



2. 파이썬

2022

Table of Contents

I. Python 개발환경 준비 및 주요 모듈이해

II. 데이터 구조 이해 및 실습

III. 데이터 구조의 처리

I. Python 개발환경 준비 및 주요 모듈이해

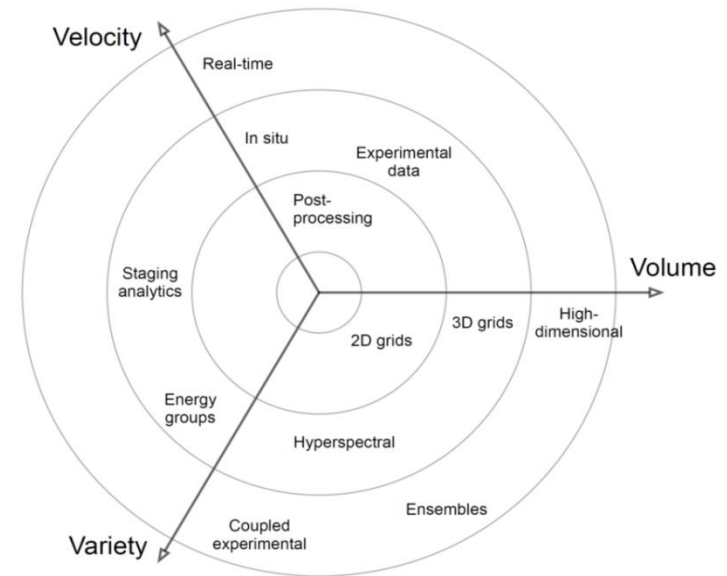
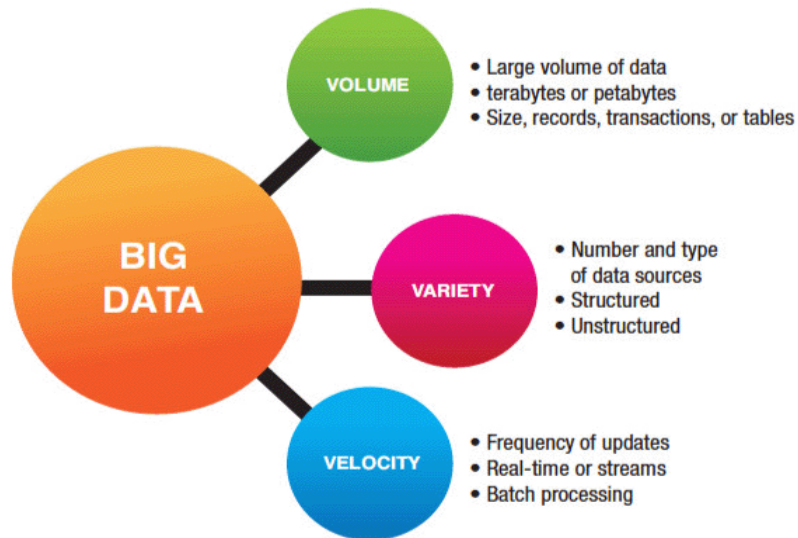
1. 빅데이터 개요 및 플랫폼
2. Python 설치 및 패키지 설치

1.빅데이터 개요 및 플랫폼

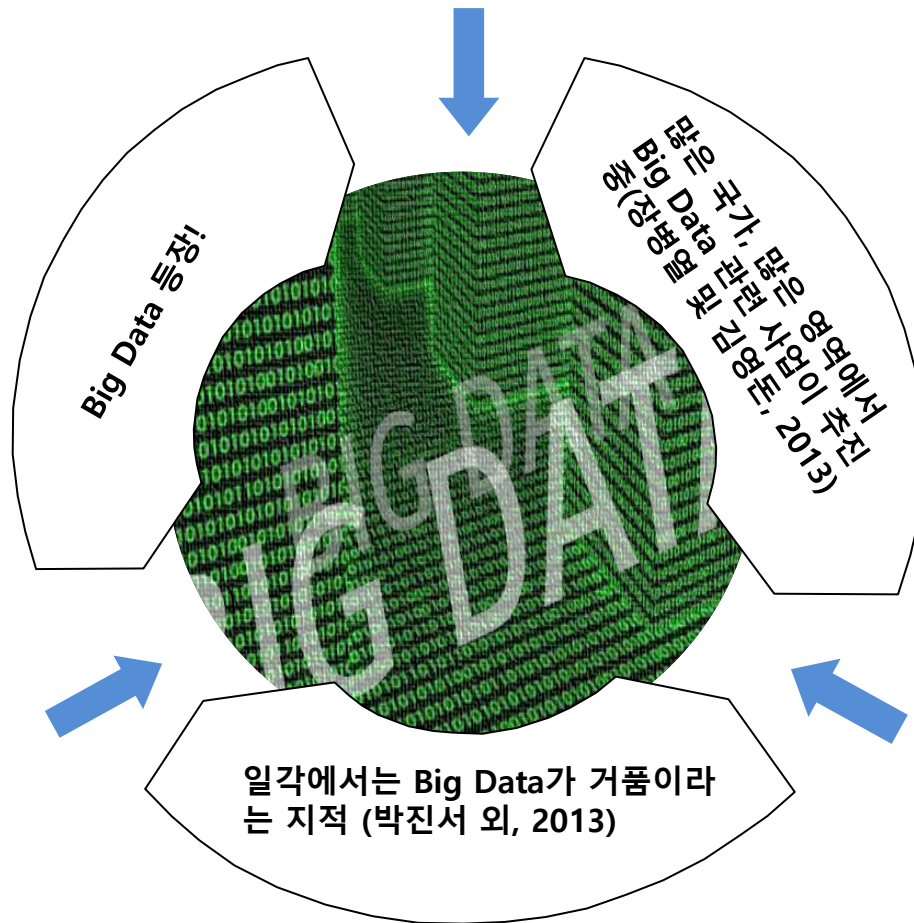
➤ Big Data 정의

- ✓ 더 나은 의사결정, 시사점 발견 및 프로세스 최적화를 위해 사용되는 새로운 형태의 정보처리가 필요한 대용량,
초고속 및 다양성의 특성을 가진 정보자산(Gartner, 2012)
- ✓ 일반적인 데이터베이스 소프트웨어 도구가 수집, 저장, 관리, 분석하기 어려운 대규모의 Data.
3V를 가진 Data(Velocity, Volume, Variety) (Mckinsey, 2011)

The Three Vs of Big Data



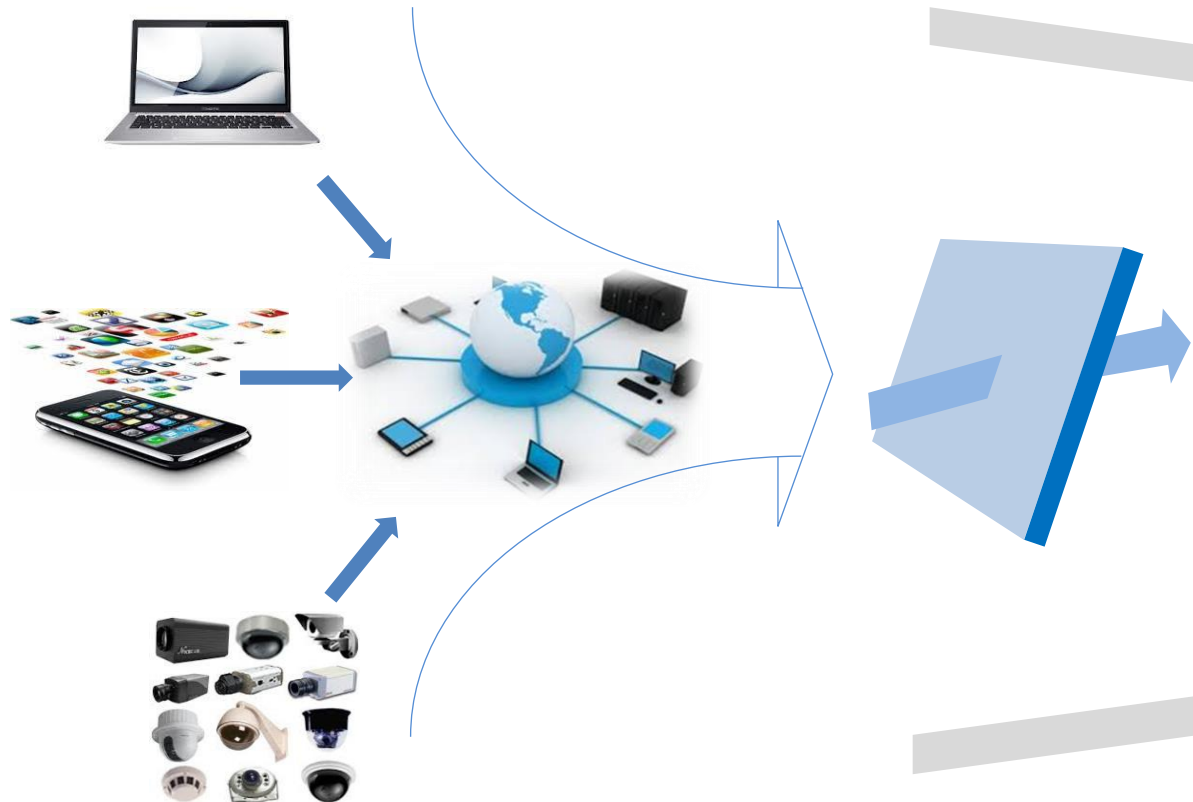
1.빅데이터 개요 및 플랫폼



- ❖ Big Data는 **전혀 새로운 개념이 아님**
- ❖ Big Data의 **본질은 결국 Data**
- ❖ Big Data를 잘 하기 위해서는 **Data 이해 필수**

1.빅데이터 개요 및 플랫폼

Data는 계속 존재해왔지만, 그 동안은 Data의 측정 및 수집이 어려웠음



Insight from Data!

- ✓ Data는 단순히 크기와 관련된 문제가 아님
- ✓ 새로운 Data처리 및 분석방법을 통해 새로운 통찰력을 찾을 수 있는 기회로 이해(양혜영, 2012)

많은 기회를 이해(양혜영, 2012)
올해 내로 올랐던 것을 찾을 수
있을 Data처리 및 분석방법

Taming Data

기술과 장비의 발달로 인해 점차 Data의 측정과 수집이 가능하게 됨

1.빅데이터 개요 및 플랫폼

➤ Data Analytics 부각 배경

- ✓ 자료 저장 기술의 발달로 인한 **대용량 데이터** 수집
- ✓ 지식 서비스기반의 산업구조 도래
- ✓ 계산 능력 발달로 인한 **대용량 데이터의 분석 가능**

➤ 최근 Data Analytics의 특징

- ① 대용량 관측 가능한 데이터를 대상
- ② 분석모형의 일반화 및 예측이 중요
- ③ 컴퓨터 계산 능력을 활용하여 경험적으로 지식 발견
- ④ 다학제적 기반: 통계학+컴퓨터과학+경영학+...



1.빅데이터 개요 및 플랫폼

➤ Data Analytics 모형 구분

지도학습 (Supervised Learning)

종속 및 독립변수를 이용하여 주어진 독립(설명)변수를 바탕으로 종속(반응)변수 예측 모형 제시

예: 회귀/분류 모형

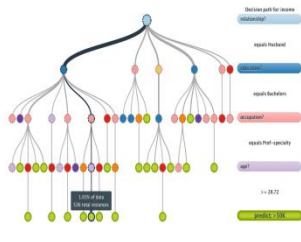
비지도학습 (Unsupervised Learning)

Target(종속변수/반응변수)이 없으며, 독립(설명)변수 간의 관계나 이를 바탕으로 개체들을 구분하여 의미 있는 결과를 제시

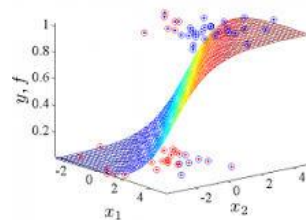
예:군집 분석, 연관성 분석, 주성분 / 요인분석



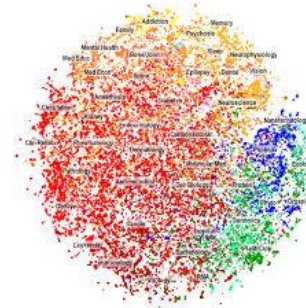
[decision tree]



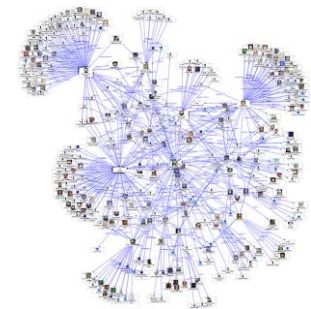
[logistic regression]



[clustering analysis]

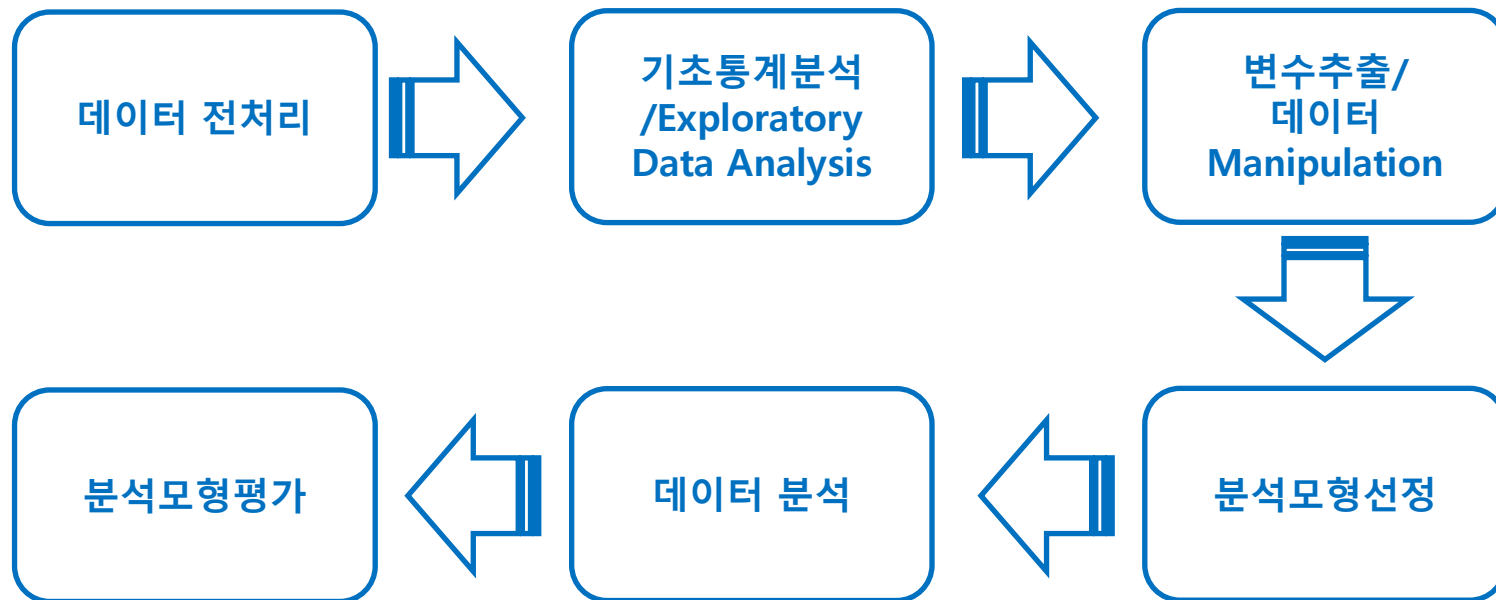


[link analysis]



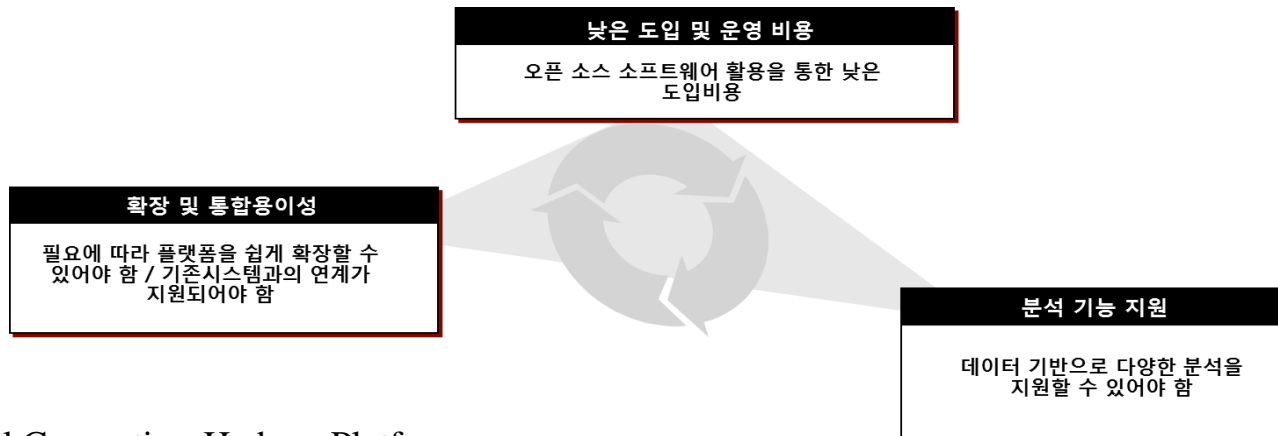
1.빅데이터 개요 및 플랫폼

➤ Data Analytics 세부 절차

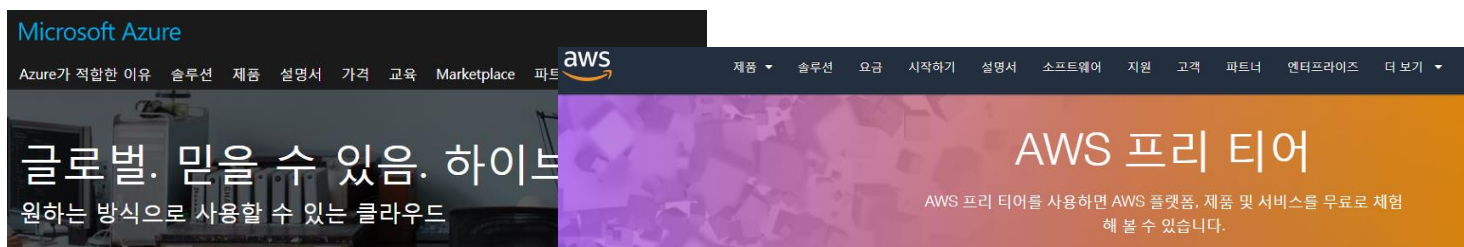
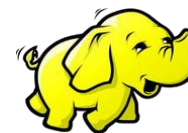


1.빅데이터 개요 및 플랫폼

➤ 분석 플랫폼과 도구

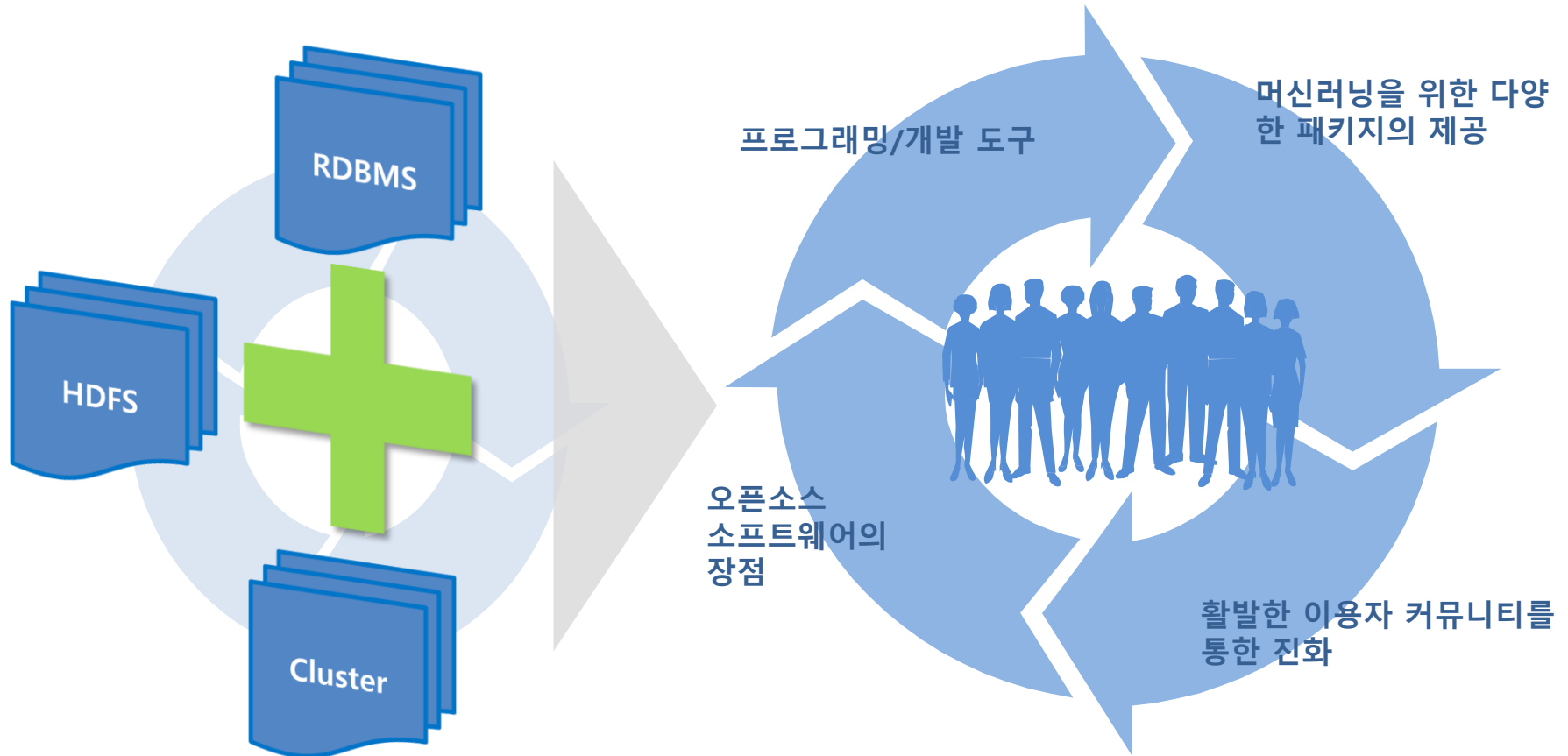


- ✓ Local Computing-Hadoop Platform
 - Low cost , Scalability and Failure-Tolerant structure
 - Open Source Project for Distributed File System / in Java and MapReduce framework
- ✓ Cloud Computing-Hadoop, HPC, ...
 - Low cost , Scalability and Failure-Tolerant structure
 - Amazon Web Service, Microsoft Azure etc.



1.빅데이터 개요 및 플랫폼

- Data Platform에 대한 이해 및 구축 방안: Python
 - 데이터플랫폼에서 분석엔진으로 자리잡음

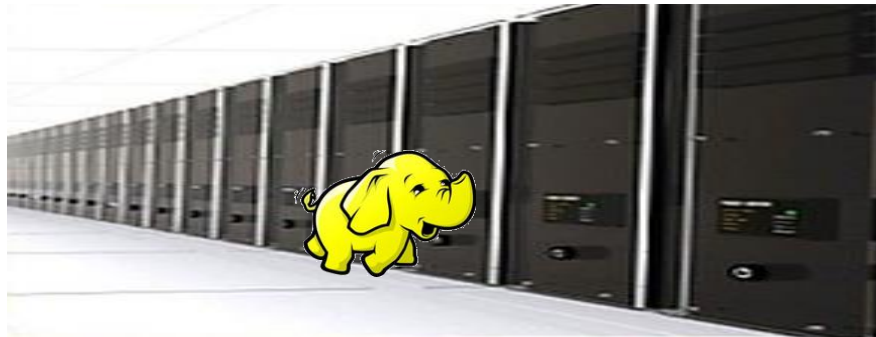


1.빅데이터 개요 및 플랫폼

➤ 빅데이터 분석 환경

Hadoop Platform

- Low cost , Scalability and Failure-Tolerant structure
- Open Source Project for Distributed File System based on Google's MapReduce-based computational infrastructure
- Written in Java and MapReduce processing framework
- Support Streaming API
 - Any tool/language, that can support Standard In & Out, can execute MapReduce!

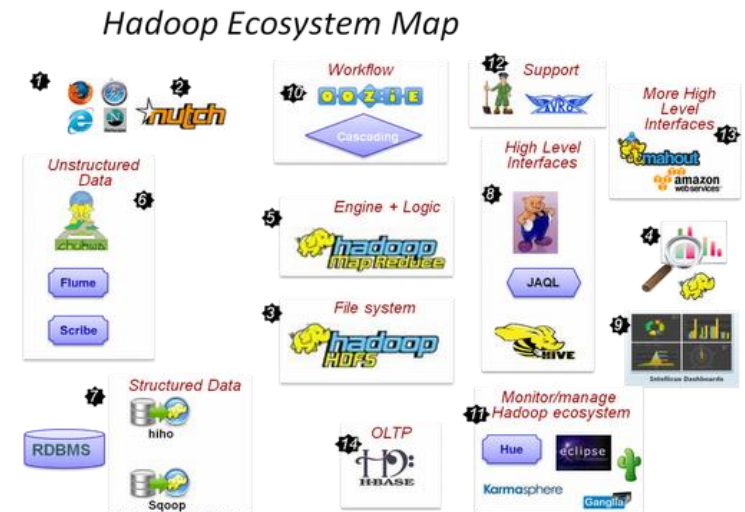


1.빅데이터 개요 및 플랫폼

2) 빅데이터 기술에 대한 이해 : Big Data 기술: 다양한 기술이 모인 Ecosystem

- ✓ 대상: Web의 대용량 데이터
- ✓ Nutch: 웹 데이터 수집
- ✓ Hadoop: DFS(Distributed File System)로서 데이터를 저장
- ✓ Map/Reduce Framework: 분산환경에서 데이터 분석에 사용
- ✓ Hive/Pig: Map/Reduce 이용을 지원하는 고수준의 Interface tool
- ✓ Oozie: Work flow 관리
- ✓ Zoo Keeper: Framework 및 클러스터 관리
- ✓ Hbase: NoSQL
- ✓ Flume: 비정형 데이터 수집

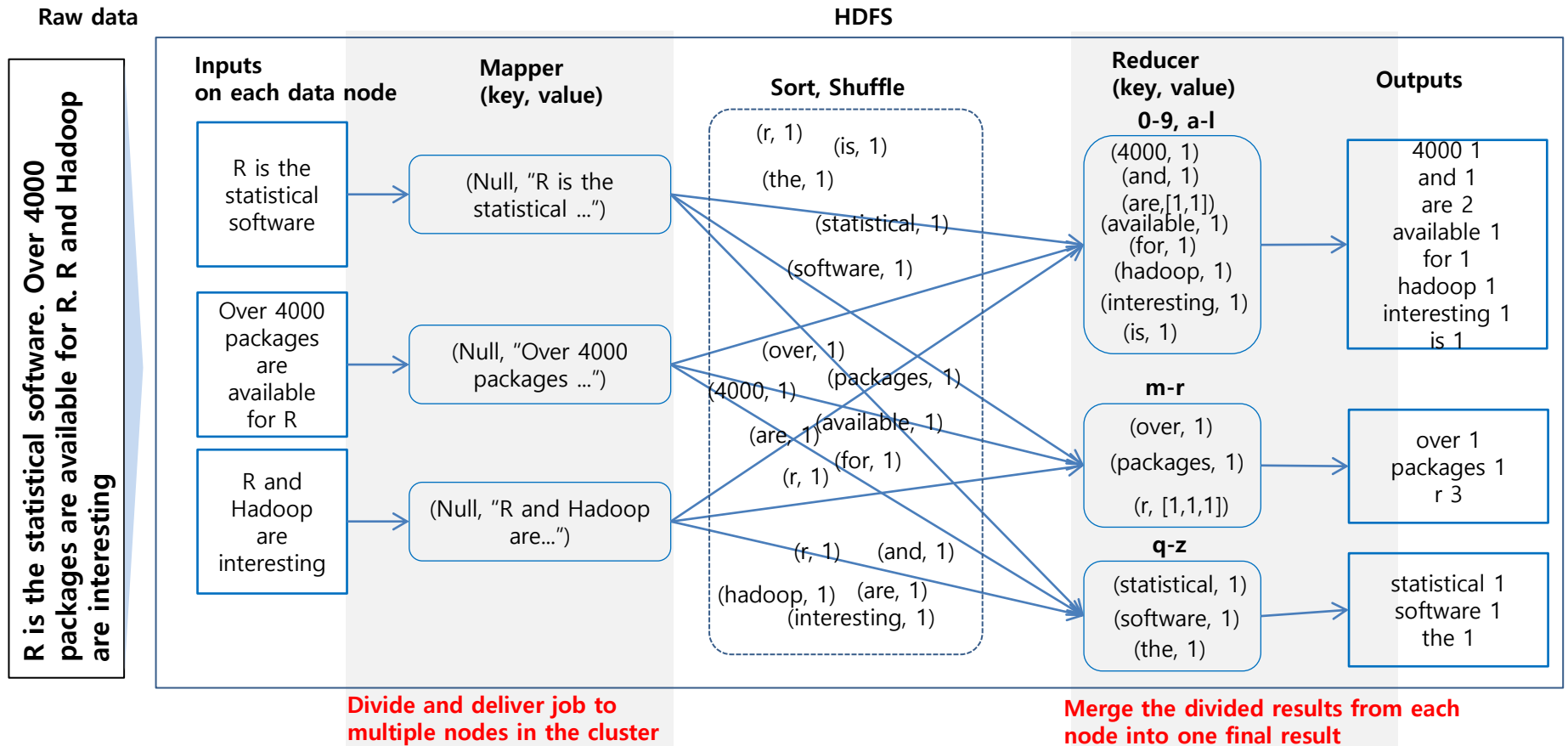
※ 분석을 위해 Hadoop, Map/Reduce, Hive, Pig 이해 필요



1.빅데이터 개요 및 플랫폼

➤ Hadoop platform

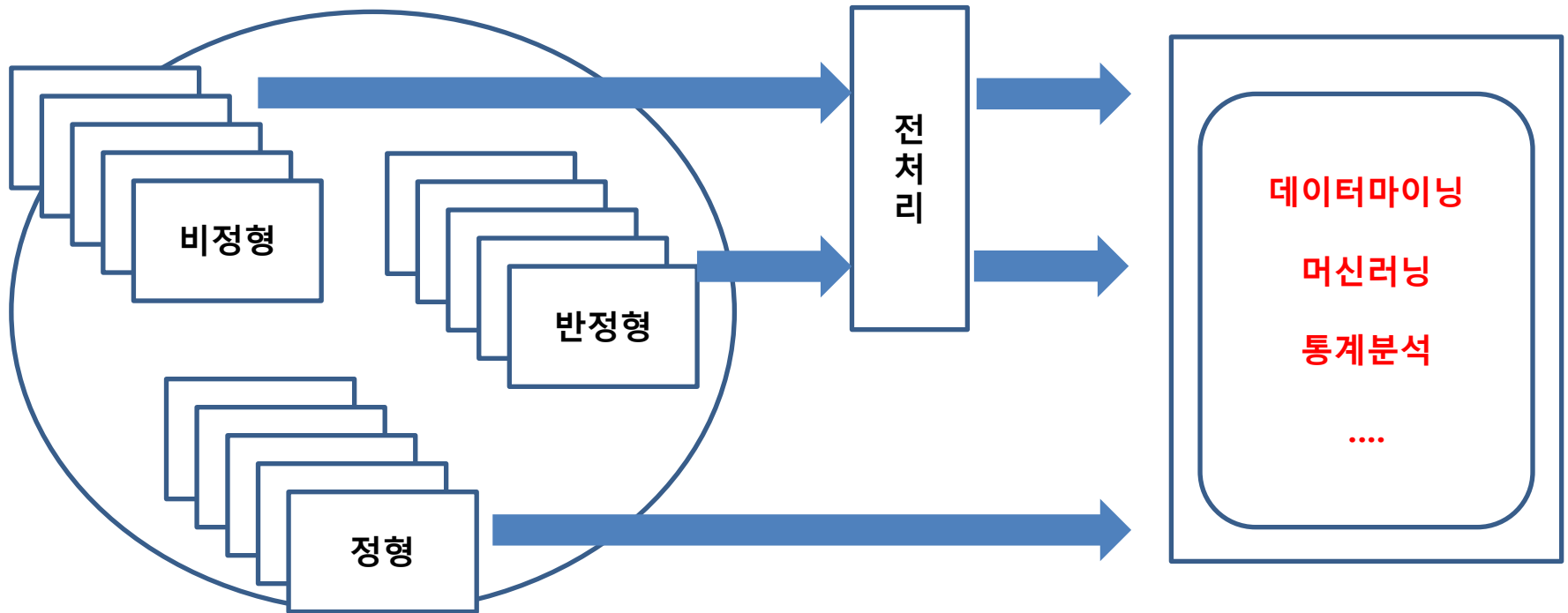
- MapReduce: Framework for processing large-scale data on DFS
- Example of word count using MapReduce



1.빅데이터 개요 및 플랫폼

➤ Big Data 분석

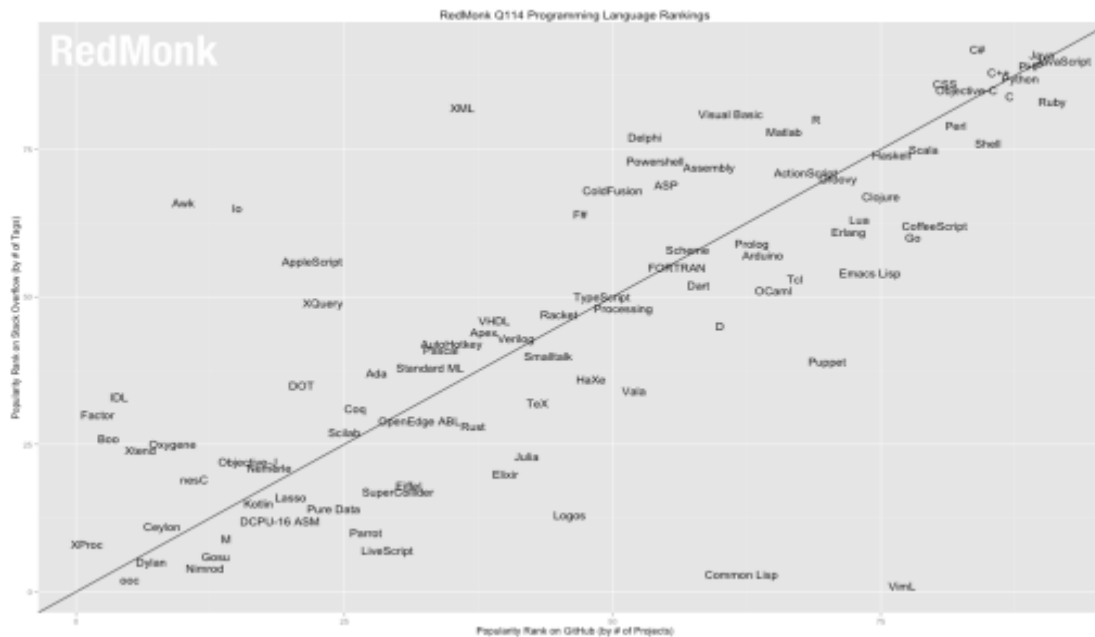
- 비정형/반정형 데이터의 경우 분석에 앞서 데이터 전처리가 중요
- 적용되는 기법의 머신러닝 알고리즘이 병렬 처리를 지원해야 함



2. Python 설치 및 패키지 설치

"Python is 5th of top programming languages in latest RedMonk ranking"

– <http://redmonk.com/>



Rank	Language(변동)
1	JavaScript (+1)
2	Java (-1)
3	PHP
4	C# (+2)
5	Python (-1)
6	C++ (+1)
7	Ruby (-2)
8	C
9	Objective-C
10	CSS (new)
11	Perl
12	Shell (-2)
13	Scala (-1)
14	Haskell
15	R (1)
16	Matlab (+3)
17	Clojure (+5)
18	CoffeeScript (-1)
19	Visual Basic (+1)
20	Groovy (-2)

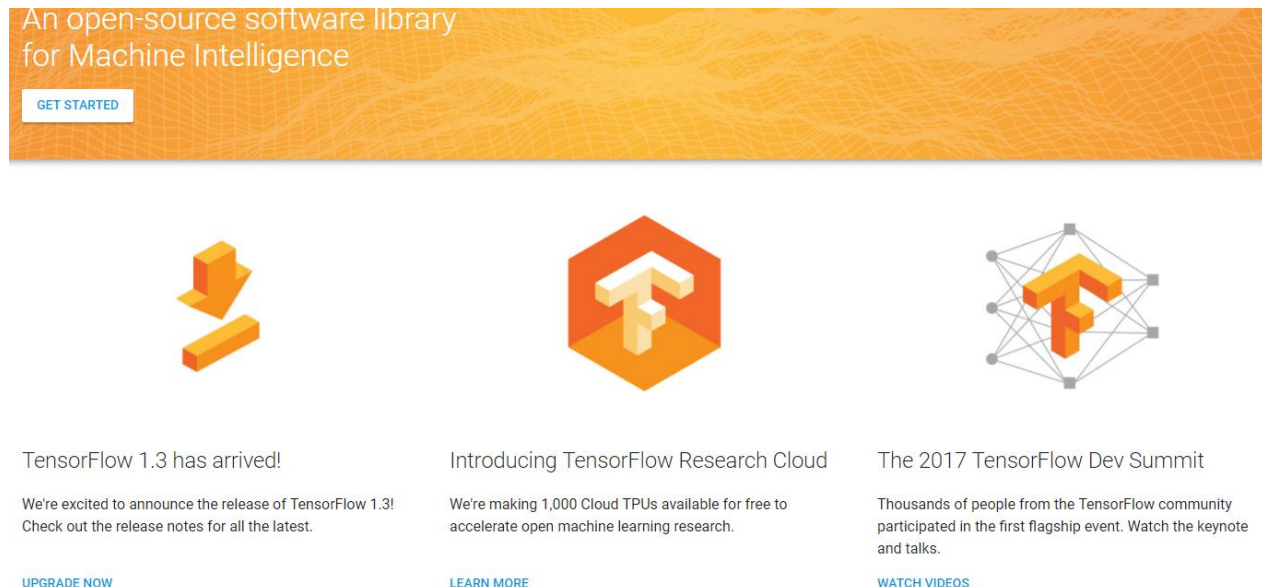
2. Python 설치 및 패키지 설치

➤ Python

- 1990년대 후반 Guido van Rossum이 개발 (네덜란드)
- 오픈소스 platform: General Public License (GPL)라이선스
- Python의 특징: High-level / Interactive / 객체지향 / Scripting 언어
- Easy-to-learn / Easy-to-read / Easy-to-maintain




➤ 확장성 및 범용성: Tensor flow!

A promotional banner for TensorFlow. The top part is an orange banner with the text "An open-source software library for Machine Intelligence" and a "GET STARTED" button. Below the banner are three promotional cards. Each card has a TensorFlow logo (a 3D orange 'T' shape) and a title. The first card is titled "TensorFlow 1.3 has arrived!" and mentions the release of TensorFlow 1.3. The second card is titled "Introducing TensorFlow Research Cloud" and mentions making 1,000 Cloud TPUs available for free. The third card is titled "The 2017 TensorFlow Dev Summit" and mentions the first flagship event. Each card also has a "WATCH VIDEOS" link.

An open-source software library
for Machine Intelligence


[GET STARTED](#)



TensorFlow 1.3 has arrived!

We're excited to announce the release of TensorFlow 1.3! Check out the release notes for all the latest.


[UPGRADE NOW](#)



Introducing TensorFlow Research Cloud

We're making 1,000 Cloud TPUs available for free to accelerate open machine learning research.

[LEARN MORE](#)



The 2017 TensorFlow Dev Summit

Thousands of people from the TensorFlow community participated in the first flagship event. Watch the keynote and talks.

[WATCH VIDEOS](#)

2. Python 설치 및 패키지 설치

- python
 - 1990년대 후반 Guido van Rossum이 개발 (네덜란드)
 - 오픈소스 cross-platform
 - General Public License (GPL)라이선스)
- Python의 특징
 - High-level 언어
 - Interpreted 언어
 - Interactive 언어
 - 객체지향 언어
 - Scripting 언어

2. Python 설치 및 패키지 설치

- python 종류
 - "Python" or "CPython" written in C/C++
 - Version 2.7
 - Version 3.5
 - "Jython"
 - Java for the JVM
 - "IronPython"
 - C# for the .Net environment
- python 사용
 - 대화식 (Interactive Interpreter)
 - 명령어 줄을 통한 Script 프로그램의 활용

2. Python 설치 및 패키지 설치

➤ python 장점


- Easy-to-learn
- Easy-to-read
- Easy-to-maintain



➤ Tensor flow!

An open-source software library
for Machine Intelligence


[GET STARTED](#)



TensorFlow 1.3 has arrived!

We're excited to announce the release of TensorFlow 1.3! Check out the release notes for all the latest.


[UPGRADE NOW](#)



Introducing TensorFlow Research Cloud

We're making 1,000 Cloud TPUs available for free to accelerate open machine learning research.

[LEARN MORE](#)



The 2017 TensorFlow Dev Summit

Thousands of people from the TensorFlow community participated in the first flagship event. Watch the keynote and talks.

[WATCH VIDEOS](#)

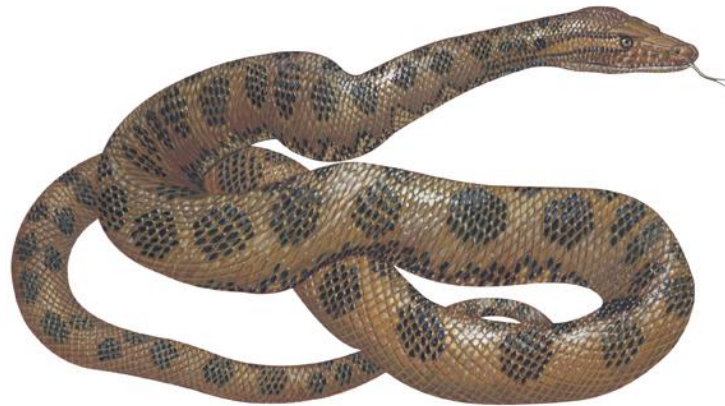
2. Python 설치 및 패키지 설치

- Python 설치

- 개발 환경

- python 다운로드 후 설치: 각종 필수 패키지를 직접 설치해야 함
 - pycharm 설치: 전문적인 개발용, 상용 툴이지만 community 버전은 무료

- 아나콘다 설치

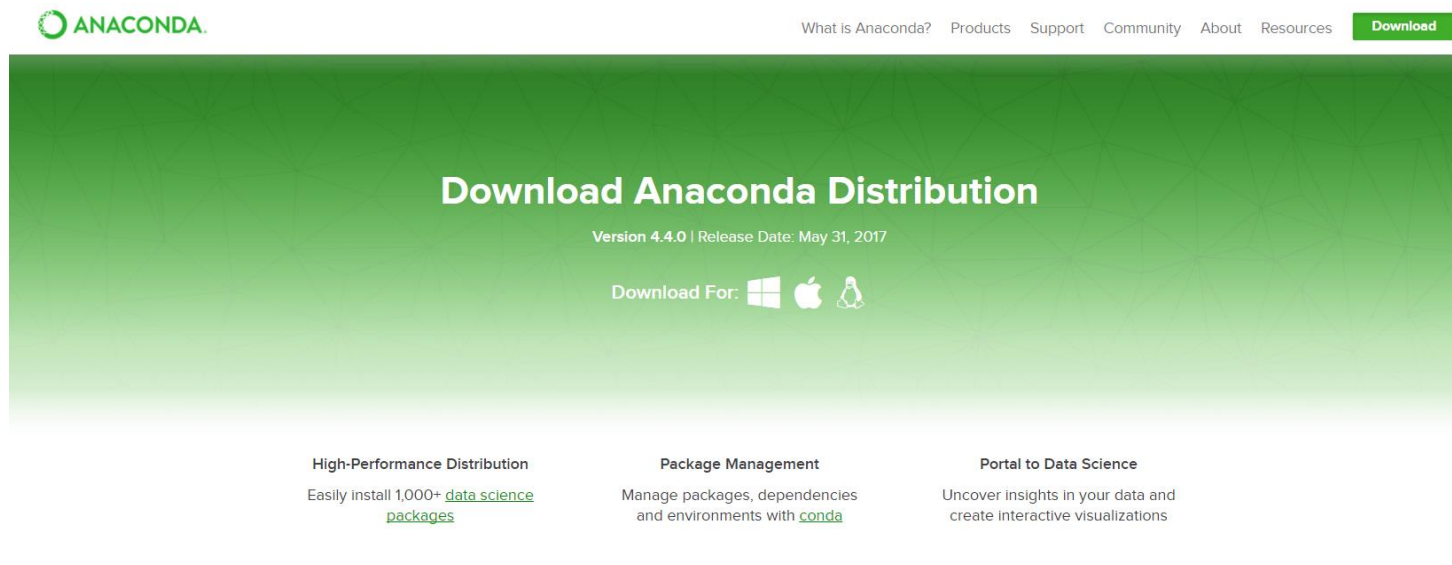


2. Python 설치 및 패키지 설치

➤ Python 설치: 아나콘다

➤ 개요

- 분석에 많이 사용되는 패키지를 미리 포함해서 배포
- conda: 패키지 의존성 및 환경 관리자 제공하여, 720 개가 넘는 패키지 접근 및 사용



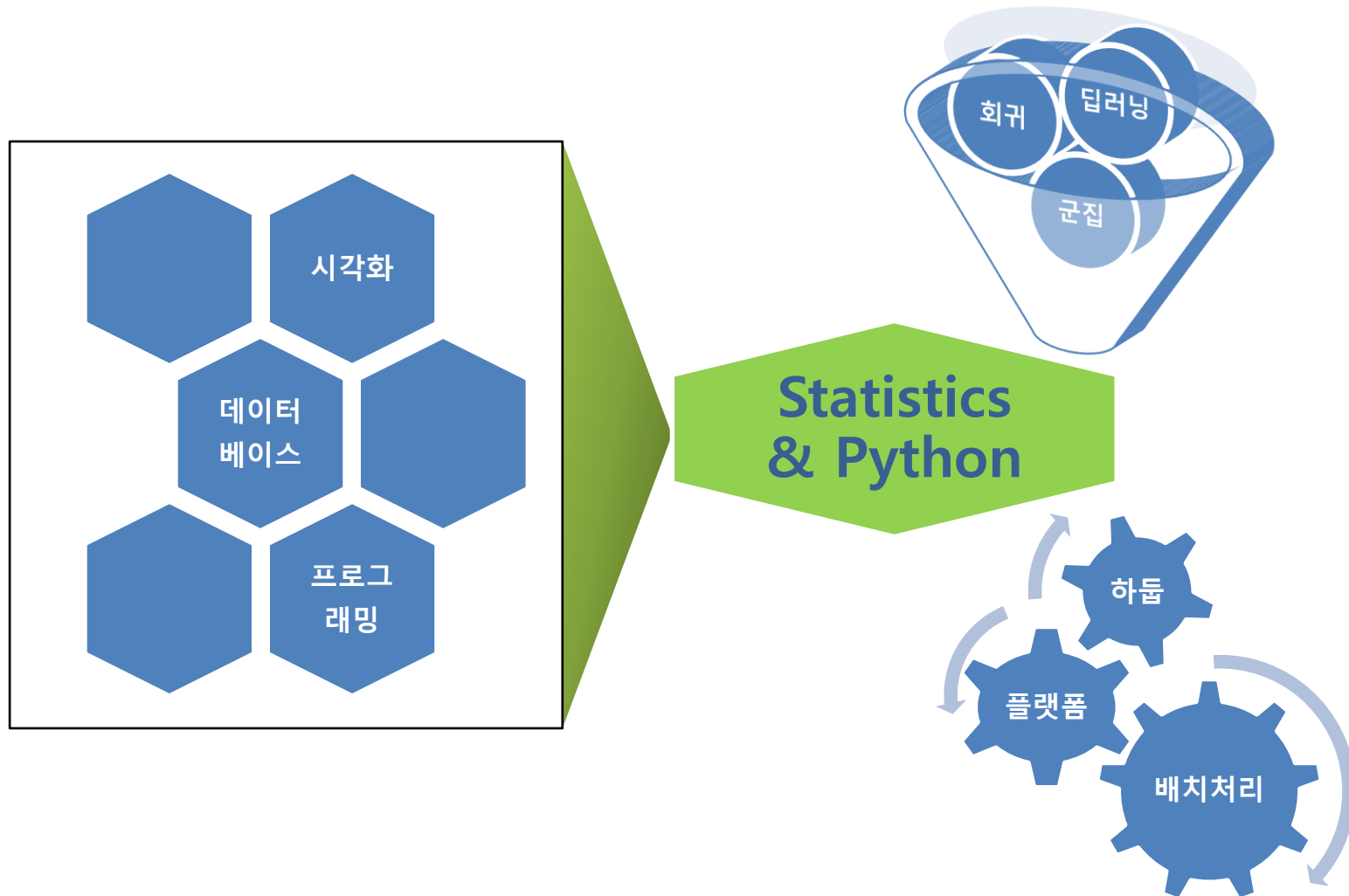
<https://www.anaconda.com/download/>

II. 데이터 구조 이해 및 실습

1. 데이터 이해
2. 주요 데이터 구조 처리

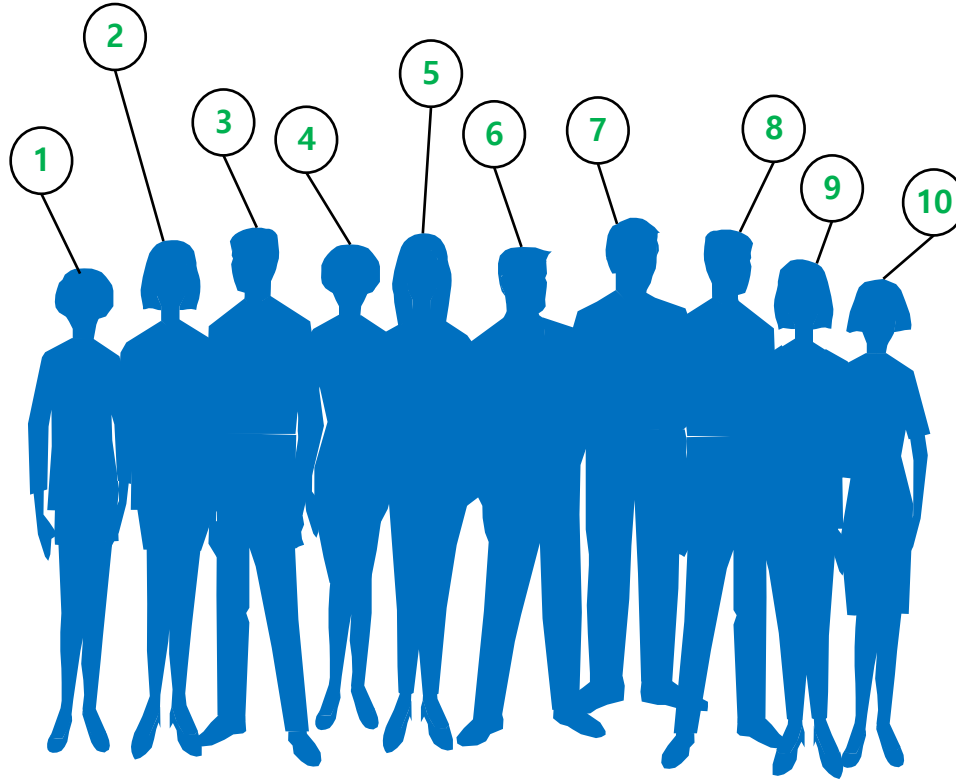
1. 데이터 이해

➤ 데이터 분석



1. 데이터 이해

➤ 데이터의 유형 : 10명의 고객

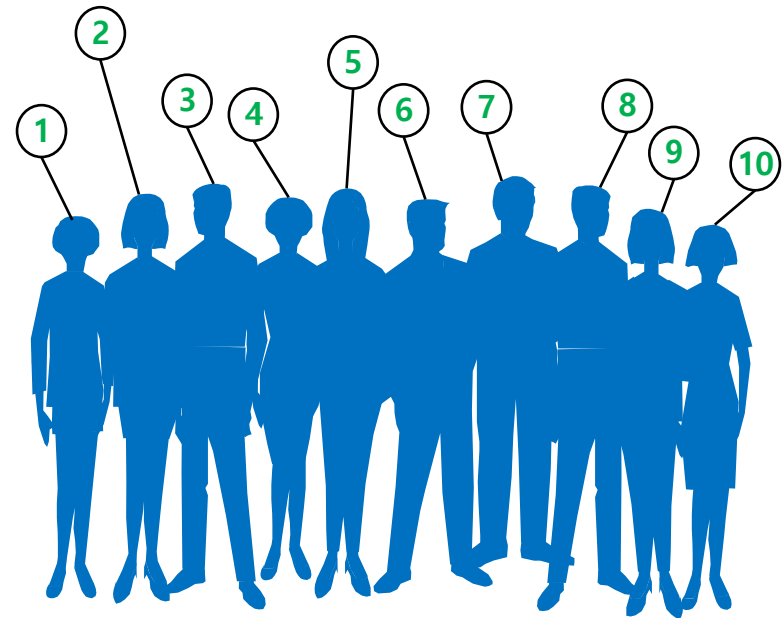


10명의 이용자들을
이해한다면?

1. 데이터 이해

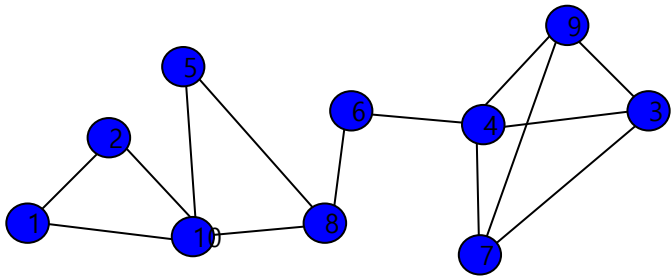
➤ 데이터의 유형

- ✓ 이름/성별/나이/거주지/직업
- ✓ 월별 요금/데이터사용량
- ✓ 휴대폰 교체주기/휴대폰 선호도
- ✓ 서로 전화하는 관계 여부
- ✓ 이용자의 VOC /서비스 선호도
- ✓ 주로 방문하는 인터넷 사이트



1. 데이터 이해

➤ 데이터의 유형



반정형데이터



비정형데이터

어제 전화가
잘 안되었어요...

인터넷이 느렸어요...

통화품질이 끊겼어요...

상담원이 불친절해요...

정형데이터

이름	성별	나이	거주지	직업	요금	데이터 사용량	휴대폰 선호도	서비스 선호도
AAA	F	20	서울	회사원	55000	3GB	LG	5
BBB	F	19	인천	자영업	45000	9GB	삼성	4
CCC	M	25	김포	회사원	35000	1GB	샤오미	3
DDD	F	42	대전	회사원	75000	4GB	LG	5
EEE	F	27	서울	자영업	65000	2GB	소니	4
FFF	M	20	서울	회사원	55000	3GB	LG	5
GGG	M	43	서울	자영업	45000	9GB	삼성	4
HHH	M	25	대전	회사원	95000	11GB	샤오미	3
III	F	42	김포	회사원	45000	3GB	LG	5
JJJ	F	27	인천	자영업	40000	4GB	소니	4

1. 데이터 이해

➤ 비정형 데이터의 예시

"450조 전세보증금, 월세화에 가계부채 '뇌관'되나"

전세의 월세화가 진행되면서 450조원에 달하는 전세보증금이 가계부채 문제의 뇌관이 될 수 있다는 지적이 나왔다. 세입자들은 월세에 부담을 느껴 주택 매입에 나서고, 집주인들은 보증금을...

다시 늘어난 아파트 미분양...부동산 열기 냉각?

뜨겁게 달아오르던 부동산 시장이 다시 냉각되는 조짐입니다. 아파트 미분양이 최근 다시 크게 늘어난 것인데요. 앞으로 부동산시장에 찬물을 끼얹을 것으로 우려됩니다. 김대도 기자가 취재...

노후주택 많은 지역 '신규분양' 주목

최근 몇 년간 신규 공급이 없었던 경기 안산·의정부·포천·오산 등지에서 하반기 신규 분양이 이뤄질 예정이어서 눈길을 끌고 있다. 공급 가뭄 지역은 기존 주택의 노후화로 새 아파트로 갈아타...

"가계부채 관리-부동산 경기부양" 두마리 토끼 잡을 수 있을까?

정부가 작년 8월부터 시행중인 주택담보대출비율(LTV)과 총부채상환비율(DTI) 규제 완화 조치를 내년까지 1년 연장키로 확정했다. 이미 행정예고 등의 절차를 거쳐 기정사실화된 사안이지만 ...

2. 주요 데이터 구조의 처리

➤ 데이터 구조?

- 데이터를 일정한 기준에 의해 모아 놓은 것 (collection of data elements, structured in some way)
- Built-in 구조 및 확장구조
 - Sequence: mapping by element position (위치값인 index 활용)
 - Indexing / Slicing / Adding / Multiplying / Membership 확인
 - Mapping : mapping by element name (key활용)

➤ List

- 여러 같은 type의 값을 하나의 객체로 모음
- 관련 함수: methods (object에 관련된 함수): len, max, min
- object.method(arguments) 방식으로 사용: lst.append(값), lst.count(값), lst.insert(위치, 값), lst.pop(), lst.reverse(), lst.remove(값), lst.sort()

2. 주요 데이터 구조의 처리

➤ List

- 숫자형 값 또는 문자열 등 여러 값의 모임
 - indexing: Sequence 내의 element에 대해 순서 별 index로 접근
 - slicing: 일정 범위 내의 element를 access하는 것

[1,2,3,4]

[1, "two", 3, 4.0, ["a","b"],(5,6)]

- 값의 추가: 같은 type의 값을 추가
- element의 포함여부: in 연산자

permission = 'rw'

'w' in permission

users = ['hky','foo','bar']

input('Enter your name: ') in users

2. 주요 데이터 구조의 처리

➤ Set

- list와 같은 값의 모임
- 값의 순서가 없음, 집합으로 이해
- {}로 생성
- 예시

`a={1,2,3}`

`b={3,4,5}`

`type(a)`

`a.union(b)`

`a.intersection(b)`

`a-b`

`a|b`

`a&b`

2. 주요 데이터 구조의 처리

➤ Tuple

- 다른 형태의 sequence, list와 유사, 0 index에서 시작하는 값
- (, ,)로 표현
- Immutable sequence: 일단 생성 후에는 값을 변경할 수 없음. 문자열과 유사
- sort, append, reverse 등의 method는 사용 불가
- 예:

```
x = (3, 2, 1)
```

```
x[1]=5      #error
```

➤ 참고:

```
x = [9, 8, 7]          #list의 생성
```


2. 주요 데이터 구조의 처리

➤ Tuple

➤ 효율적인 자료 구조

➤ 생성 후 값의 변경이 불가하여, 메모리의 사용 및 속도 등에서 list보다 효율적

➤ 임시 변수 등을 생성 시에 유리

➤ dictionary 자료구조에서의 key, value의 반환 시 tuple 구조 사용

➤ Map에서의 key (list는 key가 될 수 없음)

➤ 일부 내장함수에서의 반환값 형식

```
d = dict()
```

```
d['csev'] = 2
```

```
d['cwen'] = 4
```

```
tups = d.items()
```

```
print tups
```

결과: [('csev', 2), ('cwen', 4)]

2. 주요 데이터 구조의 처리

➤ Tuple

➤ 크기 비교

➤ 튜플 간 크기 비교 시, 튜플 내 각 값을 비교, 첫번째 비교값이 같으면 다음 값을 비교

`(0, 1, 2) < (5, 1, 2)`

`(0, 1, 2000000) < (0, 3, 4)`

`('Jones', 'Sally') < ('Jones', 'Sam')`

`('Jones', 'Sally') > ('Adams', 'Sam')`

➤ tuple 내의 값은 sort가 안되지만, tuple을 값으로 갖는 list는 sort가 가능

```
d = {'a':10, 'b':1, 'c':22}
```

```
t = d.items() #[('a', 10), ('c', 22), ('b', 1)]
```

```
t.sort() #[('a', 10), ('b', 1), ('c', 22)]
```

2. 주요 데이터 구조의 처리

➤ Set, List, Tuple?

➤ 리스트, 세트, 튜플 사용 및 변환 (세트<리스트<튜플)

```
a = set( (1,2,3) )
```

```
type(a)
```

```
b=list(a)
```

```
b
```

```
type(b)
```

```
c=tuple(b)
```

```
c
```

```
type(c)
```

```
d = set(c)
```

```
d
```

```
type(d)
```

2. 주요 데이터 구조의 처리

➤ Dictionary

- 각 값마다 이름이 있는 값의 모음(A "bag" of values, each with its own label)
- 파이썬의 유용한 기능으로, key-value 구조를 지원
- Dictionary는 { }를 이용하여 생성 / 값으로는 key:value 쌍을 , 로 나열
- 빈 dictionary는 {}를 통해 생성
- 유사한 자료 구조가 다른 언어에서는 다른 이름
 - Associative Arrays - Perl / Php
 - Properties or Map or HashMap - Java
 - Property Bag - C# / .Net

2. 주요 데이터 구조의 처리

➤ Dictionary

➤ 예시

```
purse = dict()
purse['money'] = 12
purse['candy'] = 3
purse['tissues'] = 75
print purse #{'money': 12, 'tissues': 75, 'candy': 3}
print purse['candy']    #3
purse['candy'] = purse['candy'] + 2
print purse #{'money': 12, 'tissues': 75, 'candy': 5}
```

2. 주요 데이터 구조의 처리

➤ Dictionary

- Lists에서는 값을 위치에 따라 index하는 반면, Dictionaries에서는 위치에 따른 index가 없음
- 특정 이름의 값을 lookup tag로 출력

➤ List

```
lst = list()
lst.append(21)
lst.append(183)
print lst[21, 183]
lst[0] = 23
print lst[23, 183]
```

➤ Dictionary

```
ddd = dict()
ddd['age'] = 21
ddd['course'] = 182
print ddd    #{'course': 182, 'age': 21}
ddd['age'] = 23
print ddd    #{'course': 182, 'age': 23}
```

2. 주요 데이터 구조의 처리

➤ Dictionary

➤ 값의 수정

```
ccc = dict()
ccc['csev'] = 1
ccc['cwen'] = 1
print ccc # {'csev': 1, 'cwen': 1}
ccc['cwen'] = ccc['cwen'] + 1
print ccc #{'csev': 1, 'cwen': 2}
print 'csevv' in ccc    #False
```

2. 주요 데이터 구조의 처리

➤ bool

➤ TRUE, FALSE값을 나타내는 type

isRight=False

type(isRight)

1<2

1!=2

1==2

True & False

False | False

not True

bool(0)

bool(1)

bool('test')

bool(None)

➤ 비교연산자

==

<, >

<=, >=

!=

is와 is not

in과 not in

➤ ==와 is 의 구분 : identity vs. equality

x = y = [1,2,3]

z = [1,2,3]

x==y #True

x==z #True

x is y #True

x is z #False

2. 주요 데이터 구조의 처리

- shallow & deep copy

- 얕은 복사

- 1객체 생성 후 1의 주소가 a에 저장

a=1

- 변수를 복사하면, 같은 객체를 다른 변수가 공유

a=[1,2,3]

b=a

a[0]=38

a

b

id(a)

id(b)

2. 주요 데이터 구조의 처리

- **shallow & deep copy**

- **deep copy**

- list는 슬라이싱을 이용

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

```
b = a[:]
```

```
a[0]=38
```

```
id(a)
```

```
id(b)
```

- list 외의 자료는 deepcopy를 이용

```
import copy
```

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

```
b = copy.deepcopy(a)
```

```
a[0] = 38
```

2. 주요 데이터 구조의 처리

➤ 조건문과 반복문

➤ 조건문: 조건의 참/거짓여부에 따라 문장을 각각 실행, if else, if elif else를 사용

➤ 반복문: 파라미터 값을 변경하면서 동일한 작업을 여러 번 실행, for 사용

➤ 예시

```
counts = dict()
```

```
names = ['csev', 'cwen', 'csev', 'zqian', 'cwen']
```

```
for name in names :
```

```
    if name not in counts:
```

```
        counts[name] = 1
```

```
    else :
```

```
        counts[name] = counts[name] + 1
```

2. 주요 데이터 구조의 처리

➤ 조건문과 반복문

- 조건문-bool을 사용하여 조건 충족 여부 판단
- block은 { }로 표현하지 않고 indentation으로 표현(참고: 한 레벨당 공백 4개)

```
var = 100
```

```
if var < 200:
```

```
    print("Expression value is less than 200")
```

```
    if var == 150:
```

```
        print("Which is 150")
```

```
    elif var == 50:
```

```
        print("Which is 50")
```

```
    elif var < 50:
```

```
        print("Expression value is less than 50")
```

```
    else:
```

```
        print("Could not find true expression")
```

if에 대한 block

하위 block

2. 주요 데이터 구조의 처리

➤ 조건문과 반복문

- 반복문-while과 for를 사용
- for: set/sequence(또는 iterable object(iterate가능 객체)의 각 값에 대해 Code block 수행
- ***block은 { }로 표현하지 않고 indentation으로 표현(참고: 한 레벨당 공백 4개)***

```
x =1
```

```
while x <=100:
```

```
    print(x)
```

```
    x+=1
```

```
words = ['this','is','a','wonder']
```

```
for word in words:
```

```
    print(word)
```

2. 주요 데이터 구조의 처리

➤ 조건문과 반복문의 들여쓰기

- 같은 level의 statement들은 같은 들여쓰기(indentation)값을 갖아야 함

```
if a==1:
```

```
    print(1)
```

```
    print(0)
```

X

```
if a==1:
```

```
    print(1)
```

```
    print(0)
```

O

2. 주요 데이터 구조의 처리

➤ 함수 정의

- def 라는 예약어와 함께 정의
- 함수의 사용은 함수명(입력값)의 형태

```
def Times(a,b):  
    return a*b
```

- 함수 객체 확인: globals()
- return 없으면 None 반환

```
def setValue( newValue):  
    x=newValue  
  
retval = setValue(10)  
print(retval)
```

III. 데이터 구조의 처리

1. 모듈 활용 데이터 표현 및 처리
2. NumPy 활용 데이터 처리
3. Pandas 활용 데이터 처리
4. 파일/DB 입출력

1. 모듈 활용 데이터 표현 및 처리

➤ Python 모듈의 활용

➤ 모듈: 확장기능을 담은 파일

➤ Numpy, scipy, sklearn, statmodels, mlxtend, ...

➤ 예: math 모듈, 수학 관련 확장 기능

```
import math
```

```
math.floor(32.9)
```

```
from math import sqrt
```

```
sqrt(-1)
```

```
import cmath
```

```
cmath.sqrt(-1)
```

1. 모듈 활용 데이터 표현 및 처리

- Built-in 자료구조의 한계

$[1,2,3] * 3$?

- 모듈을 통한 데이터 표현 및 처리가 중요!

numpy

pandas

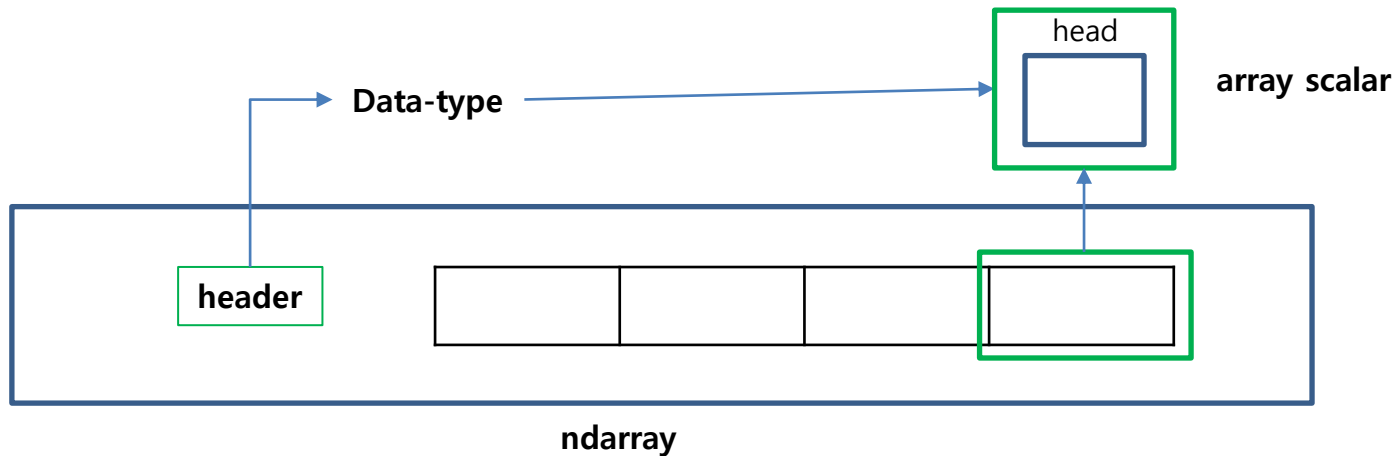
...

2. NumPy 활용 데이터 처리

- 데이터 처리를 위한 주요 패키지
 - numpy
 - pandas
- NumPy
 - 수치 데이터 처리 기능을 확장
 - Numeric과 Numarray 를 대체
 - 주요 기능:
 - fast built-in object (ndarray)!
 - large, multi-dimensional arrays and matrices
 - high-level 수학 함수 지원

2. NumPy 활용 데이터 처리

- ndarray
 - Matlab 또는 R과 같이 자료를 다루는 기능 제공
 - 동일한 data type을 갖는 다차원 array
 - N-dimensional array of rectangular data
 - Fast algorithms on machine data-types (int, float, etc.)



2. NumPy 활용 데이터 처리

- Array 생성
 - array 함수 사용
 - asarray: ndarray로 변환
 - arange: ndarray를 만드는 range 기능(정해진 구간 내 일정한 간격 값 생성)
 - ones, ones_like: 1로 채워진 ndarray
 - zeros, zeros_like: 0으로 채워진 ndarray

2. NumPy 활용 데이터 처리

➤ array 생성

```
>>> a = array([0,1,2,3])
>>> a
array([0, 1, 2, 3])
```

➤ type 확인

```
>>> type(a)
<type 'array'>
```

➤ NUMERIC 'TYPE' OF ELEMENTS

```
>>> a.dtype
dtype('int32')
```

➤ BYTES PER ELEMENT

```
>>> a.itemsize # 원소별 크기
4
```

➤ array 모양(shape)->차원 확인

```
# tuple로 반환
# array의 차원별 길이
>>> a.shape
(4,)
>>> shape(a)
(4,)
```

➤ 크기 확인

```
# array의 원소 개수
>>> a.size
4
>>> size(a)
4
```

2. NumPy 활용 데이터 처리

➤ 사용한 메모리 확인

```
# array가 차지하는 바이트 정보
>>> a.nbytes
12
```

➤ 차원 출력

```
>>> a.ndim
1
```

➤ array 복사

```
# array값의 복사
>>> b = a.copy()
>>> b
array([0, 1, 2, 3])
```

➤ LIST로 변환

```
# array를 list로 변환
>>> a.tolist()
[0, 1, 2, 3]
```

```
# 위와 동일하지만, list함수가 더 느림
>>> list(a)
[0, 1, 2, 3]
```

2. NumPy 활용 데이터 처리

➤ INDEXING

```
>>> a[0]
0
>>> a[0] = 10
>>> a
[10, 1, 2, 3]
```

➤ 값 채우기

array내 값 채우기

```
>>> a.fill(0)
>>> a
[0, 0, 0, 0]
```

위와 동일하지만 속도가 느림

```
>>> a[:] = 1
>>> a
[1, 1, 1, 1]
```

➤ 형 변환 시 유의

```
>>> a.dtype
dtype('int32')
```

실수를 정수형값에 할당 시 정수로 변환

```
>>> a[0] = 10.6
>>> a
[10, 1, 2, 3]
```

fill 역시 위와 동일하게 작동

```
>>> a.fill(-4.8)
>>> a
[-4, -4, -4, -4]
```


2. NumPy 활용 데이터 처리

➤ 다차원 array

```
>>> a = array([[ 0, 1, 2, 3],
               [10,11,12,13]])
```

```
>>> a
array([[ 0, 1, 2, 3],
       [10,11,12,13]])
```

➤ (ROWS,COLUMNS)

```
>>> a.shape
```

```
(2, 4)
```

```
>>> shape(a)
```

```
(2, 4)
```

➤ ELEMENT COUNT

```
>>> a.size
```

```
8
```

```
>>> size(a)
```

```
8
```

➤ NUMBER OF DIMENSIONS

```
>>> a.ndim
```

```
2
```

➤ GET/SET ELEMENTS

```
>>> a[1,3]
```

```
13
```

```
>>> a[1,3] = -1
```

```
>>> a
array([[ 0, 1, 2, 3],
       [10,11,12,-1]])
```

column
row

➤ index 사용하여 첫 행 출력

```
>>> a[1]
```

```
array([10, 11, 12, -1])
```

2. NumPy 활용 데이터 처리

➤ Data-types

- "type"에 대한 2개의 개념
 - The data-type object (dtype in NumPy): 예) int32 등 값에 대한 type
 - The Python "type" of the object
- dtype object:
 - 객체에 대한 상세한 정보 제공
 - instance of a single dtype class
- "type" of the extracted elements
 - 원래 파이썬 클래스
 - 파이썬 클래스 계층구조에 존재

2. NumPy 활용 데이터 처리

➤ NumPy dtypes 개요

Basic Type	Available NumPy types	Comments
Boolean	bool	Elements are 1 byte in size
Integer	int8, int16, int32, int64, int128, int	int defaults to the size of int in C for the platform
Unsigned Integer	uint8, uint16, uint32, uint64, uint128, uint	uint defaults to the size of unsigned int in C for the platform
Float	float32, float64, float, longfloat,	Float is always a double precision floating point value (64 bits). longfloat represents large precision floats. Its size is platform dependent.
Complex	complex64, complex128, complex	The real and complex elements of a complex64 are each represented by a single precision (32 bit) value for a total size of 64 bits.
Strings	str, unicode	Unicode is always UTF32 (UCS4)
Object	object	Represent items in array as Python objects.
Records	void	Used for arbitrary data structures in record arrays.

2. NumPy 활용 데이터 처리

➤ 단순 계산

```
>>> a = array([1,2,3,4])
>>> b = array([2,3,4,5])
>>> a + b
array([3, 5, 7, 9])
```

pi와 e를 다음처럼 미리 정의

```
pi = 3.14159265359
e = 2.71828182846
```

➤ math 함수 활용

```
# array from 0 to 10
>>> x = arange(11.)

# array와 scalar 곱하기
>>> a = (2*pi)/10.
>>> a
0.62831853071795862
>>> a*x
array([ 0., 0.628, ..., 6.283])

# x=x*a
>>> x *= a
>>> x
array([ 0., 0.628, ..., 6.283])

# array에 함수 적용
>>> y = sin(x)
```

2. NumPy 활용 데이터 처리

➤ 합계 함수

```
>>> a = array([[1,2,3],  
               [4,5,6]], float)
```

```
# array내 값 합하기
```

```
>>> sum(a)  
21.
```

```
# axis=0이면 열 방향 계산
```

```
>>> sum(a, axis=0)  
Array([5., 7., 9.])
```

```
# axis=1이면 행 방향 계산
```

```
>>> sum(a, axis=1)  
array([6., 15.])
```

➤ 합계 method

```
# sum(a)와 동일
```

```
>>> a.sum()  
21.
```

```
# axis로 sum의 방향 지정
```

```
>>> a.sum(axis=0)  
array([5., 7., 9.])
```

➤ 곱셈

```
# 열 방향 곱셈
```

```
>>> a.prod(axis=0)  
array([ 4., 10., 18.])
```

```
# 함수 형태로 사용
```

```
>>> prod(a, axis=0)  
array([ 4., 10., 18.])
```

2. NumPy 활용 데이터 처리

➤ MIN

```
>>> a = array([2.,3.,0.,1.])
>>> a.min(axis=0)
0.
# min은 파이썬 빌트인 method
# amin이 더 빠름(다차원 array)
>>> amin(a, axis=0)
0.
```

➤ ARGMIN

```
# 최소값의 index (순서) 구하기
>>> a.argmin(axis=0)
2
# 함수형태로 사용
>>> argmin(a, axis=0)
2
```

➤ MAX

```
>>> a = array([2.,1.,0.,3.])
>>> a.max(axis=0)
3.
```

```
# functional form
>>> amax(a, axis=0)
3.
```

➤ ARGMAX

```
# 최대값의 index.
>>> a.argmax(axis=0)
1
# functional form
>>> argmax(a, axis=0)
1
```

2. NumPy 활용 데이터 처리

➤ 평균

```
>>> a = array([[1,2,3],  
               [4,5,6]], float)
```

열방향 평균

```
>>> a.mean(axis=0)  
array([ 2.5,  3.5,  4.5])  
>>> mean(a, axis=0)  
array([ 2.5,  3.5,  4.5])  
>>> average(a, axis=0)  
array([ 2.5,  3.5,  4.5])
```

가중 평균

```
>>> average(a, weights=[1,2],  
            axis=0)  
array([ 3.,  4.,  5.])
```

➤ 표준편차와 분산

Standard Deviation

```
>>> a.std(axis=0)  
array([ 1.5,  1.5,  1.5])
```

Variance

```
>>> a.var(axis=0)  
array([2.25, 2.25, 2.25])  
>>> var(a, axis=0)  
array([2.25, 2.25, 2.25])
```

2. NumPy 활용 데이터 처리

➤ Comparison 연산자

`equal` `(==)`
`greater_equal` `(>=)`
`logical_and`
`logical_not`

`not_equal` `(!=)`
`less` `(<)`
`logical_or`

`greater` `(>)`
`less_equal` `(<=)`
`logical_xor`

➤ 2D EXAMPLE

```
>>> a = array(((1,2,3,4),(2,3,4,5)))
>>> b = array(((1,2,5,4),(1,3,4,5)))
>>> a == b
array([[True, True, False, True],
       [False, True, True, True]])
# functional equivalent
>>> equal(a,b)
array([[True, True, False, True],
       [False, True, True, True]])
```


3. Pandas 활용 데이터 처리

- 데이터 처리를 위한 주요 패키지
 - numpy
 - **pandas**
- pandas
 - Data munging/preparation/cleaning /integration
 - Rich data manipulation tool (Numpy 이용)
 - Fast, intuitive data structures
 - R의 data.frame과 유사
 - Easy-to-use, highly consistent API

3. Pandas 활용 데이터 처리

➤ 상세 기능

- DataFrame object – Integrated indexing을 이용한 데이터 분석
- 여러 포맷 지원 (CSV, text files, Excel, SQL databases, HDF5)
- data alignment 및 결측 데이터를 위한 통합 기능
- 데이터셋의 reshaping 및 pivoting
- 대규모 dataset 용 label-based slicing, indexing, subsetting
- 데이터 Aggregating/ transforming data (group by 엔진)
 - split-apply-combine operations on data sets;
- Hierarchical axis indexing
- Time series 기능


3. Pandas 활용 데이터 처리

- 데이터 구조
 - Data structures
 - Series (1D)
 - DataFrame (2D)
 - Panel (3D)
 - NA-friendly statistics
 - Index implementations/label-indexing
 - GroupBy engine
 - Time series tools
 - Data range generation
 - Extensible data offsets
 - Hierarchical indexing stuff

3. Pandas 활용 데이터 처리

➤ Series:


- 1D label
- numpy array
- Subclass of numpy.ndarray
- Data: any dtype
- Index labels need not be ordered
- Duplicates are possible (but result in reduced functionality)



index	value
A	5
B	6
C	12
D	-5

➤ DataFrame

- 2D table with rows and column labels
- [potentially heterogeneous columns](#)
- ndarray-like, but not ndarray
- column 별로 서로 다른 dtype을 가질 수 있음
- Row and column index
- Size mutable: insert and delete columns



index	Columns		
	score	id	test
A	5	ID1	2.1
B	6	ID2	3.2
C	12	ID3	NA
D	-5	ID4	-1.2

3. Pandas 활용 데이터 처리

➤ Index

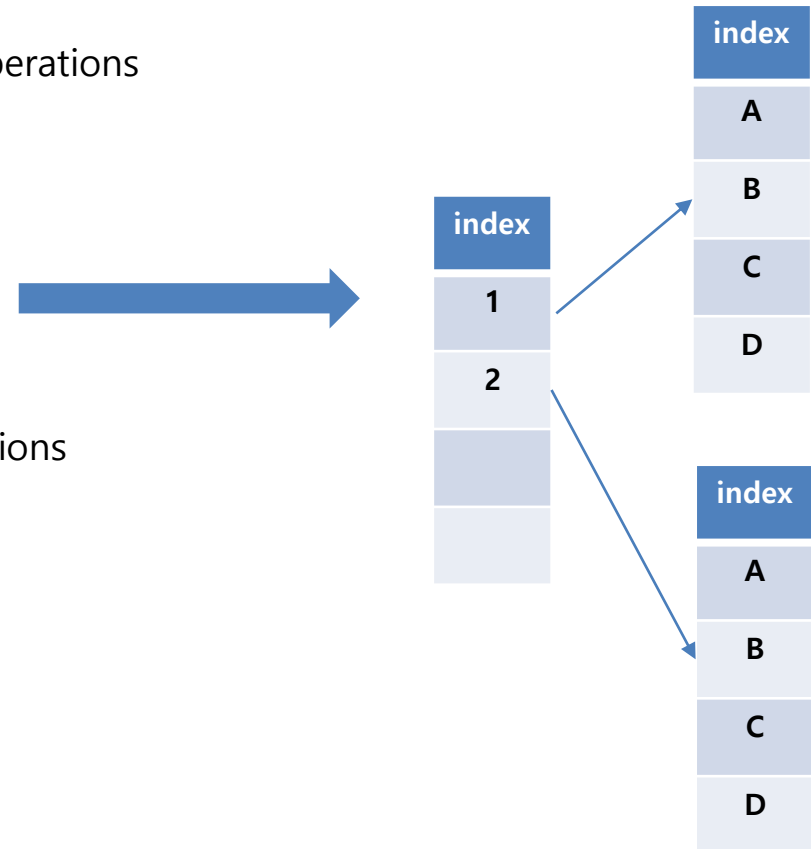
- Every axis has an index
- 신속한 lookup과 Data alignment and join operations

➤ Hierarchical indexes

- Semantics: a tuple at each tick
- Enables easy group selection
- Terminology: "multiple levels"
- Natural part of GroupBy and reshape operations

➤ Data Alignment

- Binary operations are joins!
- "Outer join by default"
- Data Alignment
- DataFrame joins/aligns on both axes
- Irregularly-indexed data



3. Pandas 활용 데이터 처리

➤ Data Alignment

- Binary operations are joins!
- ["Outer join by default"](#)
- DataFrame joins/aligns on both axes

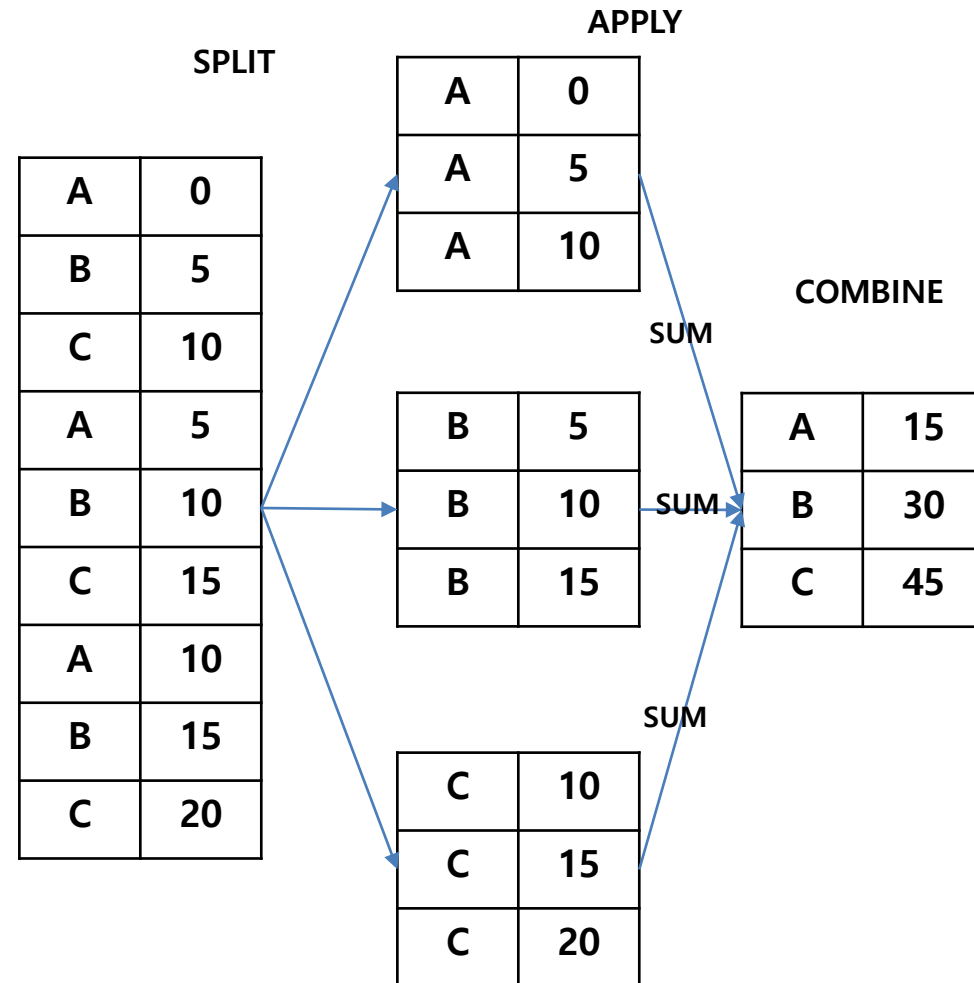
A	0	+	A	0	=	A	0
B	1		B	1		B	2
C	2		C	2		C	4
D	3		D	3		D	6

B	1	+	A	0	=	A	NA
C	2		B	1		B	2
D	3		C	2		C	4
E	4		D	3		D	6
						E	NA

3. Pandas 활용 데이터 처리

➤ Group By

- Splitting axis into groups
 - DataFrame columns
 - Arrays of labels
 - Functions, applied to axis labels
- grouped data 작업방식은 다양함
 - Iterate: "for key group, in grouped"
 - Aggregate: grouped.agg(f)
 - Transform: grouped.transform(f)
 - Apply: grouped.apply(f)



4. 파일/DB 입출력

➤ Data 가져오기

➤ 테이블 형태 텍스트 파일(Comma Separated Value 파일)

- 일반적으로 데이터는 DBMS로 부터 불러오거나, 웹으로 수집하거나, 주어진 데이터 파일을 통해 얻을 수 있음
- 주어진 데이터 파일은 여러 형식을 가질 수 있으며(엑셀, SAS파일, 텍스트파일 등) 그 중에서도 특정 소프트웨어에 종속되지 않는 텍스트 파일을 많이 사용
- 텍스트 파일 안에는 표 형태의 자료가 주어짐
 - 각 열의 값은 구분자(Separator)로 구분되어야 R에서 데이터를 읽으며 값을 구분해서 인식할 수 있음. 일반적으로는 comma나 Tab이 많이 사용되며, 경우에 따라서는 고정된 넓이로 구분할 수 있음
 - 각 행의 값은 줄바꿈을 나타내는 개행 문자로 구분되어서 인식됨(Wn)

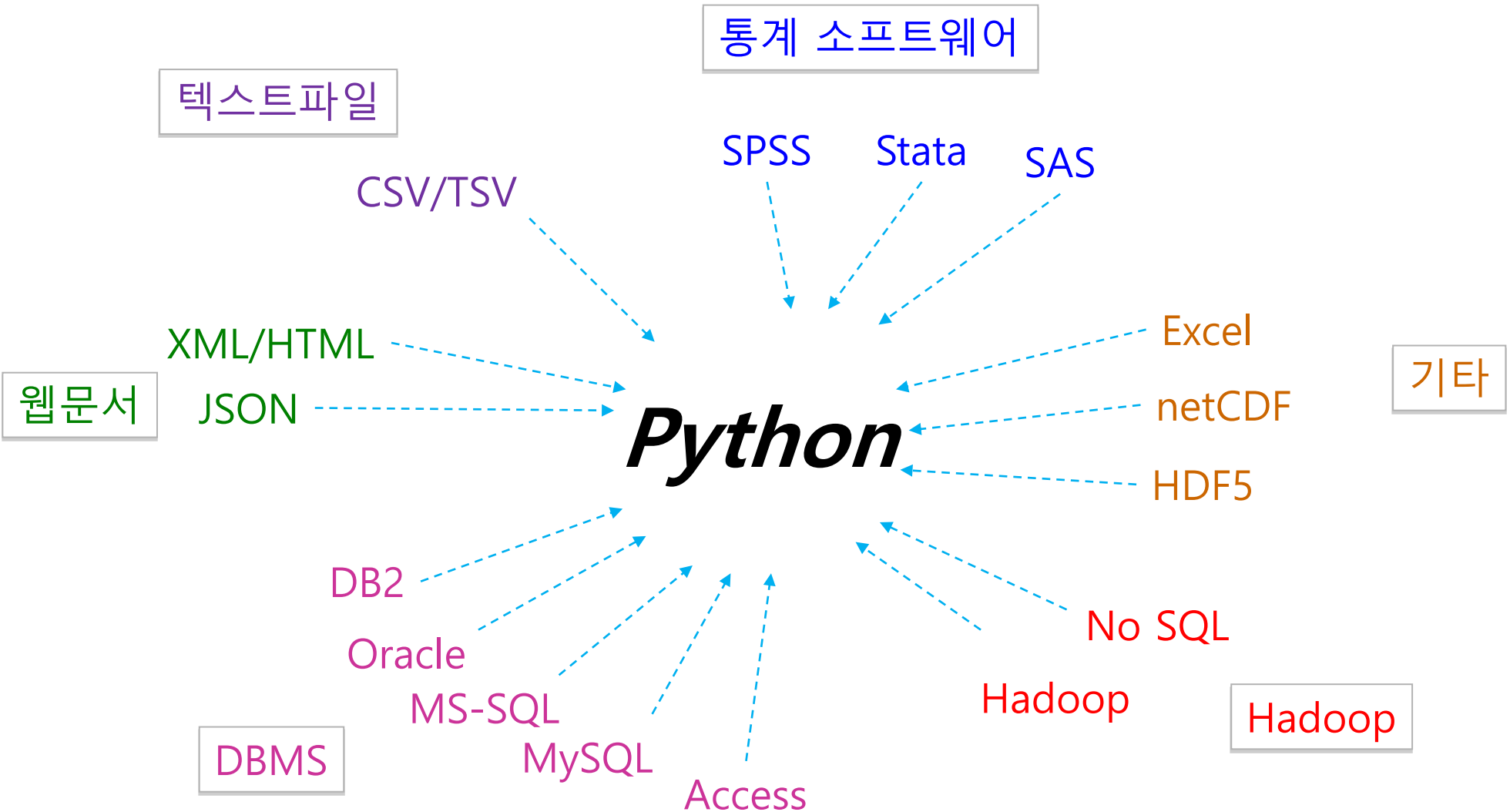
➤ 예시:

Year	Make	Model	Description	Price
1997	Ford	E350	ac, abs, moon	3000.00
1999	Chevy	Venture "Extended Edition"		4900.00
1999	Chevy	Venture "Extended Edition, Very Large"		5000.00
1996	Jeep	Grand Cherokee	MUST SELL! air, moon roof, loaded	4799.00



```
Year,Make,Model,Description,Price
1997,Ford,E350,"ac, abs, moon",3000.00
1999,Chevy,"Venture ""Extended Edition""",,,4900.00
1999,Chevy,"Venture ""Extended Edition, Very Large""",,5000.00
1996,Jeep,Grand Cherokee,"MUST SELL!
air, moon roof, loaded",4799.00
```


4. 파일/DB 입출력



Q&A

