2023 TABA 파이썬 기초

Presented by: Park JunYeong

Contents

I. 파이썬 기초

II. 데이터 과학 라이브러리

(numpy, pandas, ..)

파이썬 기초

A. 파이썬 시작하기 (Colab)
B. 자료형
C. 조건문
D. 반복문
   - 리스트+반복문
   - 딕셔너리+반복문
   - for문과 while문
E. 함수
F. 클래스

A. 파이썬 시작하기

장점

1. 문법의 간결성
2. 다양한 분야에의 활용성
3. 운영체제 일관성

단점

“느리다” (램을 왕창왕창 잡아먹는다.)

A. 파이썬 시작하기

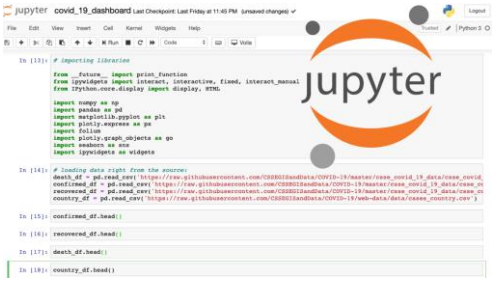
개발환경

텍스트 에디터(Text Editor) + 코드 실행기(Python Interpreter)

IDE와 Text Editor의 차이

- IDE (Integrated Development Environment, 통합개발환경) : 텍스트에디터+ 코드실행기를 모두 포함하고 있는 개발 환경 (ex. Visual Studio ...)
- Text Edition: 텍스트를 해당 언어의 문법에 맞게 작성하도록 도와주는 프로그램 (ex. Visual Studio Code …)

A. 파이썬 시작하기 - 실습환경



A. 파이썬 시작하기 - 실습환경

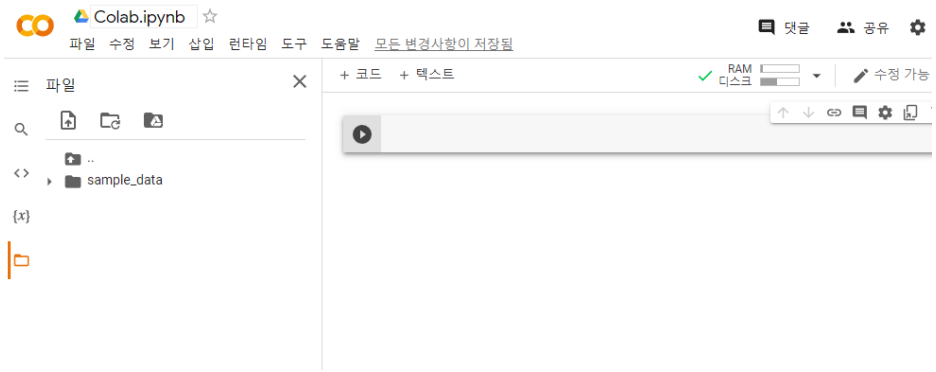
colab

구글 코랩은 구글 클라우드 기반의 무료 Jupyter Notebook 이다.

접근성이 쉬우며 개개인의 PC보다 좋은 성능의 GPU를 무료로 액세스 할 수 있어 많은 사람들이 딥러닝을 학습하는 데 많이 사용하고 있다.

- Jupyter Notebook은 대화형 파이썬 인터프리터(Interpreter)로서 웹 브라우저 환경에서 파이썬 코드를 작성 및 실행할 수 있는 툴이다.

https://colab.research.google.com/



A. 파이썬 시작하기 - 실습환경

colab

https://colab.research.google.com/

B. 자료형(Data Type) - 문자열(string)

자료형 확인 메소드 : **type()**

문자열(string)

- “안녕하세요”, ‘안녕하세요’
- 작은 따옴표(‘’)와 큰 따옴표(“”)를 구별하지 않음.
- 단, 동일한 문장 내에서 따옴표를 중복으로 사용한다면, 서로 다른 종류의 기호를 사용해야 함.
  혹은, ’ **이스케이프 문자**’를 사용

```
# 문자열
st = "안녕하세요."
st2 = "반갑습니다."

print(st)
print(st2)
```

```
안녕하세요.
반갑습니다.
```

```
type(st)
```

```
str
```

B. 자료형(Data Type) - 문자열(string)

이스케이프 문자

- 역슬래시(\) 기호와 함께 조합해서 사용하는 특수한 문자

  (한국어 키보드에선 원화 기호(₩))

```
print(""안녕하세요." 라고 말했습니다.")
```

```
  File "<ipython-input-4-ef507dd3dcea>", line 1
    print(""안녕하세요." 라고 말했습니다.")
                ^
SyntaxError: invalid syntax
```

```
print("'안녕하세요.' 라고 말했습니다.")
```

```
'안녕하세요.' 라고 말했습니다.
```

```
print("\"안녕하세요\"라고 말했습니다.")
```

```
"안녕하세요"라고 말했습니다.
```

- \t : 탭
- \n : 줄바꿈(개행문자)
- \\ : 역슬래시 ➔ window 운영체제의 파일경로 표시 할 때 주로 사용

  (ex1. ”C:\User\pycharm_project” ➔ 에러 발생 위험)

B. 자료형(Data Type) - 문자열(string)

```
print(""안녕하세요." 라고 말했습니다.")
```

```
  File "<ipython-input-4-ef507dd3dcea>", line 1
    print(""안녕하세요." 라고 말했습니다.")
            ^
SyntaxError: invalid syntax
```

Syntax Error : 코드 작성이 잘못되어서 실행되지 않는 경우

→ 주로 문법 오류

B. 자료형(Data Type) - 문자열(string)

이스케이프 문자

- \t : 탭
- \n : 줄바꿈(개행문자)
- \\ : 역슬래시 ➔ window 운영체제의 파일경로 표시 할 때 주로 사용
  (ex1. ”C:\User\pycharm_project” ➔ 에러 발생 위험)

```
# 이스케이프 문자
print("이름\t나이\t지역")
print("====================")
print("김동훈\t20\t서울")
print("김형식\t20\t분당")
print("이현지\t20\t죽전")
```

```
이름	나이	지역
====================
김동훈	20	서울
김형식	20	분당
이현지	20	죽전
```

```
print("C:\User\tmp\")
```

```
  File "<ipython-input-17-d9347c598dd2>", line 1
    print("C:\User\tmp\")
                        ^
SyntaxError: EOL while scanning string literal
```

```
# 이스케이프 문자2 (window 파일경로)
# 에러 발생: print("C:\User\tmp\")
print("C:\\User\\tmp\\")
```

```
C:\User\tmp\
```

B. 자료형(Data Type) - 문자열(string)

여러줄 쓰기

- 이스케이프 문자 '\n'

```
# 여러 줄 쓰기
print("동해물과 백두산이 마르고 닳도록\n하느님이 보우하사 우리나라 만세\n무궁화 삼천리 화려강산\n대한사람 대한으로 길이 보전하세.")
```

```
동해물과 백두산이 마르고 닳도록
하느님이 보우하사 우리나라 만세
무궁화 삼천리 화려강산
대한사람 대한으로 길이 보전하세.
```

- """ """ 기호

```
print("""
동해물과 백두산이 마르고 닳도록
하느님이 보우하사 우리나라 만세
무궁화 삼천리 화려강산
대한사람 대한으로 길이 보전하세.
""")
```

```
동해물과 백두산이 마르고 닳도록
하느님이 보우하사 우리나라 만세
무궁화 삼천리 화려강산
대한사람 대한으로 길이 보전하세.
```

```
print('''
동해물과 백두산이 마르고 닳도록
하느님이 보우하사 우리나라 만세
무궁화 삼천리 화려강산
대한사람 대한으로 길이 보전하세.
''')
```

```
동해물과 백두산이 마르고 닳도록
하느님이 보우하사 우리나라 만세
무궁화 삼천리 화려강산
대한사람 대한으로 길이 보전하세.
```

```
print("""\
동해물과 백두산이 마르고 닳도록
하느님이 보우하사 우리나라 만세
무궁화 삼천리 화려강산
대한사람 대한으로 길이 보전하세.\
""")
```

```
동해물과 백두산이 마르고 닳도록
하느님이 보우하사 우리나라 만세
무궁화 삼천리 화려강산
대한사람 대한으로 길이 보전하세.
```

B. 자료형(Data Type) - 문자열(string)

연산자

- +(문자열 연결 연산자), *(문자열 반복 연산자)
- 문자열 선택연산자(인덱싱): []

```
# 문자열 연산자 : +, *
print("안녕하세요. " + "반갑습니다.")
print("안녕하세요." * 3)
```

```
안녕하세요. 반갑습니다.
안녕하세요.안녕하세요.안녕하세요.
```

```
# 문자열 인덱싱
print("문자열 인덱싱에 대해 알아볼까요?")
print("안녕하세요."[0])
print("안녕하세요."[1])
print("안녕하세요."[2])
print("안녕하세요."[3])
print("안녕하세요."[4])
print("안녕하세요."[5])
```

```
문자열 인덱싱에 대해 알아볼까요?
안
녕
하
세
요
.
```

```
# 문자열 인덱싱(역순)
print("문자열 인덱싱에 대해 알아볼까요?")
print("안녕하세요."[-6])
print("안녕하세요."[-5])
print("안녕하세요."[-4])
print("안녕하세요."[-3])
print("안녕하세요."[-2])
print("안녕하세요."[-1])
```

```
문자열 인덱싱에 대해 알아볼까요?
안
녕
하
세
요
.
```

B. 자료형(Data Type) - 문자열(string)

연산자

- 문자열 범위 선택연산자(슬라이싱) [:]

(원본은 변하지 않음)

```
# 문자열 슬라이싱
print("안녕하세요"[1:4])
print("안녕하세요"[:4])
print("안녕하세요"[2:])
```

```
녕하세
안녕하세
하세요
```

```
print("안녕하세요"[10])
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-44-31d7842c8b75> in <cell line: 1>()
----> 1 print("안녕하세요"[10])

IndexError: string index out of range
```

- 문자열 길이 구하기: len() ➔ 리스트 ,딕셔너리 등도 사용 가능

```
# 문자열 길이
print(len("안녕하세요"))
```

```
5
```

B. 자료형(Data Type) - 문자열(string)

```
print("안녕하세요"[10])
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-44-31d7842c8b75> in <cell line: 1>()
----> 1 print("안녕하세요"[10])

IndexError: string index out of range
```

Index Error (index out of range): 리스트/문자열의 수를 넘는 요소/글자를 선택할 때 발생

B. 자료형(Data Type) - 숫자(number)

종류

- int(integer) : 정수
- float(floating point) : 부동소수점(실수)

cf) C언어 : char, shot, int, long, long long / float, double, long double

- 파이썬을 활용해 데이터 분석을 할 때는 int16, int32, int64, float32, float64 등을 지정해서 사용하기도 함. 그러나 np.float32, torch.float32 등 라이브러리에 종속된 방법으로 지정.

- 지수표현 :

  0.52273 x 10^2 = 0.52273e2 = 52.273

  0.52273 x 10^-2 = 0.62273e-2 = 0.0052273

B. 자료형(Data Type) - 숫자(number)

연산자

- 사칙연산자 : + - * /
- 정수 나누기 연산자 : //

```
# 정수 나누기 연산자 : //
print("3 / 2 = ", 3/2)
print("3 // 2 = ", 3//2)
```

```
3 / 2 =  1.5
3 // 2 =  1
```

- 나머지 연산자(mod) : %

```
# 나머지 연산자
print("5 % 2 = ", 5%2)
```

```
5 % 2 =  1
```

제곱 연산자: **

```
# 제곱 연산자
print("2 ** 1 = ", 2**1)
print("2 ** 2 = ", 2**2)
print("2 ** 3 = ", 2**3)
print("2 ** 4 = ", 2**4)
```

```
2 ** 1 =  2
2 ** 2 =  4
2 ** 3 =  8
2 ** 4 =  16
```

B. 자료형(Data Type) - 숫자(number)

```
string = "문자열"
number = 273
print(string + number)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-50-93f6d045e87b> in <cell line: 3>()
      1 string = "문자열"
      2 number = 273
----> 3 print(string + number)

TypeError: can only concatenate str (not "int") to str
```

TypeError : 서로 다른 자료형을 연산했을 때 발생

B. 자료형 (변수와 입력)

변수

- 값을 저장할 때 사용하는 식별자
- C, C++, Java, C# 등과 달리 변수 자료형을 미리 선언할 필요가 없음.
  (단, 코드가 길어질수록 유지보수나 환경설정을 위해 최신 파이썬에서는 'type annotation'을 지원)

```
name: str = "John Doe"
age: int = 25
emails: list: ["john1@doe.com", "john2@doe.com"]
address: dict = {
  "street" : "-",
}
```

- 장점: 동일한 변수 명에 여러 타입의 변수들이 대입될 수 있으므로 다른 언어보다 유연성이 높음.
- 주의: TypeError 발생 가능성이 높으므로 하나의 변수에는 되도록 하나의 자료형을 넣어 활용하는 것이 좋음.

B. 자료형 (변수와 입력)

변수 - 예시(원의 둘레와 넓이 구하기)

```
# 변수 선언과 할당
pi = 3.141592265
r = 10

# 변수 참조
print("원주율  = ", pi)
print("반지름 = ", r)
print("원의 둘레 = ", 2*pi*r)
print("원의 넓이 = ", pi*r*r)
```

```
원주율  =  3.141592265
반지름 =  10
원의 둘레 =  62.8318453
원의 넓이 =  314.1592265
```

B. 자료형 (변수와 입력)

복합 대입 연산자

연산자 이름	설명
+=	숫자 덧셈 후 대입 (문자열 연결 후 대입)
-=	숫자 뺄셈 후 대입
*=	숫자 곱셈 후 대입 (문자열 반복 후 대입)
/=	숫자 나눗셈 후 대입
%=	숫자의 나머지를 구한 후 대입
**=	숫자 제곱 후 대입

```
# 복합대입 연산자(숫자)
number = 100
number += 10
number += 20
number += 30
print("number : ", number)
```

```
number :  160
```

```
# 복합대입 연산자(문자열)
string = "안녕하세요"
string += "!"
string += "!"
print("string : ", string)
```

```
string :  안녕하세요!!
```

B. 자료형 (변수와 입력)

자료형변환(cast)

- Int(변수), float(변수), str(변수)

```
# cast (문자열 -> 숫자)
string_a = "1024"
int_a = int(string_a)

string_b = "1004"
int_b = int(string_b)

print("문자열 자료: ", string_a + string_b)
print("숫자 자료: ", int_a + int_b)
```

```
문자열 자료:  10241004
숫자 자료:  2028
```

int() 함수

```
# int(), float()
output_a = int("52")
output_b = float("52.273")

print(type(output_a), output_a)
print(type(output_b), output_b)
```

```
<class 'int'> 52
<class 'float'> 52.273
```

int(), float() 함수

B. 자료형 (변수와 입력)

자료형변환(cast)

- Int(변수), float(변수), str(변수)

```
# int 함수와 float 함수 조합
int_a = int(273)
int_b = int(52)
float_c = float(273)
float_d = float(52)

print("int_a / int_b= ", int_a/int_b)
print("int_a // int_b= ", int_a//int_b)
print("float_c / float_d ", float_c/float_d)
```

```
int_a / int_b=  5.25
int_a // int_b=  5
float_c / float_d  5.25
```

int(), float()의 결과 연산

```
int("안녕하세요.")
float("안녕하세요.")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-5-46105c5e7b9f> in <cell line: 1>()
----> 1 int("안녕하세요.")
      2 float("안녕하세요.")

ValueError: invalid literal for int() with base 10: '안녕하세요.'
```

ValueError

B. 자료형 (숫자와 문자열의 기능)

대소문자 변환

upper(), lower()

```
# 대소문자 변환
a = "Hello Python Programming...!"
print(a.upper())
print(a.lower())
```

```
HELLO PYTHON PROGRAMMING...!
hello python programming...!
```

파괴적 함수 vs. 비파괴적 함수

- 파괴적 함수: 원본을 변화시키는 함수
- 비파괴적 함수: 원본을 변화시키지 않는 함수

B. 자료형 (숫자와 문자열의 기능)

문자열 찾기

- find() : 왼쪽부터 찾아서 처음 등장하는 위치를 반환
- rfind() : 오른쪽부터 찾아서 처음 등장하는 위치를 반환

```
# 문자열 찾기
a = "안녕안녕하세요"
print(a.find("안녕"))
print(a.rfind("안녕"))
```

```
0
2
```

문자열 in 연산자

- 문자열 내부에 어떤 문자열이 있는지 확인
- 출력 : bool 연산자 (True or False)

```
# 문자열 in 연산자
print("안녕" in "안녕하세요.")
print("잘자" in "안녕하세요.")
```

```
True
False
```

B. 자료형 (숫자와 문자열의 기능)

문자열 자르기

- split()
- 공백(띄어쓰기)를 기준으로 자름
- 실행 결과로 리스트(list)가 반환

```
# 문자열 자르기
a = "10 20 30 40 50 60"
print(a.split(" "))
```

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

C. 조건문

Boolean 자료형

- True or False (첫 글자를 반드시 대문자로)

```
print(1 == True)
print(0 == False)
print(0 == True)
```

```
True
True
False
```

- ‘어떤 명제 ’ 의 결과를 나타냄.

비교 연산자

- 결과가 Boolean 타입으로 반환.
- 문자열에도 적용 가능.

  (문자열의 크기는 사전순서(가나다순)으로 작은 값을 가짐)

```
# 비교 연산자
print("가방" == "가방")
print("가방" < "하마")
print("가방" > "하마")
```

```
True
True
False
```

==	같다
!=	다르다
〈	작다
〉	크다
〈=, 〉=	작거나(크거나) 같다

C. 조건문

논리 연산자

- Boolean 끼리 연산 할 때 사용

```
# 논리 연산자
print(not True)
print(not False)

print(True and True)
print(True and False)

print(True or True)
print(True or False)
```

```
False
True
True
False
True
True
```

not	아니다(True, False 전환)
and	피연산자 두 개가 모두 참일 때 True, 그 외는 모두 False
or	피연산자 두 개 중 하나만 참이더라도 True, 그 외는 모두 False

C. 조건문

IF 조건문

- 조건에 따라 코드를 실행하거나 실행하지 않게 만들고 싶을 때 사용 (➔ 조건분기)
- if문 조건부의 내용이 True 일 때 실행됨

```
# 조건문
if True:
  print("True 입니다.")
if False:
  print("False 입니다.")

print("Default 문장입니다.")
```

```
True 입니다.
Default 문장입니다.
```

```
if inputs.size(0) != 1:
    output_batch = torch.stack(output_batch, dim=0)
    return output_batch, output_lengths

else:
    return outputs, output_lengths
```

C. 조건문

if~else

- 조건이 하나일 때 (A일 때와 아닐 때)

```
# if~elif
number = input("정수입력> ")
number = int(number)

if number % 2 == 0:
  print("짝수입니다.")
else:
  print("홀수입니다.")
```

```
정수입력> 13
홀수입니다.
```

```
if inputs.size(0) != 1:
    output_batch = torch.stack(output_batch, dim=0)
    return output_batch, output_lengths

else:
    return outputs, output_lengths
```

if~elif

- 확인해야 할 조건이 2개 이상일 때

```
# if~elif (계절 구하기)
import datetime

now = datetime.datetime.now()
month = now.month

if 3 <= month <= 5:
  print("현재는 봄 입니다.")
elif 6 <= month <= 8:
  print("현재는 여름 입니다.")
elif 9 <= month  <= 11:
  print("현재는 가을 입니다.")
else:
  print("현재는 겨울 입니다.")
```

```
현재는 봄 입니다.
```

C. 조건문

pass

- 나중에 개발하려고 잠시 비워 두는 키워드

  (잊어버리고 구현 안 할 까봐 걱정된다면 raise NotImplementedError 지정)

```
# pass
number = 13

if number > 0:

else:
```

```
  File "<ipython-input-21-cedabb970ba9>", line 6
    else:
    ^
IndentationError: expected an indented block
```

IndentationError: 들여쓰기가 잘못 되어있을 때 발생하는 에러

```
# pass
number = 13

if number > 0:
  pass
else:
  pass
```

```
# raise NotImpementedError
# pass
number = 13

if number > 0:
  raise NotImplementedError("구현필요")
else:
  raise NotImplementedError("구현필요")
```

```
---------------------------------------------------------------------------
NotImplementedError                       Traceback (most recent call last)
<ipython-input-24-89cc439bf577> in <cell line: 5>()
      4
      5 if number > 0:
----> 6   raise NotImplementedError("구현필요")
      7 else:
      8   raise NotImplementedError("구현필요")

NotImplementedError: 구현필요
```

D. 반복문

리스트(list)와 반복문

- 여러 자료형의 값 저장 가능 (각 요소는 element 라고 부름)

```
# 리스트
array = [273, 32, 103, "문자열", True, False]
print(array)
```

```
[273, 32, 103, '문자열', True, False]
```

- zero-index로 시작, 문자열처럼 인덱싱/슬라이싱/음수슬라이싱 모두 가능

```
print("array[0] : ", array[0])
print("array[3][0] : ", array[3][0])
print("array[-1] : ", array[-1])
```

```
array[0] :  273
array[3][0] :  문
array[-1] :  False
```

- 리스트 안에 리스트(딕셔너리) 선언 가능

```
array = [[1,2,3,4], [5,6,7,9]]
array_dict = [{"array": 1, "arr": 2}, {"python": 3}]

print(array)
print(array_dict)
```

```
[[1, 2, 3, 4], [5, 6, 7, 9]]
[{'array': 1, 'arr': 2}, {'python': 3}]
```

D. 반복문

리스트(list) 연산자

- 문자열에 적용할 수 있는 연산자는 리스트에서도 적용 가능

```
# 리스트 연산자
list_a = [1,2,3]
list_b = [4,5,6]

# 연결 연산자 +
print("list_a + list_b = ", list_a + list_b)

# 반복 연산자 *
print("list_a * 3 = ", list_a * 3)

# 리스트 길이 len
print("length of list : ", len(list_a))
```

```
list_a + list_b =  [1, 2, 3, 4, 5, 6]
list_a * 3 =  [1, 2, 3, 1, 2, 3, 1, 2, 3]
length of list :  3
```

- 리스트에 요소 추가하기: append(), insert()
  - append(element) : 리스트 마지막 자리에 요소를 추가
  - insert(index, element) : 정해진 자리에 요소 추가

```
list_a = [1,2,3]
list_a.append(4)
print(list_a)

list_a.insert(2, 5)
print(list_a)
```

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

D. 반복문

리스트 연결 연산자

- + or extend
  - + : 비 파괴적 연산 처리 (list_a, list_b가 변하지 않음)
  - extend : 파괴적 연산 처리 (원본의 변화)

```
list_a = [1,2,3]
list_b = [4,5,6]

print("list_a + list_b = ", list_a + list_b)
print("list_a : ", list_a)
print("list_b : ", list_b)
print("list_a.extend(list_b) = ", list_a.extend(list_b))
print("list_a after extend : ", list_a)
```

```
list_a + list_b =  [1, 2, 3, 4, 5, 6]
list_a :  [1, 2, 3]
list_b :  [4, 5, 6]
list_a.extend(list_b) =  None
list_a after extend :  [1, 2, 3, 4, 5, 6]
```

D. 반복문

append vs. extend

- 공통점 : 파괴적 함수
- append : 원본 형태로 리스트에 요소 추가
- extend : 요소를 하나씩 빼서 추가

```
# append vs. extend
list_a = [1,2,3]
list_b = [4,5,6]

list_c = []
list_c.append(list_a)
list_c.append(list_b)
print(list_c)

list_c = []
list_c.extend(list_a)
list_c.extend(list_b)
print(list_c)
```

```
[[1, 2, 3], [4, 5, 6]]
[1, 2, 3, 4, 5, 6]
```

리스트 요소 제거

- del 리스트명[index]

  (매개변수 입력 안 하면 마지막 요소 제거, 범위 지정 가능)

- 리스트명.pop(index)

```
# 리스트 요소 제거
list_a = [1,2,3]
del list_a[:2]
print(list_a)
```

```
[3]
```

```
list_a = [1,2,3]
list_a.pop(2)
print(list_a)
```

```
[1, 2]
```

- 리스트명.remove(값)

```
list_a = [1,2,3]
list_a.remove(2)
print(list_a)
```

```
[1, 3]
```

D. 반복문

리스트에 요소 있는지 확인

- 값 in 리스트

```
# 리스트에 요소 있는지 확인
list_a = [1,2,3]
print(3 in list_a)
print(5 in list_a)
```

```
True
False
```

D. 반복문

딕셔너리(Dictionary)

Key : Value 형태로 이루어진 자료형

```
# 딕셔너리
a = {
    "키A" : 10,
     "키B" : 20,
     "키C" : 30,
     1 : 40,
     False: 50
}
print(a)
```

```
{'키A': 10, '키B': 20, '키C': 30, 1: 40, False: 50}
```

```
a = {
    "키A" : 10,
     "키B" : 20,
     "키C" : 30,
     1 : 40,
     False: 50,
     "키D" : ['망고', '설탕', '메타중아황산나트륨', '치자황색소']
}
print(a['키D'][0])
```

```
망고
```

```
# 딕셔너리
a = {
    "키A" : 10,
     "키B" : 20,
     "키C" : 30,
     1 : 40,
     False: 50
}

print(a["키A"])
```

```
10
```

D. 반복문

딕셔너리(Dictionary) 요소 추가/제거

```
# 딕셔너리 요소 추가/제거
dictionary = {}
dictionary["name"] = "새로운 이름"
dictionary["head"] = "새로운 정신"
dictionary["body"] = "새로운 몸"

print(dictionary)

del dictionary["name"]
print(dictionary)
```

```
{'name': '새로운 이름', 'head': '새로운 정신', 'body': '새로운 몸'}
{'head': '새로운 정신', 'body': '새로운 몸'}
```

딕셔너리(Dictionary) 요소 확인

```
# 딕셔너리 요소 확인
a = {
    "키A" : 10,
     "키B" : 20,
     "키C" : 30,
     1 : 40,
     False: 50,
     "키D" : ['망고', '설탕', '메타중아황산나트륨', '치자황색소']
}

print("키A" in a)
```

```
True
```

D. 반복문

for문

```python
# for 반복문
for i in range(5):
  print(str(i) + "=반복 변수")
print()

for i in range(5, 10):
  print(str(i) + "=반복 변수")
print()

for i in range(0, 10, 3):
  print(str(i) + "=반복 변수")
print()
```

```
0=반복 변수
1=반복 변수
2=반복 변수
3=반복 변수
4=반복 변수

5=반복 변수
6=반복 변수
7=반복 변수
8=반복 변수
9=반복 변수

0=반복 변수
3=반복 변수
6=반복 변수
9=반복 변수
```

```python
# for 반복문 (element, key로 반복하기)
list_a = [1,2,3]

for elem in list_a:
  print(elem)
print()

dict_a = {
    "키A" : 10,
     "키B" : 20,
     "키C" : 30,
     1 : 40,
     False: 50,
     "키D" : ['망고', '설탕', '메타중아황산나트륨', '치자황색소']
}

for key in dict_a:
  print(str(key), " : ", str(dict_a[key]))
```

```
1
2
3

키A  :  10
키B  :  20
키C  :  30
1  :  40
False  :  50
키D  :  ['망고', '설탕', '메타중아황산나트륨', '치자황색소']
```

D. 반복문

for문

```
def sentence_to_target(sentence: str, char2id: dict) → str:
    target = str()

    for ch in sentence:     # 각 글자를 id로 바꿔서 target sentence를 생성
        try:
            target += (str(char2id[ch]) + ' ')
        except KeyError:
            logging.warning(f"Out Of Vocabulary. Sentence is {sentence} and Character is {ch}")
            continue

    return target[:-1]
```

continue

D. 반복문

while문

```
# while 반복문
while True:
  print(".", end="")
```

```
....................
```

break 문

```
# while 반복문
i = 0
while i < 10:
  print(f"{i}번째 반복입니다.")
  i += 1
```

```
0번째 반복입니다.
1번째 반복입니다.
2번째 반복입니다.
3번째 반복입니다.
4번째 반복입니다.
5번째 반복입니다.
6번째 반복입니다.
7번째 반복입니다.
8번째 반복입니다.
9번째 반복입니다.
```

```
# while 반복문 break

i = 0
while True:
  print("반복 중..")
  if (i != 0) & (i % 5 == 0):
    break;
  i += 1
```

```
반복 중..
반복 중..
반복 중..
반복 중..
반복 중..
반복 중..
```

E. 함수

기본 형태

```
# 함수 기본 형태
def 함수이름(매개변수, 매개변수, ...):
    수행할 문장
    수행할 문장
    ...
```

리턴값

```
def generate_character_scripts(audio_paths: str, transcripts: list, vocab_dest: str) → None:
    """*** generate_character_script ***..."""
    logging.info("Creat script started...")
    char2id, id2char = load_vocab(os.path.join(vocab_dest, "character_vocab.csv"))

    transcripts_path = vocab_dest.replace("vocab", "transcripts")
    with open(os.path.join(transcripts_path, "character_transcripts.txt"), "w", encoding="utf8") as f:
        for audio_path, transcript in zip(audio_paths, transcripts):
            vectorized_transcript = sentence_to_target(transcript, char2id)
            f.write(f"{audio_path}\t{transcript}\t{vectorized_transcript}\n")
```

```
파이썬 프로그래밍

안녕하세요
즐거운
파이썬 프로그래밍

안녕하세요
즐거운
파이썬 프로그래밍
```

```
A :  550
B :  5050
C :  2550
```

E. 함수

```python
def generate_character_scripts(audio_paths: str, transcripts: list, vocab_dest: str) → None:
    """*** generate_character_script ***..."""
    logging.info("Creat script started...")
    char2id, id2char = load_vocab(os.path.join(vocab_dest, "character_vocab.csv"))

    transcripts_path = vocab_dest.replace("vocab", "transcripts")
    with open(os.path.join(transcripts_path, "character_transcripts.txt"), "w", encoding="utf8") as f:
        for audio_path, transcript in zip(audio_paths, transcripts):
            vectorized_transcript = sentence_to_target(transcript, char2id)
            f.write(f"{audio_path}\t{transcript}\t{vectorized_transcript}\n")
```

F. 클래스

```python
# 클래스
class Rectangle:
  count = 0          # 클래스 변수

  def __init__(self, width, height):
    self.width = width
    self.height = height
    self._area = width * height
    Rectangle.count += 1

  # 인스턴스 메소드
  def calcArea(self):
    area = self.width * self.height
    return area

  @staticmethod
  def isSquare(rectWidth, rectHeight):
    return rectWidth == rectHeight

  @classmethod
  def printCount(cls):
    print(cls.count)
```

```python
square = Rectangle.isSquare(5,5)
print(square)

rect1 = Rectangle(5,5)
rect2 = Rectangle(2,5)
rect1.printCount()
```

ParkJun-Yeong *

```python
class DeepSpeech2(EncoderModel):
    """..."""

    ParkJun-Yeong
    def __init__(
            self,
            input_dim: int,
            num_classes: int,
            rnn_type='gru',
            num_rnn_layers: int = 5,
            rnn_hidden_dim: int = 512,
            dropout_p: float = 0.1,
            bidirectional: bool = True,
            activation: str = 'hardtanh',
            device: torch.device = 'cuda',
    ):
        super(DeepSpeech2, self).__init__()
        self.device = device
        self.conv = DeepSpeech2Extractor(input_dim, activation=activation)
        self.rnn_layers = nn.ModuleList()
        rnn_output_size = rnn_hidden_dim << 1 if bidirectional else rnn_hidden_dim
```