

Python & Data Engineering

인공지능 직무전환자 과정

- Day 2

Advanced Python



Sanghyun Seo

0. Review

Course Descriptions

- Python & Data Engineering

Day 1 – Python Basic	Day 2 – Advanced Python	Day 3 – Numpy, Pandas	Day 4 – Visualization, Data Analysis
<ul style="list-style-type: none">• Getting started• Introduction to Python• Data Type & Variable• Flow control• Function• Python programming practice	<ul style="list-style-type: none">• Review• File• Class• Exception Handling• Advanced python	<ul style="list-style-type: none">• Review• Module, Package• Numpy Basic• Advanced Numpy• Pandas Basic• Advanced Pandas	<ul style="list-style-type: none">• Review• Introduction to machine learning• Data preprocessing• Visualization• Course summarization

Contents

1. File
2. Class
3. Exception Handling
4. Advanced Python

1. File

Python 출력

- print(출력대상1, 출력대상2 ...)
 - python에서는 표준 출력 함수로서 print()함수를 지원
 - sep: 출력시 출력대상들의 사이에 구분자를 삽입 (기본값: 공백)
 - end: 마지막 문자열을 출력하고 이어서 출력할 문자 기술 (기본값 : 줄바꿈문자)

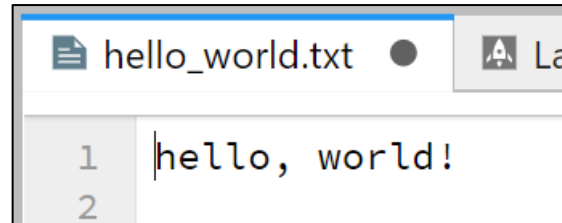
```
print("하나", "둘", "셋", 1, 2, 3)
print("하나", "둘", "셋", 1, 2, 3, sep='-')
print("첫번째 값")
print("두번째 값") # 다른 줄에 출력
print("첫번째 값", end=" ---> ")
print("두번째 값") # 같은 줄에 출력
```

```
하나 둘 셋 1 2 3
하나-둘-셋-1-2-3
첫번째 값
두번째 값
첫번째 값 ---> 두번째 값
```

Python 출력

- print(출력대상1, 출력대상2 ...)
 - file: 출력 위치를 변경(기본값 : sys.stdout(표준출력장치,모니터))
 - flush: 스트림을 강제로 flush할지를 지정 (기본값 : False)

```
file = open("hello_world.txt","w")  
print("hello, world!", file=file) # 파일로 출력  
file.close()
```



Python 출력

- 포매팅(formatting)
 - 데이터를 가지고 작업할 때 구조화된 출력(테이블 등)을 생성하고자 할 때 사용
 - 형식화된 문자열을 만든 다음 이를 출력하는 방식
- 주요 포매팅 방식
 - %
 - format()
 - f-string
 - 딕셔너리 포매팅

Python 출력

- % formatting
 - 포매팅 코드
 - **%d**: int
 - **%f**: float
 - **%s**: string
 - **%c**: 문자 1개
 - **%%**: %자체
 - 괄호 사용
 - 2개이상 포매팅

```
num = 1
```

```
print('문자열 포매팅 하기 예시 %d' %num)
```

문자열 포매팅 하기 예시 1

```
ch = '이름'
```

```
print('내 이름은 : %s' %ch)
```

내 이름은 : 이름

```
fl = 2.566132
```

```
print('반올림 하기 : %0.1f' %fl)
```

반올림 하기 : 2.6

```
a = 3
```

```
b = 5
```

```
print('%d 와 %d 를 더하면 %d' %(a,b,a+b))
```

3 와 5 를 더하면 8

Python 출력

- format() 메소드 → {표현식:포맷}
 - '문자열'.format(포맷팅 할 자료)

```
print('문자열 포맷팅 하기 예시 {}'.format(2))
```

문자열 포맷팅 하기 예시 2

```
print('내 이름은 : {}'.format(ch))
```

내 이름은 : 이름

- 실수(소수점) 포맷팅

```
fl = 2138.8728829
```

```
print('소수점 포맷팅하기 : {:.3f}'.format(fl))
```

소수점 포맷팅하기 : 2138.873

```
round(fl, 3)
```

2138.873

Python 출력

- format() 메소드
 - 괄호를 이용한 두 개 이상의 변수 포매팅

```
ln = '서'  
fn = '상현'
```

```
print('성은 {}, 이름은 {}'.format(ln, fn))
```

성은 서, 이름은 상현

- 인덱스를 통한 포매팅에 사용할 변수의 순서 지정

```
print('성은 {0}, 이름은 {1}'.format(ln, fn))
```

성은 서, 이름은 상현

```
print('First name : {1}, Last name : {0}'.format(ln, fn))
```

First name : 상현, Last name : 서

Python 출력

- format() 메소드
 - 변수명을 이용한 포매팅

```
커피 = '커피'  
print('나는 오늘 {음료}를 {개수} 잔이나 마셨다'.format(음료=커피, 개수=3))
```

나는 오늘 커피를 3 잔이나 마셨다

Python 출력

- format() 메소드
 - 길이와 정렬
 - **{:길이}** : 출력할 데이터의 길이를 지정합니다. 문자열(왼쪽 정렬), 숫자(오른쪽 정렬)
 - **{:<길이}** : 왼쪽 정렬
 - **{:>길이}** : 오른쪽 정렬
 - **{:^길이}** : 가운데 정렬

Python 출력

- format() 메소드
 - 길이와 정렬

```
print('Python is [{:15}]'.format('good'))
print('Python is [{:<15}]'.format('good'))
print('Python is [{:>15}]'.format('good'))
print('Python is [{:^15}]'.format('good'))
print('당신의 나이는 [{:15}]세'.format(22))
print('당신의 나이는 [{:<15}]세'.format(22))
print('당신의 나이는 [{:>15}]세'.format(22))
print('당신의 나이는 [{:<15}]세'.format(22))
```

```
Python is [good          ]
Python is [good          ]
Python is [              good]
Python is [      good      ]
당신의 나이는 [          22]세
당신의 나이는 [22         ]세
당신의 나이는 [          22]세
당신의 나이는 [22         ]세
```

Python 출력

- format() 메소드
 - 0으로 채우기, 부호 출력

```
# 0으로 채우기
for x in range(1, 4):
    print('{0:02d} {1:03d} {2:04d}'.format(x, x*x, x*x*x))
print()
# 부호 출력
print('{0:05d} {1:05d} {2:05d}'.format(1,-2,3))    # 음수만 부호
print('{0:+05d} {1:+05d} {2:+05d}'.format(1,-2,3)) # 부호
print('{0:<5d} {1:<5d} {2:<5d}'.format(1,-2,3))    # 정렬
```

```
[01] [001] [0001]
[02] [004] [0008]
[03] [009] [0027]
```

```
[00001] [-0002] [00003]
[+0001] [-0002] [+0003]
[1     ] [-2     ] [3     ]
```

Python 출력

- format() 메소드

- 채움문자와 숫자 표현식

- {[인덱스]:[채움문자][정렬][길이][.][_][형식문자]}

- 공백을 채움문자로 채움
 - , : 천단위 마다 콤마를 삽입
 - _ : 천단위 마다 밑줄을 삽입
 - E : 숫자를 지수 형식으로 변환
 - = : 부호를 항상 제일 앞에서 출력 (숫자 형식에만 사용)

Python 출력

- format() 메소드

- 채움문자와 숫자 표현식

→ {{인덱스}:{채움문자}[정렬][길이][|_][형식문자]}

```
# 채움문자
print('{0:05d} {1:05d} {2:05d}'.format(1,-2,3))
print('{0:★<5d} {1:★<5d} {2:★<5d}'.format(1,-2,3))
print('{0:★>5d} {1:★>5d} {2:★>5d}'.format(1,-2,3))
print('{0:★^5d} {1:★^5d} {2:★^5d}'.format(1,-2,3))
print('{0:5d} {1:5d} {2:5d}'.format(1,-2,3))
print('{0:=5d} {1:=5d} {2:=5d}'.format(1,-2,3))
print('{0:=05d} {1:=05d} {2:=05d}'.format(1,-2,3))
# 숫자 표시 형식( , _)
print('{0:>5,} {1:>5,} {2:>5,}'.format(11544435,-2544254,35454343))
print('{0:>5_} {1:>5_} {2:>5_}'.format(11544435,-2544254,35454343))
print('{0:>5e} {1:>5e} {2:>5e}'.format(11544435,-2544254,35454343))
```

```
[00001] [-0002] [00003]
[1★★★★] [-2★★★★] [3★★★★]
[★★★★1] [★★★★-2] [★★★★3]
[★★1★★] [★-2★★] [★★3★★]
[  1] [ -2] [  3]
[  1] [- 2] [  3]
[00001] [-0002] [00003]
[11,544,435] [-2,544,254] [35,454,343]
[11_544_435] [-2_544_254] [35_454_343]
[1.154444e+07] [-2.544254e+06] [3.545434e+07]
```

Python 출력

- f-string 포매팅
 - python 3.6 버전 이상부터 지원
 - string 앞에 f를 붙여서 사용 → **f'문자열'**

```
year = 2020
```

```
f'올해는 {year}'
```

```
'올해는 2020'
```

Python 출력

- f-string 포매팅
 - 다른 포매팅 방식과는 달리 표현식을 지원

```
drink = '커피'  
nums = 3
```

```
f'나는 오늘 {drink}를 {nums + 1} 잔이나 마셨다'
```

```
'나는 오늘 커피를 4 잔이나 마셨다'
```

```
'나는 오늘 {음료}를 {개수+1} 잔이나 마셨다'.format(음료='커피', 개수=3)
```

```
-----  
-----  
KeyError                                Traceback (most recent  
t call last)  
<ipython-input-5-21356a636e64> in <module>()  
----> 1 '나는 오늘 {음료}를 {개수+1} 잔이나 마셨다'.format(음료='커피', 개수=3)  
KeyError: '개수+1'
```

Python 출력

- 딕셔너리 포매팅

```
s = {'name': 'IBM',  
     'shares': 100,  
     'price': 91.1}
```

```
'{name:>10s}{shares:10d}{price:10.2f}'.format_map(s)
```

```
'          IBM          100      91.10'
```

사용자 입력 (user input)

- 사용자 입력 → **input()**
 - 사용자가 입력한 값을 특정 변수 등에 할당하여 처리하고 싶을 때 사용

```
# Life is too short, you need python 입력  
a = input()
```

```
Life is too short, you need python
```

```
a
```

```
'Life is too short, you need python'
```

사용자 입력 (user input)

- 사용자 입력

- 사용자로부터 입력을 받을 시 어떠한 자료를 입력해야 하는지 안내를 해야 할 필요성이 있음
- `Input()`의 인자로 문자열을 입력하면 해당 문자열이 출력되면서 사용자 입력을 받게 됨

```
number = input("숫자를 입력하세요: ")
```

```
숫자를 입력하세요: 123
```

```
number
```

```
'123'
```

사용자 입력 (user input)

- 사용자 입력

- Input()은 모든 입력에 대해서 string으로 처리
- 사용자로부터 입력받은 자료를 숫자형 자료로 활용하고 싶으면 타입 캐스팅이 필요

```
integer = int(input("정수형 숫자를 입력하세요: "))  
integer
```

정수형 숫자를 입력하세요: 13

13

```
real = float(input("실수형 숫자를 입력하세요: "))  
real
```

실수형 숫자를 입력하세요: 21.5

21.5

사용자 입력 (user input)

- 사용자 입력

- 타입 캐스팅시 에러가 발생할 수 있음을 유의 → 예외처리 기능 사용 가능

```
integer = int(input("정수형 숫자를 입력하세요: "))  
integer
```

정수형 숫자를 입력하세요: 21.5

```
-----  
ValueError                                Traceback (most recent call last)  
<ipython-input-40-84556bfdaed5> in <module>()  
----> 1 integer = int(input("정수형 숫자를 입력하세요: "))  
      2 integer  
  
ValueError: invalid literal for int() with base 10: '21.5'
```


파일 (file)

- Colab과 google drive 연동
 - google.colab의 drive 라이브러리를 이용하여 구글 드라이브 mount수행
→ 첫 실행시 구글 계정에 대한 권한을 입력해야 함

```
[4] 1 import os  
    2 from google.colab import drive  
    3 drive.mount('/content/drive')
```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True)

파일 (file)

- Colab과 google drive 연동
 - 현재 워킹 디렉토리(working directory)를 .ipynb를 작업 중이 경로로 변경 필요
 - `'/content/drive/My Drive'`가 구글 드라이브 실행시 기본 위치
 - (예시 경로)
`'/content/drive/My Drive/Lectures/LG_Electronics/python/day2'`

```
1 print(os.getcwd())  
  
/content  
  
1 os.chdir('/content/drive/My Drive/Lectures/LG_Electronics/python/day2')  
2 print(os.getcwd())  
  
/content/drive/My Drive/Lectures/LG_Electronics/python/day2
```

파일 (file)

- Colab과 google drive 연동
 - google.colab의 drive 라이브러리를 이용하여 구글 드라이브 mount수행
→ 첫 실행시 구글 계정에 대한 권한을 입력해야 함

```
[4] 1 import os  
    2 from google.colab import drive  
    3 drive.mount('/content/drive')
```

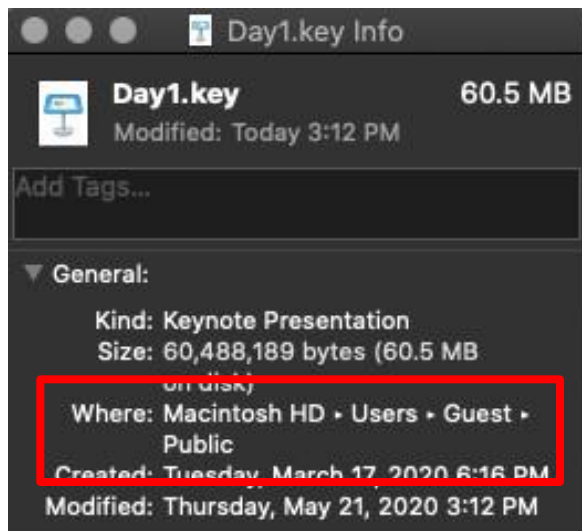
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive?force_remount=True")

파일 (file)

- 파일 경로(file path)
 - file path(파일 경로)는 string
 - 파일은 두 가지 방식으로 표현할 수 있음
 - 절대 경로 : 파일이 가지고 있는 고유한 경로
 - 상대 경로 : 현재 내 위치에서부터 해당 파일까지의 경로

파일 (file)

- 파일 경로(file path)
 - 절대 경로 : 파일이 가지고 있는 고유한 경로
 - 최초의 시작점으로부터 경유한 경로를 전부 기입
 - 현재 위치와 무관하게 항상 해당 파일에 접근 가능



→ 'Users/Guest/Public/Day1.key'

파일 (file)

- 파일 경로(file path)

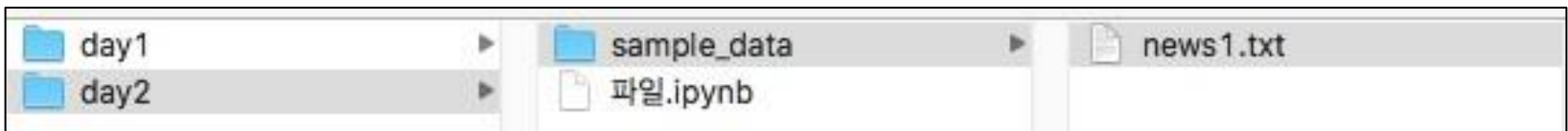
- 상대 경로 : 현재 내 위치에서부터 해당 파일까지의 경로

- 현재의 파일 위치를 기준으로 경로를 계산

- 현재 파일의 위치: **./**

- 현재 파일의 상위 폴더: **../**

- Quiz 1. 현재 '파일.ipynb'에서 작업 중일 때, 'news1.txt' 파일에 접근하기 위해서 필요한 상대 경로는 무엇인가?



파일 (file)

- 파일 경로(file path)

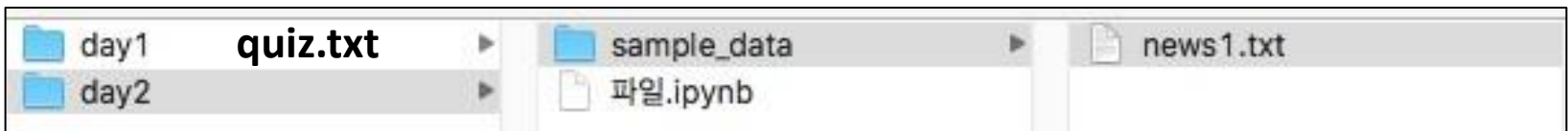
- 상대 경로 : 현재 내 위치에서부터 해당 파일까지의 경로

- 현재의 파일 위치를 기준으로 경로를 계산

- 현재 파일의 위치: **./**

- 현재 파일의 상위 폴더: **../**

- Quiz 1. 현재 '파일.ipynb'에서 작업 중일 때, 'day1' 디렉토리(폴더)에 있는 'quiz.txt' 파일에 접근하기 위해서 필요한 상대 경로는 무엇인가?



파일 (file)

- 파일 접근 → `open(file_path, mode="", encoding="")`
 - `open`은 파일에 접근하는데 사용하는 함수
 - `file_path`: 파일의 주소
 - `mode`: 파일을 열 때 필요한 파일 모드 (읽기/쓰기/수정)
 - `encoding`: 파일을 열 때 필요한 인코딩 (생략가능)
 - `open`으로 연 파일은 파일 객체를 반환

```
1 file_path_1 = '/content/drive/My Drive/Lectures/LG_Electronics/python/day2/practice/new1.txt'  
2 new_file = open(file_path_1, 'w')
```

```
1 print(new_file)
```

```
<_io.TextIOWrapper name='/content/drive/My Drive/Lectures/LG_Electronics/python/day2/practice/new1.txt'
```


파일 (file)

- open()
 - 파일 모드 비교

파일 모드	설명	비고
r	읽기 모드	파일에 새로운 내용을 쓰거나 수정이 불가능
w	쓰기 모드	파일에 새로운 내용을 덮어쓰기(기존의 내용 모두 삭제)
a	수정 모드	파일에 새로운 내용이 추가됨(기존의 내용 + 새로운 내용)
rb	바이너리 읽기 모드	바이너리 파일을 읽을 때 사용
wb	바이너리 쓰기 모드	바이너리 파일을 쓸 때 사용

- 인코딩
 - 지정하지 않으면 플랫폼을 따름
 - UTF-8: 대표적인 유니코드 인코딩 방식

파일 (file)

- 파일쓰기(file write) → 파일.write(내용)
 - 파일: open으로 열었던 파일 객체
 - 내용: 파일에 쓰고 싶은 내용

```
data = '새로운 내용을 쓰고 싶다면,\n이렇게 작성하면 됩니다.'
```

```
new_file.write(data)
```

```
29
```

```
new_file.close()
```

파일 (file)

- with문을 활용하여 파일 쓰기
 - With 블록을 벗어나는 순간 파일이 자동으로 closed 됨
 - as 변수: open으로 연 파일 객체를 변수로 할당 받음

```
with open(file_path, mode= ) as 변수 :  
    수행할 문장 1  
    수행할 문장 2
```

```
1 file_path_2 = '/content/drive/My Drive/Lectures/LG_Electronics/python/day2/practice/new2.txt'  
2 with open(file_path_2, 'w') as f :  
3     data = '이렇게도 작성이 가능합니다. close를 따로 안해도 돼요.'  
4     f.write(data)
```

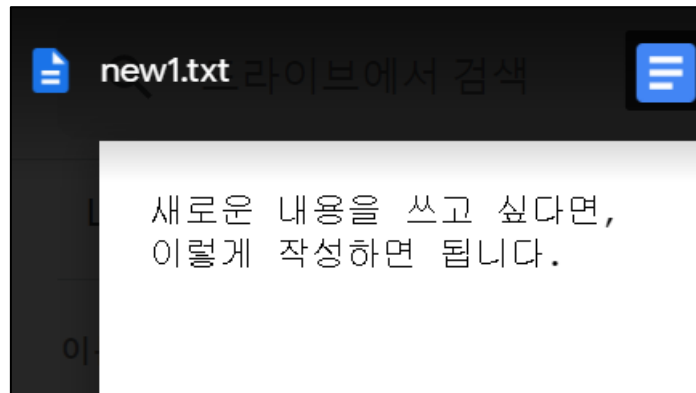
파일 (file)

- 파일 읽기 → read / readline / readlines

함수	설명
read	파일 전체 를 한번에 읽어서 string 으로 반환
readline	파일을 한 줄만 읽어서 string 으로 반환
readlines	파일을 줄단위 로 모두 읽어서 list 로 반환

파일 (file)

- read를 이용하여 파일 읽기
 - 파일 전체를 스트링으로 받음



```
with open(file_path_1, 'r') as f:  
    lines = f.read()  
    print(lines)  
    print(type(lines))
```

새로운 내용을 쓰고 싶다면,
이렇게 작성하면 됩니다.
<class 'str'>

파일 (file)

- readline를 이용하여 파일 읽기
 - 파일에서 한 줄만 string으로 불러옴

```
with open(file_path_1, 'r') as f:  
    line = f.readline()  
    print(line)  
    print(type(line))
```

새로운 내용을 쓰고 싶다면,

```
<class 'str'>
```

- While문을 적용하여 전체파일을 불러오기 가능

```
with open(file_path_1, 'r') as f:  
    while True :  
        line = f.readline()  
        if not line : break  
        print(line)
```

새로운 내용을 쓰고 싶다면,

이렇게 작성하면 됩니다.

파일 (file)

- readlines를 이용하여 파일 읽기
 - 파일에서 줄단위로 읽어서 list로 할당
 - 개행문자(\n)이 남아 있음

```
with open(file_path_1, 'r') as f:  
    lines = f.readlines()  
    print(lines)  
    print(type(lines))
```

['새로운 내용을 쓰고 싶다면,\n', '이렇게 작성하면 됩니다.']
<class 'list'>

Quiz 1

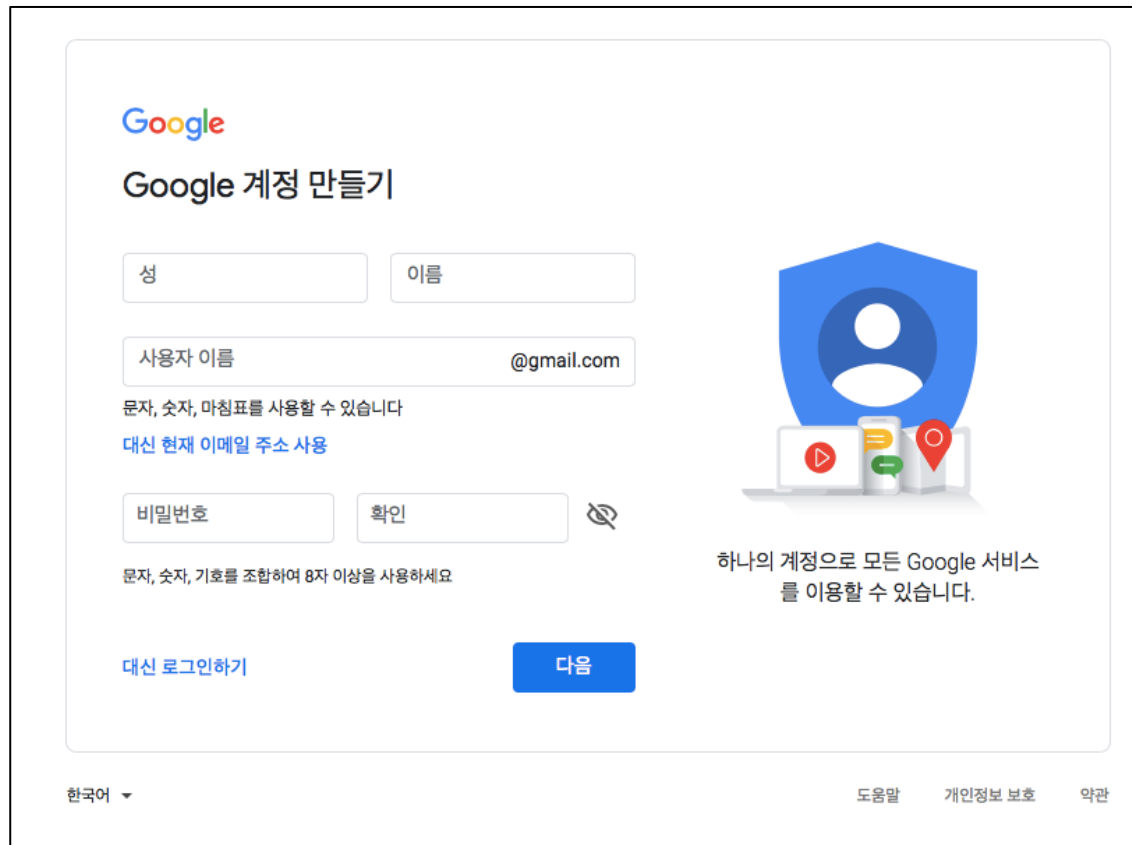
- Q1, Q2, Q3
 - 파일을 읽고 정보를 처리하기

뷔	50	70	91
정국	80	90	100
지민	83	92	86
슈가	95	100	97
지민	92	72	73
RM	77	100	87
제이홉	66	81	87

2. Class

클래스 (class)

- 클래스(class)
 - 계정 만들기



The image shows the Google Account creation interface. It features the Google logo at the top left, followed by the title 'Google 계정 만들기'. The form includes input fields for '성' (Last Name), '이름' (First Name), '사용자 이름' (Username), and '비밀번호' (Password). The '사용자 이름' field is pre-filled with '@gmail.com'. Below the username field, there is a note about using letters, numbers, and hyphens, and a link to '대신 현재 이메일 주소 사용' (Use current email address instead). The password field has a '확인' (Confirm) sub-field and a toggle for visibility. A note below the password field states that passwords must be at least 8 characters long, including letters, numbers, and symbols. To the right of the form is a blue shield icon with a white person silhouette, and below it, icons for YouTube, Gmail, and Google Maps. Text to the right of the shield says '하나의 계정으로 모든 Google 서비스를 이용할 수 있습니다.' (You can use all Google services with one account). At the bottom of the form is a blue '다음' (Next) button and a link for '대신 로그인하기' (Sign in instead). The footer contains a language dropdown set to '한국어', and links for '도움말' (Help), '개인정보 보호' (Privacy), and '약관' (Terms).

Google

Google 계정 만들기

성 이름

사용자 이름 @gmail.com

문자, 숫자, 마침표를 사용할 수 있습니다
[대신 현재 이메일 주소 사용](#)

비밀번호 확인

문자, 숫자, 기호를 조합하여 8자 이상을 사용하세요

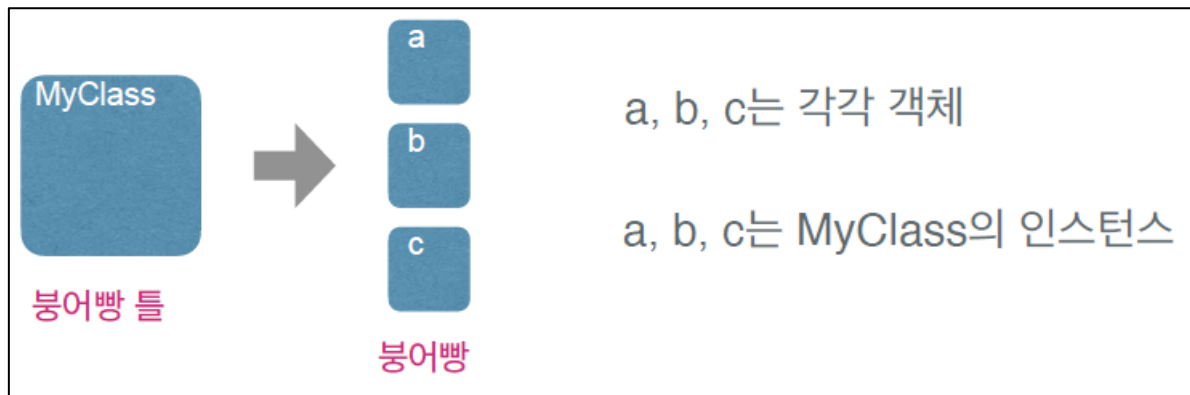
[대신 로그인하기](#) 다음

하나의 계정으로 모든 Google 서비스를 이용할 수 있습니다.

한국어 ▼ 도움말 개인정보 보호 약관

클래스 (class)

- 클래스(class)
 - 객체(Object)
 - 객체 = 속성(attribute) + 기능(method)
 - 객체 = 변수 + 함수
 - 클래스(Class) : 객체를 만드는 구조 / 틀
 - 인스턴스(Instance) : 클래스가 실질적으로 객체를 만들었을 때 그 객체를 부르는 용어



클래스 (class)

- 클래스(class)

- **class** 클래스 이름() :

- 클래스 변수,
메소드 인스턴스 변수,
메소드 등 클래스의 구조와 내용

```
class MyClass() :  
    class_var = '클래스 변수'  
  
    @classmethod  
    def class_method(cls):  
        print('클래스의 메소드')  
  
    def instance_method(self):  
        self.instance_var = '인스턴스 변수'  
        print("인스턴스의 메소드")
```

클래스 (class)

- 인스턴스(instance)
 - 클래스가 실질적으로 객체를 만들었을 때 그 객체를 부르는 용어
 - 클래스 선언
 - 클래스 호출을 통한 객체 할당

```
class MyClass():  
    pass
```

```
a = MyClass()  
b = MyClass()  
c = MyClass()
```

```
print(a,b,c)
```

```
<__main__.MyClass object at 0x7ff058193eb8> <__main__.MyClass object at 0x7ff058193e80> <__main__.MyClass object at 0x7ff058193ef0>
```

클래스 (class)

- 인스턴스(instance)

- 클래스가 실질적으로 객체를 만들었을 때 그 객체를 부르는 용어
 - instance variable(인스턴스 변수)
 - 인스턴스 고유의 변수
 - instance method(인스턴스 메소드)
 - 인스턴스에 적용되는 함수

```
class Account():  
    def make_account(self):  
        self.balance = 0  
  
    def deposit(self, money):  
        self.balance += money  
  
    def draw(self, money):  
        self.balance -= money
```

self: 인스턴스를 위한 placeholder
→ 객체 자신을 가리킴

```
a1 = Account()  
a1.make_account()  
a1.deposit(1000)  
print(a1.balance)
```

1000

인스턴스

클래스 (class)

- 인스턴스 메소드(instance method)
 - 인스턴스 메소드의 매개변수 중 가장 첫번째는 인스턴스 의 자리
 - 파이썬은 인스턴스를 자동으로 넘김

```
class TestClass2():  
    def __init__(in1, in2):  
        self.v1 = in1  
        self.v2 = in2
```

```
t_cls2 = TestClass2(10, 20)
```

```
-----  
TypeError                                Traceback (most recent call last)  
<ipython-input-5-32e19ab7347c> in <module>()  
----> 1 t_cls2 = TestClass2(10, 20)  
  
TypeError: __init__() takes 2 positional arguments but 3 were given
```

클래스 (class)

- 인스턴스 메소드(instance method)
 - 인스턴스 메소드의 매개변수 중 가장 첫번째는 인스턴스 의 자리
 - 파이썬은 인스턴스를 자동으로 넘김

```
class TestClass1():  
    def __init__(self, in1, in2):  
        self.v1 = in1  
        self.v2 = in2
```

```
t_cls1 = TestClass1(10, 20)
```

```
t_cls1.v1
```

```
10
```


클래스 (class)

- 인스턴스 메소드(instance method)
 - `__init__` (생성자)
 - 인스턴스가 생성될 때, 객체의 초기값을 설정 하는 메소드로 자동으로 호출됨
 - `__del__` (소멸자)
 - 인스턴스가 소멸될 때, 자동으로 호출되는 메소드

```
class TestClass1():  
    def __init__(self, in1, in2):  
        self.v1 = in1  
        self.v2 = in2
```

```
t_cls1 = TestClass1(10, 20)
```

```
t_cls1.v1
```

```
10
```

클래스 (class)

- 인스턴스(instance)
 - 인스턴스 변수와 인스턴스 메소드 예시

```
class Account():  
    def make_account(self) :  
        self.balance = 0  
  
    def deposit(self, money) :  
        self.balance += money  
  
    def draw(self, money) :  
        self.balance -= money
```

클래스 (class)

- 인스턴스(instance)
 - 인스턴스 변수와 인스턴스 메소드 예시

```
a1 = Account()  
a1.make_account()  
a1.deposit(1000)  
print(a1.balance)
```

1000



통장1에 1000원을 넣음

```
a2 = Account()  
a2.make_account()  
a2.deposit(5000)  
print(a2.balance)
```

5000



통장2에 5000원을 넣음

클래스 (class)

- 인스턴스(instance)
 - 인스턴스 변수와 인스턴스 메소드 예시

```
class Account():  
    def make_account(self) :  
        self.balance = 0  
    def deposit(self, money) :  
        self.balance += money  
    def draw(self, money) :  
        self.balance -= money
```

인스턴스 변수

Account.balance

```
-----  
AttributeError                                Traceback (most recent call last)  
<ipython-input-4-35da668b8893> in <module>()  
----> 1 Account.balance  
  
AttributeError: type object 'Account' has no attribute 'balance'
```

클래스 (class)

- Class Variable과 Class Method
 - class variable(클래스 변수)
 - 클래스와 인스턴스 전체가 공유하는 변수
 - class method(클래스 메소드)
 - 클래스와 인스턴스 전체가 공유하는 함수
 - 인스턴스와 상관없이 사용(인스턴스가 없어도 사용 가능) 클래스와 인스턴스 전체에 공유됨

```
class MyClass2() :  
    class_var = '클래스 변수'  
  
    @classmethod  
    def class_method(cls):  
        print('클래스의 메소드')
```

- @classmethod: 데코레이터(decorator) – wrapping을 통해 특정 코드를 재사용할 수 있게 해주는 역할
- cls: 클래스를 위한 placeholder (self와 유사한 역할)

클래스 (class)

- Class Variable과 Class Method
 - 인스턴스 → 인스턴스.변수 / 인스턴스.메소드
 - 클래스 → 클래스.변수 / 클래스.메소드

```
e = MyClass2()  
print(e.class_var)  
print(MyClass2.class_var)
```

클래스 변수
클래스 변수

```
e.class_method()  
MyClass2.class_method()
```

클래스의 메소드
클래스의 메소드

클래스 (class)

- Class Variable과 Class Method

```
class Account2() :  
    bank = '파이썬은행'  
    total = 0  
  
    @classmethod  
    def merge(cls, acc1, acc2) :  
        cls.total = acc1.balance + acc2.balance  
        print("당신의 재산은 %d" %cls.total)  
  
    def __init__(self) :  
        self.balance = 0  
  
    def deposit(self, money) :  
        self.balance += money  
  
    def draw(self, money) :  
        self.balance -= money
```

Account2.bank

'파이썬은행'

클래스 (class)

- Class Variable과 Class Method

```
class Account2() :  
    bank = '파이썬은행'  
    total = 0  
  
    @classmethod  
    def merge(cls, acc1, acc2) :  
        cls.total = acc1.balance + acc2.balance  
        print("당신의 재산은 %d" %cls.total)  
  
    def __init__(self) :  
        self.balance = 0  
  
    def deposit(self, money) :  
        self.balance += money  
  
    def draw(self, money) :  
        self.balance -= money
```

Account2.bank

'파이썬은행'

```
b1 = Account2()  
b1.deposit(4000)  
print(b1.balance)  
print(b1.bank)
```

4000
파이썬은행

```
b2 = Account2()  
b2.deposit(7000)  
print(b2.balance)  
print(b2.bank)
```

7000
파이썬은행

```
Account2.merge(b1, b2)
```

당신의 재산은 11000

Account2.total

11000

클래스 (class)

- 호텔 예제

```
class Hotel() :  
    def __init__(self) :  
        self.room = []  
  
    def add_person(self, name):  
        self.room.append(name)
```

```
r1 = Hotel()  
r2 = Hotel()  
  
r1.add_person('제이홉')  
r2.add_person('지민')
```

```
r1.room
```

```
['제이홉']
```

```
r2.room
```

```
['지민']
```

```
Hotel.room
```

```
-----  
AttributeError          Traceback (most recent call last)  
<ipython-input-17-b87892c63c8a> in <module>()  
----> 1 Hotel.room  
  
AttributeError: type object 'Hotel' has no attribute 'room'
```

클래스 (class)

- 호텔 예제

```
class GuestHouse() :  
  
    guest = []  
  
    def __init__(self) :  
        self.room = []  
  
    @classmethod  
    def check_in(cls, name):  
        cls.guest.append(name)  
  
    def add_person(self, name):  
        self.check_in(name)  
        self.room.append(name)
```

```
r3 = GuestHouse()  
r4 = GuestHouse()  
  
r3.check_in('슈가')  
r4.check_in('진')
```

```
GuestHouse.guest
```

```
['슈가', '진']
```

```
r3.room
```

```
[]
```

```
r4.room
```

```
[]
```

```
r3.add_person('RM')  
r4.add_person('뷔')
```

```
r3.room
```

```
['RM']
```

```
r4.room
```

```
['뷔']
```

```
GuestHouse.guest
```

```
['슈가', '진', 'RM', '뷔']
```

Practice 1

- Q1 스마트폰 클래스 작성

```
phone = MyPhone('iphone', 'red')
```

```
print(phone.model)  
print(phone.color)
```

```
iphone  
red
```

```
phone.set_name('shseo')
```

```
사용자의 이름은 : shseo
```

```
phone.set_number('010-xxxx-xxxx')
```

```
print(phone.number)
```

```
010-xxxx-xxxx
```

클래스 (class)

- 던더 메소드 (Double UNDER method)
 - <https://docs.python.org/3/reference/datamodel.html#special-method-names>
 - 미리 정의되어 있는 특별한 이름을 가진 메소드를 재정의 함으로써 파이썬 인터프리터 데이터 객체를 만들거나, 표현하거나, 연산하도록 함

범주	메소드 명
문자열/바이트표현	<code>__repr__</code> , <code>__str__</code> , <code>__format__</code> , <code>__bytes__</code>
숫자로 변환	<code>__abs__</code> , <code>__bool__</code> , <code>__complex__</code> , <code>__int__</code> , <code>__float__</code> , <code>__hash__</code> , <code>__index__</code>
컬렉션 에뮬레이션	<code>__len__</code> , <code>__getitem__</code> , <code>__setitem__</code> , <code>__delitem__</code> , <code>__contains__</code>
반복	<code>__iter__</code> , <code>__reserved__</code> , <code>__next__</code>
콜러블 에뮬레이션	<code>__call__</code>
컨텍스트 관리	<code>__enter__</code> , <code>__exit__</code>
객체 생성 및 소멸	<code>__new__</code> , <code>__init__</code> , <code>__del__</code>
속성 관리	<code>__getattr__</code> , <code>__getattribute__</code> , <code>__setattr__</code> , <code>__delattr__</code> , <code>__dir__</code>
속성 디스크립터	<code>__get__</code> , <code>__set__</code> , <code>__delete__</code>
클래스 서비스	<code>__prepare__</code> , <code>__instancecheck__</code> , <code>__subclasscheck__</code>
단항 수치 연산자	<code>__neg__</code> , <code>__pos__</code> , <code>__abs__</code> , <code>abs()</code>
항상된 비교 연산자	<code>__lt__</code> , <code>__le__</code> , <code>__eq__</code> , <code>__neq__</code> , <code>__gt__</code> , <code>__ge__</code> , <code>__eq__</code> , <code>__neq__</code>
산술 연산자	<code>__add__</code> , <code>__sub__</code> , <code>__mul__</code> , <code>__truediv__</code> , <code>__floordiv__</code> , <code>__mod__</code>
역순 산술 연산자	<code>__radd__</code> , <code>__rsub__</code> , <code>__rmul__</code> , <code>__rtruediv__</code> , <code>__rfloordiv__</code> , <code>__rmod__</code> , <code>__rpow__</code>
복합 할당 산술 연산자	<code>__iadd__</code> , <code>__isub__</code> , <code>__imul__</code> , <code>__itruediv__</code> , <code>__ifloordiv__</code> , <code>__imod__</code> , <code>__ipow__</code>
비트 연산자	<code>__invert__</code> , <code>__lshift__</code> , <code>__rshift__</code> , <code>__and__</code> , <code>__or__</code> , <code>__xor__</code> , <code>__xor__</code>
역순 비트 연산자	<code>__rlshift__</code> , <code>__rrshift__</code> , <code>__rand__</code> , <code>__rxor__</code> , <code>__ror__</code>
복합 할당 비트 연산자	<code>__ilshift__</code> , <code>__irshift__</code> , <code>__iand__</code> , <code>__ixor__</code> , <code>__ior__</code>

클래스 (class)

- 던더 메소드 (Double UNDER method)
 - 비교 연산자 예시

```
1 class Food():
2     def __init__(self, name, price):
3         self.name = name
4         self.price = price
5
6 lunch = Food('냉면', 8000)
7 dinner = Food('갈비', 12000)
8
9 print(lunch)
10 print(dinner)
11 print(lunch.price < dinner.price)
12 print(lunch < dinner)
```

<__main__.Food object at 0x7f72eb9787d0>
<__main__.Food object at 0x7f72eb978a50>
True

TypeError Traceback (most recent call last)
<ipython-input-27-f43adaa943ed> in <module>()
 10 print(dinner)
 11 print(lunch.price < dinner.price)
--> 12 print(lunch < dinner)

TypeError: '<' not supported between instances of 'Food' and 'Food'

SEARCH STACK OVERFLOW

클래스 (class)

- 던더 메소드 (Double UNDER method)
 - 비교 연산자 예시

```
1 class Food():
2     def __init__(self, name, price):
3         self.name = name
4         self.price = price
5
6     def __lt__(self, other):
7         if self.price < other.price:
8             return True
9         else:
10            return False
11
12 lunch = Food('냉면', 8000)
13 dinner = Food('갈비', 12000)
14 night = Food('보쌈', 36000)
15
16 print(lunch)
17 print(dinner)
18 print(night)
19 print(lunch < dinner)
20 print(night < dinner)
```

<__main__.Food object at 0x7f72eb969350>
<__main__.Food object at 0x7f72eb969810>
<__main__.Food object at 0x7f72eb969590>
True
False

클래스 (class)

- 던더 메소드 (Double UNDER method)
 - Tensorflow Dataloader 예시

```
1 import numpy as np
2 import math
3 from tensorflow.keras.utils import Sequence
4
5 class Dataloader(Sequence):
6
7     def __init__(self, x_set, y_set, batch_size, shuffle=False):
8         self.x, self.y = x_set, y_set
9         self.batch_size = batch_size
10        self.shuffle=shuffle
11        self.on_epoch_end()
12
13    def __len__(self):
14        return math.ceil(len(self.x) / self.batch_size)
15
16    def __getitem__(self, idx):
17        # sampler
18        indices = self.indices[idx*self.batch_size:(idx+1)*self.batch_size]
19
20        batch_x = [self.x[i] for i in indices]
21        batch_y = [self.y[i] for i in indices]
22
23        return np.array(batch_x), np.array(batch_y)
24
25    def on_epoch_end(self):
26        self.indices = np.arange(len(self.x))
27        if self.shuffle == True:
28            np.random.shuffle(self.indices)
```

```
31 train_loader = Dataloader(x, y, 128, shuffle=True)
32 valid_loader = Dataloader(x, y, 128)
33 test_loader = Dataloader(x, y, 128)
34
35 # 학습방법 1
36 model.fit(train_loader, validation_data=valid_loader, epochs=10,
37           workers=4)
38 model.evaluate(test_loader)
39
40 # 학습방법 2
41 for e in range(epochs):
42     for x, y in train_loader:
43         train_step(x, y)
44     for x, y in valid_loader:
45         valid_step(x, y)
```

클래스 (class)

- 클래스 상속(inheritance)

- 상속: 부모의 클래스를 물려받음
→ 부모 클래스가 가지고 있는 함수/변수를 그대로 사용할 수 있음
- 상속의 장점
→ 기존 클래스를 변형하지 않고 추가/변경 가능

```
class MyPhone2(MyPhone):  
    def has_case(self, val=False) :  
        self.case = val
```

```
p2 = MyPhone2('iphone', 'red')
```

```
p2.set_name("shseo")
```

사용자의 이름은 : shseo

```
p2.has_case(True)
```

```
p2.case
```

True

클래스 (class)

- 메소드 오버라이딩(overriding)

- 상속을 받은 자식 클래스가 상속해준 부모 클래스의 메소드를 변형하는 방법
- 원본(부모 클래스)의 소스코드는 그대로 유지한채 소스코드를 확장, 개인화 할 수 있다는 장점이 있음

```
p1 = MyPhone('iphone', 'red')  
p1.set_number("010-xxxx-xxxx")
```

```
class MyPhone3(MyPhone) :  
    def set_number(self, num) :  
        self.number = num  
        print("이 핸드폰의 번호는 : %s" %self.number)
```

```
p3 = MyPhone3('iphone', 'red')  
p3.set_number("010-xxxx-xxxx")
```

이 핸드폰의 번호는 : 010-xxxx-xxxx

Practice 2

- Q1 Human 클래스 작성

```
shseo = Human(20201023, 'M', '한국')
```

```
shseo.birth_date
```

20201023

```
shseo.give_name("상현")
```

이름은 상현 입니다.

```
shseo.can_sing(True)
```

Sing a Song

Practice 3

- Q2 Human 클래스를 상속받은 Child 클래스 작성

```
ch = Child(20301023, 'F', '한국', 'black')
```

```
ch.nation
```

```
'한국'
```

```
ch.can_sing()
```

```
Can't Sing
```

```
ch.can_dance()
```

```
Dance Time!
```

클래스 (class)

- super()
 - super()를 사용하면 자식클래스 내에서 코드에서도 부모클래스를 호출 가능

```
class ChildSuper(Human) :  
    def __init__(self, bd, s, n, eye) :  
        super().__init__(bd, s, n)  
        self.eye = eye  
    def can_sing(self):  
        print("Can't Sing")  
    def can_dance(self) :  
        print("Dance Time!")
```

```
ch2 = ChildSuper(200219, 'M', '미국', 'blue')
```

```
ch2.can_sing()
```

```
Can't Sing
```

```
ch2.eye
```

```
'blue'
```

```
print(ch2.birth_date)  
print(ch2.sex)  
print(ch2.nation)
```

```
200219
```

```
M
```

```
미국
```

클래스 (class)

- Quiz! 다음의 클래스는 어떠한 구조로 이루어져 있는가?
 - fit(), save() 메소드?

```
class ResNet(tf.keras.Model):  
  
    def __init__(self):  
        super(ResNet, self).__init__()  
        self.block_1 = ResNetBlock()  
        self.block_2 = ResNetBlock()  
        self.global_pool = layers.GlobalAveragePooling2D()  
        self.classifier = Dense(num_classes)  
  
    def call(self, inputs):  
        x = self.block_1(inputs)  
        x = self.block_2(x)  
        x = self.global_pool(x)  
        return self.classifier(x)  
  
resnet = ResNet()  
dataset = ...  
resnet.fit(dataset, epochs=10)  
resnet.save(filepath)
```

3. Exception Handling

예외처리 (exception handling)

- 에러 발생
 - 프로그램 작성 및 사용시 예기치 못한 상황으로 다양한 에러 발생
 - <https://docs.python.org/ko/3/library/exceptions.html>

```
123 / 0

-----
ZeroDivisionError      Traceback (most recent call last)
<ipython-input-8-76e1a9ab9410> in <module>()
----> 1 123 / 0

ZeroDivisionError: division by zero

lst = [1, 2, 3]
lst[3]

-----
IndexError             Traceback (most recent call last)
<ipython-input-5-fd1a72e9bd9a> in <module>()
      1 lst = [1, 2, 3]
----> 2 lst[3]

IndexError: list index out of range

open('./wrong_path/error.txt', 'r')

-----
FileNotFoundError      Traceback (most recent call last)
<ipython-input-4-2da26f012bfc> in <module>()
----> 1 open('./wrong_path/error.txt', 'r')

FileNotFoundError: [Errno 2] No such file or directory: './wrong_path/error.txt'
```

예외처리 (exception handling)

- try-except 구문

- try 블록 내의 어느 문장에서 에러가 발생하면, except 문으로 이동하고 예외 처리
- finally 문은 try나 except와는 독립적으로 항상 마지막에 실행

```
try:  
    예외를 유발할 수 있는 구문  
except <예외 종류>:  
    예외 처리를 수행하는 구문
```

```
try:  
    예외를 유발할 수 있는 구문  
except <예외 종류>:  
    예외 처리를 수행하는 구문  
finally:  
    마지막에 항상 수행되는 구문
```


예외처리 (exception handling)

- try-except 구문
 - 예외처리 및 에러 정보 표현

```
try:  
    print(123 / 0)  
except ZeroDivisionError:  
    print('제수는 0이 될 수 없습니다.')
```

제수는 0이 될 수 없습니다.

```
lst = [1, 2, 3]  
try:  
    print(lst[3])  
except IndexError:  
    print('제수는 0이 될 수 없습니다.')
```

제수는 0이 될 수 없습니다.

```
try:  
    print(123 / 0)  
except ZeroDivisionError as e:  
    print('에러정보: ', e)  
    print('제수는 0이 될 수 없습니다.')
```

에러정보: division by zero
제수는 0이 될 수 없습니다.

예외처리 (exception handling)

- try-except 구문
 - 예외처리 할 에러 선택 주의

```
try:
    print(open('./wrong_path/error.txt', 'r'))
except ZeroDivisionError:
    print('해당 경로에 파일이 존재하지 않습니다.')
```

FileNotFoundError Traceback (most recent call last)

<ipython-input-13-c52a4eac1a77> in <module>()

```
1 try:
----> 2     print(open('./wrong_path/error.txt', 'r'))
      3 except ZeroDivisionError:
      4     print('해당 경로에 파일이 존재하지 않습니다.')
```

FileNotFoundError: [Errno 2] No such file or directory: './wrong_path/error.txt'

```
try:
    print(open('./wrong_path/error.txt', 'r'))
except FileNotFoundError:
    print('해당 경로에 파일이 존재하지 않습니다.')
```

해당 경로에 파일이 존재하지 않습니다.

예외처리 (exception handling)

- try-except 구문
 - 여러 예외를 동일하기 처리 (1)
 - 함수를 사용시 반환값 유의

```
def get(key, dataset):  
    try:  
        value = dataset[key]  
    except IndexError as error:  
        print('index가 잘못된 예외')  
        return error  
    except KeyError as key:  
        print('key 가 잘못된 예외')  
        return key  
    else:  
        print('정상처리, value: ', value)  
        return value  
  
print('1. ', end='\t')  
print('반환값: ', get(key=3, dataset=(1, 2, 3)))  
print('2. ', end='\t')  
print('반환값: ', get(key=1, dataset=(1, 2, 3)))  
print('3. ', end='\t')  
print('반환값: ', get(key='age', dataset={'name': 'shseo'}))  
print('4. ', end='\t')  
print('반환값: ', get(key='name', dataset={'name': 'shseo'}))
```

```
1.      index가 잘못된 예외  
반환값: tuple index out of range  
2.      정상처리, value:  2  
반환값: 2  
3.      key 가 잘못된 예외  
반환값: 'age'  
4.      정상처리, value:  shseo  
반환값: shseo
```

예외처리 (exception handling)

- try-except 구문
 - 여러 예외를 동일하기 처리 (2)

```
def get(key, dataset):  
    try:  
        value = dataset[key]  
    except (IndexError, KeyError) as error_message:  
        print('index 또는 key가 잘못된 예외')  
        return error_message  
    else:  
        print('정상처리, value: ', value)  
        return value  
  
print('1. ', end='\t')  
print('반환값: ', get(key=3, dataset=(1, 2, 3)))  
print('2. ', end='\t')  
print('반환값: ', get(key=1, dataset=(1, 2, 3)))  
print('3. ', end='\t')  
print('반환값: ', get(key='age', dataset={'name': 'shseo'}))  
print('4. ', end='\t')  
print('반환값: ', get(key='name', dataset={'name': 'shseo'}))
```

```
1.      index 또는 key가 잘못된 예외  
반환값: tuple index out of range  
2.      정상처리, value:  2  
반환값: 2  
3.      index 또는 key가 잘못된 예외  
반환값: 'age'  
4.      정상처리, value:  shseo  
반환값: shseo
```

예외처리 (exception handling)

- 예외 계층 구조
 - <https://docs.python.org/ko/3/library/exceptions.html>
 - 상위 범주의 예외처리를 통한 에러 해결

```
BaseException
+-- SystemExit
+-- KeyboardInterrupt
+-- GeneratorExit
+-- Exception
    +-- StopIteration
    +-- StandardError
        |
        +-- BufferError
        +-- ArithmeticError
            |
            +-- FloatingPointError
            +-- OverflowError
            +-- ZeroDivisionError
        +-- AssertionError
        +-- AttributeError
        +-- EnvironmentError
            |
            +-- IOError
            +-- OSError
                |
                +-- WindowsError (Windows)
                +-- VMSError (VMS)
        +-- EOFError
        +-- ImportError
        +-- LookupError
            |
            +-- IndexError
            +-- KeyError
        +-- MemoryError
        +-- NameError
            |
            +-- UnboundLocalError
        +-- ReferenceError
        +-- RuntimeError
            |
            +-- NotImplementedError
        +-- SyntaxError
            |
            +-- IndentationError
            +-- TabError
        +-- SystemError
        +-- TypeError
        +-- ValueError
            |
            +-- UnicodeError
                |
                +-- UnicodeDecodeError
                +-- UnicodeEncodeError
                +-- UnicodeTranslateError
    +-- Warning
        +-- DeprecationWarning
        +-- PendingDeprecationWarning
        +-- RuntimeWarning
        +-- SyntaxWarning
        +-- UserWarning
        +-- FutureWarning
+-- ImportWarning
+-- UnicodeWarning
+-- BytesWarning
```

예외처리 (exception handling)

- 상위 범주 예외처리
 - 하위 범주의 예외를 동시에 처리

```
+-- ImportError
+-- LookupError
|   +-- IndexError
|   +-- KeyError
+-- MemoryError
```

```
def get(key, dataset):
    try:
        value = dataset[key]
    except LookupError as error_message: # 상위 범주 예외 처리
        print('look up 에러')
        return error_message
    else:
        return value

print('1. ', end='\t')
print('반환값: ', get(key=3, dataset=(1, 2, 3)))
print('2. ', end='\t')
print('반환값: ', get(key='age', dataset={'name': 'shseo'}))
```

```
1.      look up 에러
반환값:  tuple index out of range
2.      look up 에러
반환값:  'age'
```

예외처리 (exception handling)

- 상위 범주 예외처리
 - 예외의 최상위 클래스: `BaseException` (일반적으로 잘 사용하지 않음)
 - `BaseException`에 준하는 `Exception`을 이용하면 `except` 절에서 대부분의 예외 처리 가능

+-- **Exception**

```
def get(key, dataset):  
    try:  
        value = dataset[key]  
    except Exception as error_message: # 상위 범주 예외 처리  
        print('look up 에러')  
        return error_message  
    else:  
        return value  
  
print('1. ', end='\t')  
print('반환값: ', get(key=3, dataset=(1, 2, 3)))  
print('2. ', end='\t')  
print('반환값: ', get(key='age', dataset={'name': 'shseo'}))
```

```
1.      look up 에러  
반환값: tuple index out of range  
2.      look up 에러  
반환값: 'age'
```

예외처리 (exception handling)

- Raise 문 → **raise** 예외클래스('에러메시지')
 - 오류를 이미 발견한 상황에서 예외를 발생시키기 위한 명령어
 - 프로그램이 정상적으로 실행될 수 없는 상황일 때, 예외를 발생시켜 프로그래머 또는 사용자에게 알림 가능
 - 모든 에러는 `BaseException`의 하위 클래스이므로, 기존 예외에 해당하지 않는 예외의 경우 `Exception` 클래스를 상속하는 방식을 통해 커스텀 예외 클래스를 만들 수 있음

```
class MyExceptionError(Exception):  
    """Custom Exception"""  
    pass  
  
try:  
    raise MyExceptionError('custom error message')  
except MyExceptionError as error_message:  
    print(error_message)  
  
custom error message
```


예외처리 (exception handling)

- assert 문 → **assert** 검증식, 오류메시지
 - 검증식의 상태를 검증하기 위한 명령어
 - 검증식을 계산하여 결과가 참일 때에는 이후 절차로 진행하고, 결과가 거짓일 경우 AssertionError 예외를 발생
 - 정확한 오류 정보 전달을 위해서는 assert보다 raise를 이용하는 것이 나음
 - 커스텀 예외상황을 처리하기 어려움

```
assert True          # 식이 올바르면 문제 없음
```

```
assert 1 + 1 == 2    # 식이 올바르면 문제 없음
```

```
assert 1 - 1 == 2    # 식이 거짓이면 AssertionError가 발생
```

```
-----  
AssertionError      Traceback (most recent call last)  
<ipython-input-64-b3f6c85fe4bc> in <module>()  
----> 1 assert 1 - 1 == 2    # 식이 거짓이면 AssertionError가 발생  
  
AssertionError:
```

예외처리 (exception handling)

- Door class 예제

- 문이 열린 상태인데 또 여는 행동이나 문이 닫혀 있는 상태인데 또 닫는 행위를 처리해야 함

```
class Door:
    def __init__(self):
        self.is_opened = True

    def open(self):
        print('문을 엽니다.', end='\t')
        self.is_opened = True
        print('문의 상태', self.is_opened)

    def close(self):
        print('문을 닫습니다.', end='\t')
        self.is_opened = False
        print('문의 상태', self.is_opened)
```

```
door = Door()
door.close()
door.open()
door.open()
```

```
문을 닫습니다.   문의 상태 False
문을 엽니다.    문의 상태 True
문을 엽니다.    문의 상태 True
```

예외처리 (exception handling)

- Door class 예제
 - raise 문을 통한 예외처리
 - door의 상태에 대한 예외를 표현할 수 있는 커스텀 예외 클래스 선언

```
+-- Exception
+-- DoorError
|   +-- DoorOpenedError
|   +-- DoorClosedError
```

```
class DoorError(Exception):
    pass

class DoorOpenedError(DoorError):
    pass

class DoorClosedError(DoorError):
    pass
```

예외처리 (exception handling)

- Door class 예제
 - raise 문을 통한 예외처리

```
class Door:
    def __init__(self):
        self.is_opened = True

    def open(self):
        if self.is_opened:
            raise DoorOpenedError('문이 이미 열려 있음')
        else:
            print('문을 엽니다.', end='\t')
            self.is_opened = True
            print('문의 상태', self.is_opened)

    def close(self):
        if not self.is_opened:
            raise DoorClosedError('문이 이미 닫혀 있음')
        else:
            print('문을 닫습니다.', end='\t')
            self.is_opened = False
            print('문의 상태', self.is_opened)
```

예외처리 (exception handling)

- Door class 예제
 - raise 문을 통한 예외처리 결과

```
door = Door()
door.close()
door.open()
door.open()
```

문을 닫습니다. 문의 상태 False
문을 엽니다. 문의 상태 True

```
-----
DoorOpenedError      Traceback (most recent call last)
<ipython-input-83-a87e3ed77084> in <module>()
      2 door.close()
      3 door.open()
----> 4 door.open()

<ipython-input-82-aa340fc14b4f> in open(self)
      6     def open(self):
      7         if self.is_opened:
----> 8             raise DoorOpenedError('문이 이미 열려 있음')
      9     else:
     10         print('문을 엽니다.', end='\t')
```

DoorOpenedError: 문이 이미 열려 있음

예외처리 (exception handling)

- Door class 예제
 - assert 문을 통한 예외처리

```
class Door:
    def __init__(self):
        self.is_opened = True

    def open(self):
        assert not self.is_opened

        print('문을 엽니다.', end='\n')
        self.is_opened = True
        print('문의 상태', self.is_opened)

    def close(self):
        assert self.is_opened

        print('문을 닫습니다.', end='\n')
        self.is_opened = False
        print('문의 상태', self.is_opened)
```

예외처리 (exception handling)

- Door class 예제
 - assert 문을 통한 예외처리 결과

```
door = Door()  
door.close()  
door.open()  
door.open()
```

문을 닫습니다. 문의 상태 False
문을 엽니다. 문의 상태 True

AssertionError Traceback (most recent call last)

<ipython-input-92-a87e3ed77084> in <module>()
 2 door.close()
 3 door.open()
----> 4 door.open()

 2 door.close()
 3 door.open()
----> 4 door.open()

<ipython-input-91-e91f5edd43f7> in open(self)

 4
 5 def open(self):
----> 6 assert not self.is_opened
 7
 8 print('문을 엽니다.', end='\t')

AssertionError:

Practice 4

- Q1 Account 클래스 예외처리 (raise 문 사용)

- 계좌의 잔액을 초과하는 액수가 출금되어서는 안 된다. (BalanceError)
- 동결된 계좌에서 출금되어서는 안 된다. (AccountFrozenError)
- 0 이하의 액수는 입금 또는 출금할 수 없다. (NegativeInputError)

```
class Account():
    def __init__(self, balance, is_frozen):
        self.balance = balance      # 계좌 잔액
        self.is_frozen = is_frozen # 계좌 동결 여부

    def check(self):
        if self.is_frozen:
            print('계좌가 동결된 상태입니다.')
        else:
            print('계좌는 정상입니다.')
        print('계좌 잔액은', self.balance, '원 입니다.')

    def deposit(self, amount):
        self.balance += amount
        print(f"{amount}원을 입금하였습니다")

    def withdraw(self, amount):
        self.balance -= amount
        print(f"{amount}원을 출금하였습니다")
```


Practice 4

- Q1 Account 클래스 예외처리 (raise 문 사용)

계좌의 잔액을 초과하는 액수가 출금되어서는 안 된다.

```
my_account_1 = Account(balance=0, is_frozen=False)
my_account_1.deposit(1000)
my_account_1.withdraw(2000)
my_account_1.check()
```

1000원을 입금하였습니다
2000원을 출금하였습니다
계좌는 정상입니다.
계좌 잔액은 -1000 원 입니다.

동결된 계좌에서 출금되어서는 안 된다.

```
my_account_1 = Account(balance=0, is_frozen=True)
my_account_1.deposit(1000)
my_account_1.withdraw(500)
my_account_1.check()
```

1000원을 입금하였습니다
500원을 출금하였습니다
계좌가 동결된 상태입니다.
계좌 잔액은 500 원 입니다.

0 이하의 액수는 입금 또는 출금할 수 없다.

```
my_account_1 = Account(balance=0, is_frozen=False)
my_account_1.deposit(-1000)
my_account_1.withdraw(-2000)
my_account_1.check()
```

-1000원을 입금하였습니다
-2000원을 출금하였습니다
계좌는 정상입니다.
계좌 잔액은 1000 원 입니다.

Practice 4

- Q1 Account 클래스 예외처리 (raise 문 사용)

```
class AccountError(Exception):  
    pass  
class BalanceError(AccountError):  
    pass  
class AccountFrozenError(AccountError):  
    pass  
class NegativeInputError(AccountError):  
    pass
```

정답을 작성해주세요.

```
my_account_1 = Account(balance=0, is_frozen=False)  
my_account_1.deposit(1000)  
my_account_1.withdraw(2000)  
my_account_1.check()
```

1000원을 입금하였습니다
계좌의 잔고가 부족합니다.
계좌는 정상입니다.
계좌 잔액은 1000 원 입니다.

```
my_account_1 = Account(balance=0, is_frozen=True)  
my_account_1.deposit(500)  
my_account_1.withdraw(1000)  
my_account_1.check()
```

500원을 입금하였습니다
계좌의 잔고가 부족합니다.
계좌가 동결된 상태입니다.
계좌가 동결된 상태입니다.
계좌 잔액은 500 원 입니다.

```
my_account_1 = Account(balance=0, is_frozen=False)  
my_account_1.deposit(-1000)  
my_account_1.withdraw(-2000)  
my_account_1.check()
```

입금액은 0보다 작을 수 없습니다.
입금액은 0보다 작을 수 없습니다.
계좌는 정상입니다.
계좌 잔액은 0 원 입니다.

4. Advanced Python

내장함수

- 파이썬 내장함수
 - 기본제공 함수

abs()	dict ()	help ()	min ()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	import()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

내장함수

- input

```
a = input()
```

내장함수 알아보기

```
print(a)
```

내장함수 알아보기

```
b = input()
```

352

```
print('b의 타입은 {}'.format(type(b)))
```

b의 타입은 <class 'str'>

내장함수

- len()

```
len(a)
```

9

```
len(b)
```

3

- abs()

```
c = -32
```

```
c
```

-32

```
abs(c)
```

32

내장함수

- range()

```
for x in range(3) :  
    print(x+1)
```

```
1  
2  
3
```

- max() / min()

```
d = [1, 34, 74, 22, 10, 5820, -123, -47]
```

```
max(d)
```

```
5820
```

```
min(d)
```

```
-123
```

내장함수

- Pop Quiz
- 여기는 동네 유명한 빵집이다.
사람들에게 먼저 온 순서대로 번호표를 나누어주려고 한다.
번호표를 나누어주는 함수를 작성해보자.
 - 함수는 사람이름으로 되어 있는 리스트를 받아서 "대기번호 x번 : 사람이름" 를 화면에 출력하고 (번호표, 사람이름)을 원소로 이루어진 리스트를 반환한다.

내장함수

- Pop Quiz

- 대기 번호를 트래킹하는 변수 i를 별도로 업데이트 해야 함

```
people = ['지민', '뷔', 'RM', '제이홉']
```

```
def func1(line) :  
    new_lines = []  
    i = 1  
    for x in line :  
        print("대기번호 %d번 : %s" %(i, x))  
        new_lines.append((i, x))  
        i += 1  
    return new_lines
```

```
lines = func1(people)
```

```
대기번호 1번 : 지민  
대기번호 2번 : 뷔  
대기번호 3번 : RM  
대기번호 4번 : 제이홉
```

내장함수

- enumerate()
 - 반복가능한 객체의 인덱스와 원소에 함께 접근할 수 있는 함수
 - tuple(인덱스, 원소)의 형태로 객체를 반환
 - 보통 리스트와 함께 쓰임

```
lst = ['a', 'b', 'c']  
for x in enumerate(lst):  
    print(x)
```

```
(0, 'a')  
(1, 'b')  
(2, 'c')
```

내장함수

- enumerate()

```
dic = {0 : 'p', 1 : 'b' , 2 : 'd'}
```

```
for x in enumerate(dic):  
    print(x)
```

```
(0, 0)
```

```
(1, 1)
```

```
(2, 2)
```

```
st = 'abcd'
```

```
for x in enumerate(st) :  
    print(x)
```

```
(0, 'a')
```

```
(1, 'b')
```

```
(2, 'c')
```

```
(3, 'd')
```

```
se = {'a','b','c'}
```

```
for x in enumerate(se) :  
    print(x)
```

```
(0, 'b')
```

```
(1, 'c')
```

```
(2, 'a')
```

내장함수

- 대기번호 발급문제 with enumerate()

```
def func1_with_enumerate(line) :  
    new_lines = []  
    for idx, val in enumerate(line):  
        print("대기번호" + str(idx+1) + ":" + val)  
        new_lines.append((idx+1, val))  
    return new_lines
```

```
lines = func1_with_enumerate(people)
```

대기번호1:펍수
대기번호2:뽀로로
대기번호3:뚝딱이
대기번호4:텔레토비

내장함수

- zip()

```
str_list = ['one', 'two', 'three', 'four']  
num_list = [1, 2, 3, 4]
```

```
for i in zip(num_list, str_list) :  
    print(i)
```

```
(1, 'one')  
(2, 'two')  
(3, 'three')  
(4, 'four')
```

```
s1 = '123'  
s2 = 'abc'  
s3 = 'ㄱㄴㄷ'  
list(zip(s1,s2,s3))
```

```
[('1', 'a', 'ㄱ'), ('2', 'b', 'ㄴ'), ('3', 'c', 'ㄷ')]
```

내장함수

- `lambda()` → **lambda 매개변수 : 리턴값**
 - `lambda`(람다)는 식 형태로 되어있어서 `lambda expression`(람다 표현식)이라고도 불림
 - 람다는 익명의 함수로서 함수를 간편하게 작성할 수 있게 지원
 - `python3`에서 사용이 권장 되지는 않지만 머신러닝이나 데이터 분석 시 많이 사용됨

내장함수

- lambda()

```
def plus_two(num) :  
    return num + 2
```

```
a = 2  
b = plus_two(a)  
print(b)
```

4

```
lambda x : x + 2
```

```
<function __main__.<lambda>>
```

```
func2 = lambda x : x + 2
```

```
c = func2(2)
```

c

4

내장함수

- `map()` → `map(함수, 자료형)`
 - 리스트, 튜플, 스트링 등 자료형 각각의 원소에 동일한 함수를 적용

```
items = [1, 2, 3, 4, 5]
```

```
squared = []  
for i in items :  
    squared.append(i*i)
```

```
print(squared)
```

```
[1, 4, 9, 16, 25]
```

```
squared_map = list(map(lambda x : x**2, items))
```

```
print(squared_map)
```

```
[1, 4, 9, 16, 25]
```

lambda와 map을 이용하여 items의 요소들을 string(문자)로 바꾸는 것을 짜봅시다.

```
str_items = list(map(lambda x : str(x), items))
```

```
print(str_items)
```

```
['1', '2', '3', '4', '5']
```

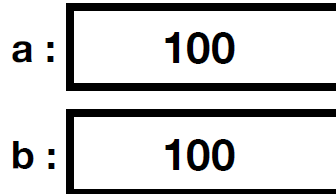

Practice 4

- Q1 함수선언 및 호출
- Q2. 길이가 정해져 있지 않은 인자 입력
- Q3. 문자열 출력
- Q4. lambda, map 함수 활용

자료형과 참조변수

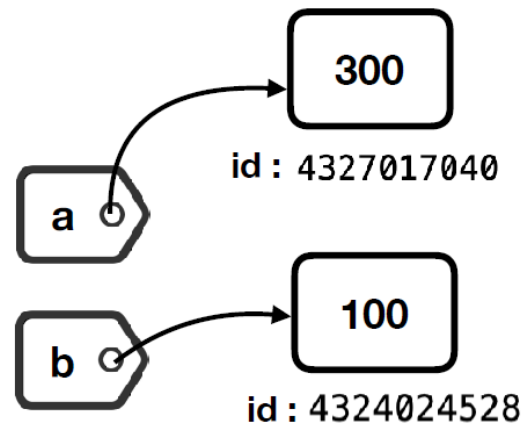
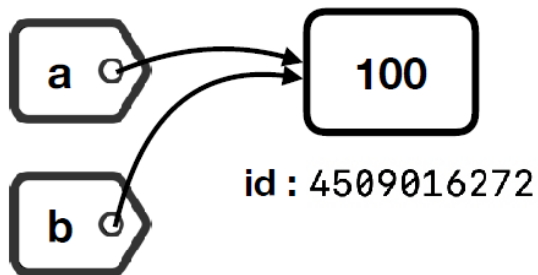
- C (변수중심)

- int a, b;
- a = 100;
- b = a;



- Python (객체중심)

- a = 100
- b = a
- a = 300



```
1 a = 100
2 b = a
3 a = 300
4 print(id(100))
5 print(id(a))
6 print(id(300))
7 print(id(b))
```

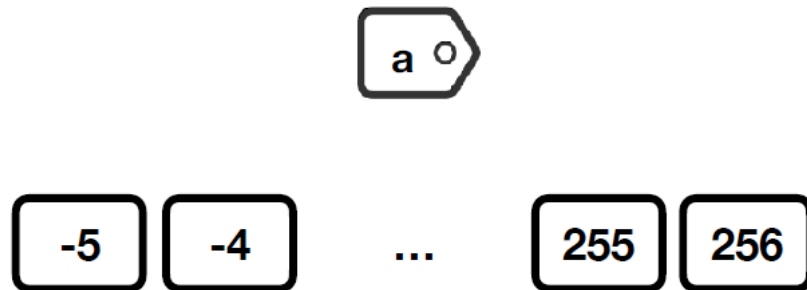
```
10917664
139922294173808
139922294174608
10917664
```

자료형과 참조변수

- 미리 생성해 둔 객체
 - -5 ~ 256의 정수객체는 미리 생성

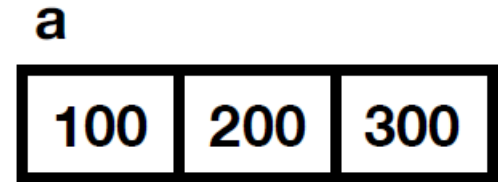
```
1 a = -100
2 b = a
3 c = -100
4
5 print(id(-100))
6 print(id(a))
7 print(id(b))
8 print(id(c))
```

```
139922294175024
139922294175472
139922294175472
139922294175216
```

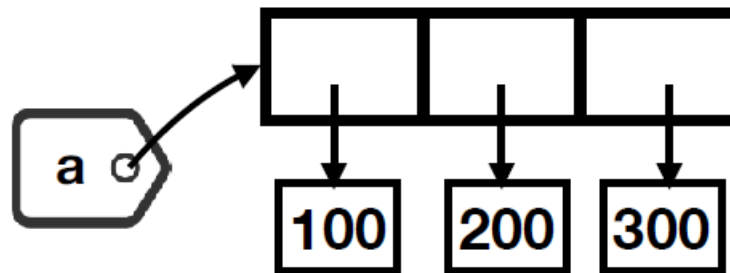


리스트의 구성요소

- C (변수중심)
 - 배열 원소의 크기는 int형으로 고정
 - `int a[3] = {100, 200, 300}`

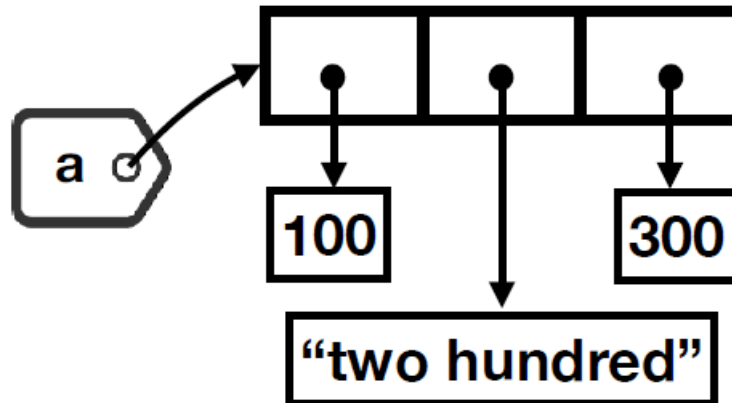


- Python (객체중심)
 - 리스트의 항목은 서로 다른 자료형도 가능
 - 리스트의 요소 `a[0]`, `a[1]`, `a[2]`는 참조형
 - `a = [100, 200, 300]`



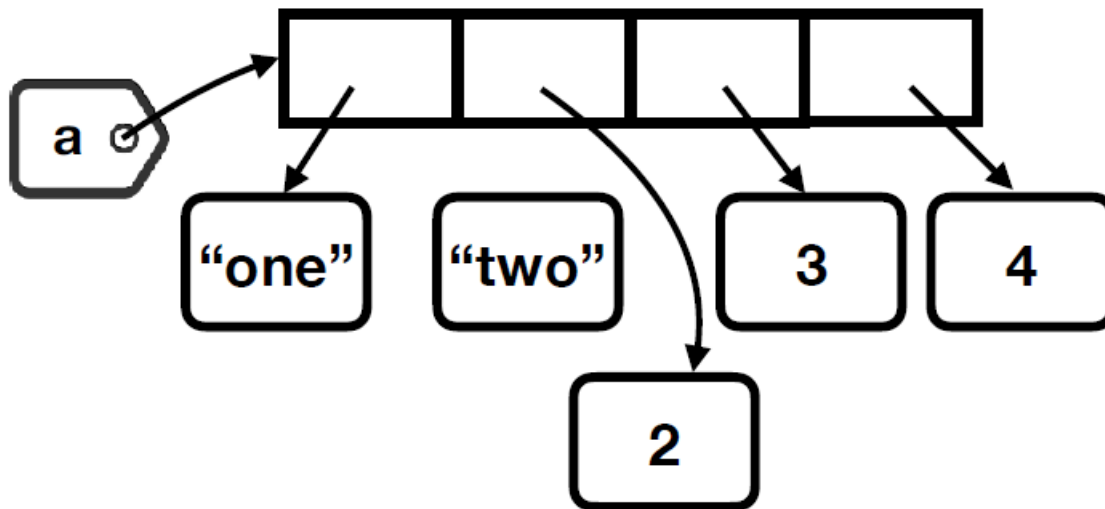
리스트의 구성요소

- Python
 - 리스트의 요소는 참조형
 - `a = [100, "two hundred", 300]`



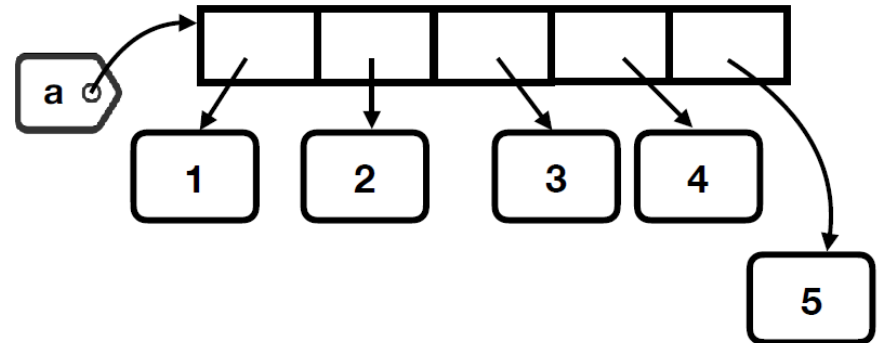
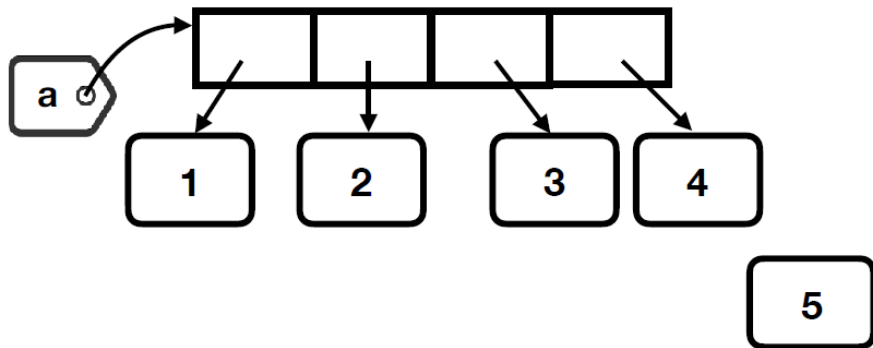
리스트의 구성요소

- 리스트 요소의 재할당
 - # 리스트 객체 생성
 - a = ["one", "two", 3, 4]
 - # 리스트 요소의 재할당
 - a[1] = 2



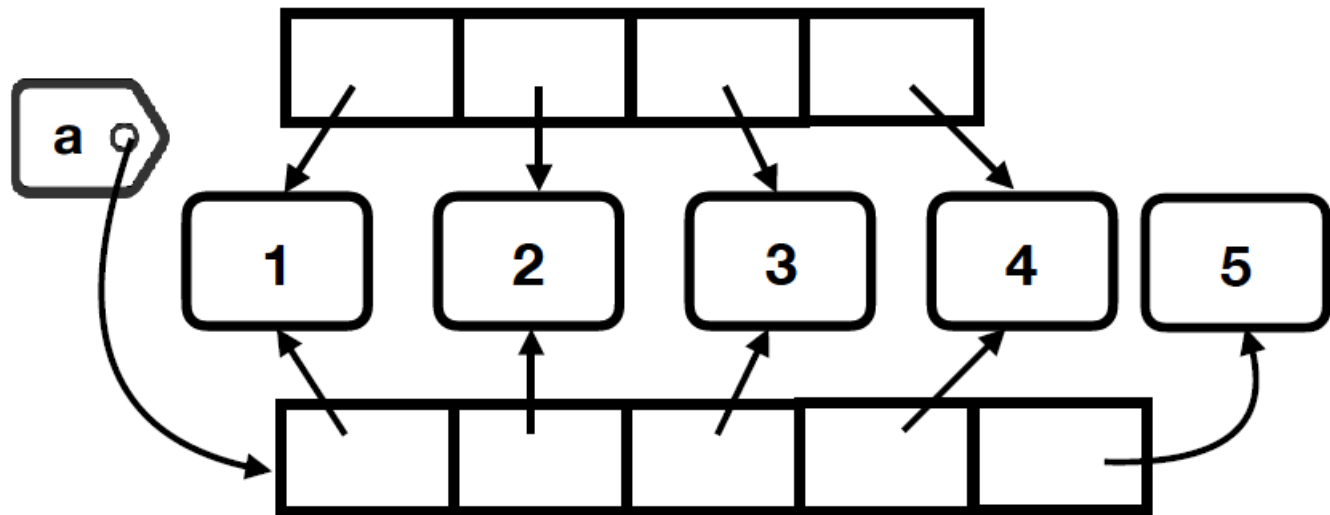
리스트의 구성요소

- 리스트 요소 추가
 - # 리스트 객체 생성
 - `a = [1, 2, 3, 4]`
 - `a.append(5)`



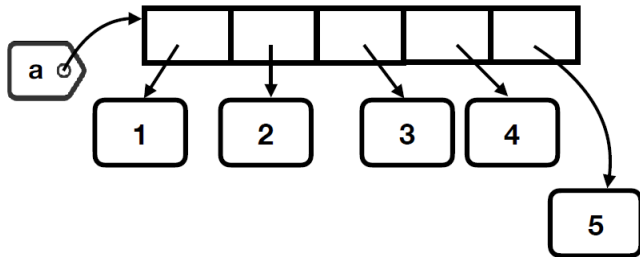
리스트의 구성요소

- 리스트의 덧셈과 재할당
 - $a = [1, 2, 3, 4]$
 - $a = a + [5]$



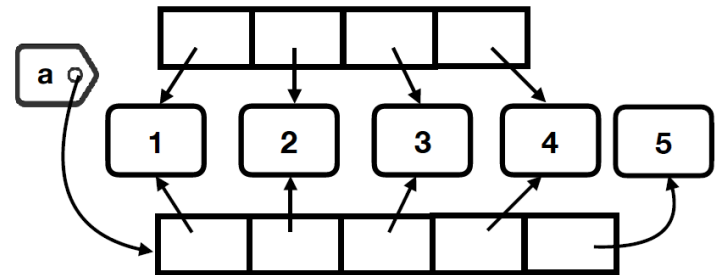
리스트의 구성요소

- append() vs 리스트의 덧셈
 - 리스트 id 비교



```
1 a = [1, 2, 3, 4]
2 print(id(a))
3 print(a)
4 a.append(5)
5 print(id(a))
6 a
```

```
139922435359944
[1, 2, 3, 4]
139922435359944
[1, 2, 3, 4, 5]
```



```
1 a = [1, 2, 3, 4]
2 print(id(a))
3 print(a)
4 a = a + [5]
5 print(id(a))
6 a
```

```
139922294081736
[1, 2, 3, 4]
139922293768840
[1, 2, 3, 4, 5]
```

List Comprehension

- List Comprehension

- 파이썬만의 독특한 문법으로 간결한 코딩을 할 수 있음
- 1) for 문 → 0부터 9까지를 순서대로 가지고 있는 리스트 생성

```
list_1 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
list_2 = []  
for x in range(10) :  
    list_2.append(x)  
print(list_2)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- list comprehension 사용

→ [저장할 값 **for** 원소 **in** 반복 가능한 객체]

```
lc_1 = [x for x in range(10)]  
print(lc_1)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

List Comprehension

- List Comprehension → for문
 - Quiz 1) 구구단 2단을 list comprehension을 이용하여 구현하고 리스트를 화면에 출력해보자.
 - Quiz 2) 다음의 문장을 분석해보자.
 - "코로나 바이러스를 예방하기 위해 사회적 거리두기를 실천합시다. 마스크를 끼고 손씻기를 생활화합시다." 라는 문장을 띄어쓰기별로 분석하려고 한다.
띄어쓰기별로 문장을 나눈 후 각 요소의 길이를 리스트에 저장하라.

List Comprehension

- 대기번호 발급 프로그램
 - list comprehension 사용

```
people = ['지민', '제이홉', '정국', '슈가']
```

```
def func1_with_lc(line) :  
    new_lines = [(idx+1, val) for idx, val in enumerate(line)]  
    return new_lines
```

```
line = func1_with_lc(people)
```

```
print(line)
```

```
[(1, '지민'), (2, '제이홉'), (3, '정국'), (4, '슈가')]
```

List Comprehension

- List Comprehension

- for문 + if 문 → [저장할 값 **for** 원소 **in** 반복 가능한 객체 **if** 조건]
- 10부터 20 사이의 숫자들 중에서 짝수만을 담은 리스트를 만들기

```
list_3 = []  
for x in range(10, 21) :  
    if x % 2 == 0 :  
        list_3.append(x)  
print(list_3)
```

```
[10, 12, 14, 16, 18, 20]
```

```
lc_2 = [x for x in range(10, 21) if x % 2 == 0]
```

```
lc_2
```

```
[10, 12, 14, 16, 18, 20]
```

List Comprehension

- List Comprehension → for문 + if문
 - Quiz 1) 1부터 10의 제곱수 중, 50 이하인 수만 리스트에 저장하라.
 - Quiz 2) 다음의 문장을 분석해보자.
 - "코로나 바이러스를 예방하기 위해 사회적 거리두기를 실천합시다. 마스크를 끼고 손씻기를 생활화합시다." 라는 문장을 띄어쓰기별로 분석하려고 한다.
띄어쓰기별로 문장을 나눈 후 각 요소의 길이가 5미만인 것들만 리스트에 저장하라.

List Comprehension

- List Comprehension

- for문 + if-else 문

- [저장할 값 if 조건 else 저장할 값 for 원소 in 반복 가능한 객체]

- 1부터 10 까지의 숫자들 중 홀수이면 제곱수를, 짝수이면 세제곱수를 담은 리스트 만들기

```
list_4 = []  
for x in range(1, 11) :  
    if x % 2 == 1 :  
        list_4.append(x ** 2)  
    else :  
        list_4.append(x ** 3)
```

```
list_4
```

```
[1, 8, 9, 64, 25, 216, 49, 512, 81, 1000]
```

```
lc_4 = [x ** 2 if x % 2 == 1 else x ** 3 for x in range(1, 11)]
```

```
lc_4
```

```
[1, 8, 9, 64, 25, 216, 49, 512, 81, 1000]
```

List Comprehension

- List Comprehension

- for문 + if-else 문

- [저장할 값 if 조건 else 저장할 값 for 원소 in 반복 가능한 객체]

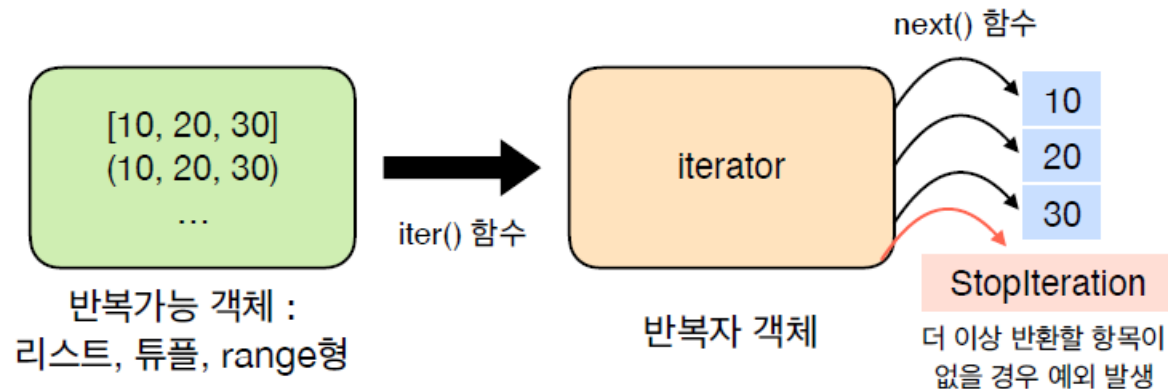
- Quiz 1) 40 이하의 숫자는 5를 더하고, 40 초과인 숫자는 41로 바꾸어 리스트로 저장하고, 리스트를 출력하라.

- Quiz 2) 컷라인이 60점일 때, 사람이름과 통과여부를 리스트로 담아 출력하라. 이름과 통과여부는 튜플로 묶어있는 자료이다.

Iterator

- 반복가능자 (Iterator)

- 하나 이상의 항목이 포함되어 있는 자료구조에서 데이터를 순차적으로 꺼내어 이용할 수 있는 객체
- 데이터를 순차적으로 활용하기 위해서는 `next()` 함수 또는 `__next__()` 특수 메소드 이용



Iterator

- 반복가능 자료형

- 리스트, 딕셔너리, 튜플, 문자열, 집합, 파일, range
- 파이썬 내장함수인 `iter()` 함수를 이용해서 반복자 객체로 변환

```
1 list_7 = [10, 20, 30]
2 list_7_iter = iter(list_7)
3 list_7_iter

<list_iterator at 0x7f423b077da0>

1 n = 100
2 n_iter = iter(n)
```

TypeError Traceback (most recent call last)
<ipython-input-2-9a510ab6d523> in <module>()
 1 n = 100
----> 2 n_iter = iter(n)

TypeError: 'int' object is not iterable

SEARCH STACK OVERFLOW

Iterator

- `next()`, `__next()`
 - 반복자 객체의 요소 추출

```
1 next(list_7_iter)
```

10

```
1 next(list_7_iter)
```

20

```
1 next(list_7_iter)
```

30

```
1 next(list_7_iter)
```

```
-----
StopIteration                                Traceback (most recent call last)
<ipython-input-7-a6097639343a> in <module>()
----> 1 next(list_7_iter)
```

StopIteration:

SEARCH STACK OVERFLOW

```
1 list_8 = [10, 20, 30]
2 list_8_iter = iter(list_8)
3 list_8_iter
```

<list_iterator at 0x7f423a7c37f0>

```
1 list_8_iter.__next__()
```

10

```
1 list_8_iter.__next__()
```

20

```
1 list_8_iter.__next__()
```

30

```
1 list_8_iter.__next__()
```

```
-----
StopIteration                                Traceback (most recent call last)
<ipython-input-15-c39869d14b41> in <module>()
----> 1 list_8_iter.__next__()
```

StopIteration:

SEARCH STACK OVERFLOW

Generator

- 제너레이터 (generator)

- 제너레이터 객체는 모든 값을 메모리에 올려두고 이용하는 것이 아니라 필요할 때마다 생성해서 반환 → 메모리 효율성 향상

- Lazy evaluation

- 반복자와 동일한 일을 하는 것처럼 보이지만 여기에서 생성된 1, 2, 3을 미리 메모리에 만들어 두는 것이 아니라 for 문에서 필요로 할 때마다 my_generator로 부터 받음

```
1 my_generator = (x**2 for x in range(5))
2 my_generator

<generator object <genexpr> at 0x7f42329dde60>

1 for n in my_generator:
2 |     print(n)

0
1
4
9
16
```

Generator

- Yield

- 일반적인 함수의 return 문과 유사하지만 yield문은 제너레이터를 반환
- 제너레이터는 한번 생성해서 반환한 객체를 보관하지 않기 때문에 이전의 코드를 실행한 후 추가한 코드를 실행하면 아무런 객체도 출력되지 않음

```
1 def create_generator(max_num):
2     for x in range(max_num):
3         yield x**2
4
5 my_generator = create_generator(5)
6 my_generator

<generator object create_generator at 0x7f42329c05c8>

1 for n in my_generator:
2     print(n)

0
1
4
9
16
```

```
1 for n in my_generator:
2     print(n)

0
1
4
9
16

1 for n in my_generator:
2     print(n)
```

Decorator

- 데커레이터(Decorator)

- 다른 함수를 인자로 받아 새롭게 수정된 함수로 대체하는 함수
- 함수를 중심으로 반복되는 코드를 리팩터링할 때 주로 사용

```
1 import datetime
2
3 def main_function_1():
4     print datetime.datetime.now()
5     print "MAIN FUNCTION 1 START"
6     print datetime.datetime.now()
7
8
9 def main_function_2():
10    print datetime.datetime.now()
11    print "MAIN FUNCTION 2 START"
12    print datetime.datetime.now()
13
14 # .... X 1000번
15
16 def main_function_1000():
17    print datetime.datetime.now()
18    print "MAIN FUNCTION 3 START"
19    print datetime.datetime.now()
```

```
1 import datetime
2
3 def datetime_decorator(func):
4     def decorated():
5         print datetime.datetime.now()
6         func()
7         print datetime.datetime.now()
8     return decorated
9
10 @datetime_decorator
11 def main_function_1():
12     print "MAIN FUNCTION 1 START"
13
14 @datetime_decorator
15 def main_function_2():
16     print "MAIN FUNCTION 2 START"
17
18 # ..... X 1000번
19
20 @datetime_decorator
21 def main_function_1000():
22     print "MAIN FUNCTION 3 START"
```

Decorator

- 데커레이터(Decorator)
 - 관리자 상태확인 예시

```
1 class Store():
2     def __init__(self):
3         self.storage = ['ice_cream', 'snack']
4
5     def get_food(self, username, food):
6         if username != 'admin':
7             raise Exception("This user is not allowed to get food")
8         return self.storage.append(food)
9
10    def put_food(self, username, food):
11        if username != 'admin':
12            raise Exception("This user is not allowed to put food")
13        self.storage.remove(food)
14
15 a = Store()
16 print(a.storage)
17
18 a.get_food("admin", "bread")
19 print(a.storage)
20
21 a.get_food("shseo", "beverage")
```

['ice_cream', 'snack']
['ice_cream', 'snack', 'bread']

Exception Traceback (most recent call last)
<ipython-input-18-4d4752c3bba5> in <module>()
19 print(a.storage)
20
--> 21 a.get_food("shseo", "beverage")

<ipython-input-18-4d4752c3bba5> in get_food(self, username, food)
5 def get_food(self, username, food):
6 if username != 'admin':
--> 7 raise Exception("This user is not allowed to get food")
8 return self.storage.append(food)
9

Exception: This user is not allowed to get food

```
1 def check_is_admin(username):
2     if username != 'admin':
3         raise Exception("This user is not allowed to get or put food")
4
5 class Store():
6     def __init__(self):
7         self.storage = ['ice_cream', 'snack']
8
9     def get_food(self, username, food):
10        check_is_admin(username)
11        return self.storage.append(food)
12
13    def put_food(self, username, food):
14        check_is_admin(username)
15        self.storage.remove(food)
16
17 a = Store()
18 print(a.storage)
19
20 a.get_food("admin", "bread")
21 print(a.storage)
22
23 a.get_food("shseo", "beverage")
```

['ice_cream', 'snack']
['ice_cream', 'snack', 'bread']

Exception Traceback (most recent call last)
<ipython-input-19-fb3b6f7cbff9> in <module>()
21 print(a.storage)
22
--> 23 a.get_food("shseo", "beverage")

1 frames
<ipython-input-19-fb3b6f7cbff9> in check_is_admin(username)
1 def check_is_admin(username):
2 if username != 'admin':
--> 3 raise Exception("This user is not allowed to get or put food")
4
5 class Store():

Exception: This user is not allowed to get or put food

Decorator

- 데커레이터(Decorator)
 - 관리자 상태확인 예시

```
1 def check_is_admin(f):
2     def wrapper(*args):
3         # print(args)
4         if args[1] != 'admin':
5             raise Exception("This user is not allowed to get or put food")
6         return f(*args)
7     return wrapper
8
9 class Store():
10     def __init__(self):
11         self.storage = ['ice_cream', 'snack']
12
13     @check_is_admin
14     def get_food(self, username, food):
15         return self.storage.append(food)
16
17     @check_is_admin
18     def put_food(self, username, food):
19         self.storage.remove(food)
20
21 a = Store()
22 print(a.storage)
23
24 a.get_food("admin", "bread")
25 print(a.storage)
26
27 a.get_food("shseo", "beverage")
```

```
['ice_cream', 'snack']
['ice_cream', 'snack', 'bread']
-----
Exception                                 Traceback (most recent call last)
<ipython-input-26-5bdfb00bd3b5> in <module>()
    25 print(a.storage)
    26
--> 27 a.get_food("shseo", "beverage")

<ipython-input-26-5bdfb00bd3b5> in wrapper(*args)
     3         # print(args)
     4         if args[1] != 'admin':
--> 5             raise Exception("This user is not allowed to get or put food")
     6         return f(*args)
     7     return wrapper

Exception: This user is not allowed to get or put food
```


Decorator

- 데커레이터(Decorator)
 - Tensorflow 데커레이터 사용 예시
 - https://www.tensorflow.org/guide/effective_tf2?hl=ko

텐서플로 2.0의 권장 사항

작은 함수로 코드를 리팩토링하세요.

텐서플로 1.x의 일반적인 사용 패턴은 "키친 싱크(kitchen sink)" 전략입니다. 먼저 모든 연산을 결합하여 준비한 다음 `session.run()` 을 사용해 선택한 텐서를 평가합니다. 텐서플로 2.0에서는 필요할 때 호출할 수 있는 작은 함수로 코드를 리팩토링(refactoring)해야 합니다. 모든 함수에 `tf.function` 데코레이터를 적용할 필요는 없습니다. 모델 훈련의 한 단계(step)나 정방향 연산(forward pass) 같은 고수준 연산에만 `tf.function` 데코레이터를 적용하세요.

케라스 층과 모델을 사용해 변수를 관리하세요.

케라스 모델과 층(layer)은 재귀적으로 의존하는 모든 변수를 수집하여 `variables` 와 `trainable_variables` 속성으로 제공합니다. 따라서 변수를 지역 범위로 관리하기 매우 쉽습니다.

기본 버전:

```
def dense(x, W, b):  
    return tf.nn.sigmoid(tf.matmul(x, W) + b)  
  
@tf.function  
def multilayer_perceptron(x, w0, b0, w1, b1, w2, b2 ...):  
    x = dense(x, w0, b0)  
    x = dense(x, w1, b1)  
    x = dense(x, w2, b2)  
    ...  
  
# 여전히 w_i, b_i 변수를 직접 관리해야 합니다. 이 코드와 떨어져서 크기가 정의됩니다.
```

Thank you!
Q&A