# 인공지능을 위한 파이썬 프로그래밍

01 파이썬 (Python)이란?

Deep & High Learning

Copyright 2021. Deep&High learning. All rights reserved.

# Contents

## 파이썬


"Life is short, you need Python."

1990년 귀도 반 로섬이 개발한 인터프리터 언어

인터프리터 언어: 한 줄씩 소스 코드를 해석해서 실행 결과를 바로 확인할 수 있는 언어

구글에서 만든 소프트웨어의 50%이상이 파이썬으로 작성됨

파이썬의 창시자, 귀도 반 로섬
(Guido van Rossum)


파이썬이 최근 급격한 성장을 보이는 이유?

## 파이썬의 특징

```c
#include <stdio.h>

int main(){
     printf("Hello world!");
     return 0;
}
```

<C>

```java
public class Helloworld{
     public static void main(String[] args){
          System.out.println("Hello world!");
     }
}
```

<Java>

```cpp
#include <iostream>
using namespace std;

int main(){
     cout <<("Hello world!") << endl;
     return 0;
}
```

<C++>

```python
print("Hello world!")
```

<Python>

언어 형태가 매우 간단하다!

## 파이썬의 특징

C언어의 자료형 – 복잡하다

| 구분 | 자료형 | 범위 |
|---|---|---|
| 정수형 | short | -32678~32767 |
| | int | -2,147,483,648~2,147,483,647 |
| | long | -2,147,483,648~2,147,483,647 |
| | unsigned short | 0~65535 |
| | unsigned int | 0~4,294,967,295 |
| | unsigned long | 0~4,294,967,295 |

## 파이썬의 특징

파이썬의 자료형 – 간단하다

| 구분 | 자료형 | 범위 |
|---|---|---|
| 정수형 | int | 무제한 |

타 언어에 비해 편리하게 사용할 수 있다!

## Pythonic

파이썬은 인간다운 언어이다

파이썬은 문법이 쉬워 빠르게 배울 수 있다

파이썬은 무료이지만 강력하다

파이썬은 간결하다

파이썬은 프로그래밍을 즐기게 해준다

파이썬은 개발 속도가 빠르다

↓

"Pythonic"

```
if 4 in [1,2,3,4]:
    print("yes")
```

# Pythonic

```
Python Console
C:\Users\unthi\anaconda3\envs\new\python.exe "C:\Program Files\JetBrains\PyCharm 2021.2\plugins\python\helpers\pydev\pydevconsole.py" --mode=client --port=61380

import sys; print('Python %s on %s' % (sys.version, sys.platform))
sys.path.extend(['D:\\구글 드라이브-동기화 폴더\\File정리', 'D:/구글 드라이브-동기화 폴더/File정리'])

Python Console>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Zen of Python, Tim Peters 저
아름다운 것이 못생긴 것보다 낫습니다.
명시적인 것이 암시적인 것보다 낫습니다.
단순한 것이 복잡한 것보다 낫습니다.
복잡한 것보다 복잡한 것이 낫습니다.
플랫은 중첩보다 낫습니다.
희소가 조밀보다 좋습니다.
가독성이 중요합니다.특별한 경우는 규칙을 어길 만큼 특별하지 않습니다.
실용성이 순수함을 능가하지만.오류는 자동으로 전달되어서는 안 됩니다.
명시적으로 침묵하지 않는 한, 모호함에 직면하여 추측하려는 유혹을 거부하십시오.
그것을 할 수 있는 확실한 방법이 하나 있어야 하고 가급적이면 하나만 있어야 합니다.
당신이 네덜란드 사람이 아닌 한 처음에는 그 방법이 분명하지 않을 수 있습니다.
지금은 결코 없는 것보다 낫습니다.
결코 지금보다 *지금*보다 낫지만,구현이 설명하기 어렵다면 나쁜 생각입니다.
구현이 설명하기 쉬우면 좋은 아이디어일 수 있습니다.
네임 스페이스는 경적을 울리는 훌륭한 아이디어 중 하나입니다. 더 많이 해 봅시다!

See you in the next lecture!

# 인공지능을 위한 파이썬 프로그래밍

02 Google Colab

Deep & High Learning

Copyright 2021.  Deep&High learning. All rights reserved.

# Contents

## Colab이란?

구글에서 개발한 Google Colaboratory의 약자.

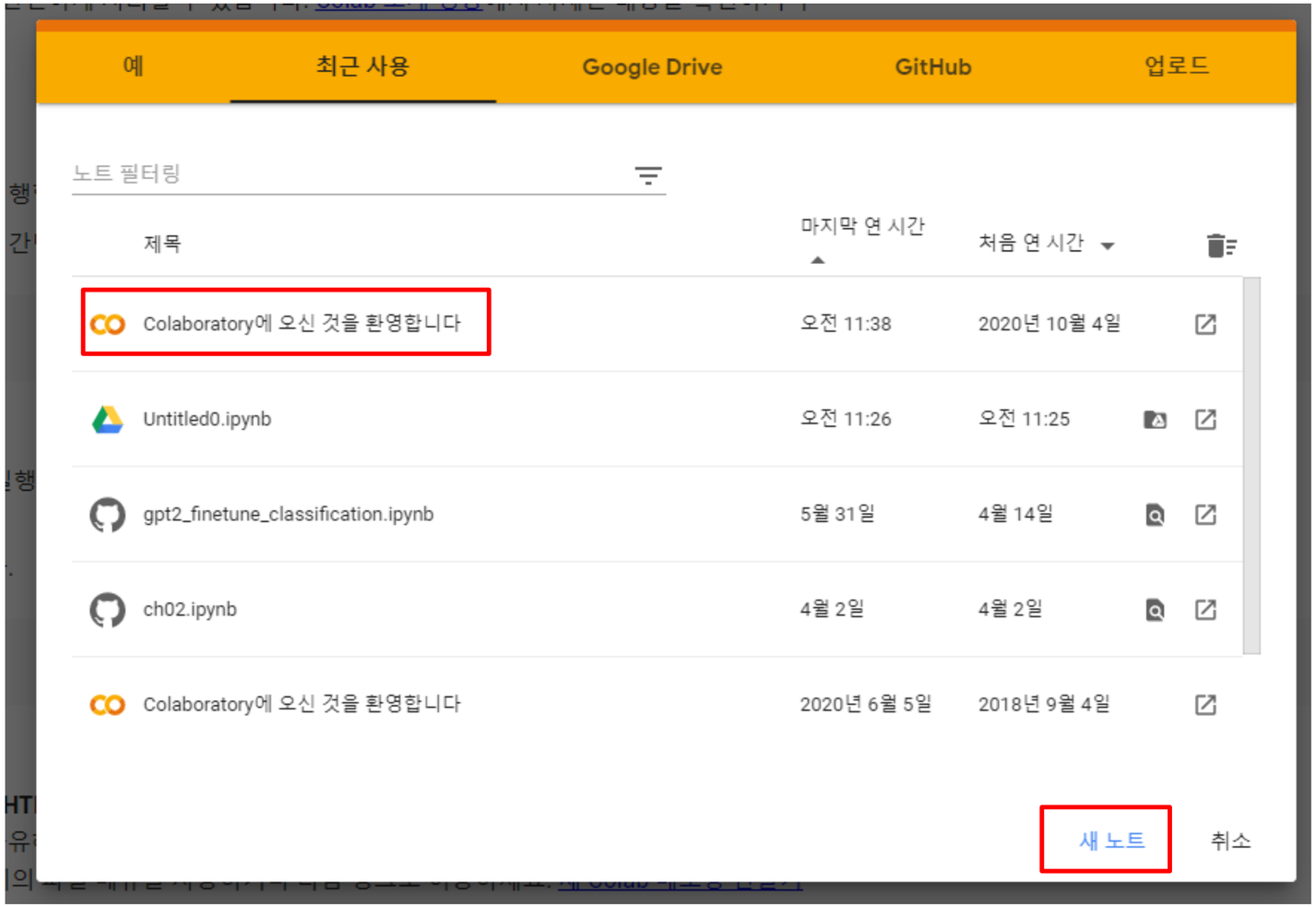
웹 브라우저 상에서 Python을 실행할 수 있게 하는 서비스

GPU 무료 사용 가능

(Jupyter Notebook보다 설치 및 사용 간편)

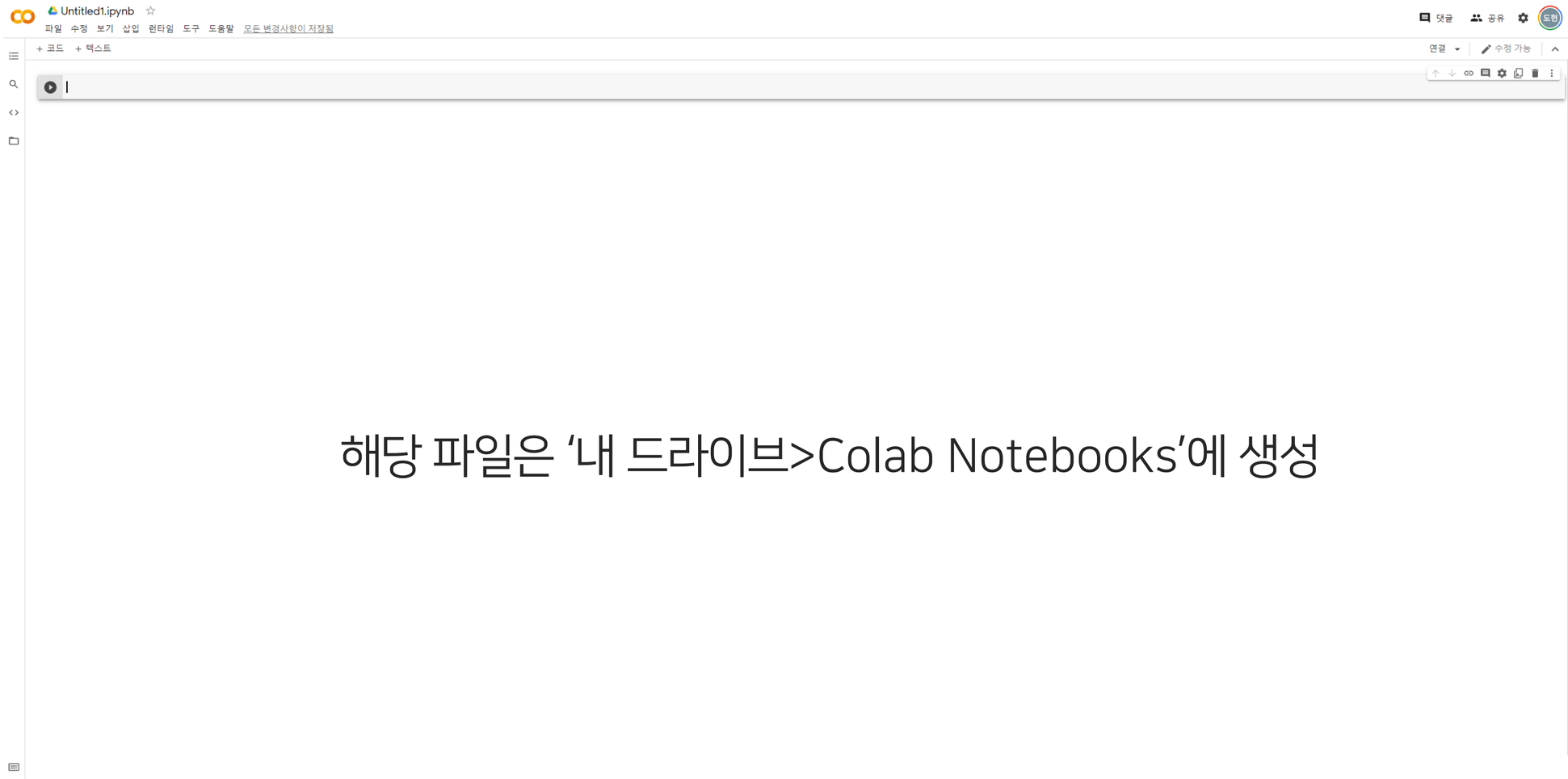
# Colab 시작하기

https://colab.research.google.com/ 접속

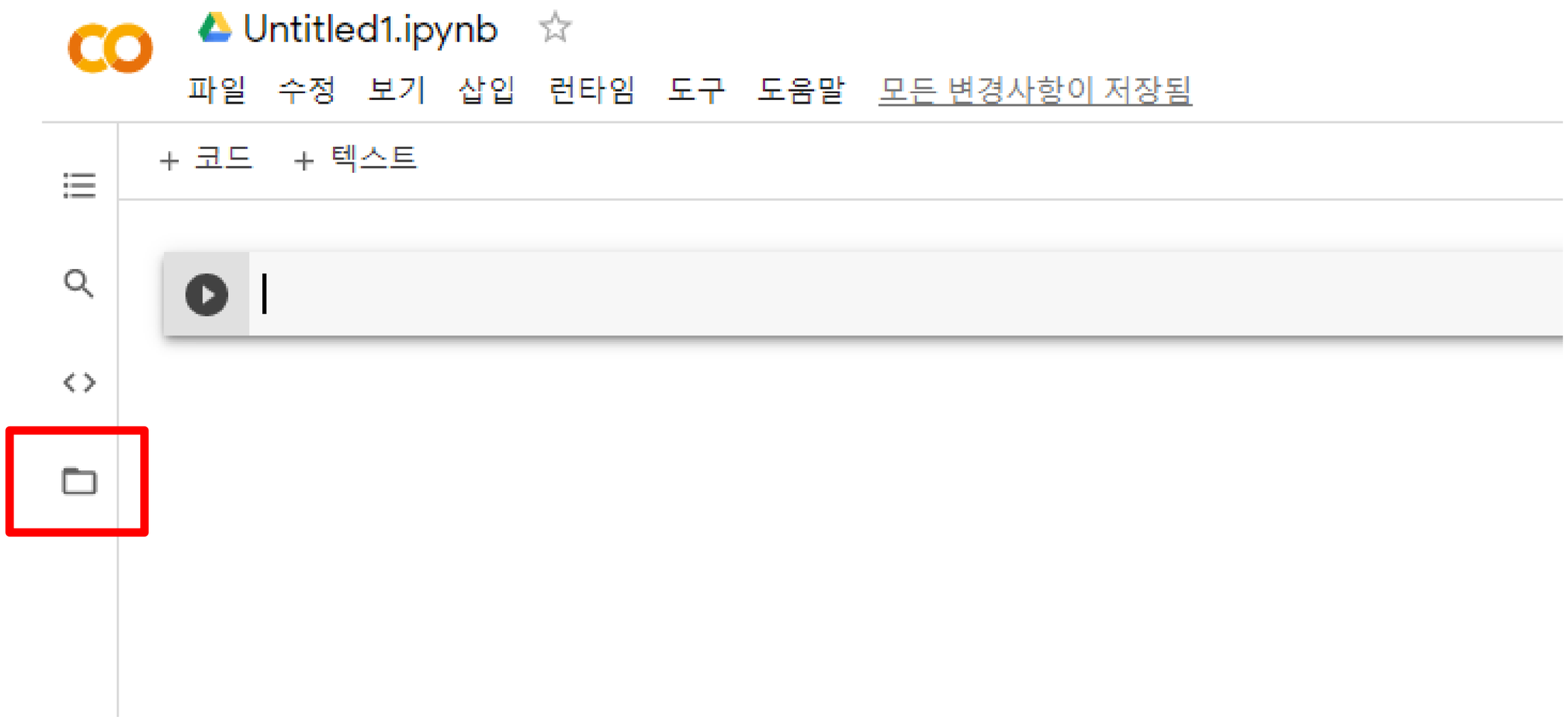


# Colab 시작하기

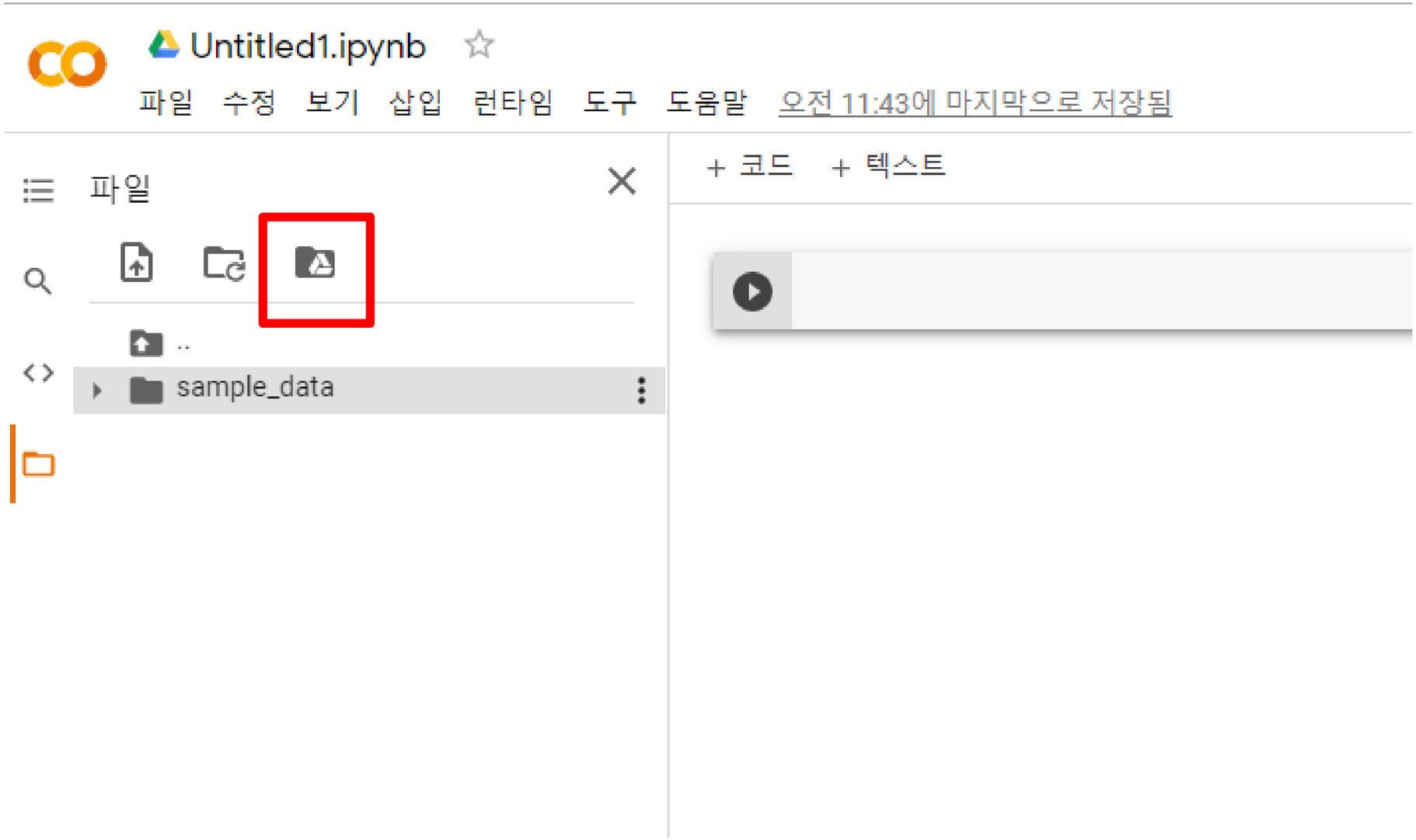
이제 Colab을 시작할 준비가 되었습니다.



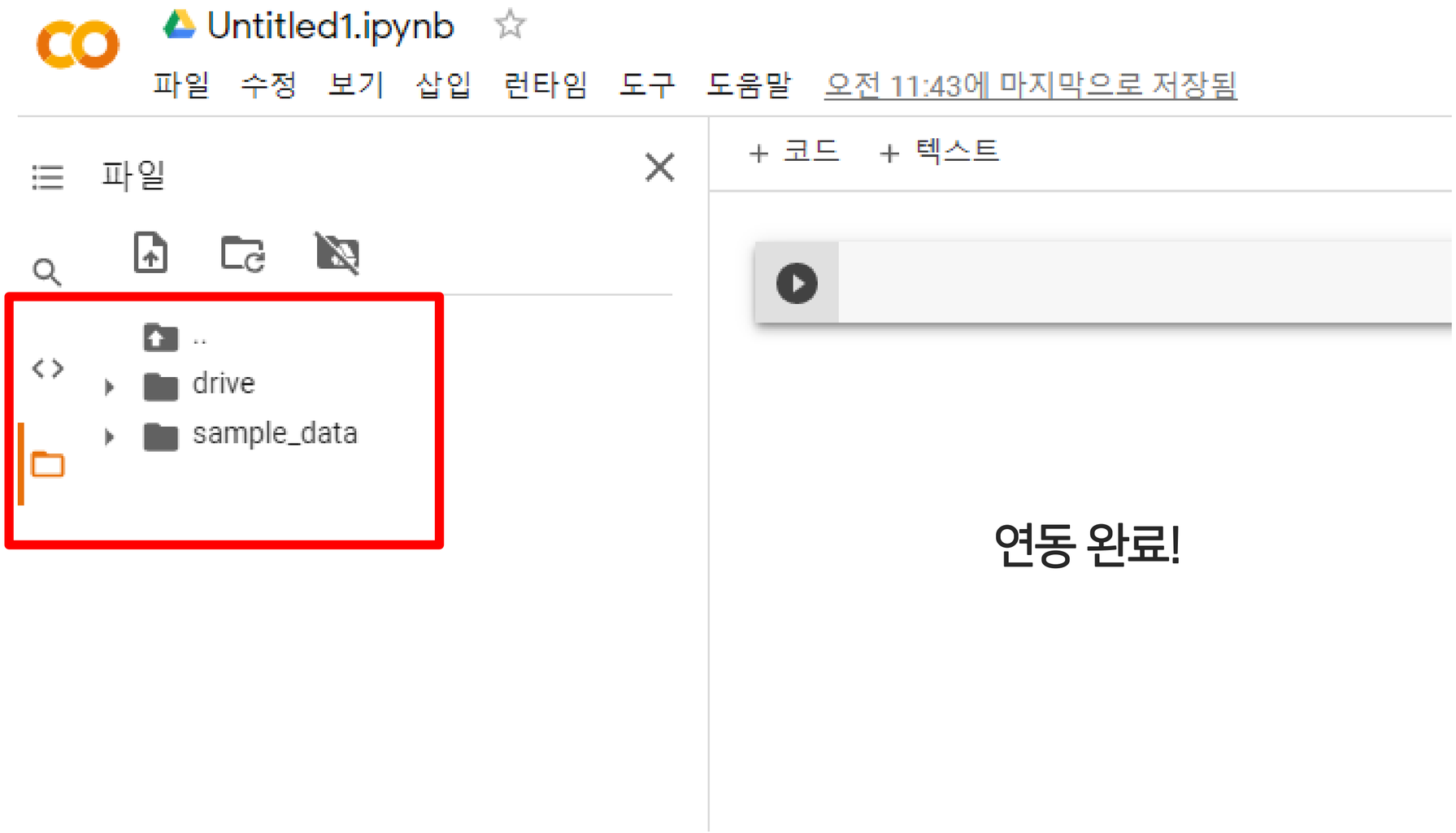
# 구글 드라이브 마운트



# 구글 드라이브 마운트


Untitled1.ipynb
파일 수정 보기 삽입 런타임 도구 도움말 오전 11:43에 마지막으로 저장됨
파일
+ 코드 + 텍스트
..
sample_data

# 구글 드라이브 마운트


Untitled1.ipynb
파일 수정 보기 삽입 런타임 도구 도움말 오전 11:43에 마지막으로 저장됨
파일
+ 코드 + 텍스트
..
drive
sample_data
연동 완료!

# CPU 정보 확인

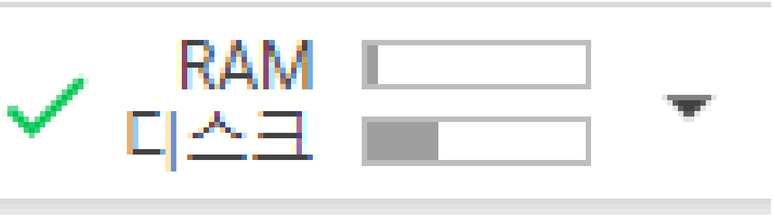
```
cat /proc/cpuinfo
```

```
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 63
model name      : Intel(R) Xeon(R) CPU @ 2.30GHz
stepping        : 0
microcode       : 0x1
cpu MHz         : 2299.998
cache size      : 46080 KB
physical id     : 0
siblings        : 2
core id         : 0
cpu cores       : 1
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
```

## 메모리 정보 확인

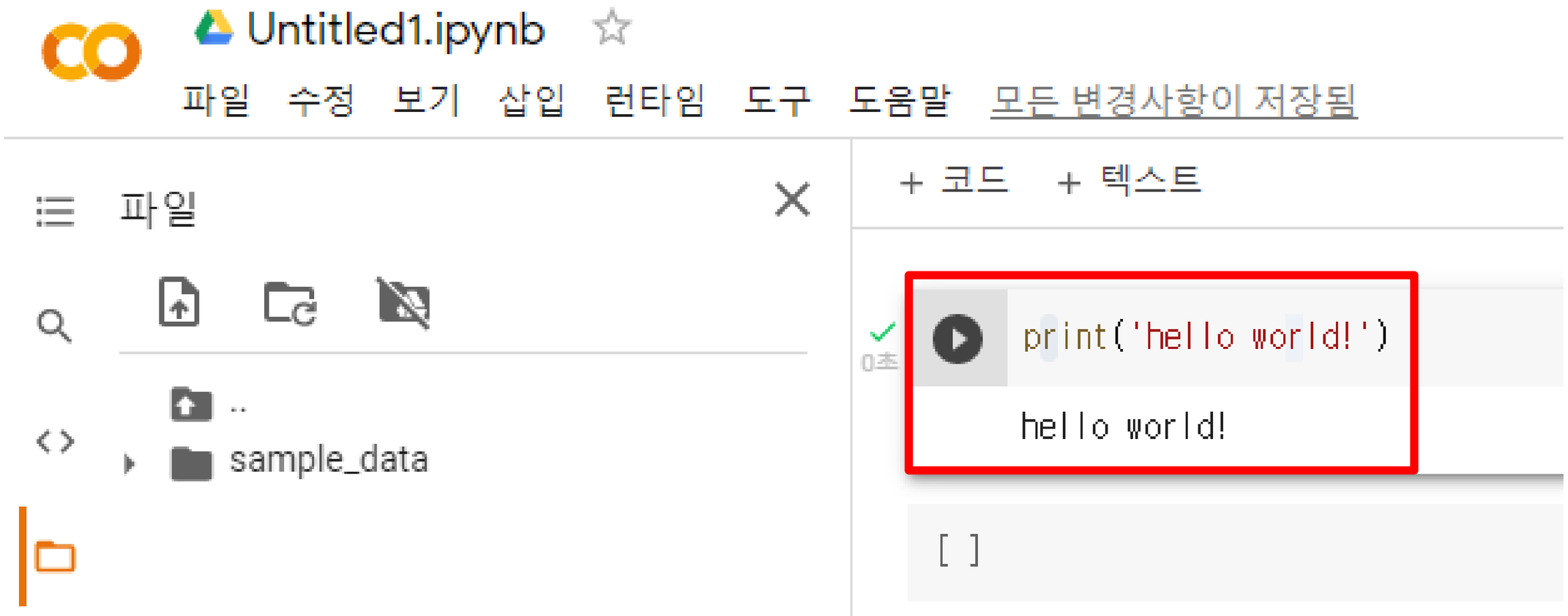
```
cat /proc/meminfo
```

```
MemTotal:       13302928 kB
MemFree:        10454700 kB
MemAvailable:   12497228 kB
Buffers:          124928 kB
Cached:          2045812 kB
SwapCached:            0 kB
Active:          1033924 kB
Inactive:        1575928 kB
Active(anon):     396604 kB
Inactive(anon):      424 kB
Active(file):     637320 kB
```



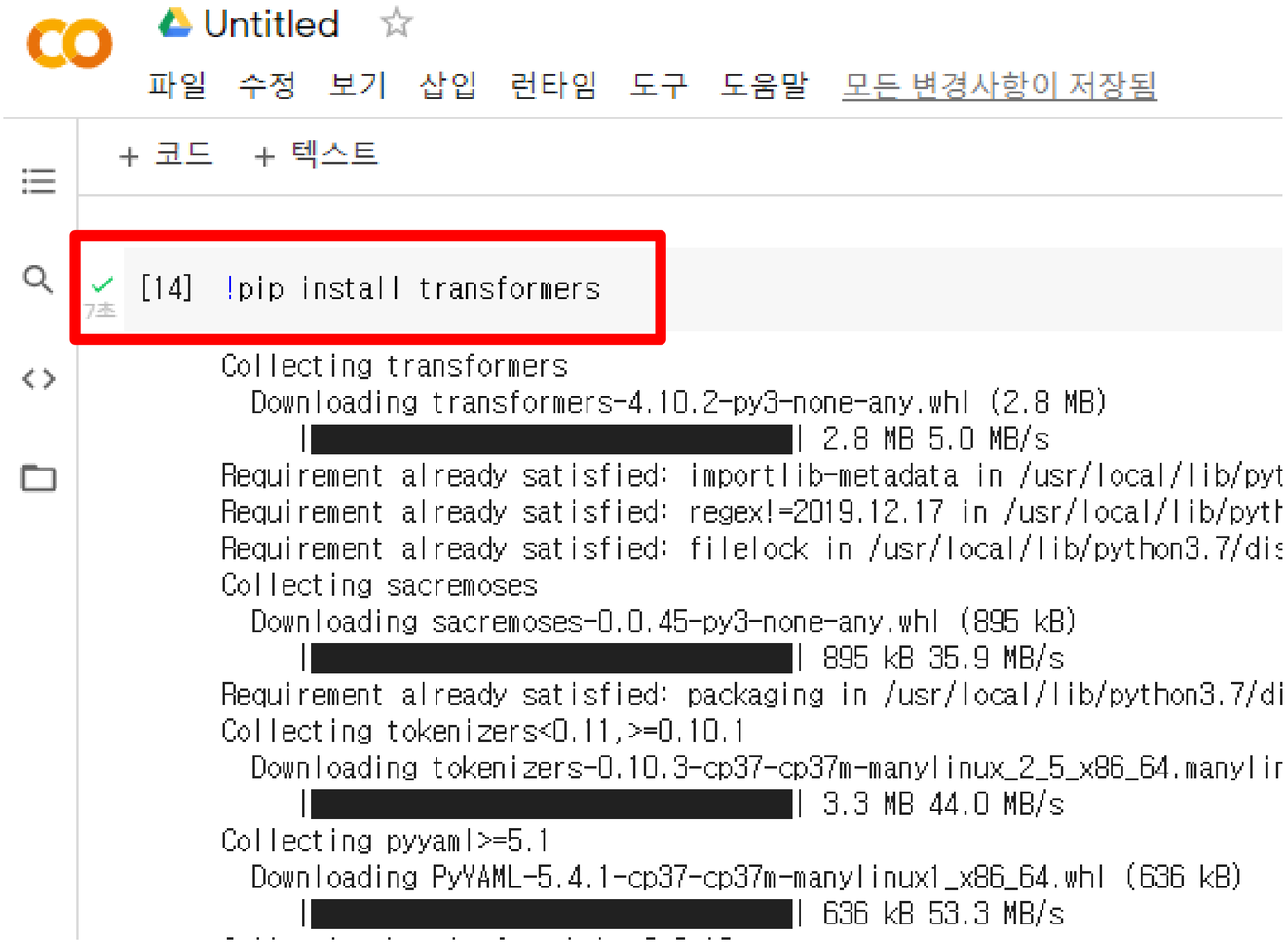

노트북 오른쪽 위를 보면 시각화되어 있음

## print("hello world!")



# Colab에 패키지 설치하기

패키지 설치 명령어 : !pip install



See you in the next lecture!

# 인공지능을 위한 파이썬 프로그래밍

## 03 자료형-1 (숫자형과 문자형)

Deep & High Learning

Copyright 2021. Deep&High learning. All rights reserved.

# Contents

## 자료형의 정의

자료형은 프로그래밍을 할 때 쓰이는 숫자, 문자열 등

**자료 형태로 사용하는 모든 것**을 뜻함

변수: 메모리에 값을 저장하기 위해 **할당하는 공간**

(할당 후 내부의 값 변경 가능)

```
x = 10
```

변수 이름 값

→ x 변수에 10이라는 값을 할당한다.

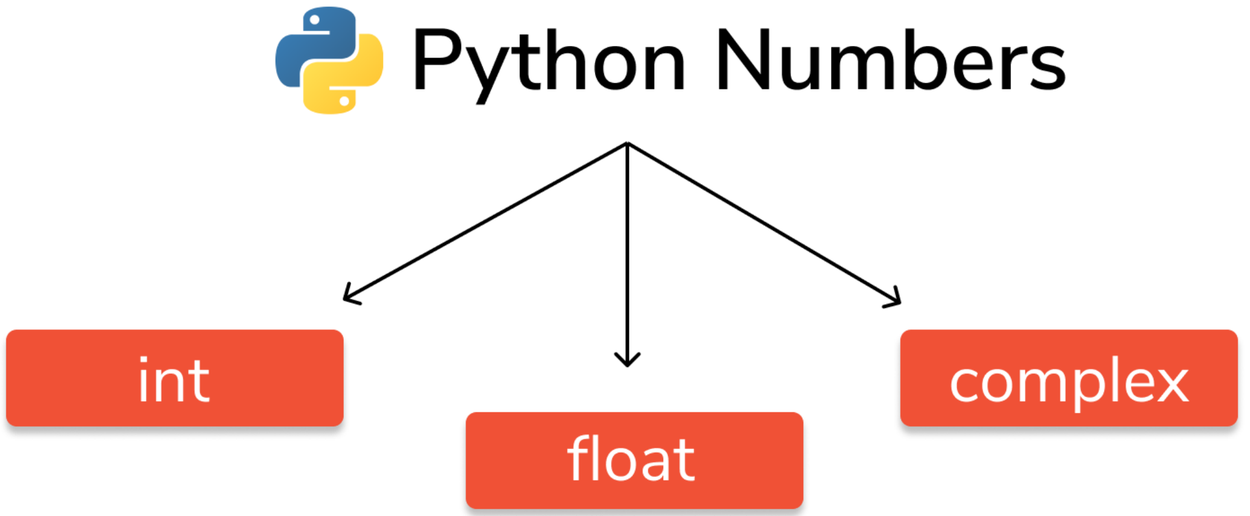
## 자료형의 종류

| | Example |
|---|---|
| Numbers | 1, 23, -34.22, 1e-05 |
| Boolean | True, False |
| String | '딥앤하이러닝', 'Deep&HighLearning' |
| List | [1,3,5], ['a', 'b', 'c'], [23.5, 'ai', True] |
| Tuple | (1,3,5), ('a', 'b', 'c'), (23.5, 'ai', True) |
| Dictionary | {'a':1, 'b':2, 'c':3, 'd':4} |
| Set | (kim, choi, chang, lee) |

## 숫자 자료형의 종류

Number : 숫자 형태로 이루어진 자료형

정수인 integer와 실수인 float, 복소수인 complex 세 가지로 구분



## int 자료형

int : 정수형. 범위는 -∞ ~ +∞

**int 선언**

```
a = 30
b = 2
c = 0
d = -12
print(a, b, c, d)
print(type(a))
```

**Result**

```
30 2 0 -12
<class 'int'>
```

## float 자료형

float : 실수형. 범위는 4.9×10^-324 ~ 1.8×10^308

**float 선언**

```
a = 3.4
b = -2.5
c = 3.5e-3
d = 2e+3
print(a, b, c, d)
print(type(a))
```

**Result**

```
3.4 -2.5 0.0035 2000.0
<class 'float'>
```

## int와 float 사이 형변환

float to int : 소수점이 버려짐

int to float : 뒤에 .0이 붙음

**float to int**

```
a = 3.7
b = int(a)
print(a, b)
```

**Result**

```
3.7 3
```

**int to float**

```
a = 3
b = float(a)
print(a, b)
```

**Result**

```
3 3.0
```

## complex 자료형

complex : 복소수형. 실수부와 허수부, 켤레복소수, 복소수의 크기, 복소수의 연산 등 수행 가능

**complex 선언**

```
a = 2 + 3j
b = complex(3, -4)
print(a, b)
```

**Result**

```
(2+3j) (3-4j)
```

**실수부, 허수부**

```
a = 2 + 3j
b = complex(3, -4)
print(a.real, a.imag,b.real, b.imag)
```

**Result**

```
2.0 3.0 3.0 -4.0
```

## complex 자료형

### 복소수 연산

```
a = 2 + 3j
b = complex(3, -4)
print(a + b, a - b)
print(a * b, a / b)
```

**Result**

```
(5-1j) (-1+7j)
(18+1j) (-0.24+0.68j)
```

### 켤레복소수와 크기

```
a = 2 + 3j
b = complex(3, -4)
print(a.conjugate(), b.conjugate())
print(abs(a), abs(b))
```

**Result**

```
(2-3j) (3+4j)
3.60555127546398 5.0
```

## 숫자형 연산자

숫자형 연산자

```
a = 8
b = 3
print(a + b, a - b)
print(a * b, a / b)
print(a ** b)
print(a % b)
print(a // b)
```

Result

```
11 5
24 2.6666666666666665
512
2
2
```

## 연습 문제!

언어 = 90, 영어 = 60, 수학 = 81

위 학생의 평균 성적을 구하는 코드를 작성하세요.

**Code**

```
kor = 90
eng = 60
mat = 81

...

print(average)
```

## string 자료형

문자열(String) : 문자, 단어 등으로 구성된 문자들의 집합

"" 나 '' 안에 문자열을 넣어 선언

**string 선언**

```
a = "딥앤하이러닝"
b = "딥앤하이\"러닝"
c = '딥앤하이"러닝'
d = "딥앤하이'러닝"
print(a, b)
print(c, d)
print(type(a))
```

**Result**

```
딥앤하이러닝 딥앤하이"러닝
딥앤하이"러닝 딥앤하이'러닝
<class 'str'>
```

## string 자료형

긴 문자열 : """ """ 나 ''' '''를 사용하여 표현

multiline

```
multiline = """
Life is too short
You need python
"""
print(multiline)
```

Result

```
Life is too short
You need python
```

## string 자료형

+와 * 연산자를 활용해서 문자열 반복 저장 가능

**multiline (+)**

```
print(1 + 1)
print('a' + 'b')
```

**Result**

```
2 ab
```

**multiline (*)**

```
a = "Deep " * 4
print(a)
```

**Result**

```
Deep Deep Deep Deep
```

## Offset (오프셋)

Offset (오프셋) : 컴퓨터 내 특정 주소로부터의 간격

문자열을 자르거나 특정 위치의 문자를 출력 가능

```
D e e p & H i g h L e a r n i n g
0 1 2 3 4 5 6     ...         15 16
```

## Offset (오프셋)

Indexing : 문자열의 특정 위치 문자 가져오기

Slicing : 문자열의 특정 부분 가져오기

오프셋 형태 : [start:end:stride]

양수는 첫째 문자부터 시작, 음수는 가장 뒤 문자부터 시작

**offset**

```
a = "abcdefghijk"
print(a[1:3], a[:5], a[-3:], a[:], a[::2])
```

**Result**

```
bc abcde ijk abcdefghijk acegik
```

## Offset (오프셋)

**offset 활용**

```
teacher = "Kim's "
title = "Deep&High Learning"
print(teacher + title)
print("=" * 30)
print(len(title))
print(title[0])
print(title[-1])
print(title[:2])
print(title[3:])
```

**Result**

```
Kim's Deep&High Learning
==============================
18
D
g
De
p&High Learning
```

## 주요 문자열 메서드 (method)

문자열에 여러 가지 변환을 가하는 데에 사용

count() : 문자열 갯수 리턴
find() : 해당 문자열 위치 리턴
(없으면 -1 리턴)
index() : 해당 문자열 위치 리턴
(없으면 value error 리턴)
upper() : 대문자로 변환
lower() : 소문자로 변환

strip() : 양쪽 공백 제거
lstrip() : 왼쪽 공백 제거
rstrip() : 오른쪽 공백 제거
replace() : 특정 문자열 치환
split() : 특정 문자열로 분리하여 리턴
join() : 문자열 리스트를 결합

## 주요 문자열 메서드 (method)

메서드 - 1

```
a = "apple"
print(a.count("p"))
print(a.find("p"))
print(a.index("p"))
print(".".join(a))
a = a.upper()
print(a)
print(a.lower())
```

Result

```
2
1
1
a.p.p.l.e
APPLE
apple
```

## 주요 문자열 메서드 (method)

메서드 - 2

```
b = " How can I improve my
coding skills? "
print(b)
b = b.strip()
print(b)
b = b.replace("?", "")
print(b)
word_list = b.split(" ")
print(word_list)
```

Result

```
 How can I improve my coding skills?
How can I improve my coding skills?
How can I improve my coding skills
['How', 'can', 'I', 'improve', 'my',
'coding', 'skills']
```

## 연습 문제!

1) Mary's cosmetics 을 출력하세요.

2) "dk2jd923i1jdk2jd93jfd92”의 길이를 구하세요.

3) t1 = 'python’, t2 = 'java’일 때 문자열 더하기와 곱하기를 이용하여 ”python java python java python java”를 출력 하세요.

4) id = "890910-1157963"에서 성별을 나타내는 수를 출력하세요.

5) license_plate = "24가 2210”에서 번호판 뒷자리만 출력하세요.

6) url = portal.ac.kr 에서 kr만 출력하세요. (split 함수 사용)

See you in the next lecture!

# 인공지능을 위한 파이썬 프로그래밍

04 자료형-2 (리스트와 튜플)

Deep & High Learning

Copyright 2021. Deep&High learning. All rights reserved.

# Contents

## 리스트의 정의

List : 순서가 있는 데이터들의 집합을 가지는 데이터 타입

## 리스트의 선언

대괄호([ ]) 를 사용하여 선언

원소로는 모든 데이터 타입 설정 가능

**list 선언**

```
a = ["deep", "and", "high"]
b = [1, 2, [3, 4]]
c = [1, "deep", True]
print(type(a))
print(a, b, c)
```

**Result**

```
<class 'list'>
['deep', 'and', 'high'] [1, 2, [3, 4]] [1, 'deep', True]
```

## 리스트의 인덱싱과 슬라이싱

**list 인덱싱**

```
a = ["deep", "and", "high"]
print(a[1], a[-1])
a[-2] = "pdj"
print(a)
```

**Result**

```
and high
['deep', 'pdj', 'high']
```

## 리스트의 인덱싱과 슬라이싱

**list 슬라이싱**

```
a = ["deep", "and", "high"]
print(a[1:3])
print(a[:])
print(a[::-2])
```

**Result**

```
['and', 'high']
['deep', 'and', 'high']
['high', 'deep']
```

## 리스트 메서드

리스트에 여러 가지 변환을 가하는 데에 사용

append() : 가장 마지막에 데이터 추가

sort() : 오름차순 정렬

reverse() : 순서 뒤집기

index() : 데이터 위치 리턴

insert() : 특정 위치에 데이터 추가

remove() : 해당되는 데이터 값을 삭제

pop() : 가장 마지막 값을 리턴하고
마지막 값을 삭제

extend() : 가장 마지막에 데이터 추가
(데이터 내부 원소를 추가)

## 리스트 메서드

### append()

```
a = ["deep", "and", "high", "learning"]
a.append("fighting!")
print(a)
```

### Result

```
['deep', 'and', 'high', 'learning', 'fighting!']
```

## 리스트 메서드

**reverse()**

```
a = ["deep", "and", "high", "learning"]
a.reverse()
print(a)
```

**Result**

```
['learning', 'high', 'and', 'deep']
```

## 리스트 메서드

**sort()**

```
a = ["deep", "and", "high", "learning"]
a.sort()
print(a)
```

**Result**

```
['and', 'deep', 'high', 'learning']
```

## 리스트 메서드

**index()**

```
a = ["deep", "and", "high", "learning"]
print(a.index('high'))
```

**Result**

```
2
```

## 리스트 메서드

### insert()

```
a = ["deep", "and", "high", "learning"]
a.insert(2, 'index1')
a.insert(5, 'index2')
print(a)
```

### Result

```
['deep', 'and', 'index1', 'high', 'learning', 'index2']
```

## 리스트 메서드

**remove()**

```
a = ["deep", "and", "high", "learning"]
a.remove('high')
print(a)
```

**Result**

```
['deep', 'and', 'learning']
```

## 리스트 메서드

### pop()

```
a = ["deep", "and", "high", "learning"]
print(a.pop())
```

### Result

```
learning
```

## 리스트 메서드

**remove()**

```
a = ["deep", "and", "high", "learning"]
a.remove('high')
print(a)
```

**Result**

```
['deep', 'and', 'learning', 'high']
```

## 리스트 메서드

**del**

```
a = ["deep", "and", "high", "learning"]
del a[2]
print(a)
```

**Result**

```
['deep', 'and', 'learning']
```

## 리스트 메서드

**extend()**

```
a = ["deep", "and", "high", "learning"]
b = ['artificial', 'intelligence']
a.extend(b)
print(a)
```

**Result**

```
['deep', 'and', 'high', 'learning', 'artificial', 'intelligence']
```

## 리스트 메서드

### append와의 비교

```
a = ["deep", "and", "high", "learning"]
b = ['artificial', 'intelligence']
a.append(b)
print(a)
```

### Result

```
['deep', 'and', 'high', 'learning', ['artificial', 'intelligence']]
```

## 리스트 원소 수정

오프셋을 사용하여 리스트의 데이터 수정 가능

**리스트 원소 수정**

```
a = [1, 2, 3, 4, 5]
a[1] = "Deep"
print(a)
a[1:4] = "Hear"
print(a)
```

**Result**

```
[1, 'Deep', 3, 4, 5]
[1, 'H', 'e', 'a', 'r', 5]
```

## 리스트 복사 : 얕은 복사와 깊은 복사

b = a 형태로 복사를 하게 되면, a를 변경하게 되면 b도 변경됨 (얕은 복사)

별도의 저장공간을 가지게 하려면 copy() 함수 사용 (깊은 복사)

**얕은 복사**

```
a = [1, 2, 3]
b = a
print(a, b)
a[2] = 4
print(a, b)
```

**Result**

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

a만 바꿨는데 b의 데이터도 함께 수정됨

## 리스트 복사 : 얕은 복사와 깊은 복사

b = a 형태로 복사를 하게 되면, a를 변경하게 되면 b도 변경됨 (얕은 복사)

별도의 저장공간을 가지게 하려면 copy() 함수 사용 (깊은 복사)

**깊은 복사**

```
a = [1, 2, 3]
c = a.copy()
print(a, c)
a[2] = 5
print(a, c)
```

**Result**

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

a만 바뀌고 c는 불변

## 연습 문제!

1) language1 = ["C", "C++", "JAVA"], language2 = ["Python", "Go", "C#"]
두 리스트의 원소를 모두 갖는 languages를 만드세요.

2) nums = [12, 245, 33, 77, 858]의 평균을 구하세요.

3) a = ["b", "a", "d", "c"] 리스트를 알파벳 순으로 정렬하세요.

## 튜플의 정의

Tuple : List와 같이 순서가 있는 데이터 타입이지만 데이터를 변경할 수 없음

List보다 컴퓨터의 자원(메모리)을 적게 사용

## 튜플의 선언

튜플은 ','(콤마) 로 구분하여 선언 또는 ',' 로 구분하고 **괄호로 묶어서** 선언

원소로는 모든 데이터 타입 설정 가능

튜플도 문자열이나 리스트와 같이 **오프셋 사용 가능**

## 튜플의 선언

**Tuple 선언**

```
a = 1, 2, 3, 4
b = "deep", "and", "high", "learning"
c = (1, "fast", True)
print(type(a), type(b), type(c))
print(a, b, c)
```

**Result**

```
<class 'tuple'> <class 'tuple'> <class 'tuple'>
(1, 2, 3, 4) ('deep', 'and', 'high', 'learning') (1, 'fast', True)
```

## 튜플의 활용

### 인덱싱과 슬라이싱

```
a = 1, 2, 3, 4
print(a[2], a[1:3])
```

**Result**

```
3 (2, 3)
```

### 데이터 수정 불가

```
a = 1, 2, 3, 4
a[1] = 3
```

**Result**

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

## 튜플의 활용

### 리스트와 메모리 크기 비교

```
import sys
# getsizeof의 단위 : byte
ls = [1, 2, 3, 4, 5]
print(type(ls), sys.getsizeof(ls), "byte")
t = tuple(ls)
print(type(t), sys.getsizeof(t), "byte")
```

### Result

```
<class 'list'> 96 byte
<class 'tuple'> 80 byte
```

See you in the next lecture!

# 인공지능을 위한 파이썬 프로그래밍

05 자료형-3 (딕셔너리와 집합)

Deep & High Learning

Copyright 2021. Deep&High learning. All rights reserved.

# Contents

## 딕셔너리의 정의

데이터의 순서가 없고 key와 value의 쌍으로 데이터가 모여 있는 데이터 타입.

## 딕셔너리의 선언

{} 기호 안에 키 : 값 형태로 선언( { 키:값 } )

키값에는 정수나 문자열의 데이터 타입 사용 가능.

**dictionary 선언**

```
dic = {
    1: "one",
    "A": ["data", "science"],
    "숫자": 1234,
}
print(type(dic))
print(dic)
```

**Result**

```
<class 'dict'>
{1: 'one', 'A': ['data', 'science'],
'숫자': 1234}
```

## 딕셔너리 데이터 수정

키 값으로 데이터 수정 가능

**list 인덱싱**

```
dic = {
    1: "one",
    "A": ["data", "science"],
    "숫자": 1234,
}
print(dic["숫자"])
dic[1] = "하나"
dic["A"] = "알파벳"
print(dic)
```

**Result**

```
1234
{1: '하나', 'A': '알파벳', '숫자': 1234}
```

## 딕셔너리 데이터 삭제

del을 사용하여 삭제 (키 값으로 접근)

딕셔너리 데이터 삭제

```
dic = {
    1:"one",
    2:"two",
}
print(dic)

del dic[1]
print(dic)
```

Result

```
{1: 'one', 2: 'two'}
{2: 'two'}
```

## 딕셔너리 메서드

딕셔너리 메서드를 이용해 딕셔너리 데이터를 가공

keys() : 키를 리턴

values() : 값을 리턴

items() : 키와 값을 리턴

clear() : dictionary 데이터를 모두 삭제

get() : 매개변수에 해당하는 값을 리턴

copy() : 리스트와 마찬가지로 다른 저장공간을 가지는 데이터를 대입

## 딕셔너리 메서드

딕셔너리 값 접근 및 전체 삭제

메서드-1

```
dic = {
    1: "one",
    "A": [“deep", “learning"],
    "숫자": 1234,
}
print(dic.keys())
print(dic.values())
print(dic.items())
dic.clear()
print(dic)
```

Result

```
dict_keys([1, 'A', '숫자'])
dict_values(['one', ['deep',
'learning'], 1234])
dict_items([(1, 'one'), ('A',
['deep', 'learning']), ('숫자',
1234)])
{}
```

## 딕셔너리 복사

깊은 복사를 위해서는 copy 함수 사용

**메서드-1**

```
dic = {
    1: "one",
    "A": [“deep", “learning"],
    "숫자": 1234,
}
dic2 = dic
dic3 = dic.copy()
dic[1] = "하나"
print(dic)
print(dic2)
print(dic3)
```

**Result**

```
{1: '하나', 'A': ['deep', 'learning'],
'숫자': 1234}
{1: '하나', 'A': ['deep', 'learning'],
'숫자': 1234}
{1: 'one', 'A': ['deep', 'learning'],
'숫자': 1234}
```

## 연습 문제!

```
name_to_age = {"Jenny": 20, "Ella":31}
```

name_to_age의 key는 이름, value는 나이를 나타냅니다.

1) name_to_age에 26살의 John, 29살의 Tom에 대한 정보를 추가하세요.
2) Jenny의 나이를 21살로 바꾸세요.
3) name_to_age의 구성원들이 가지는 나이를 전부 출력하세요.

## 집합의 정의 및 특성

중복되는 데이터가 없는 데이터 타입

교집합, 합집합, 차집합과 같은 집합의 연산 가능

리스트 데이터에서 중복을 제거할 때 사용

딕셔너리와 같이 순서가 없는 데이터 타입

특정 인덱스 값을 가져오거나 슬라이싱으로 데이터 수정이 불가능

## 집합의 선언

집합은 리스트 형태의 데이터에 set() 으로 형변환을 해주는 방법으로 선언

중복된 데이터는 제거됨

**집합의 선언**

```
ls = [1, 2, 3, 4, 5, 1, 2, 3]
s = set(ls)
print(type(s), s)
```

**Result**

```
<class 'set'> {1, 2, 3, 4, 5}
```

## 집합의 연산

### 교집합 (Intersection)

```
s1 = set([1, 2, 3, 4])
s2 = set([3, 4, 5, 6])
print(s1 & s2)
print(s1.intersection(s2))
```

### Result

```
{3, 4}
{3, 4}
```

## 집합의 연산

### 합집합 (Union)

```
s1 = set([1, 2, 3, 4])
s2 = set([3, 4, 5, 6])
print(s1 | s2)
print(s1.union(s2))
```

### Result

```
{1, 2, 3, 4, 5, 6}
{1, 2, 3, 4, 5, 6}
```

## 집합의 연산

### 차집합 (difference)

```
s1 = set([1, 2, 3, 4])
s2 = set([3, 4, 5, 6])
print(s1 - s2)
print(s1.difference(s2))
```

### Result

```
{1, 2}
{1, 2}
```

## 집합의 형변환

list로 형변환 한 후 다시 set으로 변환

**집합의 형변환**

```
ls = [1, 2, 3, 4, 5, 1, 2, 3]
s = set(ls)
s = list(s)
s[4] = 10
s = set(s)
print(s)
```

**Result**

```
{1, 2, 3, 4, 10}
```

See you in the next lecture!

# 인공지능을 위한 파이썬 프로그래밍

06 형변환 및 연산자

Deep & High Learning

Copyright 2021. Deep&High learning. All rights reserved.

# Contents

## 형변환이란?

데이터를 다른 데이터 타입으로 전환하는 것

하지만 데이터 타입에 따라서 형변환이 가능하기도, 불가능하기도 함

## 문자열→숫자 형변환

숫자로 이루어진 문자열은 숫자로 형변환 가능

**string to int**

```
s = "1234"
n = int(s)
print(type(n), n)
```

**Result**

```
<class 'int'> 1234
```

**string to float**

```
s = "1234.5678"
n = float(s)
print(type(n), n)
```

**Result**

```
<class 'float'> 1234.5678
```

## 문자열→숫자 형변환

숫자문자가 아닌 문자열을 숫자로 형변환하는 것은 불가능

**string to int error**

```
s = "a"
n = int(s)
print(type(n), n)
```

**Result**

```
ValueError: Invalid literal for
int() with base 10: ‘a’
```

## 숫자→문자열 형변환

숫자는 문자열로 자유롭게 형변환 가능

**int to string**

```
number = 123
s = str(number)
type(s), s
```

**Result**

```
<class 'str'> 123
```

## Boolean 형변환

문자열은 “”이 False, 숫자는 0이 False, 리스트에서는 []가 False

나머지 값들은 모두 True

**Boolean 형변환**

```
print(bool(""), bool("asedf"))
print(bool(-1), bool(0), bool(1))
print(bool([]), bool([1, 2]))
```

**Result**

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

## 문자열→리스트 형변환

문자 하나씩 순서대로 리스트의 데이터로 형변환

**string to list**

```
s = "deep"
ls = list(s)
print(ls)
```

**Result**

```
['d', 'e', 'e', 'p']
```

## 튜플→딕셔너리 형변환

(key, value) 형태의 나열로 되어 있는 튜플의 데이터는 딕셔너리로 형변환이 가능

**tuple to dictionary**

```
t = ((1, "one"), (2, "two"))
print(dict(t))
```

**Result**

```
{1: 'one', 2: 'two'}
```

## 딕셔너리→튜플 형변환

키 데이터만 형변환

key와 value 데이터를 모두 사용하여 형변환 하려면 items() 함수 사용

**dictionary to tuple**

```
d = {1:"one", 2:"two"}
t = tuple(d)
print(t)
```

**Result**

```
(1, 2)
```

**dictionary to tuple (items)**

```
d = {1:"one", 2:"two"}
t = tuple(d.items())
print(t)
```

**Result**

```
((1, 'one'), (2, 'two'))
```

## 연산자의 종류

1. 산술 연산자 (Arithmetic Operators) : +, -, *, /, //, %, **
2. 비교 연산자 (Comparison Operators) : ==, !=, >, <, >=, <=
3. 할당 연산자 (Assignment Operators) : =, +=, -=, *=, /=, %=, //=, **=
4. 논리 연산자 (Logical Operators) : and, or, not
5. 멤버 연산자 (Membership Operators) : in, not in
6. 식별 연산자 (Identity Operators) : is, is not

## 산술 연산자 (+, -, *, /, //, %, **)

산술 연산자

```
print(2 + 4)
print(2 - 4)
print(2 * 4)
print(10 / 3)
print(10 // 3) # 몫
print(10 % 3)  # 나머지
print(3 ** 4)  # 제곱
```

Result

```
6
-2
8 3.3333333333333335
3
1
81
```

## 비교 연산자 (==, !=, >, <, >=, <=)

비교 연산자

```
a, b = 10, 20
print(a == b)
print(a != b)
print(a > b)
print(a < b)
print(a >= b)
print(a <= b)
```

Result

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

## 할당 연산자 (=, +=, -=, *=, /=, %=, //=, **=)

할당 연산자-1

```
a = 5
print(a)
a += 10
print(a)
a -= 5
print(a)
a *= 10
print(a)
```

Result

```
5
15
10
100
```

## 할당 연산자 (=, +=, -=, *=, /=, %=, //=, **=)

**할당 연산자-2**

```
a = 50
print(a)
a /= 5
print(a)
a %= 8
print(a)
a //= 2
print(a)
a **= 3
print(a)
```

**Result**

```
50
10.0
2.0
1.0
1.0
```

## 논리 연산자 (and, or, not)

and : 둘다 True이면 True
or : 둘 중 하나가 True이면 True
not : True를 False로, False를 True로 변환

**논리 연산자**

```
print(True and True)
print(True and False)
print()
print(True or True)
print(True or False)
print()
print(not(True and True))
print(not(True and False))
```

**Result**

```
True
False

True
True

False
True
```

## 멤버 연산자 (in, not in)

**멤버 연산자**

```
a = [1, 2, 3, 4]
print(1 in a)
print(5 in a)
print(1 not in a)
a = {1:"one"}
print(1 in a)
```

**Result**

```
True
False
False
True
```

## 식별 연산자 (is, is not)

식별 연산자-1

```
a = 1
b = a
print(a is b)
a = 1
b = 1
print(a is b)
print(a is not b)
```

Result

```
True
True
False
```

## 식별 연산자 (is, is not)

식별 연산자-2

```
a = "python"
b = a
print( a is b )

a = "python"
b = "python"
print( a is b )
print( a is not b )
```

Result

```
True
True
False
```

See you in the next lecture!

# 인공지능을 위한 파이썬 프로그래밍

07 Mutable vs Immutable

Deep & High Learning

Copyright 2021.  Deep&High learning. All rights reserved.

## Mutable과 Immutable의 정의

Immutable : (값이) 변하지 않는

Mutable : (값이) 변하는

## 각 자료형의 소속

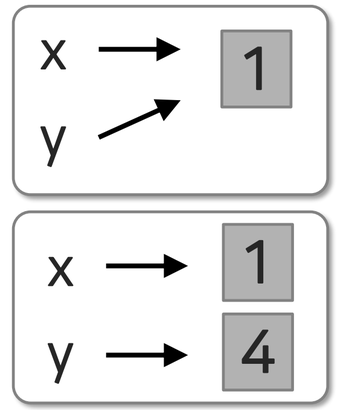
Immutable : 숫자(number), 문자열(string), 튜플(tuple), 논리(Boolean)

Mutable : 리스트(list), 딕셔너리(dictionary), 집합(set)

## Mutable과 Immutable의 정의

**Immutable**

```
x = 1
y = x
y += 3
print(x, y)
```



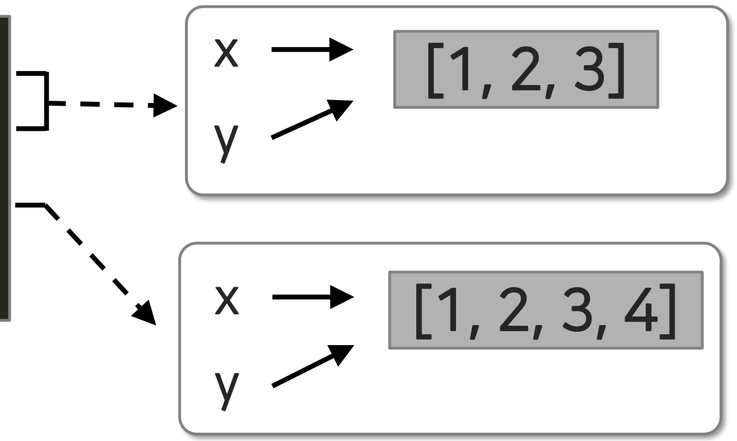

**Result**

```
14
```

## Mutable과 Immutable의 정의

**Mutable**

```
x = [1, 2, 3]
y = x
y += [4]
print(x, y)
```




**Result**

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

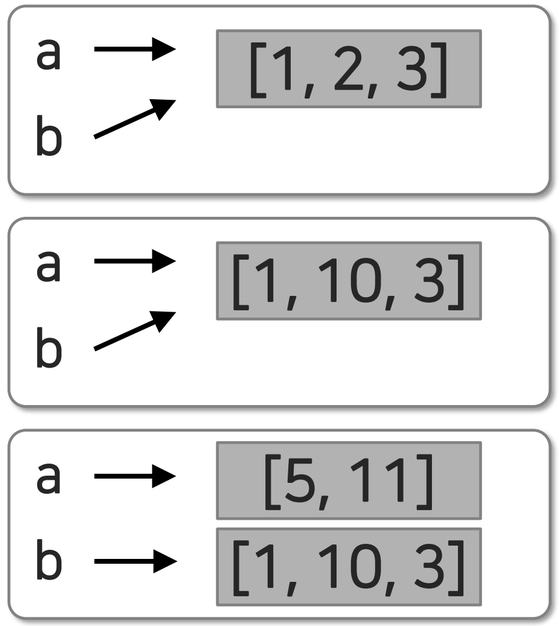
## 얕은 복사와 깊은 복사의 정확한 이해

얕은 복사

a를 새롭게 정의
b와 연결 끊어짐

**얕은 복사 (Shallow copy)**

```
a = [1, 2, 3]
b = a
print(b is a)
b[1] = 10
a = [5, 11]
print(b is a)
print(a, b)
```




**Result**

```
True
False
[5, 11] [1, 10, 3]
```

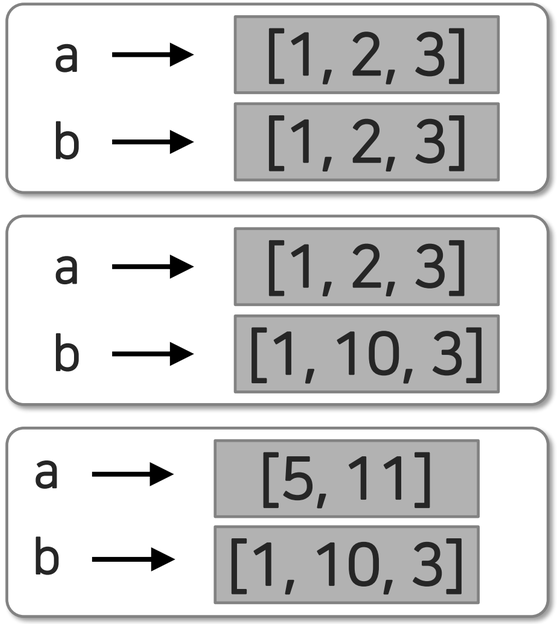
## 얕은 복사와 깊은 복사의 정확한 이해

깊은 복사
a와 b는 애초에 다름

a를 새롭게 정의

**깊은 복사 (Deep copy)**

```
a = [1, 2, 3]
b = a.copy()
print(b is a)
b[1] = 10
a = [5, 11]
print(b is a)
print(a, b)
```




**Result**

```
False
False
[5, 11] [1, 10, 3]
```

## 얕은 복사와 깊은 복사의 정확한 이해

**Mutable-list**

```
a = [1, 2, 3]
b = a
print(a is b)

a = [1, 2, 3]
b = [1, 2, 3]
print(a is b)

a = [1, 2, 3]
b = a.copy()
print(a is b)
```

**Result**

```
True
False
False
```

## 얕은 복사와 깊은 복사의 정확한 이해

**Mutable-set**

```
a = set([1, 2, 3])
b = a
print(a is b)

b |= set([4, 5])
print(b)
print(a is b)
```

**Result**

```
True
{1, 2, 3, 4, 5}
True
```

## 얕은 복사와 깊은 복사의 정확한 이해

**Mutable-dictionary**

```
a = {"a":1, "b":2}
b = a
print(a is b)

a = {"a":1, "b":2}
b = {"a":1, "b":2}
print(a is b)

a = {"a":1, "b":2}
b = a.copy()
print(a is b)
```

**Result**

```
True
False
False
```

## 얕은 복사와 깊은 복사의 정확한 이해

**Immutable-tuple**

```
a = (1, 2, 3)
b = a
print(a is b)

a = (1, 2, 3)
b = (1, 2, 3)
print(a is b)

# tuple은 copy 메소드 없음
```

**Result**

```
True
False
```

See you in the next lecture!

# 인공지능을 위한 파이썬 프로그래밍

## 08 조건문과 반복문

Deep & High Learning

Copyright 2021. Deep&High learning. All rights reserved.

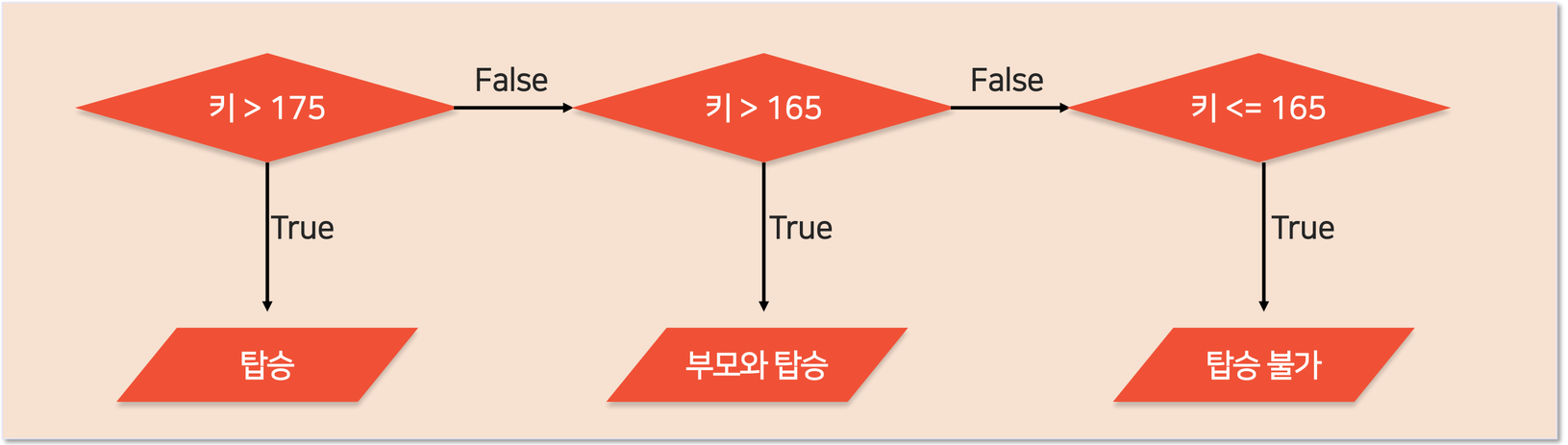
# Contents

## 조건문의 정의

조건에 따라 다른 코드를 실행할 수 있는 제어문

의도한 상황에 따라 자유롭게 분기 가능

If, elif, else 등의 키워드로 구현된다



## 조건문의 용법 - if/else

**If/else 선언**

```
if height > 175:
    print(“175cm 이상입니다. 탑승하세요”)

else:
    print(“고객님은 탑승할 수 없습니다.”)
```

**Result (height=180)**

```
175cm 이상입니다. 탑승하세요.
```

**Result (height=170)**

```
고객님은 탑승할 수 없습니다.
```

**else**: if 조건문 뒤에 오며 단독으로는 사용할 수 없음

## 조건문의 용법 - elif

**elif 추가**

```
if height > 175:
    print("175cm 이상입니다. 탑승하세요")

elif height > 165:
    print("부모님과 탑승하세요.")

else:
    print("고객님은 탑승할 수 없습니다.")
```

**Result (height=170)**

```
부모님과 탑승하세요.
```

**Result (height=160)**

```
고객님은 탑승할 수 없습니다.
```

**elif**: else와 마찬가지로 if 조건문 뒤에 오며 단독으로 사용할 수 없음

## 조건문의 용법 - 중첩

**If문 중첩**

```
if height > 175:
    print("175cm 이상입니다. 탑승하세요")

elif height > 165:
    if with_parent:
        print("부모님과 탑승하세요.")
    else:
        print("부모님을 모셔오세요.")

else:
    print("고객님은 탑승할 수 없습니다.")
```

**Result (height=170, with_parent=True)**

부모님과 탑승하세요.

**Result (height=170, with_parent=False)**

부모님을 모셔오세요.

**Result (height=160, with_parent=True)**

고객님은 탑승할 수 없습니다.

## 조건문의 용법 - 한 줄 조건문

**한 줄 조건문**

```
print(“175cm 이상입니다. 탑승하세요”) if height > 175 else print(“탑승할 수 없습니다.”)
```

**Result (height=180)**

```
175cm 이상입니다. 탑승하세요
```

else문은 필요하지 않다면 적지 않을 수 있음

## 반복문의 정의

프로그램이 특정 부분을 반복 실행하도록 제어하는 명령문

for, while 등의 키워드로 구현된다

```
while 불린(Boolean) 값으로 계산되는 조건:
    반복할 코드 블럭
```

## 반복문의 용법 - while

**while 반복문**

```
a = 3
b = []
while a:
    a -= 1
    b.append(a)
    print(a)
print(b)
```

**Result**

```
2
1
0
[2, 1, 0]
```

boolean 조건으로 걸린 a가 0(False) 되는 시점에 while 조건문 종료

## 반복문의 제어 - continue

**continue 제어**

```
a = 3
b = []
while a:
    a -= 1
    if a == 2:
        continue
    b.append(a)
    print(a)
print(b)
```

**Result**

```
1
0
[1, 0]
```

continue를 만나면 이하 코드를 실행하지 않고 다시 반복문 코드 라인으로 이동

## 반복문의 제어 - break

**break 제어**

```
a = 5
b = []
while a:
    a -= 1
    if a == 1:
        break
    b.append(a)
    print(a)
print(b)
```

**Result**

```
4
3
2
[4, 3, 2]
```

break 제어를 만나면 가장 가까운 반복문 탈출

## 반복문 활용 예시 - while

**반복문 활용**

```
a = 10
b = []
while a:
    if a == 2:
        break
    elif a % 2 == 1:
        a -= 1
        continue
    else:
        b.append(a)
        a -= 1
        print(a)
print(b)
```

**Result**

```
9
7
5
3
[10, 8, 6, 4]
```

반복문 **탈출 조건**을 적절히 선언하지 않으면 무한 loop에 빠질 수 있음

## 반복문의 용법 - for

리스트와 같은 데이터 시퀀스에 하나씩 접근하며 원하는 명령을 반복할 때 쓰임

**리스트 순회**

```
a = [1, 2, 3, 4, 5, 6, 7, 8]
for number in a:
        print(number)
```

**Result**

```
1
2
3
4
5
6
7
8
```

## 반복문의 용법 - for

딕셔너리를 순회하며 key와 value에 접근할 수 있음

**딕셔너리 순회**

```
dic = {“수학”: 100, “영어”: 90}
for key, val in dic.items():
        if val == 100:
            print(key, val, “굳굳”)
        else:
            print(key, val, “분발하세요!”)
```

**Result**

```
수학 100 굳굳
영어 90 분발하세요!
```

## 반복문의 용법 - for

range 함수를 활용해 for문을 선언할 수 있음

**range 함수 활용 반복문**

```
for val in range(1, 10, 1):
    if val % 2 == 0:
            print(val, "-> 짝수입니다.")
```

**Result**

```
2 -> 짝수입니다.
4 -> 짝수입니다.
6 -> 짝수입니다.
8 -> 짝수입니다.
```

range(1, 10, 1) 함수의 첫번째 인자(1) 부터 두번째 인자(10) 미만까지

세번째 인자(1) 만큼 늘려가며 반복

## 반복문의 용법 - for

enumerate 함수를 활용해 리스트의 순서와 값에 동시에 접근할 수 있음

**enumerate 활용 반복문**

```
subjects = [“수학”, “영어“, “국어”]
for index, val in enumerate(subjects):
    print(index, val)
```

**Result**

```
0 수학
1 영어
2 국어
```

## 반복문의 용법 - zip

zip 함수를 활용해 여러 리스트의 값을 묶어 순회할 수 있음

**zip 활용 반복문**

```
subjects = [“수학”, “영어“, “국어”, “탐구”]
scores = [100, 90, 80]
for subj, scr in zip(subjects, scores):
    print(subj, scr)
```

**Result**

```
수학 100
영어 90
국어 80
```

묶이는 리스트의 길이가 다를 경우, 가장 짧은 리스트 기준으로 묶고 이후 데이터 버림

## 반복문의 용법 - zip

zip 함수를 활용해 리스트 쌍을 딕셔너리로 변환할 수 있음

**zip 활용 딕셔너리화**

```
subjects = [“수학”, “영어“, “국어”, “탐구”]
scores = [100, 90, 80]
to_dict = dict(zip(subjects, scores))
print(to_dict)
```

**Result**

```
{‘수학’: 100, ‘영어‘: 90, ‘국어’: 80}
```

묶이는 리스트의 길이가 다를 경우, 가장 짧은 리스트 기준으로 묶고 이후 데이터 버림

## 반복문의 용법 - 리스트 내포

반복문의 결과를 바로 리스트로 생성하는 방법

**리스트 내포**

```
ls = [num*10 for num in range(1, 8)]
print(ls)
```

**Result**

```
[10, 20, 30, 40, 50, 60, 70
```

단순히 for문을 반복해 append 하는 방식보다 빠름 (function call 최소화)

## 반복문의 용법 - 리스트 내포

리스트 내포에 조건문을 활용할 수 있음

**리스트 내포 조건문**

```
ls = [num*10 for num in range(1, 6) if num % 2 == 0]
print(ls)
```

**Result**

```
[20, 40]
```

1~5까지 순회하며 짝수인 경우 10을 곱해 리스트에 저장하는 코드

# 연습 문제

1) while문을 사용해 1부터 1000까지의 자연수 중 3의 배수의 합을 구하세요.

2) while 문을 사용해 다음과 같이 *들을 출력해보세요.

```
*
**
***
****
*****
```

3) 아래 코드를 리스트 내포를 이용해 한 줄로 구현해보세요.

```
numbers = [1, 2, 3, 4, 5]
  result = []
  for n in numbers:
      if n % 2 == 0:
          result.append(n + 2)
```

See you in the next lecture!

# 인공지능을 위한 파이썬 프로그래밍

09 입력과 출력

Deep & High Learning

Copyright 2021. Deep&High learning. All rights reserved.

# Contents

## 화면 출력

**print() 함수**

```
print(“Hello world!”)
```

**Result**

```
Hello world!
```

(쌍)따옴표 안의 문자열은 자유롭게 작성 가능

## 화면 출력

**print() 함수**

```
print(“Hello world!”, end=“ “)
print(“This is beautiful!”)
```

**Result**

```
Hello world! This is beautiful!
```

end: 출력 맨 끝 문자 정의. default: ₩n(개행문자)

## 화면 출력

**print() 함수**

```
print(“Hello”, “world!”, sep=“#“)
```

**Result**

```
Hello#world!
```

**sep:** 개체 출력 사이 문자 정의. default : ' '(공백)

## 화면 입력

**Input() 함수**

```
a = input()  # 입력: kim
print("당신의 이름은 ", a)
```

**Result (입력: kim)**

```
당신의 이름은 kim
```

input 함수로 입력받은 문자열을 변수 a에 저장하여 print 함수로 화면 출력

## 화면 출력 - 이스케이프 문자

₩(역슬래시)와 바로 뒤에 이어지는 특정 문자를 합쳐 하나의 이스케이프 문자라 함

출력시 특별한 의미를 나타냄

**print() 함수**

```
print('\', \", \\, \t, \n')
```

**Result**

```
', ", \,       ,
```

₩' : 홑따옴표, ₩" : 쌍따옴표, ₩n : 줄바꿈, ₩t : 탭문자 (일정 간격), ₩₩ : 역슬래시

## 화면 출력 - 자료형별 출력 서식

출력 문자열에 자료형에 대응하는 서식을 순서대로 넣고,

이후 %() 에 원래 변수 작성

**print() 함수**

```
a = 100
b = 200
c = 0.5
print("%d"%a)
print("%d %d %f"%(a,b,c))
```

**Result**

```
100
100 200 0.5
```

%d : 정수, %f : 실수, %c : 1글자 문자, %s : 2글자 이상 문자열

## 화면 출력 - 자료형별 출력 서식

출력 문자열에 자료형에 대응하는 서식을 순서대로 넣고,

이후 %() 에 원래 변수 작성

**print() 함수**

```
a = "아메리카노"
b = 2000

print("%s 한 잔의 가격은 %d원입니다."%(a,b))
```

**Result**

```
아메리카노 한 잔의 가격은 2000원입니다.
```

%d : 정수, %f : 실수, %c : 1글자 문자, %s : 2글자 이상 문자열

## 화면 출력 - 여백을 맞추어 출력하기(정수)

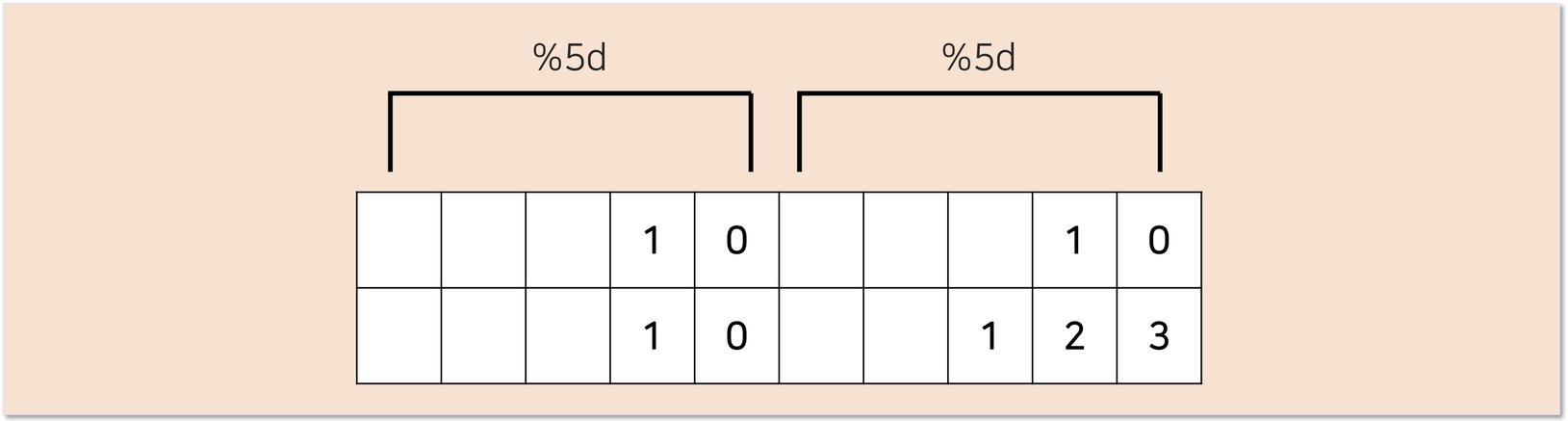
%와 문자 사이에 숫자 입력하여 여백 맞추기 가능

**print() 함수**

```
a = 10
b = 123
print("%5d %5d"%(a, a))
print("%5d %5d"%(a, b))
```

**Result**

```
   10    10
   10   123
```



## 화면 출력 - 여백 및 소수점 맞추어 출력하기(실수)

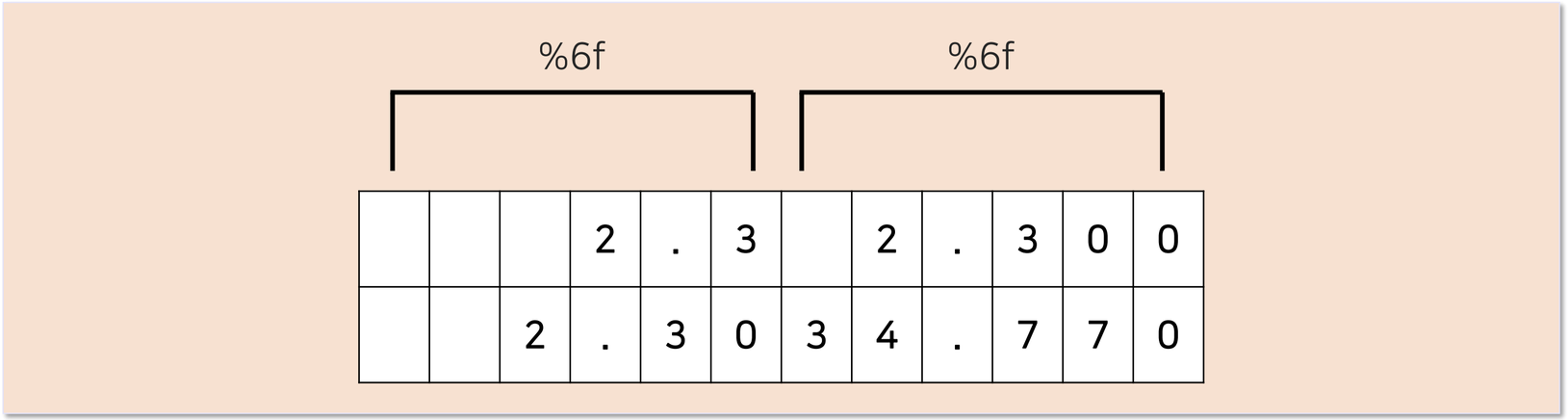
%와 문자 사이에 숫자 입력하여 여백 맞추기 가능

**print() 함수**

```
a = 2.3
b = 34.77
print("%6.1f %6.3f"%(a, a))
print("%6.2f %6.3f"%(a, b))
```

**Result**

```
  2.3 2.300
 2.3034.770
```



## 화면 출력 - format()을 이용한 출력

format 함수는 중괄호 {, } 안에 포매팅을 지정하고 인자로 값들을 삽입

**print() 함수**

```
a = 2
b = 3
s = '구구단 {0} x {1} = {2}'.format(a,b,a*b)
print(s)
```

```
0 <- a
1 <- b
2 <- a*b
```

**Result**

```
구구단 2 x 3 = 6
```

## 화면 출력 - format()을 이용한 출력

format 함수 인자로 문자, 변수, 직접 지정 등 사용 가능

**print() 함수-1**

```
s1 = 'name : {0}'.format('Deeplearning')
print(s1)
```

**Result**

```
name : Deeplearning
```

**print() 함수-2**

```
age = 55
s2 = 'age : {0}'.format(age)
print(s2)
```

**Result**

```
age : 55
```

**print() 함수-3**

```
s3 = 'number : {num}, gender :
{gen}'.format(num=111, gen='여')
print(s3)
```

**Result**

```
number : 111, gender : 여
 2 x 3 = 6
```

## 화면 출력 - format()을 이용한 출력

인덱스가 없으면 순서대로, 순서가 바뀌거나 중복도 허용 가능

### format()-1

```
s4 = 'name : {}, city : {}'.format('Kim', 'seoul')
print(s4)
```

**Result**

```
name : Kim, city : seoul
```

### format()-2

```
s5 = 'song1 : {1}, song2 : {0}'.format('love
yourself','shape of you')
print(s5)
```

**Result**

```
song1 : shape of you, song2 :
love yourself
```

### format()-3

```
s6 = 'test1 : {0}, test2 : {1}, test3 :
{0}'.format('인덱스0','인덱스1')
print(s6)
```

**Result**

```
test1 : 인덱스0, test2 : 인덱스1,
test3 : 인덱스0
```

## 화면 출력 - format()을 이용한 출력

'<' : 왼쪽 정렬, '>' : 오른쪽 정렬, '^' : 가운데 정렬

**print() 함수-왼쪽정렬**

```
s9 = 'this is {0:<10} | done {1:<5} |'.format('left', 'a')
print(s9)
```

**Result**

```
this is left       | done a     |
```

## 화면 출력 - format()을 이용한 출력

'<' : 왼쪽 정렬, '>' : 오른쪽 정렬, '^' : 가운데 정렬

**print() 함수-오른쪽정렬**

```
s10 = 'this is {0:>10} | done {1:>5} |'.format('right', 'b')
print(s10)
```

**Result**

```
this is      right | done     b |
```

## 화면 출력 - format()을 이용한 출력

'<' : 왼쪽 정렬, '>' : 오른쪽 정렬, '^' : 가운데 정렬

**print() 함수-가운데정렬**

```
s11 = 'this is {0:^10} | done {1:^5} |'.format('center', 'c')
print(s11)
```

**Result**

```
this is    center    | done    c    |
```

## 화면 출력 - f-string을 이용한 출력 (>=python3.6)

문자열 맨 앞에 f를 붙이고, 출력할 변수와 값을 중괄호 안에 삽입

**f-string**

```
s = 'study'
n = 5
result1 = f'{s}를 좋아합니다. 하루 {n}시간 합니다.'
print(result1)
```

**Result**

```
study를 좋아합니다. 하루 5시간 합니다.
```

## 화면 출력 - f-string을 이용한 출력 (>=python3.6)

변수 뒤에 :<, :^, :> 를 붙여서 정렬

**f-string-왼쪽정렬**

```
s1 = 'left'
result1 = f'|{s1:<10}|'
print(result1)
```

**Result**

```
|left      |
```

**f-string-오른쪽정렬**

```
s3 = 'right'
result3 = f'|{s3:>10}|'
print(result3)
```

**Result**

```
|     right|
```

**f-string-가운데정렬**

```
s2 = 'mid'
result2 = f'|{s2:^10}|'
print(result2)
```

**Result**

```
|   mid    |
```

## 연습 문제

1) print()를 사용하여 아래와 같이 출력하세요. (여백 만들기를 활용)

```
    +
   +++
  +++++
 +++++++
+++++++++
```

2) for과 print()를 사용하여 다음과 같이 3의 1제곱부터 3의 10제곱까지 출력하세요.

(숫자는 칸에 맞추어 정렬되어 있어야 합니다)

```
3    **    1    =      3
3    **    2    =      9
3    **    3    =     27
3    **    4    =     81
3    **    5    =    243
...
```

## 파일 입력

open() 함수와 close() 함수 이용, 'r' : 읽기 모드

**읽을 파일(txtio.txt)**

```
Hello, world!
My name is Dohyun Kim.
Nice to meet you.
```

**open() / read() 함수**

```
f = open("./txtio.txt", 'r', encoding='utf-8')
line = f.read()
print(line)
f.close()
```

**Result**

```
Hello, world!
My name is Dohyun Kim.
Nice to meet you.
```

read() 함수를 활용하여 txtio.txt 파일 전체를 읽어옴

## 파일 입력

open() 함수와 close() 함수 이용, 'r' : 읽기 모드

**읽을 파일(txtio.txt)**

```
Hello, world!
My name is Dohyun Kim.
Nice to meet you.
```

**open() / readline() 함수**

```
f = open("./txtio.txt", 'r', encoding='utf-8')
line = f.readline()
print(line)
f.close()
```

**Result**

```
Hello, world!
```

readline() 함수를 활용하여 txtio.txt 파일 한 문장을 읽어옴

## 파일 입력

open() 함수와 close() 함수 이용, 'r' : 읽기 모드

**읽을 파일(txtio.txt)**

```
Hello, world!
My name is Dohyun Kim.
Nice to meet you.
```

**open() / readline() 함수**

```
f = open("./txtio.txt", 'r', encoding='utf-8')
line = f.readline()
while line:
    print(line)
    line = f.readline()
f.close()
```

**Result**

```
Hello, world!
My name is Dohyun Kim.
Nice to meet you.
```

readline() 함수와 반복문을 활용하여 txtio.txt 파일 전체를 한 문장씩 읽어옴

## 파일 입력

open() 함수와 close() 함수 이용, 'r' : 읽기 모드

**읽을 파일(txtio.txt)**

```
Hello, world!
My name is Dohyun Kim.
Nice to meet you.
```

**open() / readlines() 함수**

```
f = open("./txtio.txt", 'r', encoding='utf-8')
line = f.readlines()
print(line)
f.close()
```

**Result**

```
['Hello, world!\n', 'My name is Dohyun Kim.\n', 'Nice to meet you.\n']
```

readlines() 함수로 파일의 모든 line을 리스트화 해서 입력

## 파일 출력

'w': 쓰기 모드 – 파일에 내용을 쓸 때 사용

'a' : 추가 모드 – 파일의 마지막에 새로운 내용을 추가할 때 사용

**open() / write() 함수**

```
music = ['Circle of life', 'Be prepared',
'The lion sleeps tonight', 'Hakuna Matata’]

with open('./Lion_king.txt', 'w') as f:
    for a in music:
        f.write(a+'\n')
```

**출력 파일(Lion_king.txt)**

```
Circle of life
Be prepared
The lion sleeps tonight
Hakuna Matata
```

## 파일 출력

format(), f-string 모두 사용 가능

**open() / write() 함수**

```
music = ['Circle of life', 'Be prepared',
'The lion sleeps tonight', 'Hakuna Matata’]

with open("./Lion_king.txt", 'w',
encoding='utf-8') as f:
    for a in music:
        data = f'{a}\n'
        f.write(data)
```

**출력 파일(Lion_king.txt)**

```
Circle of life
Be prepared
The lion sleeps tonight
Hakuna Matata
```

## 연습 문제

- 주어진 fileIO.txt 파일을 읽어 Key는 성이고 Value는 나이인 딕셔너리 name_age에 정보들을 할당한 후 출력 하세요.

**읽을 파일(fileIO.txt)**

```
Kim 32
Lee 34
Park 39
Choi 28
Cho 25
```

결과: {'Kim':32, 'Lee':34…}

See you in the next lecture!

# 인공지능을 위한 파이썬 프로그래밍

## 10 함수 (Function)

Deep & High Learning

Copyright 2021. Deep&High learning. All rights reserved.

# Contents

## 함수 (Function)의 정의

특정 작업을 수행하기 위해 독립적으로 설계된 코드의 집합

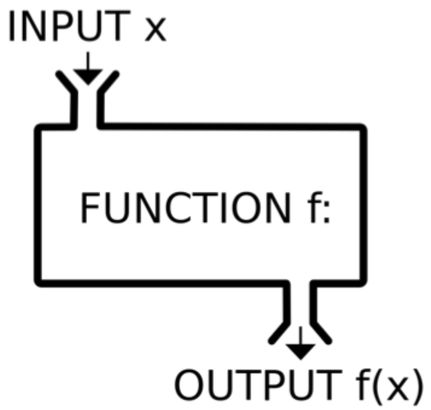
**함수 선언**

```
def data_science():
    print(“python”)

data_science()
```

**Result**

```
python
```



## 파라미터 (Parameter)와 아규먼트(Argument)

파라미터 (parameter) : 함수 호출 시 전달하는 데이터를 받아서 함수 내에서 사용되는 변수명

아규먼트 (Argument) : 함수를 호출할 때의 인수.

**함수 선언 및 실행**

```
def data_science(print_sentence):
    print(print_sentence)

string = “매개변수로 전달된 문자열 출력”

data_science(string)
```

**Result**

```
매개변수로 전달된 문자열 출력
```

print_sentence: 파라미터, string: 아규먼트

## 디폴트 파라미터 (Default Parameter)

함수 호출시 파라미터에 대한 아규먼트가 없으면  디폴트로 설정된 값이 파라미터 변수로 입력됨

**함수 선언 및 실행**

```
def minus(num1=3, num2=2):
    print(num1-num2)

minus(2)
```

**Result**

```
0
```

num1은 argument로 넘긴 2, num2은 default 값 2

## 디폴트 파라미터 (Default Parameter)

디폴트 파라미터를 사용할 때는 디폴트로 설정된 파라미터가 항상 뒤쪽에 위치해야 함

### 에러 케이스

```
def minus(num1=3, num2=2, num3):
    print(num1-num2-num3)

minus(5)
```

### Result

```
  File "<ipython-input-14-4e76455c1a92>", line 1
    def minus(num1=3, num2=2, num3):
              ^
SyntaxError: non-default argument follows default argument
```

### 디버깅

```
def minus(num3, num1=3, num2=2):
    print(num1-num2-num3)

minus(5)
```

### Result

```
-4
```

## 키워드 아규먼트 (Keyword Argument)

argument가 들어가는 부분에 (key=value) 형태로 입력

argument의 키 값과 함수 parameter 명이 같은 변수에 value 값이 들어감

**키워드 아규먼트**

```
def minus(num1, num2):
    print(num1-num2)

minus(2, 3)
minus(num2=3, num1=2)
```

**Result**

```
-1
-1
```

순서는 상관없이 매치되는 이름으로 들어감

## 리턴 (Return)

리턴 (return) 은 함수를 종료하며 결과 데이터를 반환하는 용도로 사용

**함수 return**

```
# 리턴이 없는 함수
def no_return():
    a = 1 + 2

result = no_return()
print(result)
```

**Result**

None

```
# 리턴이 있는 함수
def plus(a, b):
    return a + b

result = plus(1, 2)
print(result)
```

**Result**

3

## *args

함수를 호출할 때 보내는 argument의 개수를 특정할 수 없을 때, 함수의 파라미터를 넣는 영역에 사용

***args**

```
def print_args(*args):
    print(args)
    print(args[4])
    print(args[5][1])

print_args(1, 2, 3, "deep", "learning",
["artificial", "intelligence"])
```

**Result**

```
(1, 2, 3, 'deep', 'learning',
['artificial', 'intelligence'])
learning
intelligence
```

파라미터로 받는 데이터는 tuple 데이터 타입으로 인덱스를 통한 접근이 가능함

## *args 활용 예시

함수를 호출할 때 보내는 argument의 개수를 특정할 수 없을 때, 함수의 파라미터를 넣는 영역에 사용

***args**

```
def avg_func(*args):
    return sum(args) / len(args)

a = avg_func(100, 70, 80, 99, 85, 60, 80)

print("avg : {}".format(round(a, 2)))
```

**Result**

```
avg : 82.0
```

## **kwargs

함수에서 parameter로 키워드 argument를 받아올 수 있음 (key, value로 구성된 dictionary 타입)

**kwargs

```
def avg_func(**kwargs):
    print(kwargs)
    total=0
    count=0
    for subject, point in kwargs.items():
        print(subject, point)
        total += point
        count += 1
    return total / count

a = avg_func(korean=100, english=70,
math=80, science=90)
print("avg : {}".format(round(a, 2)))
```

Result

```
{'korean': 100, 'english': 70, 'math':
80, 'science': 90}
korean 100
english 70
math 80
science 90
avg : 85.0
```

## *args, **kwargs

함수에서 *args와 **kwargs를 함께 사용 가능

***args, **kwargs**

```
def test_func(*args, **kwargs):
    print(args)
    print(kwargs)

test_func(1, 2, 3, "deepandhigh",
"python", korean=100, english=70, math=80)
```

**Result**

```
(1, 2, 3, 'deepandhigh', 'python')
{'korean': 100, 'english': 70, 'math': 80}
```

## 범위 (Scope)

변수는 코드 전체에서 사용 가능한 global (전역 변수)과
특정 블록에서만 사용 가능한 local (지역 변수)로 나뉨

**전역변수 gv**

```
gv = 10

def print_gv():
    print(gv)

print_gv()
```

**Result**

```
10
```

print_gv 함수 안에서도 전역 변수인 gv에 접근 가능

## 범위 (Scope)

변수는 코드 전체에서 사용 가능한 global (전역 변수)과
특정 블록에서만 사용 가능한 local (지역 변수)로 나뉨

**전역/지역 변수 gv1, 2**

```
gv1, gv2 = 1, 2
def print_variable():
    gv1 = 10
    gv2 = 20
    print(gv1, gv2)
    return gv1, gv2

print_variable()
print(gv1, gv2)
```

**Result**

```
10 20
1 2
```

동일한 변수명의 전역/지역 변수가 있을 경우, 지역변수의 우선순위가 더 높음

## 범위 (Scope)

global 예약어를 사용하시면 함수 내에서 전역 변수의 값 변경이 가능

**global 변수 접근**

```
gv=12

def change_gv(data):
    global gv
    gv=data

print(gv)
change_gv(100)
print(gv)
```

**Result**

```
12
100
```

global 사용은 자칫 프로그램이 꼬일 수 있으므로 유의해야 함

## 람다 함수 (Lambda function)

파라미터를 간단한 계산으로 리턴하는 함수는 람다 함수 이용

**일반적 함수 선언**

```
def sum_func(x, y):
    return x + y

sum_func(5, 6)
```

**Result**

```
11
```

**람다 함수 선언**

```
sum_func2 = lambda x, y : x + y
sum_func2(5, 6)
```

**Result**

```
11
```

lambda 활용시 코드가 간결해지고 메모리 절약 가능

## 연습 문제

1) 아래 student2score에 학생들의 이름과 점수가 주어져 있습니다.

특별반 학생의 기준은 80점 이상입니다.

특별반 학생의 이름과 점수를 리턴하는 함수를 만드세요.

**학생들 이름과 점수**

```
student2score = {
    "Darius": 100,
    "Dr. Mundo": 80,
    "Morgana": 60,
    "Sivir": 75,
    "Yummi": 20,
    "Viktor": 97
}
```

**함수 선언**

```
def get_special_students(student2score):
    special_students = {}
    """
    your code
    """
    return special_students
```

## 연습 문제

1) 아래와 같이 text가 주어져 있을 때, text 안의 단어에 숫자로 된 id를 부여하세요
2) 각 단어의 빈도수를 세어 각각의 텍스트에 해당하는 id를 저장하는 딕셔너리와
각각의 단어 id에 해당하는 빈도수를 저장하는 딕셔너리를 리턴하세요

* 조건1) 텍스트 안의 특수 기호는 제거하세요.
조건2) 모든 단어는 소문자로 만드세요.

**주어진 text**

```
text = "Apple is fruit.
Orange is also fruit.
Tomato is fruit?"
```

**함수 선언**

```
def word_index_count(text):
    word_id = {}
    word_frequency = {}
    """
    your code
    """
    return word_id, word_frequency
print(word_index_count(text))
```

See you in the next lecture!

# 인공지능을 위한 파이썬 프로그래밍

11 클래스 (Class)

Deep & High Learning

Copyright 2021. Deep&High learning. All rights reserved.

# Contents

## 프로그래밍 패러다임

[명령형 프로그래밍 (Imperative programming)]

- 프로그래밍의 상태와 상태를 변경시키는 구문의 관점에서 연산을 설명하는 프로그래밍 패러다임
- 대부분의 컴퓨터 하드웨어의 구현 방식
- 절차적 프로그래밍, 객체지향 프로그래밍

[선언형 프로그래밍 (Declarative programming)]

- 프로그램이 어떤 방법으로 해야 하는지를 나타내기보다 무엇과 같은지를 설명하는 프로그래밍 패러다임
- Haskell, LISP, PROLOG 등

## 명령형 프로그래밍의 종류

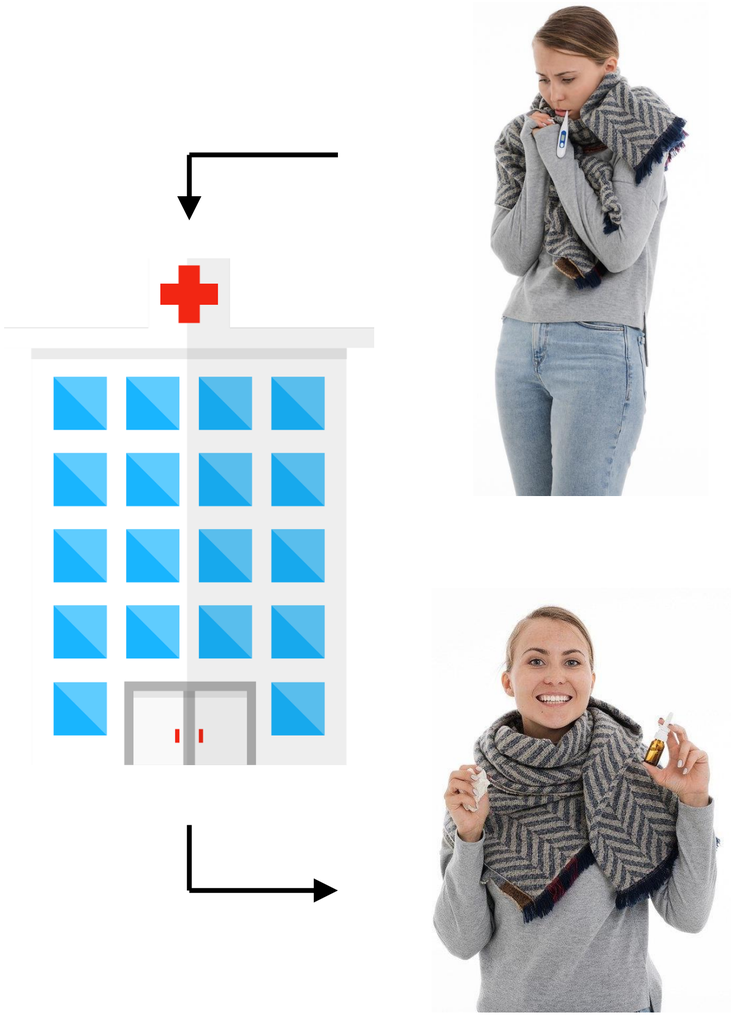
[절차적 프로그래밍 (Procedural programming)]

- 루틴, 서브루틴, 메소드, 함수 등을 이용한 프로그래밍 패러다임

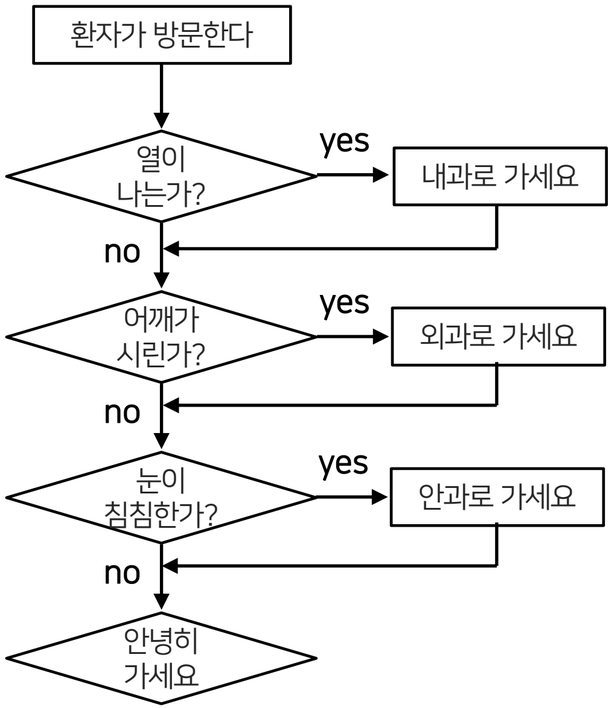
[객체지향 프로그래밍 (Object-oriented programming)]

- 프로그램을 수많은 '객체'라는 기본 단위로 나누고 이들의 상호작용으로 서술하는 방식
- 객체(Object): 하나의 역할을 수행하는 '메소드와 변수(데이터)'의 묶음
- 인스턴스(Instance) : 실제 메모리에 할당되어 사용할 수 있는, 실체가 있는 객체

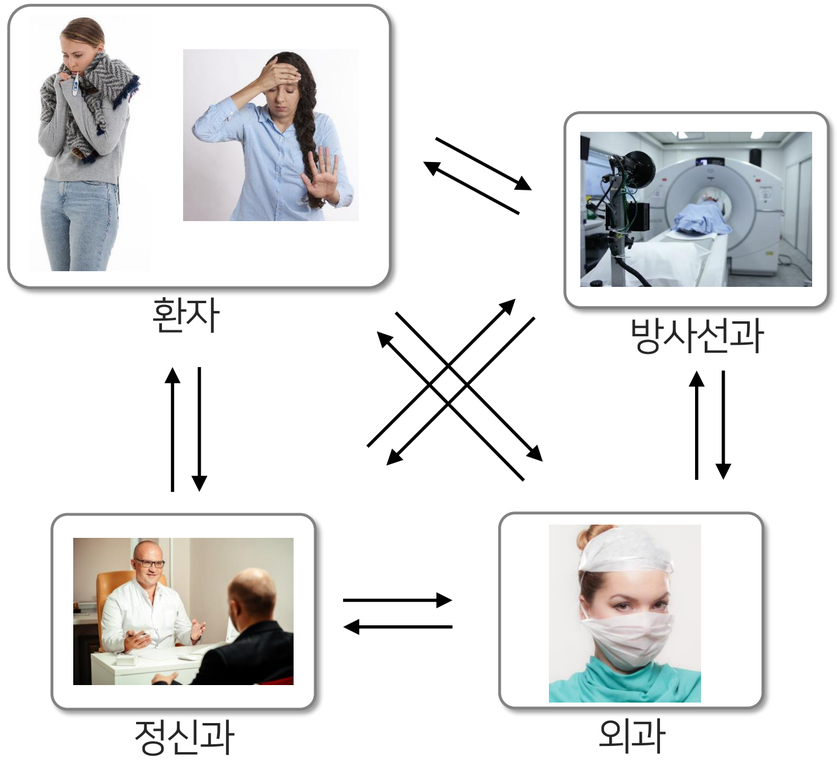
## 명령형 프로그래밍의 종류



[절차적 프로그래밍]




[객체지향 프로그래밍]



## 클래스 (class)의 정의

변수와 함수를 묶은 사용자 정의 데이터 타입

클래스는 청사진, 설계도, 빵틀

객체는 클래스로 찍어낸 형체

[클래스]

[객체]

## 클래스 (class)의 정의

class 명령어를 사용하여 정의

**class 정의**

```
class Flight:
    pass

f = Flight()
print(type(f))
```

**Result**

```
<class '__main__.Flight'>
```

어떤 변수, 함수도 없는 class도 정의 가능함

## 클래스 (class)의 정의

메소드(method) : 클래스 내 함수. 온점(.)을 통해서 접근 가능

self : 메소드의 첫번째 파라미터명. 객체 자신을 의미 (항상 필요)

**class 내부 함수 정의**

```
class Flight:
    def number(self):
        return 'KE081'

f = Flight()
print(f.number())
```

**Result**

```
KE081
```

## 생성자 (Constructor)

클래스가 객체가 될 때 실행되는 함수

객체에서 사용할 변수의 초기값 설정 시 사용

**생성자 활용**

```
class Flight:
    def __init__(self, number):
        self._number = number
    def number(self):
        return self._number

f = Flight('KE082')
print(f.number())
print(f._number)
```

**Result**

```
KE082
KE082
```

'KE082' 가 Flight 클래스 생성자의 number 변수로 넘어감

# 생성자 (Constructor)

## 생성자 활용 예외처리

```
class Flight:
    def __init__(self, number):
        if not number[:2].isalpha():
            raise ValueError("첫 두글자가 알파벳이 아닙니다.")
        if not number[:2].isupper():
            raise ValueError("첫 두글자가 대문자가 아닙니다.")
        if not number[2:].isdigit():
            raise ValueError("세번째 글자 이상이 양의 숫자가 아닙니다.")
        self._number = number

f = Flight('Ke082') # KEE82, 0KE082
```

## Result

```
<ipython-input-5-b337494b4aa0> in __init__(self, number)
      4             raise ValueError("첫 두글자가 알파벳이 아닙니다.")
      5         if not number[:2].isupper():
----> 6             raise ValueError("첫 두글자가 대문자가 아닙니다.")
      7         if not number[2:].isdigit():
      8             raise ValueError("세번째 글자 이상이 양의 숫자가 아닙니다.")

ValueError: 첫 두글자가 대문자가 아닙니다.
```

## 더블 언더바 (__)

클래스 내 변수의 외부접근을 막아줌

### 더블 언더바

```
class Flight:
    def __init__(self, number):
        self._number = number
        self.__number = number

    def number(self):
        return self.__number

f = Flight('KE082')
print(f.number())
print(f._number)
print(f.__number)
```

### Result

```
KE082
KE082
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-6-2e997313a210> in <module>()
      9 print(f.number())
     10 print(f._number)
---> 11 print(f.__number)

AttributeError: 'Flight' object has no attribute '__number'
```

## 인스턴스 속성 변경

**속성 변경**

```
class Flight:
    def __init__(self, number, passenger_num):
        self.__number = number
        self._passenger_num = passenger_num

    def number(self):
        return self.__number

    def add_passenger(self, num):
        self._passenger_num += num # 속성 변경

f1 = Flight('KE082', 0)
f2 = Flight('KE081', 0)
f1.add_passenger(2)
f2.add_passenger(3)
print(f1._passenger_num)
print(f2._passenger_num)
```

**Result**

```
2
3
```

## 상속 (Inheritance)

새로운 클래스 생성 시, 기존에 정의되어 있던 클래스의 속성과 메소드를 가져오는 기능

기존 클래스 : 부모 클래스, 새로운 클래스 : 자식 클래스

**클래스 상속**

```
class Flight:
    def __init__(self, number, passenger_num):
        self.__number = number
        self._passenger_num = passenger_num

    def number(self):
        return self.__number

    def add_passenger(self, num):
        self._passenger_num += num

class AdvancedFlight(Flight):           # Flight 상속
    def subtract_passenger(self, num):
        self._passenger_num -= num

f2 = AdvancedFlight('KE081', 0)
f2.add_passenger(3)
f2.subtract_passenger(1)
print(f2._passenger_num)
```

**Result**

```
2
```

클래스 정의 시 괄호 안에 부모 클래스 이름을 넣어 정의

## Super

상속 시 부모 클래스의 생성자를 재정의하기 위해 사용

**Super**

```
class Flight:
    def __init__(self, number, passenger_num):
        self.__number = number
        self._passenger_num = passenger_num

    def number(self):
        return self.__number

    def add_passenger(self, num):
        self._passenger_num += num

class AdvancedFlight(Flight):
    def __init__(self, number, passenger_num):
        super().__init__('KE083',10)
    def subtract_passenger(self, num):
        self._passenger_num -= num

f2 = AdvancedFlight('KE081', 0)
f2.add_passenger(3)
f2.subtract_passenger(1)
print(f2._passenger_num)
```

**Result**

```
12
```

super 사용 시 부모 클래스 생성자 형식을 맞춰야 함

## 연습 문제

1) 사칙연산을 수행하는 Calculator 클래스를 만드세요.
   num1 = 23, num2 = 4 로 설정 후 각 연산을 수행한 결과를 출력하세요.
2) Calculator 클래스를 상속받은 AdvancedCalculator 클래스를 만드세요.
   AdvancedCalculator 클래스에 제곱과 나머지 연산을 수행하는 메소드를 만드세요.
   num1 = 8, num2 = 5 로 설정 후 제곱과 나머지 연산을 수행한 결과를 출력하세요.

```
class Calculator:
      def __init__(self, num1, num2):
            '''your code'''
      def add(self):
            '''your code'''
            return result
      def sub(self):
            '''your code'''
            return result
      def mul(self):
            '''your code'''
            return result
      def div(self):
            '''your code'''
            return result
```

See you in the next lecture!