구': 60, '유리': 65, '짱구': 70, '흰둥': 75}
```


리스트 2개로 딕셔너리 만들기

- 참고: zip()을 이용하면 리스트 두 개를 묶어 딕셔너리로 만들 수 있습니다.
- zip()은 유용하지만 많이 자주 쓰이지는 않는 듯합니다.

```
[ ] 1 names = ["철수", "훈이", "맹구", "유리", "짱구"]
     2 results = [50, 55, 60, 65, 70]
     3
     4 students_result = dict(zip(names, results))
     5 students_result
```

```
➔ {'철수': 50, '훈이': 55, '맹구': 60, '유리': 65, '짱구': 70}
```

```
[ ] 1 # zip의 정체는 튜플을 원소로 하는 리스트와 비슷함
     2 list(zip(names, results))
```

```
➔ [('철수', 50), ('훈이', 55), ('맹구', 60), ('유리', 65), ('짱구', 70)]
```

Key의 중복은 허용되지 않습니다

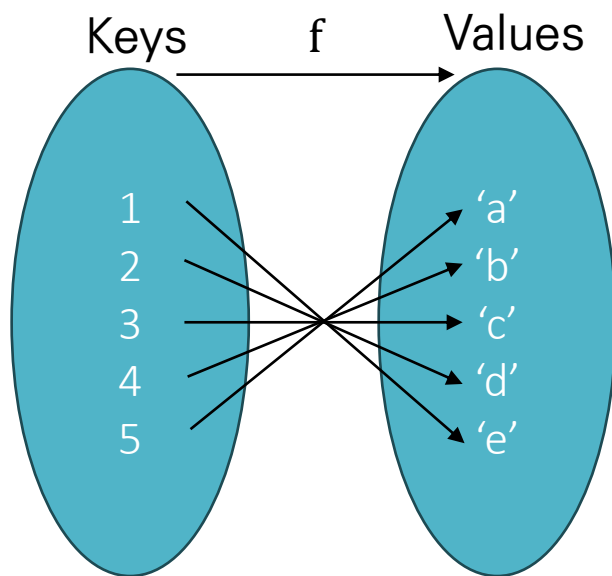
- 아래처럼 딕셔너리를 만들면 맨 마지막의 키-값 쌍인 {1: 'c'}만 남습니다.
- 이로 인해 딕셔너리는 사실 함수처럼 사용할 수도 있습니다.
- 인간적으로 저렇게 코드 쓰지 말자구요… (잠시, 화석 김현규의 끈대질 시간…)

```
[ ] 1  # 딕셔너리는 Key의 중복을 허용하지 않습니다.  
    2  
    3  # 화석 김현규의 끈대질 시간: 가급적 이렇게 코드 작성하지 마세요!  
    4  # 코드는 컴퓨터가 읽기 전에 인간이 먼저 읽습니다.  
    5  
    6  example = {1: 'a', 1: 'b', 1: 'c'}  
    7  print(example)
```

⇒ {1: 'c'}

딕셔너리를 함수처럼?

- Key들의 모임: 정의역과 비슷함 (중복 비허용)
- Value들의 모임: 치역과 비슷함
- 코드 가독성에 있어서 더 유리합니다.



```
def f(x):  
    if x == 1: return 'e'  
    elif x == 2: return 'd'  
    elif x == 3: return 'c'  
    elif x == 4: return 'b'  
    elif x == 5: return 'a'  
  
print(f(1), f(2), f(3), f(4), f(5))
```

```
# 딕셔너리는 함수의 의미를 실제로 깔끔하게 대체할 수 있습니다.  
  
f = {1: 'e', 2: 'd', 3: 'c', 4: 'b', 5: 'a'}  
  
print(f[1], f[2], f[3], f[4], f[5])
```

딕셔너리 관련 함수들

- 아래 함수들은 특히 반복문 등에서 유용합니다.
- `keys()`: 딕셔너리의 모든 Key들을 담음
- `values()`: 딕셔너리의 모든 Value들을 담음
- `items()`: 딕셔너리의 Key-Value 쌍을 튜플로 담음
- 참고: 위의 세 개의 함수들의 반환은 list가 아니라서, 리스트로 쓰고 싶다면 `list()`로 감싸주어야 합니다. 단 반복문을 통한 순회에서는 그렇게 할 필요가 없습니다.
- `update()`: 딕셔너리 두 개를 병합함 (리스트에서의 `extend` 느낌)

딕셔너리 관련 함수들

- update 메서드 사용 예시
- 솔직히 저도 이거 써본 적은 없지만... 잘 쓰면 좋아보여서 실습합니다. 참고로 알아두세요.

```
1 price_dict = {'apple': 10, 'banana': 15, 'cherry': 20}
2
3 price_dict.update({'grape': 12, 'kiwi': 15})
4
5 print(price_dict)
```

{ 'apple': 10, 'banana': 15, 'cherry': 20, 'grape': 12, 'kiwi': 15 }

컴프리헨션을 딕셔너리에서도!

- 딕셔너리 컴프리헨션을 통해 아까 소개한 함수들을 야무지게 써먹을 수 있습니다.



```
1 names = ["철수", "훈이", "맹구", "유리", "짱구"]
2 scores = [50, 55, 60, 65, 70]
3 students_result = {name: score for name, score in zip(names, scores)}
4 print(students_result)
5
6 # 훈이 모자이크가 하고 싶음 (강제 전학?) - if문 적용으로 구현 가능
7 students_result = {k: v for k, v in students_result.items() if k != "훈이"}
8 print(students_result)
```



```
{'철수': 50, '훈이': 55, '맹구': 60, '유리': 65, '짱구': 70}
{'철수': 50, '맹구': 60, '유리': 65, '짱구': 70}
```

다음 시간에 잠시 발표할 사람?

- `replace` 메서드를 이용해 다음 문자열에서 `심표(,)`만 전부 제거해보세요.

“에혀,,, 저래가지고,,,, 쓰겠나,,? 요즘 것들은,,, 이래서 안 돼,,,,, 라떼는 말이야,,,,,”

우리 스터디에서 다룰 주요 주제

- 파이썬 조감도
- 변수와 리스트
- 문자열, 튜플, 딕셔너리
- 조건문과 반복문
- 함수
- 클래스와 상속
- 파이썬을 이용한 문제 해결 (간단한 알고리즘 문제들) – 미정
- 파이썬 라이브러리로 할 수 있는 일 (이미지 처리, 엑셀 데이터 처리, 그래프 그리기, ...)

다음 시간에 다룰 주제

- 조건문과 반복문

