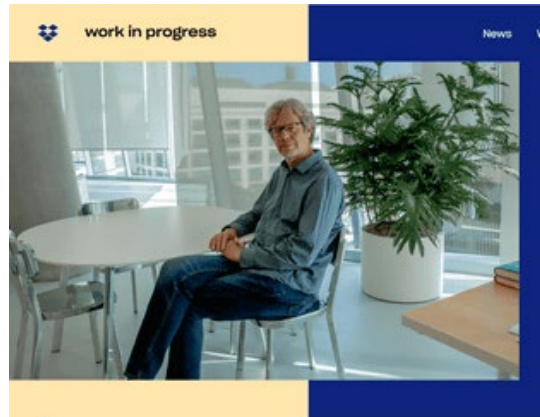


# 딥러닝 - AI 기초 (Deep Learning)

Hyungmin Jun, Ph. D.

Multiphysics Systems Design Laboratory  
Department of Mechanical System Engineering  
Jeonbuk National University

- ❖ Python은 1989년 12월 네덜란드 개발자 **Guido van Rossum**에 의해 개발  
(1년의 개발기간, 크리스마스 전후에 취미로 개발)
- ❖ Guido van Rossum은 1956년생, 네덜란드 CWI 및 미국 NIST등의 연구소 및 Google(약 7년)에서 근무 → Dropbox (2014) → Microsoft (2020)



## Python creator Guido van Rossum joins Microsoft

Frederic Lardinois

@frederic / 2:32 AM GMT+9 • November 13, 2020

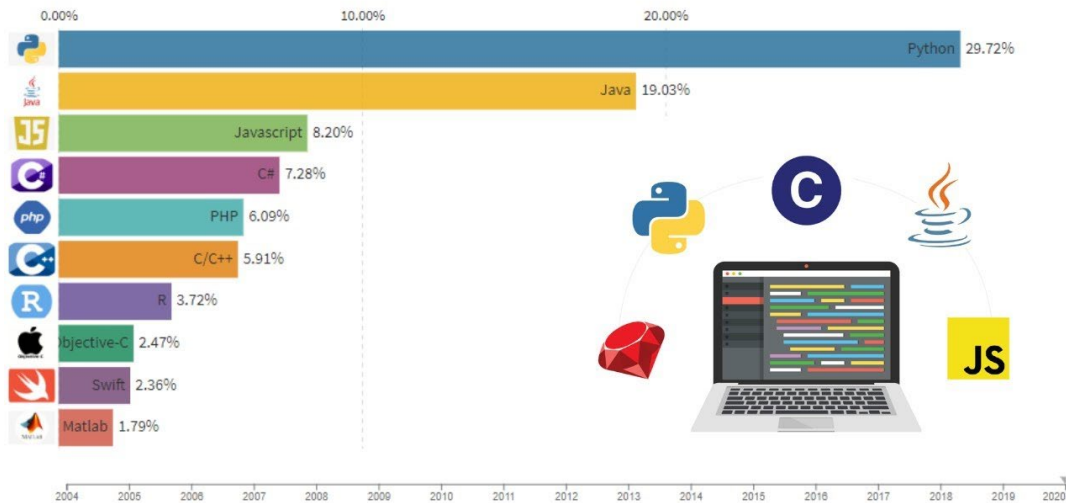
Comment



- ❖ 1991년 Python 0.9, 1994년 Python 1.0(정식버전), 2000년 Python 2.0, 2008년 Python 3.0 출시
- ❖ Python 특징
  - ✓ 높은 가독성(Readability), 간결한 코딩을 강조한 **인터프리터(Interpreter) 언어**
  - ✓ 배우기 쉬운 언어
  - ✓ **동적타입(Dynamic Type)** 프로그래밍 언어(예, Ruby, JavaScript 등)
  - ✓ 호환성이 좋은 범용 프로그래밍 언어
  - ✓ 리눅스(Linux), macOS, 윈도우(Windows)등의 다양한 OS환경에서 사용 가능
  - ✓ 값비싼 공학용 계산기(?) MATLAB을 대체할 수 있는 **다양한 패키지 제공**
  - ✓ 현업에서 자주 사용되는 언어
  - ✓ 인공지능분야에서 큰 각광을 받고 있는 프로그래밍 언어

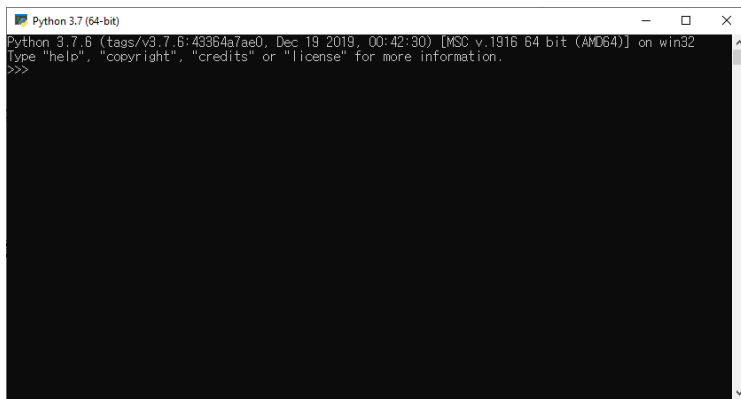


**Top 10 Most Popular Programming Languages**  
2004 to 2020

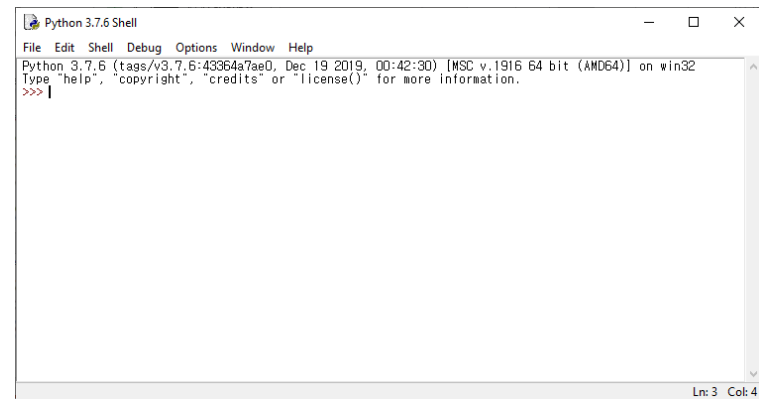


<https://medium.com/@davhana.r/all-to-know-about-c-in-2020-4330994e4b15>

- ✓ Windows [시작] → [프로그램] → [Python 3.x]에서 “Python 3.x” “콘솔프로그램” 혹은 “IDLE (Python 3.x)” 윈도우 프로그램 실행
- ✓ 팝업되는 윈도우 창 → 대화형 인터프리터 = Python Shell
- ✓ Terminal에서 python을 입력하면 2.7버전이 실행되고, python3을 입력하면 3.8 버전이 실행 (Linux 또는 Mac의 경우 - OS 설치 버전에 따라 상이)



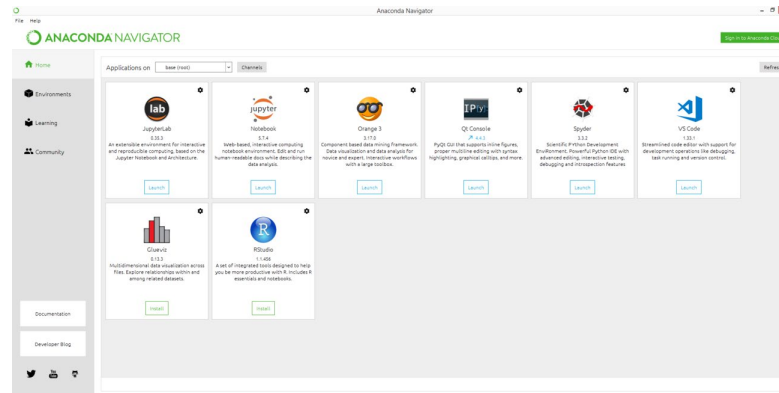
```
Python 3.7 (64-bit)
Python 3.7.6 (tags/v3.7.6:43364a7ae0, Dec 19 2019, 00:42:30) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
```



```
Python 3.7.6 Shell
File Edit Shell Debug Options Window Help
Python 3.7.6 (tags/v3.7.6:43364a7ae0, Dec 19 2019, 00:42:30) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> |
Ln: 3 Col: 4
```

© Python Distribution, Anaconda

- ❖ 과학 분야에서 주로 사용되는 여러 Python 패키지(Package)들을 포함하여 Python과 함께 제공
  - ✓ 데이터 사이언스(Data Science)와 머신러닝(Machine Learning) 분야에서 자주 활용되는 패키지들을 기본적으로 포함
  - ✓ Jupyter Notebook, Spyder IDE, NumPy, SciPy, Pandas, Pillow, Matplotlib, TensorFlow, Scikit-Learn 등등등..



## © Python 코드 개발 환경 - Jupyter Notebook

- ✓ 크롬, 익스플로러, 파이어폭스 등의 웹브라우저에서 Python 코드 작성
- ✓ Anaconda 또는 pip(Pip Install Packages)를 통해 설치
- ✓ 8888 포트를 기본으로 사용 → 접속: <http://localhost:8888>



- 변수와 자료형(정수, 실수, 불)
- 자료구조(리스트, 튜플, 딕셔너리, 셋)
- 입력 및 출력
- 조건문
- 반복문
- 함수
- 클래스
- 모듈



- ✓ Python은 동적타입 프로그래밍 언어
- ✓ 변수(Variable)는 데이터를 저장하는 컴퓨터 메모리 공간의 이름
- ✓ 변수에 저장된 데이터는 프로그램이 실행되면서 값이 변함
- ✓ 변수의 이름을 정의한 후, "=" 기호를 사용하여 값을 대입
  - $x=10 \rightarrow$  x라는 변수에 10 저장
  - $a=50 \rightarrow$  a라는 변수에 50 저장
- ✓ Python 변수 이름 작성법
  - 영문, 문자 또는 숫자 사용 가능
    - 반드시 문자부터 시작되어야 함; Python은 대소문자 구분

- 특수 문자(+, -, ₩, \*, & 등) 사용 불가능
  - 단, \_(밑줄) 시작은 가능
- Python의 키워드(if, while, for, or 등) 사용 불가능
- ✓ C나 Java에서는 변수를 만들 때 자료형을 직접 지정
  - 정적타입(Static Typing) 프로그래밍 (C, C++, C#, Java 등)
- ✓ Python은 변수에 저장된 값을 스스로 판단하여 **자동으로 자료형을 지정**
  - 동적타입(Dynamic Typing) 프로그래밍 (Python, Ruby, MATLAB, Perl, PHP, JavaScript 등)
- ✓ 정적타입 언어 **VS** 동적타입 언어

- ✓ 숫자형태로 이루어진 자료형 - 정수(123, -345), 실수(123.45, -123.5, 3.4e10), 8진수(0o34, 0o25), 16진수(0x2A, 0xFF)

```
a = 123
a = -178
a = 0
a = 1.2
a = -3.45
a = 4.24E10
a = 4.24e-10
```

## ❖ 정수형(Integer)

- ✓ 정수형: -2, -1, 0, 1, 2, ...
- ✓ Python3에서는 정수형에 대한 **오버플로우(Overflow)**가 발생되지 **않음**
  - 정수(int)를 4 byte 크기의 메모리로 할당할 경우 → int는 총  $2^{32}$ 가지 수를 표현
    - 4 byte 정수: -2,147,483,648 ~ 2,147,483, 647
    - 2 byte 정수: -32,768 ~ 32,767
  - unsigned int의 경우 →  $0 \sim 2^{32}-1$  ( $2^{32}$ 가지 수)
  - Python2는 정수 자료형으로 int와 long(arbitrary precision) 사용 → Python3에서는 int가 Arbitrary Precision으로 대체

## ❖ 실수형(Floating-point)

- ✓ 실수형: 소수점이 포함된 숫자, 0.12345, 12.3456, -0.1234, -12.3456
- ✓ 컴퓨터식 지수 표현
  - $4.24e10$  또는  $4.24E10 \rightarrow 4.24 \times 10^{10}$
  - $4.24e-10$  또는  $4.24E-10 \rightarrow 4.24 \times 10^{-10}$

## ❖ 숫자형 연산자

- ✓ 사칙연산: +, -, \*, /
- ✓ x의 y 제곱:  $x**y$
- ✓ 나눗셈 후 나머지를 반환: %
- ✓ 나눗셈 후 몫을 반환: //

예)  $7 / 4 \rightarrow 1.75$ ,  $7 \% 4 \rightarrow 3$ ,  $7 // 4 \rightarrow 1$

- ❖ 문자열: 문자, 숫자 및 기호로 구성된 문자 집합
- ❖ 문자열을 만드는 4가지 방법
  - ✓ 큰따옴표(") 사용 → "Hello world"
  - ✓ 작은따옴표(') 사용 → 'Hello world'
  - ✓ 큰따옴표 3개 연속(""") 사용 → """Hello world"""
  - ✓ 작은따옴표 3개 연속('''') 사용 → '''Hello world'''
- ❖ 작은/큰 따옴표가 포함된 문자열
  - ✓ 백슬래시(\) 사용

```
var = 'Hyungmin\'s favorite programming language is Ruby'
```

- ❖ 여러 줄의 문자열을 포함하는 문자열 변수
  - ✓ 이스케이프(Escape) 코드 \n 사용
  - ✓ 작은따옴표 3개('') 또는 큰따옴표 3개(""") 사용

```
mline = "Programming is difficult\nYou need Python"
mline='
... Programming is difficult
... You need Python
...'
```

- ❖ 이스케이프(Escape): 프로그래밍에서 정의해 둔 문자 조합
  - ✓ 주로 출력을 보기 좋게 하기 위한 용도로 사용, 백슬래시(\)와 함께 사용
  - ✓ \n → 줄을 변경
  - ✓ \t → 문자 사이에 탭 간격
  - ✓ \\ → 문자에서 “\” 사용
  - ✓ \' → 작은따옴표(') 사용
  - ✓ \" → 큰따옴표(“) 사용
  - ✓ \r → 줄 바꿈 문자, 현재 커서를 가장 앞으로 이동
  - ✓ \f → 줄 바꿈 문자, 현재 커서를 다음 줄로 이동
  - ✓ \a → 출력할 때 '뵁' 소리 재생
  - ✓ \b → 백 스페이스
  - ✓ \000 → 널 문자



## ❖ 문자열 연산

- ✓ 문자열 더하기(Concatenation) → "+" 사용하여 문자를 직접 결합
- ✓ 문자열 곱하기 → a = "python" \* 2 → "pythonpython"
- ✓ 문자열 길이 → len() 함수, 문자열 길이를 반환

## ❖ 문자열 인덱싱(Indexing)

- ✓ 대괄호 "[" 사용, 0부터 a-1까지 인덱싱
- ✓ Python은 0부터 인덱싱 숫자 카운트한다. Why?
- ✓ a[-1] → 마지막 위치에 있는 문자 인덱싱

## ❖ 문자열 슬라이싱(Slicing)

- ✓ 구간의 문자열을 반환(슬라이싱) → 예, a[0:4]

```
a = "2020Jan27"  
year = a[:4] # '2020'
```

- ✓ a = "Programming is difficult, you need to learn Python" → a[-1] #'n'

```
month = a[4:7] # 'Jan'
```

## ✓ 문자열 포매팅(Formatting)

- 정수형 숫자 포맷 → "There are %d apples." % 3
  - num = 3; "There are %d apples." % num
- 문자열 포맷 → "There are %s apples." % "five"
  - "There are %d apples and %d oranges." % (num1, num2)
- 문자열 포맷코드
  - %d (정수), %f (실수)
  - %s (문자열), %c (문자)
  - %% (문자 % 자체) → "The error is %d%%." % 99
- 정렬을 포함하는 포맷코드
  - 오른쪽 정렬 및 나머지는 공백 → "%10s" % "hi"
  - 왼쪽 정렬 및 나머지는 공백 → "%-10s." % 'hi'
  - 실수 표현
    - "%0.4f" % 3.42134234 → '3.4213'
    - "%10.4f" % 3.42134234 → ' 3.4213'

✓ format() 함수 사용

○ 인덱스 사용

```
"I ate {0} apples. so I was sick for {1} days.".format(number, day)
'I ate 10 apples. I was sick for three days.'
```

○ 변수 이름 사용

```
"Those are {num1} apples and {num2} bananas.".format(num1=10, num2=3)
'Those are 10 apples and 3 bananas.'
```

○ 인덱스와 이름을 혼용

```
"I had {0} apples. so I was sick for {day} days.".format(10, day=3)
'I had10 apples. I was sick for 3 days.'
```

○ 왼쪽 정렬

```
"{0:<10}".format("hi") # 'hi '
```



- 오른쪽 정렬

```
"{:>10}".format("hi") # ' hi '
```

- 가운데 정렬

```
"{:^10}".format("hi") # ' hi '
```

- 특정 문자로 공백 채우기

```
"{::=^10}".format("hi") # '====hi===='  
"{:!  
":<10}".format("hi") # 'hi !!!!!!!!!'
```

- 소수점 표현

```
y = 3.42134234  
"{:0.4f}".format(y) # '3.4213'
```

- ✓ 참(True) 또는 거짓(False)을 표현하는 자료형
- ✓ 항상 첫 문자를 대문자로 사용 할 것

```
a = True
b = False
type(a)           # <class 'bool'>
type(b)           # <class 'bool'>
```

- type() 함수: Python의 자료형을 확인하는 내장 함수

```
print(1 == 1)     # True
print(2 > 1)      # True
print(2 < 1)      # False
```

- ✓ 자료들의 그룹화 가능 → 모음(집합) 표현 가능 (c 언어의 배열 vs 리스트)

```
odd = [1, 3, 5, 7, 9]
```

- ✓ 리스트 자료형은 대괄호 "["]를 사용하고, 각 요소는 쉼표 ","로 구분

```
리스트명 = [요소1, 요소2, 요소3, ...]
```

- ✓ 리스트 자료형 활용

```
a = []  
b = [1, 2, 3]  
c = ['Life', 'is', 'too', 'short']  
d = [1, 2, 'Life', 'is']  
e = [1, 2, ['Life', 'is']]
```

- 리스트 a는 요소(원소)를 포함하지 않은 빈 리스트, a = list()과 동일
- 어떠한 자료형으로도 리스트 생성이 가능

## ✓ 리스트의 인덱싱(Indexing)

- 문자열의 인덱싱과 동일

```
a = [1, 2, 3]           # [1, 2, 3]
a[0] + a[2]           # 4
```

- 예) 리스트 a의 인덱싱

```
a = [1, 2, 3, ['a', 'b', 'c']]
a[0]           # 1
a[-1]         # ['a', 'b', 'c']
a[3]         # ['a', 'b', 'c']
a[-1][0]     # 'a'
```

## ✓ 리스트의 슬라이싱(Slicing)

- 문자열의 슬라이싱과 동일

```
a = [1, 2, 3, 4, 5]
a[0:2]           # [1, 2]
```

- 예) 리스트 a의 슬라이싱

```
a = [1, 2, 3, 4, 5]
a1 = a[:2]          # a1 = [1, 2]
a2 = a[2:]         # a2 = [3, 4, 5]
```

- 예) 리스트 b의 슬라이싱

```
b = [1, 2, 3, ['a', 'b', 'c'], 4, 5]
b1 = b[2:5]        # b1 = [3, ['a', 'b', 'c'], 4]
b2 = b[3][:2]     # b2 = ['a', 'b']
```

- ✓ 리스트 연산; "+", "\*"

- 리스트 자료 더하기, "+"

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

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



- 리스트 자료 반복, "\*"

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

# [1, 2, 3, 1, 2, 3, 1, 2, 3]
```

- 리스트 자료의 길이 구하기

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

# 3
```

⇒ len() 함수는 문자열(String), 리스트(List), 튜플(Tuple) 및  
딕셔너리(Dictionary) 모두에서 사용가능

- ✓ 리스트 자료 - 수정 및 삭제

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

# [1, 2, 4]
```

- del 함수 - 리스트 요소(원소) 삭제

```
a = [1, 2, 3]
del a[1]
print(a)          # [1, 3]
```

- 리스트 멤버함수를 통한 원소를 삭제

➤ remove(), pop() 리스트 멤버 함수, **멤버 함수란?**

- ✓ 리스트 관련 중요 멤버함수

- 리스트 원소 추가 → append()
- 리스트 원소 정렬 → sort()
- 리스트 원소 전환 → reverse()

- 리스트 원소 전환 → `reverse()`
- 리스트 원소 위치 반환 → `index()`
  - `index(x)`는 리스트에 `x` 값이 있으면 `x`의 위치 값을 돌려주는 함수
- 리스트 원소 삽입 → `insert()`
  - `insert(a, b)`는 리스트의 `a`번째 위치에 `b`를 삽입하는 함수
- 리스트 원소 제거 → `remove()`
  - `remove(x)`는 리스트에서 첫 번째로 나오는 `x`를 삭제하는 함수
- 리스트 원소 끄집어 내기 → `pop()`
- 리스트 원소 개수 반환 → `count()`
  - `count(x)`는 리스트 안에 `x`가 몇 개 있는지 조사하여 그 개수를 반환
- 리스트 확장 → `extend()`

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

a.append([5,6])
print(a)           # [1, 2, 3, 4, [5, 6]]

a = [1, 4, 3, 2]
a.sort()
print(a)           # [1, 2, 3, 4]

a = ['a', 'c', 'b']
a.sort()
print(a)           # ['a', 'b', 'c']

a = ['a', 'c', 'b']
a.reverse()
print(a)           # ['b', 'c', 'a']

a = [1,2,3]
a.index(3)          # return 2
a.index(1)          # return 0
```

```
a = [1, 2, 3]
a.insert(0, 4)
print(a)           # [4, 1, 2, 3]

a.insert(3, 5)
print(a)           # [4, 1, 2, 5, 3]

a = [1, 2, 3, 1, 2, 3]
a.remove(3)
print(a)           # [1, 2, 1, 2, 3]

a.remove(3)
print(a)           # [1, 2, 1, 2]

a = [1,2,3]
a.pop()            # return 3
print(a)           # [1, 2]

a = [1,2,3]
a.pop(1)           # return 2
print(a)           # [1, 3]
```

```
a = [1,2,3,1]
a.count(1)           # return 2

a = [1,2,3]
a.extend([4,5])
print(a)            # [1, 2, 3, 4, 5]

    b = [6, 7]
a.extend(b)
print(a)            # [1, 2, 3, 4, 5, 6, 7]
```

- ✓ 튜플(Tuple)은 리스트와 동일, **But 2개의 큰 차이**
  - 리스트 - [], 튜플 - ()를 활용하여 자료 생성
  - 리스트의 원소는 수정(추가, 삭제) 가능, **튜플의 원소는 변경 불가**

```
t1 = ()  
t2 = (1,)  
t3 = (1, 2, 3)  
t4 = 1, 2, 3  
t5 = ('a', 'b', ('ab', 'cd'))
```

- 1개의 요소로 구성된 튜플은 **반드시 원소에 ","를 사용**
  - 튜플 자료 생성시 **"()"를 생략 가능**
- ✓ 튜플의 활용; 인덱싱(Indexing), 슬라이싱(Slicing), 더하기, 길이 반환

- 튜플 인덱싱(Indexing)

```
t1 = (1, 2, 'a', 'b')
print(t1[0])      # 1
print(t1[3])     # 'b'
```

- 튜플 슬라이싱(Slicing)

```
t1 = (1, 2, 'a', 'b')
print(t1[1:])

# (2, 'a', 'b')
```

- 튜플 더하기

```
t1 = (1, 2, 'a', 'b')
t2 = (3, 4)
print(t1 + t2)

# (1, 2, 'a', 'b', 3, 4)
```



- 튜플 곱하기

```
t2 = (3, 4)
print(t2 * 3)

# (3, 4, 3, 4, 3, 4)
```

- 튜플 길이 반환

```
t1 = (1, 2, 'a', 'b')
len(t1)

# return 4
```

- ✓ 딕셔너리(Dict): 원소 간의 대응 관계로 자료를 표현
  - 예) "이름" = "superman", "소속" = "MSE" 등으로 자료 표현
- ⇒ 연관 배열(Associative Array) = 해시(Hash)
- ✓ "Key"와 "Value"를 한 쌍으로 갖는 자료형
- ✓ **딕셔너리(Dict)는 Key를 통해 Value에 접근** ← 딕셔너리의 특징
  - cf) 리스트나 튜플은 순차적으로(Sequential) 해당 요소에 접근
  - 영어사전에서 단어가 있는 곳에 내용을 찾는 것과 동일
- ✓ 딕셔너리(Dict) 자료형 생성 - "{}" 사용

```
# {Key1:Value1, Key2:Value2, Key3:Value3, ...}  
dic = {'name':'hyungmin', 'home':'csdl', 'phone': '2408'}
```

- { }안에 Key:Value의 쌍 원소들로 생성
- 각각의 원소는 쉼표(,)로 구분
- Key는 변하지 않는 값, Value는 변하는 값 또는 변하지 않는 값

```
a = {1: 'hi'}  
a = { 'a': [1,2,3]}
```

- ✓ 딕셔너리(Dict) 자료에 원소 쌍 추가

```
a = {1: 'a'}  
a[2] = 'b'  
print(a)           # {1: 'a', 2: 'b'}  
  
a['name'] = 'john'  
print(a)           # {1: 'a', 2: 'b', 'name': 'john'}
```

- ✓ 딕셔너리(Dict) 자료의 원소 쌍 삭제

```
del a[1]  
print(a)           # {2: 'b', 'name': 'john'}
```

- ✓ 딕셔너리(Dict) - Key를 사용해 Value 접근

```
grade = {'Paul': 90, 'Mark': 12}
grade['Paul']           # 90
grade['Mark']           # 12
```

- 딕셔너리는 Key를 사용해서 Value에 접근

- cf) 리스트, 튜플, 문자열은 인덱싱 또는 슬라이싱으로 원소에 접근

```
a = {1:'a', 2:'b'}
print(a[1])           # 'a'
print(a[2])           # 'b'
```

- ✓ 딕셔너리(Dict) 사용시 주의 사항

- 딕셔너리의 Key는 고유한 값

- 중복되는 Key 값은 하나를 제외하고 모두 무시

```
a = {1:'a', 1:'b'}
print(a)              # {1: 'b'}
```

✓ 딕셔너리(Dict) 주요 멤버 함수

- keys ← 딕셔너리의 Key를 리스트로 변환

```
a = {'name': 'hyungmin', 'home': 'csdl', 'phone': '2408'}  
print(a.keys())          # dict_keys(['name', 'home', 'phone'])
```

- a.keys()는 a의 Key만을 모아 dict\_keys 객체를 반환

```
list(a.keys())           # ['name', 'phone', 'birth']
```

- values ← 딕셔너리의 value를 리스트로 변환

```
print(a.values())        # dict_values(['hyungmin', 'msdl', '2408'])  
print(list(a.values())) # ['hyungmin', 'msdl', '2408']
```

- items ← 딕셔너리의 Key와 Value 쌍 얻기

```
print(a.items())  
# dict_items([('name', 'hyungmin'), ('home', 'msdl'), ('phone', '2408')])
```

- clear ← Key: Value 쌍 모두 지우기

```
a.clear()
print(a)      # {}
```

- get ← 딕셔너리의 Key로 Value 얻기

```
a = {'name': 'hmjeon', 'home': 'msdl', 'phone': '2408'}
print(a.get('name'))      # 'hmjeon'
print(a.get('phone'))    # '2408'
```

- a.get('name')와 a['name']는 동일, 하지만 key가 존재하지 않을 경우

- a.get('nokey') → None반환
- a['nokey'] → Error 발생

- ✓ 해당 Key가 딕셔너리 안에 있는지 조사하기 → in

```
a = {'name': 'hmjeon', 'home': 'msdl', 'phone': '2408'}
print('name' in a)      # True
print('nickname' in a)  # False
```

- ✓ 원소의 집합 형태로 구성된 Python 자료형
- ✓ set() 함수를 사용해서 집합 자료 생성, 인자는 **리스트** 또는 **문자열**

```
a1 = set([1,2,3])  
print(a1)           # {1, 2, 3}  
  
a2 = set("Hello")  
print(a2)           # {'e', 'H', 'l', 'o'}
```

- 빈 집합 자료형 생성 → a3 = set()
- a2 셋 자료형의 특징
  - ◆ 중복을 허용하지 않음
  - ◆ 순서가 없음
- ✓ 딕셔너리와 셋은 순서가 없는 자료형 → **인덱싱 불가**

- ✓ 셋(set) 자료형의 변환 → 리스트 또는 튜플

```
s1 = set([1,2,3])

l1 = list(s1)
print(l1)           # [1, 2, 3]
print(l1[0])       # 1

t1 = tuple(s1)
print(t1)          # (1, 2, 3)
print(t1[0])      # 1
```

- ✓ 셋(set) 자료형 - 교집합(&, intersection())

```
s1 = set([1, 2, 3, 4, 5, 6])
s2 = set([4, 5, 6, 7, 8, 9])
print(s1 & s2)           # {4, 5, 6}
```

- s1과 s2의 교집합 - "&" 사용
- intersection() 멤버 함수 사용

```
s1.intersection(s2)     # {4, 5, 6}
```



✓ 셋(set) 자료형 - **합집합(|, union)**

```
s1 | s2          # {1, 2, 3, 4, 5, 6, 7, 8, 9}
s1.union(s2)    # {1, 2, 3, 4, 5, 6, 7, 8, 9}
```

- s1과 s2의 합집합 - "|" 사용
- union() 멤버 함수 사용

✓ 셋(set) 자료형 - **차집합(-, difference)**

```
s1 - s2          # {1, 2, 3}
s2 - s1          # {8, 9, 7}
s1.difference(s2) # {1, 2, 3}
s2.difference(s1) # {8, 9, 7}
```

- s1과 s2의 차집합 - "-" 기호 사용
- difference() 멤버 함수 사용

✓ 셋(set) 자료구조의 멤버함수

○ **add** 멤버 함수 - 원소 1개 추가

```
s1 = set([1, 2, 3])
s1.add(4)
print(s1)           # {1, 2, 3, 4}
```

○ **update** 멤버 함수 - 원소 여러 개 추가

```
s1 = set([1, 2, 3])
s1.update([4, 5, 6])
print(s1)           # {1, 2, 3, 4, 5, 6}
```

○ **remove** 멤버 함수 - 특정 값 제거

```
s1 = set([1, 2, 3])
s1.remove(2)
print(s1)           # {1, 3}
```

- ✓ 조건에 해당되는 문장을 선택적으로 수행

```
money = True
if money:
    print("by bus")
else:
    print("on foot")
```

- ✓ if문 - 기본 구조

```
if 조건문:
    수행문1
    수행문2
    ...
else:
    수행문3
    수행문4
    ...
```

- ✓ "수행문" 앞은 항상 들여쓰기(탭;Tab) 또는 공백(Spacebar 4개 권장) 사용
  - 공백(Spacebar) vs 탭(Tab) → 2가지를 혼용하지 말자

✓ '조건문' #1 - 비교(관계)연산자 (<, >, ==, !=, >=, <=)

- $x < y$  → x가 y보다 작다
- $x > y$  → x가 y보다 크다
- $x == y$  → x와 y가 같다
- $x != y$  → x와 y보다 같지 않다
- $x >= y$  → x와 y보다 크거나 같다
- $x <= y$  → x와 y보다 작거나 같다

```
x = 3
y = 2
print(x > y)      # True
print(x < y)      # False
print(x == y)     # False
print(x != y)     # True
```

✓ 프로그래밍 언어에서는 0을 False로 1을 True로 정의 → 다른 정수는?

- ✓ '조건문' #2 - 논리 연산자 (and, or, not)

연산자	설명
x or y	x와 y 둘 중에 하나만 참 이어도 참
x and y	x와 y 모두 참이어야 참
not x	x가 거짓이면 참

- ✓ if문 활용 예 #1 (비교 연산자 및 논리 연산자 사용)

- "3000원 이상" 있거나 "카드"가 있다면 → 버스를 타고
- 그렇지 않으면 → 걸어라

```
money = 2000
card = True

if money >= 3000 or card:
    print("by bus")
else:
    print("on foot")
```

- ✓ "x in s" 및 "x not in s" 활용

in	not in
x in 리스트	x not in 리스트
x in 튜플	x not in 튜플
x in 문자열	x not in 문자열

```
1 in [1, 2, 3]           # True
1 not in [1, 2, 3]      # False
```

- ✓ if문 활용 예 #2 (x in s 사용)

- 주머니에 돈이 있으면 버스를 타고, 없으면 걸어 가라

```
pocket = ['paper', 'cellphone', 'money']

if 'money' in pocket:
    print("by bus")
else:
    print("on foot")
# by bus
```

- ✓ 조건문 if에서 **수행문이 아무런 일도 하지 않을 때** → **pass** 키워드

```
pocket = ['paper', 'money', 'cellphone']  
  
if 'money' in pocket:  
    pass  
else:  
    print("by bus")
```

- ✓ 다중 조건 판단 - **elif** 키워드

```
pocket = ['paper', 'cellphone']  
card = True  
  
if 'money' in pocket:  
    print("by bus")  
elif card:  
    print("by Uber")  
else:  
    print("on foot")  
# 택시를 타고 가라
```

- ✓ 수행문 간결화(콜론 뒤 수행문 작성), 조건부 표현(Conditional Expression)

```
pocket = ['paper', 'money', 'cellphone']

if 'money' in pocket: pass
else: print("by bus")

if score >= 80:
    mes = "A"
else:
    mes = "F"

mes = "A" if score >= 80 else "F"
```



- ✓ 반복문(while, for): 특정 구간의 수행문장들을 반복
- ✓ 기본 구조 - while

```
while <조건문>:  
    <수행문1>  
    <수행문2>  
    <수행문3>  
    ...
```

- ✓ while문 활용 예)
  - 열 번 찍어 안 넘어가는 나무 없다

```
Hit = 0  
  
while Hit < 10:  
    Hit = Hit + 1  
    print( " 나무를 %d번 찍었습니다." % Hit)  
    if Hit == 10:  
        print("나무 넘어갑니다.")  
  
# 나무를 1번 찍었습니다.  
# 나무를 2번 찍었습니다.  
...  
# 나무를 10번 찍었습니다.  
# 나무 넘어갑니다.
```

- ✓ While문의 컨트롤 #1 → 강제 종료 - **break** 키워드

```
coffee = 10
while 1:
    coffee = coffee - 1
    print("남은 커피는 %d개" % coffee)
    if coffee == 0:
        print("판매 중지!")
        break
```

- ✓ While문의 컨트롤 #2 → 처음으로 돌아가기 - **continue** 키워드

```
a = 0
while a < 10:
    a = a + 1
    if a % 2 == 0: continue
    print(a)          # 1, 3, 5, 7, 9
```

- ✓ While문의 컨트롤 #3 → 무한루프(Infinity Loop)

```
while True:
    수행문1
    수행문2
    ...
```

- ✓ While문 vs for문
- ✓ For문 기본 구조

```
for 변수 in 리스트, 튜플, 또는 문자열:  
    수행문1  
    수행문2  
    ...
```

- ✓ 리스트, 튜플 또는 문자열의 **첫 번째 요소부터 마지막 요소까지** 차례로 **변수**에 대입되어 "수행문1", "수행문2" 등이 수행
- ✓ for문 활용 #1)

```
digit = ['one', 'two', 'three']  
  
for i in digit:  
    ...  
    print(i)  
    ...  
# one, two, three
```

- ✓ for문 활용 #2)

```
a = [(1,2), (3,4), (5,6)]  
  
for (first, last) in a:  
    print(first + last)  
  
# 3  
# 7  
# 11
```

- ✓ for문 컨트롤 - **continue, break** 키워드

```
score = [90, 25, 67, 45, 80]  
  
index = 0  
for val in score:  
    index = index + 1  
    if val < 60:  
        continue  
    print("%d" % index)
```

- ✓ 함수 작성의 목적 → 모듈화 → 가독성 증가 / 유지보수 편리
- ✓ 내장 함수 vs 사용자 정의 함수
- ✓ 함수 vs 멤버 함수
- ✓ 파이썬 함수의 기본 구조

```
def 함수명(매개변수):  
    <수행문1>  
    <수행문2>  
    return 변수명
```

- **def**: 함수 작성을 위한 Python 키워드
- **함수명**: 함수 기능을 잘 표현하는 이름으로 프로그래머가 직접 작성
- **매개변수**: 입력으로 전달되는 변수
- **return 변수명**: 함수 밖으로 결과를 돌려주는 변수 값

## ✓ 매개변수와 인수

- 매개변수: 함수에 입력으로 전달된 값을 받는 변수
- 인수: 함수를 호출할 때 전달하는 입력 값

```
def add(a, b):      # a, b는 매개변수
    return a+b

print(add(3, 4))   # 3, 4는 인수
```

## ✓ 함수의 다양한 형태

- 함수 형태 #1 - 입력(O), 반환(O)

```
def add(a, b):
    result = a + b
    return result
```

- 함수 형태 #2 - 입력(X), 반환(O)

```
def say():
    return 'Hello'
```

- 함수 형태 #3 - 입력(O), 반환(X)

```
def add(a, b):  
    print("%d + %d = %d" % (a, b, a+b))
```

- 함수 형태 #4 - 입력(X), 반환(X)

```
def say():  
    print('Hello')
```

- ✓ 함수 고급 #1 - 매개변수 지정

```
result = add(a=3, b=7)  
print(result)  
# 10
```

- 매개변수 사용 → 순서에 무관

```
result = add(b=5, a=3)  
print(result)  
# 8
```

✓ 함수 고급 #2 - 가변 인자

```
def 함수이름(*매개변수):  
    <수행문1>  
    <수행문2>  
    ...
```

- Add(1, 2, 3) 및 add(1, 2, 3, ..., 9, 10)를 동시에 사용할 수 있는 add() 함수

```
def add(*args):  
    result = 0  
    for i in args:  
        result = result + i  
    return result  
  
result = add_many(1,2,3)  
print(result) # 6  
result = add_many(1,2,3,4,5,6,7,8,9,10)  
print(result) # 55
```

- \*args는 튜플로 자료를 받아 함수 내부로 전달



- ✓ 함수 고급 #3 - 튜플을 통한 다변수의 결과 반환

```
def add_and_mul(a, b):  
    return a+b, a*b  
  
result = add_and_mul(3, 4)      # (7,12)
```

- 결과를 받는 변수 result → 튜플 자료형 (a + b, a \* b)

```
result1, result2 = add_and_mul(3, 4)
```

- result1 = 7, result2 = 12

- ✓ 함수 고급 #4 - 매개변수 초기값 설정

```
def intro_myself(name, old, student=True):  
    print("My name is %s." % name)  
    print("I am %d years old." % old)  
    if student:  
        print("I am a student.")  
    else:  
        print("I am a teacher.")
```

- “global” 키워드 사용 → 가급적 사용하지 말 것

```
a = 1
def vartest():
    global a
    a = a + 1

vartest()
print(a)
```

## ✓ Lamda 함수

- 함수를 생성할 때 사용하는 예약어 → def와 동일한 역할
- 한 줄로 간결하게 함수를 작성
- 특수한 상황의 함수 작성

```
add = lambda a, b: a+b
result = add(3, 4)
print(result) # 7
```

⇒ return 명령어가 없어도 결과를 반환

- 사용 예) 매개변수에 초기값을 설정

```
say_myself("Paul", 40)
say_myself("Mark", 30, False)
```

- 매개변수 초기화 시 → 초기화된 매개변수는 항상 뒤에 위치

- ✓ 변수 접근범위 (함수 내 변수 ≠ 함수 밖의 변수)

```
a = 1
def vartest(a):
    a = a + 1

vartest(a)
print(a)
```

- ✓ 함수 내부에서 함수 밖의 변수 제어

- "return" 키워드 사용

```
a = 1
def vartest(a):
    a = a + 1
    return a

a = vartest(a)
print(a)
```

- ✓ 사용자 입력 - "input" 키워드 사용

```
a = input()
Typing → Python is easy to learn
Result → a = 'Python is easy to learn'

input("type the sentence")
```

- ✓ 화면 출력 - print() 함수

- 큰 따옴표("")로 둘러싸인 문자열 == "+" 연산

```
print("life" "is" "too short")    # lifeistoo short
print("life"+"is"+"too short")   # lifeistoo short
```

- 문자열 띄어쓰기 == "," 사용

```
print("life", "is", "too short") # life is too short
```

- 한 줄에 결과 출력 == end = '' 사용

```
for i in range(10):
    print(i, end=' ')

# 0 1 2 3 4 5 6 7 8 9
```

✓ 파이썬 모듈 작성

```
# mod1.py
def add(a, b):
    return a + b

def sub(a, b):
    return a - b
```

○ 파일 mod1.py을 작성 → 파이썬 파일 = 파이썬 모듈

✓ 파이썬 모듈 사용 #1

```
import mod1
print(mod1.add(3, 4))      # 7
print(mod1.sub(4, 2))     # 2
```

✓ 파이썬 모듈 사용 #2

```
from mod1 import add
add(3, 4)                  # 7
```

✓ 파이썬 모듈 사용 #3

```
from mod1 import add, sub
from mod1 import *
```

✓ if \_\_name\_\_ == "\_\_main\_\_"

```
# mod1.py
def add(a, b):
    return a+b

def sub(a, b):
    return a-b

print(add(1, 4))
print(sub(4, 2))
```

○ Import 시 → 결과가 출력

```
# mod1.py
def add(a, b):
    return a+b

def sub(a, b):
    return a-b

if __name__ == "__main__":
    print(add(1, 4))
    print(sub(4, 2))
```

- ✓ 꼭 다뤄 봐야 할 Python 모듈
  - matplotlib
  - numpy
  - pandas
  - seaborn

