



# Large Scale Data 분석을 위한 Python Programming

---

August 23, 2019

# Large Scale Data

- 규모가 큰 데이터
  - 사례 수가 많은 긴 데이터(**tall data**)
  - **feature**의 개수가 많은 넓은 데이터(**wide data**)
  - 길고 넓은 데이터
  - 희소 행렬 데이터
- 대용량 데이터 분석의 장애요인
  - 대규모 연산에 따른 분석 시간의 지연
  - 데이터의 **I/O**로 인한 시간의 지연
  - 컴퓨터 메모리의 제한으로 한번에 처리 가능한 데이터 량의 제한
- 해결 방안
  - 소프트웨어 또는 더 많은 메모리, 더 빠른 **CPU**, 더 빠른 **storage disk**, **GUP** 등 하드웨어를 교체(**scale up**)
  - 다수의 머신으로 연산을 분산(**scale out**)
  - 최적의 스케일 업 기법과 스케일 아웃 기법을 함께 사용

# 확장성 문제

- 확장성
  - 매일 증가하는 데이터로부터 학습
  - 새로운 데이터를 반영하기 위한 머신러닝 모델의 갱신
- 확장성 문제의 해결방안
  - 차원 축소를 통한 스케일 업 (feature selection etc.)
  - 효율적인 알고리즘을 통한 스케일 업
  - 멀티프로세싱, 효율적 벡터화를 통한 병렬처리 등의 스케일 업
  - 다수의 머신을 활용한 스케일 아웃
- 스케일 업과 스케일 아웃을 위한 효율적인 프로그래밍 언어가 필수적
- Python could be an answer
  - 뛰어난 메모리 관리 기능
  - 머신 러닝을 위한 수준 높은 패키지 시스템이 제공
  - 데이터 사이언티스트들이 선호

# Big data: New era of analytic



- zeta byte 데이터
- Garbage in garbage out
- Big 데이터 분석을 위한 새로운 paradigm이 요구

# Classical data analysis is inadequate for Big data

- Data 분석 방법
  1. Top-down 방식: Confirmative data analysis
  2. Bottom-up 방식: Exploratory data analysis
- 빅데이터 분석: 가설검정과 같은 CDA 방법의 적용은 적합치 않은 경우가 많음.
- Why?  $n$ 이 크면, 귀무 가설은 거의 항상 기각
- $X_1, \dots, X_n \stackrel{iid}{\sim} N(\theta, 1)$ , and test  $H_0 : \theta = 0$  vs.  $H_1 : \theta \neq 1$ .
- At 5% significance level, we reject  $H_0$  if  $\sqrt{n}|\bar{X}| > 1.96$ .
- Suppose  $\theta = 10^{-10}$  (a meaningless difference from zero).
- Note that  $\sqrt{n}(\bar{X} - 10^{-10}) \sim N(0, 1)$ .

$$\begin{aligned}
\Pr[\sqrt{n}|\bar{X}| > 1.96] &= \Pr[Z > 1.96 - \sqrt{n}10^{-10} \text{ or} \\
&\quad Z < -1.96 - \sqrt{n}10^{-10}] \\
&= \Pr[Z > 1.96 - 10^2] + \Pr[Z < -1.96 - 10^2] \\
&\approx 1 + 0 \text{ if } n = 10^{24}
\end{aligned}$$

- 평균이 귀무가설에 주어진 값과 매우 가깝기 때문에 실제로 귀무가설이 맞는다고 하여도 유의수준에 관계없이 기각

# A historical example

- Kepler의 행성운동 1 법칙: 행성의 궤도는 타원
  - Kepler 시대의 데이터는 1 법칙을 잘 설명
  - 현재의 관찰 데이터를 이용하여 Kepler의 1 법칙을 가설 검정하면 행성들 간의 상호작용에 의한 섭동 때문에 가설이 기각
  - 섭동에 의한 오차는 크지 않으므로 케플러의 법칙은 행성운동을 잘 설명
- The concern here is that an essentially correct model can be rejected by too accurate data if statistical significance tests are blindly applied without regard to the actual size of discrepancies.
- Big data 분석에서는 분석의 새로운 paradigm이 요구됨.
  - 탐색적 데이터 분석(Exploratory Data Analysis, EDA)
  - 기계학습(machine learning), 데이터마이닝

# 기계학습

- 통계학습(statistical learning) 또는 예측분석(predictive analytics)
- 세 개 학문 영역의 공통분모
  - 통계학(statistics)
  - 인공지능(Artificial Intelligence; AI)
  - 전산과학
- 특성 (compared with statistical methodology)
  - 직관적이고 간단, but no sound theoretical basis
  - 컴퓨터 연산에 크게 의존
  - 데이터 중심적(data-driven)
  - 큰 데이터를 요구



# Statisticians versus Data scientists

- “Data science”이라는 용어가 등장한 이래, 전통적인 통계학자와 데이터 과학자를 정의하고 구별하기 위한 논쟁이 있었다.
- A personal opinion:
  - “A data scientist is someone who is better at statistics than any software engineer and better at software engineering than any statistician.”
- 데이터 사이언스:
  - 통계학과 전산학에 대한 지식을 바탕으로 데이터를 분석하는 학문 영역
- 어느 분야에 더 가까울까? 또는 어느 분야에서 접근하는 것이 좋을까?
- My opinion
  - 데이터 분석에 대한 이해가 필요
  - 데이터 분석이라는 관점에서 통계학에 근접
  - 전산학은 분석을 위한 수단
- 두 분야에 대한 지식은 반드시 필요

# Data scientist를 위한 8가지 skills

- **Statistics:** 최소한 기본적인 통계학의 이해는 반드시 필요
- **Machine Learning:** 기계학습의 방법론을 이해하고, 문제에 따라 적절한 방법을 선택할 수 있는 것이 중요
- **Software Engineering:**
- **Data Munging:** 불안정한 데이터를 다룰 수 있는 것이 매우 중요
  - 분석 시간의 60% - 80% 이상을 데이터 클리닝 (data cleaning), 데이터 구성 (data organization) 등 업무에 할애
- **Data Visualization**
- **Multivariable Calculus and Linear Algebra**
- **Thinking Like A Data Scientist:**
- **Communication**

# Data scientist가 갖추어야 할 업무 역량 5가지

- 데이터 분석 및 양적추론 능력
- 데이터에 대한 스토리텔링
  - 데이터의 수집, 분석 및 예측 결과에 대해 스토리텔링 (story telling) 을 하듯이 설명할 수 있어야 한다.
- 팀워크를 살릴 수 있는 요원
  - 데이터 사이언티스트의 업무는 주로 팀 워크를 통해 수행되고 팀 플레이를 통해 새로운 과제를 발굴, 새로운 영역을 개척.
- 창의력
  - 데이터 사이언티스트는 데이터에 대한 탐정이 되어야 한다.
- 호기심이 가득한 사람
  - 호기심으로 시작돼 데이터를 분석하게 되고 또한 자신의 직관력, 데이터 수집력, 표준화, 통계, 모델링, 비주얼라이제이션, 커뮤니케이션 능력을 발휘할 수 있다

# Data Scientist(<http://flip.it/10-7cY>)

- 데이터 사이언티스트, ‘인기 폭발’ (이강봉/2017년 8월 17일)
  - 빅데이터 시대를 맞아 ‘데이터 사이언티스트(Data Scientist)’의 인기가 하늘로 치솟고 있고, 거의 모든 산업에 걸쳐 많은 기업들이 엄청난 보수를 제시하며 숙련된 전문가를 찾고 있는 중
  - 많은 기업들이 데이터 전문가를 찾고 있지만 유능한 전문가를 찾기는 어려운 상황
  - IBM 연구에 따르면 오는 2020년까지 미국에서 필요로 하게 될 데이터 사이언티스트의 수는 272만 명에 이른다.
  - 현재 충원되고 있는 데이터 사이언티스트의 수는 36만4000명에 불과
  - 데이터 사이언티스트 수를 보충하려면 매년 약 70만 명의 데이터 전문가를 양성하거나 다른 나라 등에서 초빙해야 한다.
  - 많은 기업들이 자사 업무에 맞는 뛰어난 데이터 사이언티스트를 찾고 있지만 쉽지 않은 일
  - 인사 책임자들은 뛰어난 능력의 소유자가 아니라, 자사 업무에 손쉽게 적응할 수 있는 데이터 사이언티스트를 구한다.

# What is Python?

- Python: Guido van Rossum에 의한 만들어진 **script** 언어
  - Rossum이 즐겨보던 Monty Python's Flying Circus라는 코미디언 프로그램에서 이름이 유래
  - 그리스 신화에 나오는 거대한 뱀
  - **Very High Level Language**
- 인터프리트 언어
- 동적인 데이터 타입 결정 지원
- **Platform independent**
- 개발기간 단축에 유리
- 간단한 문법
- 다양한 데이터 타입
- 팀워크에 유리: 모듈단위의 코드를 작성하고 결합
- 다양한 라이브러리
- 빅데이터 분석을 위한 표준 프로그래밍 언어의 하나

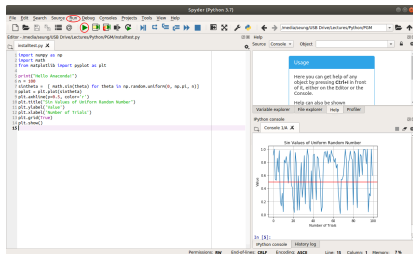
# Why Python?

- 머신러닝(machine learning)을 위한 다수의 패키지가 존재
- 빅데이터 분석을 위한 다수의 프레임워크들을 제어 가능
  - H2O 프레임워크
  - Hadoop 프레임워크
  - HDFS, MapReduce, YARN과 같은 Hadoop의 컴포넌트
- Data Scientist들이 선호

# Anaconda

- Anaconda는 python의 과학용 배포판으로 Numpy, SciPy, pandas, IPython, matplotlib, Sciit-learn, StatsModels 등, 200여개의 패키지를 포함
- <https://www.anaconda.com/distribution/> 에서 운영체제에 맞는 최신 버전의 python을 download.
- download 받은 “Anaconda...exe” file을 관리자 권한으로 실행
  - 관리자 권한이 아니면 c:\users\사용자이름\에 설치. 이름이 영문이 아니거나 빈칸이 있으면 문제가 발생할 소지가 있다.
  - 관리자 권한으로 실행하면 hidden directory인 c:\ProgramData에 설치(path)
  - Open conda 터미널
    - 시작->Anaconda3->Anaconda Powershell Prompt
  - terminal에서 conda update --all 입력 후 return
  - terminal에서 conda activate 입력 후 return

- spyder는 python의 IDE(Integrated Development Environment)의 하나로써 Anacondad와 같이 설치된다.
- 시작 --> Anaconda3 --> Spyder



- Anaconda의 설치 확인을 위해 자료실에 upload된 anacondaTest.py를 spyder에서 불러 run

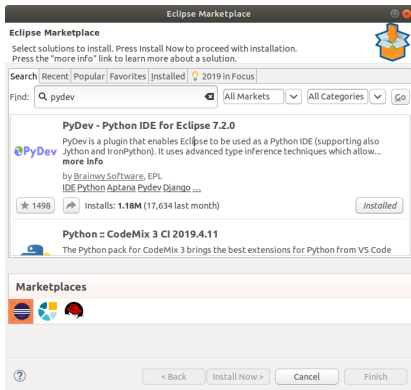


# Eclipse

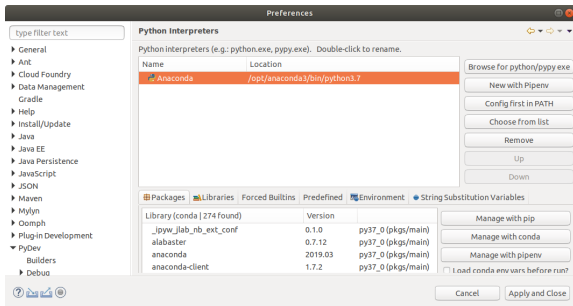
- 이클립스 재단에 의해 만들어진 **Java** 기반의 통합 개발 환경 (**Java**, **C++**, **python** 등 다양한 프로그래밍 언어의 **IDE**로 널리 사용).
- Eclipse는 **Java** 기반 IDE이므로 설치에 앞서 **Java**를 설치 또는 update(<https://www.java.com>).
- <https://www.eclipse.org/downloads>에서 운영체제에 맞는 eclipse installer를 다운 후, 실행
  - 어떤 종류의 eclipse package를 설치해도 관계없으나 여기에서는 Eclipse IDE for Java Developers를 선택



- Eclipse 실행 -> workspace 선택 -> close “welcome” window
- Help -> Eclipse Marketplace
- Eclipse Marketplace 창에서 Pydev 검색하고 설치
- Restart Eclipse



- Open perspective(Eclipse 창, 우상단의 버튼 또는 Window -> Perspective -> Open Perspective) -> Pydev 선택
- Windows -> preference -> Pydev(double click) -> Interpreters(double click) -> Python Interpreter -> Choose from list -> 설치된 Anaconda의 python3.7 선택 -> Apply and Close



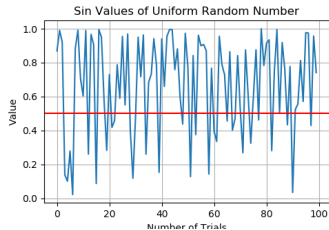
- Eclipse Pydev 설치 확인을 위해 자료실의 `anacondaTest.py`를 실행
  - File -> New -> Pydev project -> give project name (say, Test)
  - File -> New -> File -> give file name(say, installTest.py)
  - `anacondaTest.py`를 `installTest.py`에 복사
  - Run

The screenshot shows the Eclipse IDE interface. The 'PyDev Package Explorer' on the left shows a project named 'Test' with a file 'installTest.py'. The 'Outline' view on the right shows the structure of the script. The 'Console' at the bottom shows the output of the script, which includes the message 'Hello Anaconda!'.

```

1 import numpy as np
2 import math
3 from matplotlib import pyplot as plt
4
5 print("Hello Anaconda!")
6 n = 100
7 s1atheta = [ math.sin(theta) for theta in np.random.un
8 plt = plt.plot(s1atheta)
9 plt.xlabel('Value')
10 plt.title("Sin Values of Uniform Random Number")
11 plt.ylabel('Value')
12 plt.xlabel('Number of Trials')
13 plt.grid(True)
14 plt.show()
15

```



# Python

---

## Python Overview

# 식별자 (identifier)

- 식별자는 변수, 함수 클래스 등에 주어지는 이름
- 변수, 함수 및 클래스 이름을 만드는 규칙은 대체로 다른 언어와 유사
  - 언더스코아(`_`), 또는 **unicode** 문자로 시작, 두 번째 문자부터는 **unicode**, 숫자, 언더스코어. **but** 특수문자 사용불가
  - 이름크기의 제한 없음
  - **case sensitive**
- 변수명의 선택은 프로그램의 이해를 돕는 중요한 요소
  - `x = y * z`
  - `force = mass * acceleration`
- **Tips for variable name:**
  - 변수명의 첫 단어는 소문자(클래스는 대문자)로 표기하고 복합단어일 경우 각 단어의 첫 번째 문자만 대문자로 표기: **iAge, isSingle**
  - 상수 이름은 이름 전체를 대문자로 표기: **PI = 3.141592**

# Python keywords

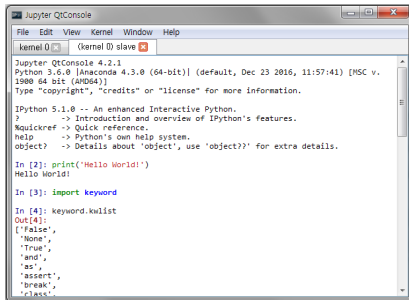
- 30개의 예약어(또는 키워드)는 python에서 사용하고 있기 때문에 변수명, 클래스명 등의 식별자(identifier)로 사용하면 안된다.

**Table 1:** python 예약어

|        |         |       |       |          |
|--------|---------|-------|-------|----------|
| and    | assert  | break | class | continue |
| def    | del     | elif  | else  | except   |
| exec   | finally | for   | from  | global   |
| if     | import  | in    | is    | lambda   |
| not    | or      | pass  | print | raise    |
| return | try     | while | with  | yield    |

# Interactive Running

- Run Jupyter QtConsole
  - 시작메뉴에서 마우스 right click, 윈도우 cmd 창을 열고  
jupyter qtconsole↔
- 또는 spyder의 console 창을 이용
- Type import keyword ↔ keyword.kwlist ↔



```
Jupyter QtConsole 4.2.1
Python 3.6.0 [Anaconda 4.3.0 (64-bit)] (default, Dec 23 2016, 11:57:41) [MSC v.
1900 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 5.1.0 -- An enhanced Interactive Python.
?                -> Introduction and overview of IPython's features.
%quickref        -> Quick reference.
help             -> Python's own help system.
object?         -> Details about 'object', use 'object??' for extra details.

In [2]: print('Hello World!')
Hello World!

In [3]: import keyword

In [4]: keyword.kwlist
Out[4]:
['False',
 'None',
 'True',
 'and',
 'as',
 'assert',
 'break',
 'class',
```



# 주석, 연속라인

- # 이후의 문자들은 주석
  - 주석은 코드의 내용을 설명하거나 참조사항을 기록
  - 프로그램의 이해를 돕는데 주석은 매우 중요
- 줄바꾸기 앞의 “\”는 다음 라인을 현재라인과 연결
  - 프로그램의 가독성을 위해 줄바꾸기가 필요

```
>>> if (a == 1) and                                     # 여기서 ↩ 은 에러 발생
```

```
SyntaxError: invalid syntax
```

```
>>> if (a == 1) and \  
    (b == 3):  
    print 'True'
```

# 치환

- '='은 우변의 객체 또는 값을 좌변의 할당('='는 비교연산자)

```
>>> a = 12
```

```
>>> a = a + 1
```

```
>>> 4 == 5
```

```
False
```

```
>>> a, b = 12, 16 # 여러 개를 한번에
```

```
>>> a = b = c = 0 # 같은 값으로 치환
```

```
>>> a = 17; b = 16 # ; 로 구분(not recommended)
```

```
>>> a, b = b, a # 교환, 다른 언어와 구분됨
```

- **Java** 등과는 달리 변수를 선언할 필요가 없고, 변수의 유형의 변환도 자유롭다. (변수의 자료형은 등호 오른쪽의 자료형에 의해 결정)

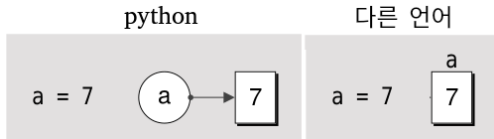
```
>>> a = 17
>>> type(a)
<type 'int'>
>>> a = 'dynamic'
>>> a
'dynamic'
>>> type(a)
<type 'str'>
```

확장 치환: **Java** 또는 **C++**와 같은 확장 치환연산자가 있다.

- +=, -=, \*=, /= etc
- a += 4와 a = a + 4는 같은 연산

# 이름과 객체

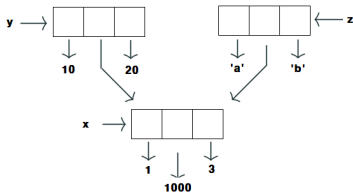
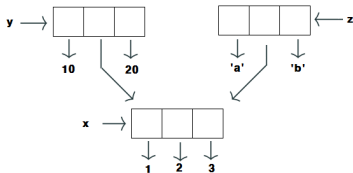
- 다른 언어와는 달리 변수 이름과 값 객체가 분리 (이런 이유로 유형의 변환이 자유로움)
- `a = 7`는 숫자 7이 변수 `a`에 저장되는 것이 아님 (`a`는 이름, 7은 객체)



# 변수와 값 객체 분리의 **side effect**(객체 공유)

```
>>> x = [1, 2, 3]; y = [10, x, 20]; z = ['a',x,'b']
```

```
>>> x
[1, 2, 3]
>>> y
[10, [1, 2, 3], 20]
>>> z
['a', [1, 2, 3], 'b']
>>> x[1] = 1000
>>> x
[1, 1000, 3]
>>> y
[10, [1, 1000, 3], 20]
>>> z
['a', [1, 1000, 3], 'b']
```



# 문자열로 된 코드 실행

- 문자열 Python 식(expression) 실행: `eval()`

```
>>> x = 2
>>> x = eval('x + 3')
>>> x
5
```

- 문자열 Python 문(statement) 실행: `exec`

```
>>> exec 'x = x + 3'
8
>>> s '''    # 여러 줄에 걸친 문자열
a = 1
if a > 0:
    print('True')
'''
>>> exec(s)
True
```

# Console I/O

- 키보드로부터 문자열 입력(input)

```
>>> name = input('name? ')
name? 홍길동
>>> print(name)
홍길동
```

- 수치형데이터는 문자열 입력 후 수치형으로 변환

```
>>> k = int(input('input data = '))
input data = 17
>>> k
17
```

- Console 출력(print)

```
>>> print('add = ' 4+5, 'sub = ', 4-5)
add = 9 sub = -1
```

# Console and file I/O

- `print`는 출력 후 줄이 바뀌지만 인수 `end`로 마지막 출력 문자 변경

```
>>> print(1, 2); print(3, 4)
1 2
3 4
>>> print(1, 2, end = ' '); print(3, 4)
1 2 3 4
>>> print(1, 2, 3, 4, sep = ', ')
1, 2, 3, 4
>>> print(1, 2, 3, 4, sep = ' ')
1 2 3 4
>>> f = open('c:/out.txt', 'w')      # 파일 출력
>>> print(1, 2, 3, 4, file = f)
>>> f.close()
>>> open('c:/out.txt').read()
'1 2 3 \n'
```



# 서식 출력

- `format(value, format_spec)`

```
>>> print(format(3.141593, '10.3f'))  
3.142
```

- 10자리에 소수 3 째 자리까지 출력 (반올림)
- f: 실수, d: 정수 s: 문자열

```
>>> 'Name: {0}, ID: {1}'.format('Lee', 'A')  
'Name: Lee, ID: A' ← 결과, not 출력
```

- {0}과 {1}는 `format()` 메소드의 첫 번째와 두 번째 인수

```
import math
for k in range(1,3) :
    print('sqrt({0}) = {1}'.format(k, math.sqrt(k)))

sqrt(1) = 1.0
sqrt(2) = 1.4142135623730951

import math
for k in range(1,3) :
    print('sqrt({0})={1:5.3f}'.format(k,math.sqrt(k)))

sqrt(1) = 1.000
sqrt(2) = 1.414
```

# Python

---

## Python Data Type

# 데이터의 종류

**Table 2: Python의 주요 내장 데이터 타입**

| 데이터 타입              | 설명                                 | Example               |
|---------------------|------------------------------------|-----------------------|
| bool                | True와 False                        | True, False           |
| int, float, complex | 정수, 실수 및 복소수                       | 123, 3.14, 5+4j       |
| str                 | 유니코드 문자열, 내용변경 불가                  | 'spams', "ham", "egg" |
| byte                | 0 ~ 255 사이 코드의 모임을 표현              | b'Python'             |
| list                | 순서가 있는 Python 객체의 집합               | ['ham', 'spam']       |
| dict                | 순서가 없는 Python 객체의 집합<br>key로 값을 참조 | {'ham':4, 'spam':5}   |
| tuple               | 순서가 있는 Python 객체의 집합,<br>내용변경 불가   | ('ham', 'spam')       |
| set                 | 집합을 표현                             | {1, 2, 3}             |

- 참 또는 거짓을 나타내는 **True**과 **False**, 주로 조건문에서 사용

```
>>> a = 1; a < 0
```

```
False
```

- bool** 값은 정수로 취급(**True**는 1, **False**는 0)

```
>>> True + True
```

```
2
```

```
>>> True * False
```

```
0
```

- 식(**expression**)의 **bool** 값은 **bool()** 함수 이용

```
>>> bool(3)
```

```
True
```

```
>>> bool(0)
```

```
False
```

```
>>> a = 2; bool( a > 3)
```

```
False
```

# str(문자열)

- 문자열은 따옴표(' ')나 중 따옴표(" ")로 묶인 문자들의 모임

```
>>> s = "Hello World!"
```

- Indexing으로 문자열 안의 문자를 선택할 수 있다.

```
>>> s[0]
```

```
'H'
```

```
>>> s[-1]
```

```
'!'
```

- Slicing(start:stop:step)

```
>>> s[1:]      # 1 위치에서 끝까지
```

```
>>> s[:3]      # 처음 (0)에서 2까지
```

```
>>> s[:]       # 처음 (0)부터 끝까지
```

```
>>> s[::2]     # 하나씩 건너 뛰어서
```

```
>>> s[::-1]    # 역으로
```

- 문자열의 연결(+)과 반복(\*)

```
>>> 'Hello' + 'World!'
'HelloWorld!'
>>> 'Hello' * 3
'HelloHelloHello'
```

- 문자열의 특정 요소의 값은 변경되지 않는다.

```
>>> s[0] = 'h'
TypeError: ...
```

- 문자열을 변경하고 싶은 경우 문자열 전체를 새로 정의

```
>>> s = 'h' + s[1:]
>>> s
'hello World!'
```

- len() 함수: 문자열의 길이

```
>>> len(s)
12
```

- `in` 연산자: 부분 문자열의 존재 확인

```
>>> 'world' in s
```

```
False
```

```
>>> 'World' in s
```

```
True
```

```
>>> 'World' not in s
```

```
False
```



# str의 메소드(method)

- 문자열 객체에서 사용할 수 있는 메소드

```
>>> s = 'Hello World!'
>>> s.upper()                # 대문자로 전환
'HELLO WORLD!'
>>> s.split()                # 공백을 기준으로 분리
['Hello', 'World!']          # return list
>>> s.find('World')          # 부분 문자열의 위치
6
>>> s.find('world')
-1
>>> s.startswith('Hello')    # 시작 부분 문자열 확인
True
>>> s.endswith('ld')
False
```

- 문자열은 유니코드, 바이트는 0 ~ 255의 코드 (ASCII code)

```
>>> b = b"Hello World!"
>>> b
b'Hello World!'
>>> type(b)
<class 'bytes'>
```

- 문자열에서 사용할 수 있는 메소드 및 연산은 **byte**에서도 사용 가능
- 한글 문자 사용할 수 없다.
- 문자열로 변환하려면 **decode()** 메소드 사용

```
>>> b.decode()           # UTF-8: default
>>> b.decode('euc-kr')   # 인코딩 지정
>>> s.encode()           # str을 byte로 변경
```

# list

- 대괄호([ ])를 사용하여 객체를 저장하는 집합적인 자료형

```
>>> L = [1, 2, 3] # list
```

```
>>> len(L)      # 원소 개수
```

```
>>> L[1]        # 2 번째 요소
```

```
2
```

```
>>> L[-1]       # 마지막 요소
```

```
3
```

```
>>> L[1:3]      # slicing
```

```
[2, 3]
```

```
>>> L + L
```

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

```
>>> L * 2
```

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

```
>>> L = L * 3
```

```
>>> L[:2]
```

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

```
>>> 4 in L      # 멤버 검사
```

```
False
```

# Slicing

- 시퀀스 자료형의 일정 영역에서 새로운 객체를 반환, 결과의 데이터형은 원래의 데이터형과 같다

```
>>> s = 'abcdef'; L = [1,2,3]
```

```
>>> s[1:3]    # 2번째에서 4번째?
```

```
'bc'
```

- 1 번 위치에서 3 번 위치까지

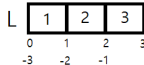
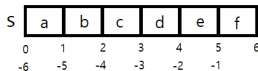
```
>>> L[:-1]    # 맨 오른쪽을 제외한 나머지
```

```
[1, 2]
```

```
>>> s[-100:100] # 범위 밖
```

```
'abcdef'
```

- 범위를 넘어서면 범위내의 값으로 자동 처리



# list method

- 리스트 객체에서 사용할 수 있는 메소드

```
>>> L = [1, 2, 3]           # 3 개의 데이터로 구성된 list
>>> L.append(4)             # 구성요소 추가
>>> L
[1, 2, 3, 4]
>>> del L[0]                # 데이터 삭제
>>> L
[2, 3, 4]
>>> L.reverse()             # 역순
>>> L
[4, 3, 2]
>>> L.sort()                # 오름차순 정렬
>>> L
[2, 3, 4]
```

# tuple

- 괄호()를 사용하여 객체를 저장하는 집합적인 자료형, **list**와 유사한 특성이 있으나 값을 변경할 수 없다.

```
>>> t = (1, 2, 3)
>>> len(t)      # 원소 개수
>>> t[1]        # 2 번째 요소
2
>>> t[-1]       # 마지막 요소
3
>>> t[0:2]      # slicing
(1, 2)
```

```
>>> t + t
(1, 2, 3, 1, 2, 3)
>>> t * 2
(1, 2, 3, 1, 2, 3)
>>> 3 in t      # 멤버 검사
True
>>> t[0] = 100
Error message
```

# 사전(dictionary)

- 중괄호({ })를 사용하여 객체를 저장하는 집합적인 자료형으로 키를 이용하여 값을 참조(사전(dictionary)에서는 순서가 무의미)

```
>>> d = {'one': '하나', 'two': '둘'}           # key:값
>>> d=dict('one': '하나', 'two': '둘')       # 동일
>>> d['one']                                   # 키에 의한 값 추출
'하나'
>>> d['three'] = '셋'                         # 요소 추가
>>> d                                          # 전체 출력
{'three': '셋', 'one': '하나', 'two': '둘'}
>>> d['one'] = 1                              # 값 변경
>>> d.keys()                                 # key만 추출
dict_keys(['one', 'two'])
>>> d.values()                               # 값만 추출
dict_values(['하나', '둘'])
>>> d.items()                                # tuple로 변환
dict_items([('one', '하나'), ('two', '둘')])
```

# 집합 (set)

- 중복되지 않는 데이터를 순서없이 저장하는 자료형
- 멤버 검사와 중복된 항목 제거에 유용

```
>>> s1 = {1, 2, 3}          # 집합 생성
>>> L = [1, 4, 3, 2, 3]
>>> s = set(L)              # set으로 변환, 중복제거
>>> s
{1, 3, 4}
>>> 3 in s                  # True
>>> s.union(s1)             # 합집합
>>> s | s1                  # 합집합
>>> s.intersection(s1)     # 교집합
>>> s & s1                  # 교집합
>>> s - s1                  # 차집합
>>> s.add(4)                # 원소 추가
>>> s.discard(4)            # 원소 제거
```



# Python

---

자료형의 분류

# 자료형의 분류

**Table 3:** 저장 방법에 따른 자료형의 분류

| 자료형       | 설명                          |   |
|-----------|-----------------------------|---|
| Direct형   | 직접 데이터를 표현하는 자료형            | int, float, complex                       |
| Sequence형 | 다른 데이터를 포함하고, 순서가 있는 집합 자료형 | list, str, tuple, bytes, bytearray, range |
| Mapping형  | 다른 데이터를 포함하는 자료형            | dict                                      |
| Set형      | 순서와 중복이 없는 자료형              | set, frozenset                            |

**Table 4:** 변경 가능성에 따른 자료형의 분류

| 자료형                 | 설명             |  |
|---------------------|----------------|--|
| 변경 가능형 (Mutable)    | 데이터의 값이 변경 가능  | list, dict, set                            |
| 변경 불가능형 (Immutable) | 데이터의 값이 변경 불가능 | int, float, complex, str, tuple, frozenset |

**Table 5:** 저장 개수에 따른 자료형의 분류

| 자료형        | 설명           |  |
|------------|--------------|--|
| Literal형   | 한 가지 객체만 저장  | str, bytes, bytearray, int, float, complex |
| Container형 | 여러 가지 객체를 저장 | list, tuple, dict, set, frozenset          |

# 변경 가능성

- mutable: list, dict, set

```
>>> s = [1, 2, 3]
>>> s[1] = 10
>>> s
[1, 10, 3]
```

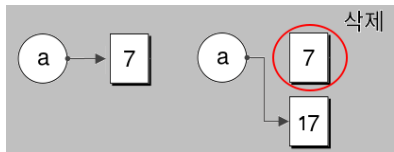
- immutable: int, float, tuple 등

```
>>> s = (1, 2, 3)
>>> s[1] = 10
TypeError                                Traceback (most recent call...
<ipython-input-21-d8b471273ca8> in <module>
----> 1 s[1] = 10
```

TypeError: 'tuple' object does not support

```
>>> a = 7
>>> a = 17
```

- 값이 변경?
- 값 7은 변경되지 않고 다만 a가 다른 객체(17)를 참조



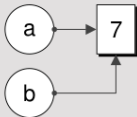
## Relationship between objects and object references

a = 7

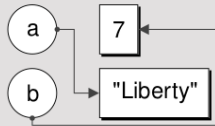


The circles represent object references.  
The rectangles represent objects in memory.

a = 7  
b = a



a = 7  
b = a  
a = "Liberty"



# Python

---

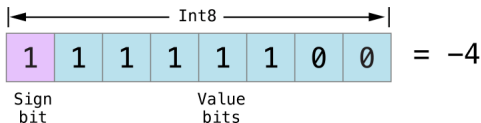
연산

# 수치형 자료

- 수치형 자료에는 정수, 실수(float), 복소수가 있다.
- 정수
  - 정수형은 유효 숫자에 제한이 없다.

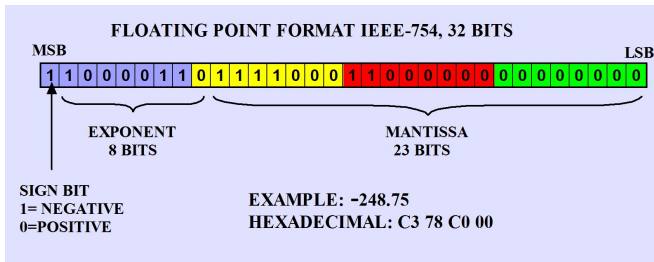
```
>>> 2**1024
```

```
179769313486231590772930519078902473361797697894230657  
430081157732675805500963132708477322407536021120113879  
393357658789768814416622492847430639474124377767893424  
485276302219601246094119453082952085005768838150682342  
881473913110540827237163350510684586298239947245938479  
304835356329624224137216
```



- 실수

- 실수는 8byte(64bit)로 표현
- 유효숫자는 대략 15자리,  $2.225078e-308 \sim 1.79769313e+308$



- 유효 숫자의 제한으로 컴퓨터 연산은 오차를 가질 수 밖에 없다.
- 정수와 정수, 실수와 실수의 연산은 각각 정수 및 실수가 된다.
- 정수와 실수의 연산은 실수이다.

# 정수

```
>>> a = 17 # 10진수
>>> b = 0o17 # 8진수
>>> c = 0x17 # 16진수
>>> d = 0b10 # 2진수
>>> print(a, b, c, d)
17 15 23 2 # 10 진수 출력
>>> int('123') # 10 진수로 변환
>>> bin(17) # 2 진수로 변환
>>> oct(17) # 8 진수로 변환
>>> hex(17) # 16 진수로 변환
>>> int(3.14) # 실수를 정수로 변환
>>> int('123.45') # error
>>> int(float('123.45')) # 실수 변환 후 정수 변환
```



# 실수

```
>>> a = 1.7                # 소수점 포함한 수
>>> a = 17e-1              # 실수
>>> a = 17e-1.             # 오류(지수부는 정수여야)
>>> float('inf')           #  $\infty \rightarrow$  실수
>>> float('-inf')/1000.
-inf
>>> round(1.2)             # 반올림
>>> import math            # math module
>>> math.ceil(1.2)          # 올림
>>> math.ceil(-1.2)         # ?
>>> math.floor(1.2)         # 내림
```

# 누적 오차

- 실수는 제한된 유효숫자 때문에 누적 오차가 발생할 수 있다.

```
xSum = 0. # 초기화
```

```
for k in range(1000):  
    xSum = xSum + 0.1
```

```
print(xSum)
```

99.99999999999986

- 컴퓨터에서의 연산은 항상 근사값을 구한다.
- 오차없는 연산이 필요하면 **assembler** 언어를 구사하여야 한다.

- Python에서는 decimal module 이용할 수 있다.

```
from decimal import *
```

```
xSum = Decimal('0.')
```

```
eps = Decimal('0.1')
```

```
for k in range(1000):  
    xSum = xSum + eps
```

```
print(xSum)
```

```
100.0
```

- Decimal 객체와 정수형 자료와는 연산이 가능하나 실수형과는 연산이 안된다.

```
>>> Decimal('1.7')/ 3.14           # error
```

# 산술연산자

- 산술연산자: +, -, \*, /, //(몫), %(나머지), \*\*(지수)

**Table 6:** 산술 연산자 우선 순위

| 산술연산자       | 설명              | 결합순서     |
|-------------|-----------------|----------|
| +, -        | 단항 연산자          | 오른쪽에서 왼쪽 |
| **          | 지수              | 오른쪽에서 왼쪽 |
| *, /, %, // | 곱하기 나누기, 몫, 나머지 | 왼쪽에서 오른쪽 |
| +, -        | 더하기, 빼기         | 왼쪽에서 오른쪽 |

```
>>> 4 * -5
```

```
>>> 2 + 3 * 4
```

```
>>> (2 + 3) * 4
```

```
>>> 4 / 2 * 2
```

```
>>> 2 ** 3 ** 4          # 234
```

```
>>> (2 ** 3) ** 4       # (23)4
```

# 관계연산자

- 관계연산자: >, <, >=, <=, ==, !=

- 문자열의 비교는 **unicode**의 선후 비교(사전 역순)

```
>>> 'abcd' > 'abd' # False
```

- **list, tuple**은 앞에서 하나씩 비교

```
>>> (1, 2, 4) < (2, 1, 0) # True
```

- '=='는 같은 값을 갖는지 비교

```
>>> x = [1, 2, 3]; y = [1, 2, 3]
```

```
>>> x == y # True
```

- 같은 객체를 참조하는가는 **is** 연산자

```
>>> x is y # False
```

```
>>> z = y
```

```
>>> y is z # True
```

# 논리연산자

- 논리연산자: not, and, or
- 정수 0과 실수 0.은 False, 나머지는 True

```
>>> bool(0.0) # False
```
- 빈 객체((), {}, [])는 False

```
>>> bool('') # False
>>> bool([]) # False
```
- any(), all()

```
>>> x = [0, 1, 2]
>>> all(x) # 모두가 True?
>>> any(x) # 하나라도 True?
>>> all(i < 1 for i in x) # False
```
- 논리식의 결과를 판정하는 최종적 객체 반환

```
>>> 1 and 0 # 0
>>> [] and 1 # []
>>> 1 and 2 # 2
```

# 비트연산자

---

- omitt

# 수치 연산 함수

- 내장함수
  - `abs()`, `int()`, `float()`, `pow()`, `max()`, `min()`
  - `divmod(x,y) = (x//y, x%y)`
  - `complex(re, im)`, `c.conjugate()`
- `math(cmath)` module의 함수
  - `module.변수`, `module.함수로 호출`
    - `>>> print(math.pi)`
    - `>>> print(math.e)`
    - `>>> math.sqrt(2)`
    - `>>> math.exp(2)`
    - `>>> math.exp(2.)`



# 문자열 **method**

- Python에는 문자열에 대한 많은 **method**가 있다. (문자열.method 이름())
- 대소문자 변환:  
`s.upper()`, `lower`, `swapcase`, `capitalize`, `title`
- 검색: `count`, `find`, `rfind`, `index`, `rindex`, `startswith`, `endwith`
- 편집: `strip`, `rstrip`, `lstrip`, `replace`
- 분리와 결합: `split`, `join`, `splitlines`
- 정렬: `cener`, `ljust`, `rjust`
- etc

```
>>> L = ["Lee", "Kim"]
```

```
>>> ", ".join(L)
```

```
'Lee,Kim'
```

# list의 연산

- 리스트는 **mutable, and sequence** 형인 객체들의 집합으로 가장 유용하게 사용됨.
- **sequence** 형이므로 인덱싱, 슬라이싱, 연결, 반복, 멤버 검사 등이 가능
- 데이터의 크기를 동적으로 조절 가능

```
>>> L = list(range(10))    # Q: L = range(10)
>>> L[0:2] = [10, 11]     # 두 값 교체
>>> L[0:2] = [10]         # 두 값 교체(항목 1 삭제)
>>> L[0:2] = []           # 두 항목 삭제
>>> del L[1:]             # 항목 삭제
>>> a = [1,2]
>>> a[1:1] =['a', 'b']    # 1위치에 두 항목 삽입
[1, 'a', 'b', 2]
>>> a[:2]                 # 확장 slicing
[1, 'b']
```

- 확장 slicing이 왼쪽에 오는 경우는 항목 갯수가 일치하여야 한다.

```
>>> a[::2] = list(range(5)) # 갯수 불일치, error
```

```
>>> a = list(range(3)) # 갯수 불일치 but ok
```

```
>>> s = [1, 2, 3]
```

```
>>> t = ['a', s, 'b']
```

```
>>> t
```

```
['a', [1, 2, 3], 'b'] # 중첩리스트
```

# list의 method

- append, insert, index, count, sort, reverse, remove, pop, extend

```
>>> s = [1, 2, 3]
>>> s.append(5)           # 리스트의 끝에 값 5를 삽입
>>> s.insert(3, 4)        # 3위치에 4 삽입
>>> s.index(3)            # 값 3의 위치
>>> s.count(2)            # 값 2의 개수
>>> s.reverse()           # 순서를 역으로 변경
>>> s.sort()              # 크기 순으로 정렬
>>> s.remove(2)           # 값 2 삭제, 첫 번째만
>>> s.extend([6, 7])     # 끝에 항목 추가
```

- Stack: LIFO 구조

```
>>> s = [10, 20, 30, 40, 50]
```

```
>>> s.append(60)
```

```
>>> s.pop()
```

```
60
```

```
>>> s
```

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

```
>>> s.pop(0) # 첫번째 요소 추출
```

```
>>> s.sort() # ascending sort
```

```
>>> s.sort(reverse = True) # descending
```

# list 내장

- sequence형 자료로 list를 만드는 방법

```
[ <식> for x1 in 객체1  
    for x2 in 객체2  
    .  
    for xn in 객체n  
    (if 조건식) ]
```

```
>>> L = [ k*k for k in range(5) if k % 2 == 1]
```

```
>>> L
```

```
[1, 9]
```

```
>>> L = [] # Same as above
```

```
>>> for k in range(5):
```

```
...     if k % 2 == 1 L.append(k*k)
```

```
>>> seq1 = 'abc'; seq2=(1,2,3)
>>> [(x,y) for x in seq1 for y in seq2]
[('a', 1), ('a', 2), ('a', 3), ('b', 1), ... ]
>>> L=[[j+i*3 for j in [10,11,12]] for i in [0,1,2]]
[[10, 11, 12], [13, 14, 15], [16, 17, 18]]
>>> L[1][2]
15
```

# 발생자

```
>>> a = (k*k for k in range(5))    # ()이면 발생자 객체
```

```
>>> a
```

```
<generator object <genexpr> at 0x00000009777C50>
```

```
>>> sum(a)                        # 30
```

- list 내장보다 발생자가 효율적일 수 있다.

```
>>> range(10)                    # 발생자 객체
```

```
>>> range(5, 10)                 # 5 부터
```

```
>>> range(5, 10, 2)              # 5 부터 2 간격으로
```

- range 객체는 순차적인 값을 할당하는데도 사용

```
>>> Sun, Mon, Tue, Wed, Thu, Fri, Sat = range(7)
```

```
>>> Sun, Mon, Sat
```

```
(0, 1, 6)
```



# tuple

- immutable sequence형 자료

```
>>> t = ()           # 빈 tuple
>>> t = 1,           # t = (1,)
>>> t = 1, 2, 3      # t = (1,2,3)
>>> t * 2             # (1,2,3,1,2,3)
>>> t + ('a', 'b')    # (1,2,3,'a','b')
>>> t[0], t[1:3]      # (1, (2,3)) 중첩 tuple
>>> len(t)            # 길이, 3
>>> 1 in t            # 멤버 검사 True
>>> t[0] = 100        # error
>>> t.count(2)        # 2 몇개?
>>> t.index(2)        # 2의 위치
```

# 패킹, 언패킹

- 치환

```
>>> x, y, z = 1, 2, 3
```

```
>>> (x1, y1), (x2, y2) = (1,2), (3,4)
```

- 패킹: 한 데이터에 여러 개의 데이터 치환

```
>>> t = 1, 2, 'Hello'
```

- 언패킹: 여러 개의 데이터를 하나의 데이터로 치환

```
>>> x, y, x = t
```

```
>>> L = [1,2,3] # list도 언패킹 가능
```

```
>>> x, y, x = L
```

```
>>> T = 1, 2, 3, 4, 5
```

```
>>> a, b* = T # a = 1, b = [2,3,4,5]
```

- b\*는 나머지 전부를 의미

```
>>> a*, b = T # error
```

# tuple의 활용

- 함수가 하나 이상의 값을 반환할 때

```
>>> def add_prod(a, b):  
    return a + b, a * b
```

```
>>> x, y = add_prod(2, 3)
```

- tuple 값을 함수의 인수로 사용할 때

```
>>> args = (2, 3); add_prod(*args)
```

- tuple과 list는 내장함수 list()와 tuple()을 이용하여 상호 변환 가능

```
>>> L = list(T)
```

```
>>> T = tuple(L)
```

# tuple의 예제

- tuple을 이용한 경로명

```
>>> import os
```

```
>>> p = os.path.abspath('test.tex') # 절대경로 반환
```

```
>>> p
```

```
'/home/seung/test.tex'
```

```
>>> os.path.exists(p) # 파일 존재 여부 검사
```

```
>>> os.path.getsize(p) # 파일 크기 반환
```

```
>>> os.path.split(p) # 파일명과 경로 분리
```

```
('/home/seung', 'test.tex')
```

# tuple을 이용한 URL

- `urllib.parse` module은 URL을 성분별로 분해하거나 결합하는 인터페이스를 제공
- `urlparse()` 함수는 URL을 (addressing scheme, network location, path, parameters, query, fragment identifier)로 분리하여 tuple로 반환

```
from urllib.parse import urlparse
url='https://wikipedia.org/Limits_of_computation'
r = urlparse(url)
r
ParseResult(scheme='https', netloc='wikipedia.org',
path='/Limits_of_computation', params='', query='',
fragment='')
```

# set

- 여러 값을 순서 및 중복없이 모아 놓은 자료형
  - set: mutable; frozenset: immutable

```
>>> s1 = set()           # 빈 set 객체 생성
>>> s2 = {1, 2, 3}
>>> s3 = s2.copy()       # s2 복사
>>> s4 = set((1,2,3))    # tuple 또는 list로 생성
>>> s5 = set('abc')      # {'a', 'b', 'c'}
>>> set({'one':1, 'two':2}) # 사전으로 생성
{'one', 'two'}           # 키만 반환
>>> {[1,2],[2,3]}        # error
```

In Python, both str and the basic numeric types such as int are immutable, that is, once set, their value cannot be changed. At first this appears to be a rather strange limitation, but Python's syntax means that this is a nonissue in practice.

- 단어의 수는?

```
sentence = ''
```

In Python, both str and the basic numeric types such as int are immutable that is, once set, their value cannot be changed. At first this appears to be a rather strange limitation, but Python's syntax means that this is a nonissue in practice.

```
'''
```

```
len(set(sentence.upper().split()))
```

```
39
```

```
>>> s = {1,2,3}
>>> len(s)                # 원소의 개수
>>> s.add(4)               # {1,2,3,4}
>>> s.update([3,4])        # s cup {3,4}
>>> s.update({3,4})        # same as above
>>> s.clear()              # 전체 원소 제거
>>> s5 = set('abc')        # {'a', 'b', 'c'}
>>> s.discard(3)           # 원소 3 제거, 없어도 ok
>>> s.remove(3)            # 원소 3 제거, 없으면 예외 발생
>>> s.pop()                # 원소 하나를 제거하면서 이를 반환
```



# disctionary

- 임의 객체의 집합적 매핑형 자료(**key**에 의해 값을 접근)
- 값은 임의의 객체가 될 수 있으나 **key**는 **immutable**

```
>>> member = {'basketball':5, 'soccer':11}
```

```
>>> member['basketball']          # 호출 5
```

```
>>> member['baseball':9]          # 멤버 추가
```

```
>>> del member['baseball']        # 멤버 삭제
```

- 함수를 **key**나 값으로 활용할 수 있다.

```
>>> action = {0:add, 1:sub}
```

```
>>> action[0](4,5)
```

```
dict(one=1, two=2)                # dict함수에 의한 생성
```

# dictionary method

- 사전의 method(Let D be a dictionary)

`D.clear()`                      # 모든 항목 삭제

`D.copy()`                      # 사전 복사

`D.get(key [. x])`              # 값이 존재하면 `D[key]` 아니면 `x`

`D.pop(key)`                    # `key`의 값을 반환하고 삭제

- 사전의 내장

```
>>> {w:1 for w in 'abc'} # {'a':1, 'b':1, 'c':1}
```

```
>>> a1 = 'abc'; a2=(1,2,3)
```

```
>>> {x:y for x,y in zip(a1, a2)}
```

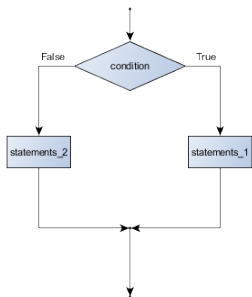
```
{'a':1, 'b':2, 'c':3}
```

# Python

---

## Control Structure

# if statement



```
if 조건식:
    statements
else:
    statements
```

```
if 조건식1:
    statements
elif 조건식2:
    statments
else:
    statements
```

```
x = 2 ; y = 3
if x > y :
    result = str(x)+" is greater than " + str(y)
else:
    result = str(y)+" is greater than or equal to "+ str(x)
print(result)
```

# Python의 들여 쓰기

- Python은 들여 쓰기에 의해 블록화(들여 쓰기에 매우 민감)

- 가장 바깥의 블록은 반드시 1열에서 시작

```
>>> a = 1          # OK
```

```
>>>   a = 2        # error
```

- 블록 안에서는 같은 거리의 들여 쓰기

```
>>> if a > 1:
```

```
    print("a is greater than 1")
```

```
    print("Is it big?")          # error
```

- 들여 쓰기 간격은 일정하게
- **tab**과 공백을 함께 사용하는 것은 바람직하지 않다.
- 대부분의 IDE는 자동적으로 들여 쓰기를 한다.

# if 문의 대체

- if 문 대신 때때로 사전을 활용하면 편리한 경우가 있다.
- python스러운 프로그래밍

```
order = "spagetti"
if order == "egg":
    price = 100
elif order == "ham":
    price = 200
elif order == "spagetti":
    price = 300
else price = 0

menu = {"egg":100, "ham":200, "spagetti":300}
price = menu.get(order, 0)    # 0: default for no key
```

```

if a > 5 :
    x = a * 2
else :
    x = a/2

```

· a if 조건식 else b

조건식이 True이면 a, False이면 b가 값이 되는 삼항 연산자

```

x = [a/2, a*2][a > 5] # python style coding

```

```

def add (a, b):
    return a + b
def sub (a, b):
    return a - 2

```

```

select = 0
x = (add, sub)[select](2, 3) # add 호출
print(x)
{True:add, False:sub}[select > 10](3, 4)

```

# 반복문: for loop

- for loop

for <target> in <sequence형 객체>:

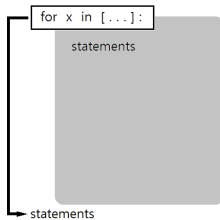
statements

( continue )

( break )

(else:)

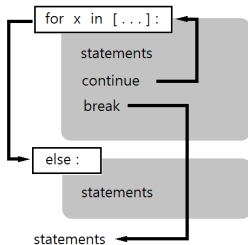
next statements



- 괄호안은 optional
- 객체의 각 항목이 **target**으로 치환되면서 블록내의 **statements** 실행
- 모든 치환이 완료되면 **for-loop**가 종료되고 다음 **statement** 실행



- **continue**문
  - 블록의 끝 or **continue**를 만나면 **loop**의 시작 부분으로
- **break**문
  - **break**문을 만나면 **next statement**
- **break, else**문(**for-loop**블록 밖에 **else**블록이 있을 때)
  - **break**를 만나면 **else**블록 다음의 **statement**로
  - **break**를 만나지 않고 정상적으로 종료되면 **else**부분으로



```
x = ["cat", "dog"]
```

```
for animal in x:
```

```
    print(animal)
```

```
cat
```

```
dog
```

- **enumerate**: 반복문 사용 시, 몇 번째 반복인지 확인이 필요할 때  
(인덱스 번호와 원소를 **tuple**로 반환)

```
for p in enumerate(x):
```

```
    print(p)
```

```
(0, 'cat')
```

```
(1, 'dog')
```

```
for k, animal in enumerate(x):
```

```
    print(k, animal)
```

```
0 cat
```

```
1 dog
```

```
sum = 0
for i in range(10) : # range(0,10), 0,1,...,9
    sum = sum + i
print(sum)
45
sum(range(10))
45
```

# 구구단

---

```
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4
1 * 5 = 5
1 * 6 = 6
1 * 7 = 7
1 * 8 = 8
1 * 9 = 9

2 * 1 = 2
2 * 2 = 4

----
```

```
for i in range(1,10) :
    for j in range(1,10) :
        print('{0} * {1} = {2}'.format(i, j, i*j))
    print()
```

1 \* 1 = 1  
1 \* 2 = 2  
1 \* 3 = 3  
1 \* 4 = 4  
1 \* 5 = 5  
1 \* 6 = 6  
1 \* 7 = 7  
1 \* 8 = 8  
1 \* 9 = 9

4 \* 1 = 4  
4 \* 2 = 8  
4 \* 3 = 12  
4 \* 4 = 16  
-----

2 \* 1 = 2  
2 \* 2 = 4  
2 \* 3 = 6  
2 \* 4 = 8  
2 \* 5 = 10  
2 \* 6 = 12  
2 \* 7 = 14  
2 \* 8 = 16  
2 \* 9 = 18

5 \* 1 = 5  
5 \* 2 = 10  
5 \* 3 = 15  
5 \* 4 = 20  
-----

3 \* 1 = 3  
3 \* 2 = 6  
3 \* 3 = 9  
3 \* 4 = 12  
3 \* 5 = 15  
3 \* 6 = 18  
3 \* 7 = 21  
3 \* 8 = 24  
3 \* 9 = 27

6 \* 1 = 6  
6 \* 2 = 12  
6 \* 3 = 18  
6 \* 4 = 24  
-----

- This is somewhat challenging.

# 반복문

- while loop

```
while<조건식>:
```

```
    statements  
    ( continue )  
    ( break )
```

```
(else:)  
next statement
```

- 조건식이 True이면 블록안의 **statement** 실행
- **break** 문을 만나면 **next statement**
- 블록의 끝 or **continue**를 만나면 **loop**의 시작부분으로
- **break**를 만나지 않고 정상적으로 종료되면 **else**부분으로

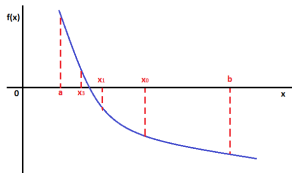
```
count = 1; xsum = 0
```

```
while count < 10:
```

```
    xsum = xsum + count
```

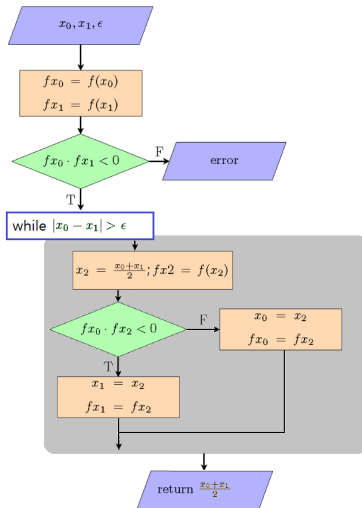
```
    count = count + 1
```

## Bisection: a Root find method



### Algorithm for Bisection Method

- I. Take initial approx. for  $x_1$  and  $x_2$ .
- II. Compute  $f(x_1)$  and  $f(x_2)$
- III. If  $f(x_1).f(x_2)>0$ , then no root lies between  $x_1$  and  $x_2$  and goto step [V] otherwise goto step IV.
- IV. Calculate  $x_0 = \frac{x_1 + x_2}{2}$  and  $f(x_0)$
- V. If  $f(x_1) f(x_0) < 0$  then take  $x_2 = x_0$  otherwise set  $x_1 = x_0$
- VI. Next approx  $= \frac{x_1 + x_2}{2}$   
Repeat step IV
- VII. Stop.





```

import math
def f(x): return math.exp(-abs(x)) - x/(1 + x * x)

def bisection(x0, x1, eps = 0.00001):
    fx0 = f(x0); fx1 = f(x1)

    if (fx0*fx1) > 0 :
        print("Wrong guess")
        return ()

    while abs(x0 - x1) > eps :
        x2 = (x0 + x1)/2 ; fx2 = f(x2)

        if (fx0*fx2) < 0:
            x1 = x2; fx1 = fx2
        else:
            x0 = x2 ; fx0 = fx2

    return ((x0+x1)/2)

print(bisection(0,2))
0.7384262084960938

```

```
import math
def f(x):
    return math.exp(-abs(x)) - x/(1 + x * x)

def bisection(x0, x1, eps = 0.00001):
    x = [[x0, f(x0)], [x1, f(x1)]]

    if x[0][1] * x[1][1] > 0 :
        print("Wrong guess")
        return()

    while abs(x[0][0] - x[1][0]) > eps :
        xn = (x[0][0] + x[1][0])/2
        fxn = f(xn)
        x[x[0][1] * fxn < 0] = [xn, fxn]

    return(x[0][0] + x[1][0])/2
```

# Magic Square

|    |    |    |    |    |    |
|----|----|----|----|----|----|
| 15 | 8  | 1  | 24 | 17 | 65 |
| 16 | 14 | 7  | 5  | 23 | 65 |
| 22 | 20 | 13 | 6  | 4  | 65 |
| 3  | 21 | 19 | 12 | 10 | 65 |
| 9  | 2  | 25 | 18 | 11 | 65 |
| 65 | 65 | 65 | 65 | 65 | 65 |

step 1 input  $n$  (an odd number), generate  $n \times n$  matrix  $M$

step 2  $i = 0$ ;  $j = \text{int}((n+1)/2)-1$ ;  $k = 1$ ;  $M[i][j] = k$

step 3  $k = k + 1$

step 4  $i = i - 1$ ;  $j = j - 1$

step 5 check conditions

1.  $i < 0 \ \& \ j \geq 0 \Rightarrow i = n$

2.  $i \geq 0 \ \& \ j < 0 \Rightarrow j = n$

3.  $i < 0 \ \& \ j < 0 \Rightarrow i = i+2 ; j = j + 1$

4.  $M[i][j] \neq 0 \Rightarrow i = i+2 ; j = j + 1$

Step 6 repeat from step 3 until  $k = n*n$

```

def mSquare(n): # n: an odd number
    nsqr = n * n
    M = [[0 for _ in range(n)] for _ in range(n)] # n x n matrix
    i = 0 ; j = int((n+1)/2) - 1 ; M[i][j] = 1

    for k in range(2, nsqr+1):
        i = i - 1 ; j = j - 1
        if ((i < 0) & (j >= 0)):
            i = n-1
        elif ((i >= 0) & (j < 0)) :
            j = n-1
        elif ((i < 0) & (j < 0)) :
            i = i + 2
            j = j + 1
        elif M[i][j] != 0 :
            i = i + 2
            j = j + 1

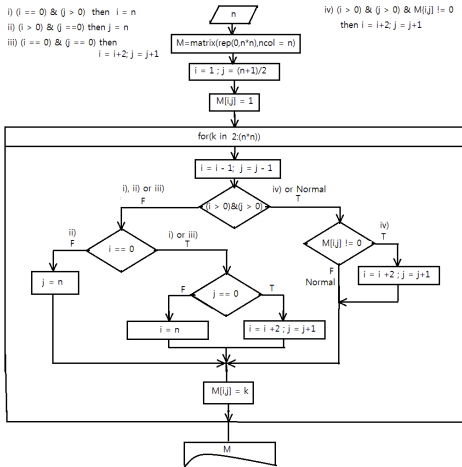
        M[i][j] = k

    for i in range(n):
        for j in range(n):
            print(format(M[i][j], '5d'), end = " ")
        print()

```

i)  $(i == 0) \ \& \ (j > 0)$  then  $i = n$   
 ii)  $(i > 0) \ \& \ (j == 0)$  then  $j = n$   
 iii)  $(i == 0) \ \& \ (j == 0)$  then  
      $i = i+2; j = j+1$

iv)  $(i > 0) \ \& \ (j > 0) \ \& \ M[i,j] != 0$   
 then  $i = i+2; j = j+1$



```

def mSquare(n):  # n: an odd number
    nsqr = n * n
    M = [[0 for col in range(n)] for row in range(n)]
    i = 0 ; j = int((n+1)/2) - 1 ; M[i][j] = 1

    for k in range(2, nsqr+1):
        i = i - 1; j = j - 1
        if (i >= 0) & (j >= 0) :  # subscripts are normal
            if (M[i][j] != 0):
                i = i + 2; j = j + 1
            else:  # subscripts out of range
                if( i == -1):
                    if (j == -1):
                        i = i + 2 ; j = j + 1
                    else: i = n-1
                else: j = n - 1

        M[i][j] = k

    for i in range(n):
        for j in range(n):
            print(format(M[i][j], '5d'), end=" ")
        print()

    return(M)

```

# try - except

- 예외(error)가 발생하면 프로그램이 종료
- 예상할 수 있는 예외에 대해, 이를 잡아내고 처리

```
>>> s = 'abcd'
```

```
>>> n = int(s)      # ValueError
```

```
Traceback (most recent call last):
```

```
File "<ipython-input-2-09e4a191d130>", line 2, in ...
```

```
n = int(s)
```

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

```
try:
```

```
    n = int(s)      # ValueError
```

```
except ValueError:
```

```
    n = 0
```

```
    print("invalid string for integer")
```

```
invalid string for integer
```

# Python

---

## Function



# 함수(function)

- 함수: 하나의 이름으로 프로그램 코드를 묶는 단위
- Why function?
  - 관리에 편리: 디버깅이 간단하고, 코드 수정도 쉽다.
  - 반복적인 코드를 없애 주어 프로그램의 작성이 쉽다.
  - 전체 작업을 논리적 구조로 나누어 프로그램을 작성할 수 있다(structure programming)

- 함수의 정의

```
def fun_name (args):  
    statements  
    ...  
    return(값)
```

- fun\_name: 함수명, identifier
- args: 매개변수 또는 인수(함수에 전달할 변수, parameter)
- return: 실행 결과를 호출 곳에 반환

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

```
>>> print(add(3,4))
```

```
7
```

```
>>> print(add('a','b'))
```

```
'ab'
```

- 인수들의 자료형은 동적으로 결정하고 자료형에 따른 연산
- **return** 문이 없으면 **None** 객체가 전달
- 인수의 기본값(**default**)을 줄 수도 있다.

```
def add(a, b = 1):  
    return a + b
```

```
add(5)      # 6
```

```
add(5, 2)   # 7
```

- 여러 개의 값을 return하면 tuple로 전달

```
def swap(a, b):  
    return b, a
```

```
c = 10; d = 20;  
swap(c, d)  
(20, 10)
```

- 사실 괄호()를 사용하지 않아도 쉼표 ", "로 데이터를 구분하면 tuple로 처리

```
>>> t = 1, 2, 3    # tuple
```

- 괄호()는 수식에서의 ()와 혼동될 가능성이 있어 주의가 필요
  - r=(1)은 r = 1로 해석
  - 데이터가 하나인 tuple은 r=(1,)와 같이 쉼표를 포함

# 함수 인자 전달 ; call by value or call by reference

```
def add(a, b):  
    a = 10 ; b = 20  
    return a + b
```

```
c = 1; d = 2  
add(c, d)  
print(c, d)
```

1, 2 # call by value?

```
def addNew (a, b):  
    a[0] = 10; b[0] = 20  
    return a + b
```

```
c = [1, 2, 3]; d = [4, 5, 6]  
addNew(c, d)  
print(c, d)
```

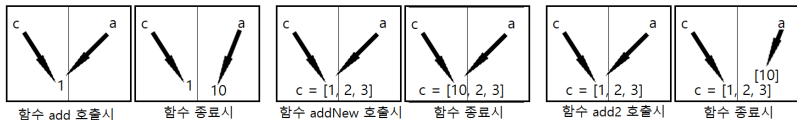
[10, 2, 3] [20, 5, 6]  
# call by reference?

- **mutable**한 객체 (list, [1,2,3])가 전달되면 함수 내부에서 값이 변경될 수 있고, **immutable** 객체 (int, 1)는 변경되지 않는다.

```
def add2 (a, b):
    a = [10]; b = [20]
    return a + b
```

```
c = [1, 2, 3]; d = [4, 5, 6]
add2(c, d)    # expect [10, 20]
print(c, d)
```

[1, 2, 3] [4, 5, 6] # mutable but no change Why?



- Python 방식: call by object reference 또는 call by sharing

# 함수 인수 전달과 객체 복사

```
>>> a = 1
>>> b = a
>>> b = 10
>>> a
1
```

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

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

- 함수의 인수 전달은 인수를 복사한 것과 같은 효과

# 함수의 인수 (arguments)

```
def incr(a, step = 1):    # (step = 1, a) not allowed
    return a + step      # 기본값이 있는 변수는 뒤 부분으로
b = 1; b = incr(b)       # b = 2
```

```
def area(height, weight):
    return height * weight
```

```
area(weight = 10, height=20)  # keyword에 의한 전달 OK
area(20, weight = 10)         # OK
area(weight = 10, 20)         # error
```

- keyword 이후는 순서에 의한 일치 불가능

- 가변 인수: 고정되지 않은 수의 인자, **tuple**으로 받는다.

```
def varg(a, *arg):    # *변수이름 형식으로  
    print(a, arg)     # 인수 마지막에 하나만 허용
```

```
varg(1)
```

```
1 ( )
```

```
varg(1, 2)
```

```
1 (2,)
```

```
varg(1, 2, 3)
```

```
1 (2, 3)
```



```
def printf(format, *args):  
    print(format % args)
```

```
printf("I've spent %d days and %d nights", 6, 5)  
I've spent 6 days and 5 nights
```

```
print("format문" % tuple)  
print("%3d %s %0.2f %3.1f" % (1,"abc",3.141, 3.141))  
1 abc 3.14 3.1
```

- 정의되지 않은 keyword 인수는 dictionary로 처리

```
def volumn(width, height, **kw):  
    print(width, height)  
    print(kw)  
  
volumn(width = 10, height = 5, depth = 2, dim = 3)  
10 5  
{'depth': 2, 'dim': 3}
```

- keyword 인수는 함수 인수 목록에 맨 마지막 부분에 위치하여야 한다.

```
def foo(a, b, *arg, **kw):  
    print(a, b)  
    print(args)  
    print(kw)
```

```
foo(1,2,3,4,c = 5, d = 6)  
1 2  
(3, 4)  
{'c': 5, 'd': 6}
```

- 튜플에 의한 전달: \*tuple

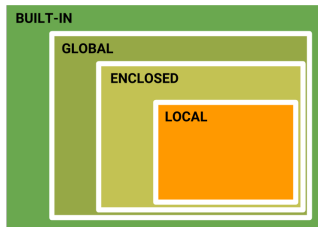
```
def foo(a, b, c)  
    print(a, b, c)  
args = (1, 2, 3)  
foo(*args)  
1 2 3
```

- 사전에 의한 전달: \*\*dict

```
def foo(a, b, c)  
    print(a, b, c)  
args = {'a':1, 'b':2, 'c':3}  
foo(**args)  
1 2 3
```

# 유효 범위: 변수 객체가 유효한 범위

- 변수가 저장되는 이름공간(name space)는 변수가 어디에서 정의되는지에 따라 결정
- Python의 이름 공간 규칙: LEGB 규칙
  - L: Local, 함수 내에서 정의된 변수 (지역변수)
  - E: Enclosing Function Local, 함수 내의 함수 영역
  - G: Global, 함수 영역에 포함되지 않은 모듈 영역
  - B: Built-in 영역



- 변수의 이름은 항상 안쪽에서 바깥쪽으로 찾는다.

```

x = 10; y = 15                                # global
def foo( ):
    x = 20                                     # foo()의 local, bar()의 E
    def bar():
        a = 30                                # bar()의 local
        print(a, x, y)

    bar()                                     # 30 20 15
    print(a, x, y)

foo()
30 20 15
some error messages                            # a: local

```

- **local** 변수: **block** 내에서만 존재

- 동일한 이름이 있으면 안쪽의 이름 공간에서 찾는다

```
>>> abs                                # built-in(내장) 함수
<function abs(x, /)>
>>> abs = 10                           # global 변수
>>> abs(-5)                             # abs 함수 call?
TypeError: 'int' object is not callable
>>> del abs                             # global 변수 삭제
>>> abs(-5)                             # built-in 함수 call
5
```

# 일급 함수

1. 변수나 자료구조에 저장할 수 있다.
2. 다른 함수에 인수로 전달 가능
3. 함수의 반환 값으로 함수가 전달 가능

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

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

```
>>> addition = add # 함수를 변수에 저장 (1)
```

```
>>> def foo(add, a, b): # 함수를 인수로 전달 (2)
    return add(a, b)
```

```
def add_sub(type):
```

```
    if type == 'add': return add # return a function (3)
```

```
    else: return sub
```

```
whatToDo = add_sub('add')
```

```
whatToDo(1, 2)
```

```
3
```

# 재귀함수

- 함수에서 자기 자신 호출 가능

·

$$n! = \begin{cases} 1 & \text{if } n = 1 \\ n * (n-1)! & \text{else} \end{cases}$$

```
def factorial(n):  
    if n == 1: return 1  
    else: return n * factorial(n-1)
```

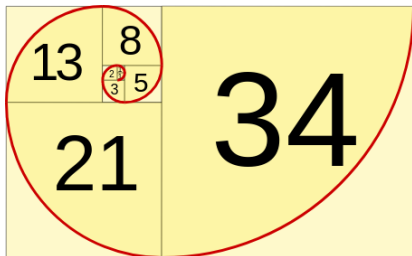
```
print(factorial(5))
```

```
120
```

# Fibonacci 수열 : 숙제

· 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

$$fib(n) = \begin{cases} fib(n-1) + fib(n-2), & n = 3, 4, \dots \\ 1, & n = 1, 2 \end{cases}$$





# Python

---

파일 입출력

# 텍스트 파일 I/O

1. `open()` 함수로 파일 객체를 얻는다.
2. 얻어진 파일 객체에서 데이터를 읽거나 쓴다.
3. `close()` 함수로 파일 객체의 사용을 종료

```
>>> f = open('/home/seung/test.tex', 'w') # write mode
>>> f.write(s)                            # s: 문자열
294                                       # 저장한 문자수
>>> f.close()                             # 파일 close
```

- 인코딩 지정

```
>>> f = open('../seung/test.tex', 'w', encoding = 'utf-8')
>>> f = open('/home/seung/test.tex', 'r') # read mode
>>> s = f.read()                         # 파일 전체 읽기
>>> f.close()                            # 파일 close
```

# 줄단위 파일 I/O

- 줄단위로 파일에 출력

```
>>> lines = ['first\n', 'second\n']
>>> f = open('../seung/test.tex', 'w') # write mode
>>> f.writelines(lines)
>>> f.close()
```

- if '\n'이 없으면 (lines = ['first', 'second'])

```
>>> f.write('\n'.join(lines))
```

- 줄단위로 파일로 부터 입력

```
>>> f.readline() # 한번에 한 줄 씩
>>> f.readlines() # 줄단위로 읽어 list에 저장
>>> with open('a.txt') as f: # alternative
    for line in f:
        print(line, end = ' ')
```

# Python

---

## Object-Oriented Programming

# 객체 지향 프로그래밍 (OOP)

- OOP는 컴퓨터 프로그래밍의 패러다임 중 하나
  - 컴퓨터 프로그램을 명령어의 목록으로 보는 시각에서 벗어나 여러 개의 독립된 단위, 즉 “객체(object)”들의 모임으로 파악
  - 각각의 객체는 메시지를 주고 받으며 데이터를 처리
  - 프로그램을 유연하고 변경이 용이하게 만들기 때문에 대규모 소프트웨어 개발에 많이 사용
  - 또한 프로그래밍을 더 배우기 쉽게 하고 소프트웨어 개발과 보수를 간편하게 하며, 보다 직관적인 코드 분석을 가능하게 하는 장점이 있다.
- OOP의 특징은 기본적으로 자료 추상화, 상속, 다형 개념, 동적 바인딩 등이 있으며 추가적으로 다중 상속 등의 특징이 존재

- 자료 추상화: 불필요한 정보는 숨기고 중요한 정보만을 표현함으로써 프로그램을 간단히 만드는 것
  - 자료 추상화를 위한 기본단위는 클래스(class)
  - 클래스를 통해 자료 표현과 연산을 캡슐화하여 자료형의 정보를 은닉
  - 클래스의 인스턴스를 객체, 정의된 연산을 메소드(함수), 메소드의 호출을 생성자로 지칭
- 상속: 새로운 클래스가 기존의 클래스의 자료와 연산을 이용할 수 있게 하는 기능
  - 상속받은 하위 클래스를 이용해 프로그램의 요구에 맞추어 클래스를 수정하고, 클래스 간의 종속 관계를 형성함으로써 객체를 조직화
- 다중 상속: 2개 이상의 클래스로부터 상속받을 수 있는 기능
  - 여러 클래스들의 기능이 동시에 필요할 때 용이(JAVA는 지원하지 않음)
- 다형성 개념: 어떤 한 요소에 여러 개념을 넣어 놓는 것
  - 일반적으로 오버라이딩(같은 이름의 메소드가 여러 클래스에서 다른 기능을 하는 것)이나 오버로딩(같은 이름의 메소드가 인자의 개수나 자료형에 따라서 다른 기능을 하는 것)을 의미
- 동적 바인딩: 프로그램의 한 개체나 기호를 실행 과정에 여러 속성이나 연산에 바인딩함으로써 다형 개념을 실현

# 클래스

- 객체 지향 프로그래밍의 기본단위인 **class**는 다음과 같이 정의

```
class MyClass:                # class 클래스_이름:
    pass                       # 빈 클래스
```

- 클래스 이름은 PEP 8 Coding Convention에 따라 각 단어의 첫문자를 대문자로 하는 **CapWords** 방식으로 명명.
- MyClass**는 클래스 멤버를 정의하지 않은 빈 클래스로서 클래스 정의내의 **pass**는 빈 동작 또는 아직 구현되지 않았음을 의미

# Class member

```
class Rectangle:
    count = 0                                # class variable

    def __init__(self, width, height): # 초기자(initializer)
        self.width = width             # self.*:
        self.height = height           # instance variable
        Rectangle.count += 1

    def calcArea(self):                  # instance method
        area = self.width * self.height
        return area

>>> rect = Rectangle(10, 20)           # rect(instance 객체) 생성
>>> rect.count                         # class variable 호출
1
>>> rect.calcArea()                   # instance method 호출
200
```



- 클래스는 클래스 멤버(class variable and class method 또는 static method), 인스턴스 멤버(instance variable and instance method), 초기자(initializer), 소멸자(destructor) 등의 멤버로 구성
- 클래스 멤버는 클래스 이름공간에서 생성되며, 모든 인스턴스에서 공유되지만 유일한 값을 갖는다

```
>>> rect2 = Rectangle(100,200)    # 새로운 인스턴스 생성
>>> rect.count, rect2.count, Rectangle.count
(2, 2, 2)
```

- 인스턴스멤버는 인스턴스 객체의 이름공간에서 생성되고 인스턴스 객체내에서만 참조 가능

```
>>> Rectangle.calcArea()
TypeError: calcArea() missing 1 ...
```

- 이들 멤버는 크게 데이터를 나타내는 필드(field or variable)와 행위를 표현하는 메서드(method)로 구분
- Python에서는 field와 method를 객체의 attribute라고 명명
- 다른 OOP 언어와는 달리 새로운 attribute를 동적으로 추가할 수 있다.

# Methods

- 메서드는 클래스의 행위를 표현하는 것(클래스 내의 함수)
- 메서드는 인스턴스 메서드(instance method), 정적 메서드(static method), 클래스 메서드(class method)로 구분
- 인스턴스 메서드
  - 가장 흔히 쓰이며, 인스턴스 변수에 액세스할 수 있도록 메서드의 첫번째 파라미터에 항상 객체 자신을 의미하는 "self"라는 파라미터를 갖는다.
  - 여러 파라미터를 가질 수 있지만, 첫번째 파라미터는 항상 self 를 갖는다
- 정적 메서드
  - 인스턴스 메서드가 객체의 인스턴스 필드를 self를 통해 액세스할 수 있는 반면, 정적 메서드는 이러한 self 파라미터를 갖지 않고 인스턴스 변수에 액세스할 수 없다.
  - 정적 메서드는 보통 객체 필드와 독립적이지만 로직상 클래스내에 포함되는 메서드에 사용된다.
  - 정적 메서드는 메서드 앞에 @staticmethod 라는 Decorator를 표시하여 해당 메서드가 정적 메서드임을 표시한다.
  - 인스턴스 생성없이도 사용가능하다.

- 클래스 메서드
  - 클래스 메서드는 메서드 앞에 `@classmethod` 라는 Decorator를 표시하여 해당 메서드가 클래스 메서드임을 표시한다.
  - 클래스 메서드는 정적 메서드와 비슷한데, 객체 인스턴스를 의미하는 `self` 대신 `cls` 라는 클래스를 의미하는 파라미터를 전달받는다. 정적 메서드는 이러한 `cls` 파라미터를 전달받지 않는다.
  - 이렇게 전달받은 `cls` 파라미터를 통해 클래스 변수 등을 액세스할 수 있다.
- 일반적으로 인스턴스 데이터를 액세스 할 필요가 없는 경우 클래스 메서드나 정적 메서드를 사용하는데, 이때 보통 클래스 변수를 액세스할 필요가 있을 때는 클래스 메서드를, 이를 액세스할 필요가 없을 때는 정적 메서드를 사용한다.
- `Initializer` 이외에도 객체가 소멸될 때(`Garbage Collection`될 때) 실행되는 소멸자(`__del__`) 메서드, 두 개의 객체를 (+ 기호로) 더하는 `__add__` 메서드, 두 개의 객체를 (- 기호로) 빼는 `__sub__` 메서드, 두 개의 객체를 비교하는 `__cmp__` 메서드, 문자열로 객체를 표현할 때 사용하는 `__str__` 메서드 등 많은 특별한 용도의 메서드들이 있다.

# Example

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

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

    def calcArea(self): # 인스턴스 메서드
        return self.width * self.height

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

    @classmethod # 클래스 메서드
    def printCount(cls): print(cls.count)

    def __add__(self, other):
        obj = Rectangle(self.width + other.width, \
                        self.height + other.height)
        return obj
```

# 테스트

```
square = Rectangle.isSquare(5,5) # 객체 생성없이 static method 사용  
print(square)    # True
```

```
rect1 = Rectangle(5, 5)    # 1st instance 생성  
rect2 = Rectangle(2, 5)    # 2nd instance 생성  
rect1.printCount()         # 2  
r3 = rect1 + rect2         # __add__()가 호출됨  
print(r3.calcArea())       # 70
```

# 객체의 생성과 사용

- 클래스를 사용하기 위해서는 먼저 클래스로부터 객체 (Object)를 생성

```
r = Rectangle(2, 3)    # 객체 생성
```

- 만약 `__init__()` 함수가 있고, 그곳에 입력 파라미터들이 지정되어 있다면, “클래스명(입력파라미터들)”과 같이 파라미터를 괄호 안에 전달한다.
- 클래스로부터 생성된 객체 (Object)로부터 클래스 멤버들을 호출하거나 액세스할 수 있다.

```
area = r.calcArea()    # 메서드 호출
```

```
r.width = 10           # 인스턴스 변수 액세스
```

- 인스턴스 메서드는 “객체.메서드명()”, 인스턴스 변수는 “객체.인스턴스변수”으로 표현되며, 값을 읽거나 변경하는 일이 가능하다.

- 특히 클래스 변수를 액세스할 때, “클래스명.클래스변수명” 혹은 “객체명.클래스변수명”을 둘 다 허용
- **Rectangle.count** 혹은 **r.count**은 모두 클래스 변수 **count**를 액세스하는 경우로서 이 케이스에는 동일한 값을 출력
- 한 객체의 **attribute**에 값이 할당되면 (**r.count = 10**), 해당 객체에 이미 동일한 **attribute**가 있는지 체크해서 있으면 새 값으로 치환.
- 만약 그 **attribute**가 없으면 객체에 새로운 **attribute**를 생성하고 값을 할당한다. 즉, **r.count = 10**의 경우 클래스 변수인 **count**를 사용하는 것이 아니라 새로 그 객체에 추가된 인스턴스 변수를 사용하게 되므로 클래스 변수값은 변경되지 않는다.
- 한 객체의 **attribute**를 읽을 경우에는 먼저 그 객체에서 **attribute**를 찾아보고, 없으면 그 객체의 소속 클래스에서 찾고, 다시 없으며 상위 **Base** 클래스에서 찾고, 그래도 없으면 에러를 발생시킨다.
- 위 예제에서 클래스 변수값이 출력된 이유는 값을 할당하지 않고 읽기만 했기 때문에, **r** 객체에 새 인스턴스 변수를 생성하지 않게 되었고, 따라서 객체의 **attribute**가 없어서 클래스의 **attribute**를 찾았기 때문이다.
- 혼돈 방지를 위해 클래스 변수를 액세스할 때는 클래스명을 사용

# 클래스 상속과 다형성

- 파이썬은 객체지향 프로그래밍의 상속(Inheritance)을 지원
- 클래스를 상속 받기 위해서는 파생클래스(자식클래스)에서 클래스명 뒤에 베이스클래스(부모클래스) 이름을 괄호와 함께 넣어 주면 된다.

```
class Animal:
    def __init__(self, name):
        self.name = name
    def move(self):
        print("move")
    def speak(self):
        pass
```

```
class Dog (Animal): # Dog 클래스는 Animal 클래스로부터 파생
    def speak(self):
        print("bark")
class Duck (Animal): # Duck 클래스: Animal의 파생클래스
    def speak(self):
        print("quack")
```

- 복수의 부모클래스로부터 상속(multiple Inheritance)를 지원



- 파생클래스는 베이스클래스의 멤버들을 호출하거나 사용할 수 있으며, 물론 파생클래스 자신의 멤버들을 사용할 수 있다.

```
dog = Dog("doggy") # 부모클래스의 생성자
n = dog.name       # 부모클래스의 인스턴스변수
dog.move()         # 부모클래스의 메서드
dog.speak()        # 파생클래스의 멤버
```

- 객체지향 프로그래밍의 다형성(Polymorphism)을 또한 지원

```
animals = [Dog('doggy'), Duck('duck')]
```

```
for a in animals:
    a.speak()
```

- `animals` 라는 리스트에 `Dog` 객체와 `Duck` 객체를 넣고 이들의 `speak()` 메서드를 호출한 예이다. 코드 실행 결과를 보면 객체의 타입에 따라 서로 다른 `speak()` 메서드가 호출됨을 알 수 있다.

# Python

---

## Exercise

# 표본평균과 표본분산

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i, \quad s^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2$$

```
def summary(x):  
    n = len(x)  
    xbar = 0 ;    s2 = 0  
  
    for i in range(n):  
        xbar = xbar + x[i]  
  
    xbar = xbar/n  
  
    for i in range(n):  
        s2 = s2 + (x[i] - xbar)**2.  
  
    s2 = s2/(n-1)  
    return(xbar, s2)  
  
import random  
x = [random.random() for _ in range(100)]
```

# Problems

- 같은 loop가 두 번 반복

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i, \quad s^2 = \frac{1}{n-1} \left( \sum_{i=1}^n x_i^2 - n\bar{x}^2 \right)$$

- $x^2 = x * x, x ** 2, x * * 2$ .
- return 문

```
def summary(x):  
    n = len(x) ;    xbar = s2 = 0  
  
    for i in range(n):  
        xbar = xbar + x[i]  
        s2 = s2 + x[i] * x[i]  
  
    xbar = xbar/n  
    s2 = (s2 - n * xbar * xbar)/(n-1)  
  
    return {"Mean": xbar, "Variance":s2}
```

# Class example

```
class Univariate:
    def __init__(self, x):    # 초기자 (initializer)
        self.x = x

    def summary(self):
        n = len(self.x)
        xbar = 0 ; s2 = 0

        for i in range(n):
            xbar = xbar + self.x[i]
            s2 = s2 + self.x[i] * self.x[i]

        xbar = xbar/n
        s2 = (s2 - n * xbar * xbar)/(n-1)
        return {"Mean": xbar, "Variance":s2}
```

```

import numpy as np          #  $N(170, 5^2)$  개 100
x = np.random.normal(loc = 170., scale=5.0, size=100)

aClass = Univariate(x)      # an Univariate class 생성
print(aClass.summary())     # Univariate 의 class 멤버의 summary call

{'Mean': 169.99287982374085, 'Variance': 29.64058879217735}

```

# 정적메소드

```
class Univariate:
    def __init__(self, x):          # 초기자 (initializer)
        self.x = x

    @staticmethod                  # static method
    def summary(x):
        n = len(x)
        xbar = 0 ; s2 = 0

        for i in range(n):
            xbar = xbar + x[i]
            s2 = s2 + x[i] * x[i]

        xbar = xbar/n
        s2 = (s2 - n * xbar * xbar)/(n-1)
    return {"Mean": xbar, "Variance":s2}
```

```
import numpy as np
x = np.random.normal(loc = 170., scale=5.0, size=100)

print(Univariate.summary(x)) # 객체생성없이 summary call
{'Mean': 170.3175732413069, 'Variance': 22.135387833291343}
```



# 표본분위수

- Q1: 0.25 분위수, median: 0.5 분위수, Q3: 0.75 분위수
- q-분위수  $X_{(nq)} = (1 - g)X_{([nq])} + gX_{([nq]+1)}$ ,  $g = nq - [nq]$   
e.g.,  $n = 10$ ,  $q = 0.25$ , then  
 $nq = 2.5$ ,  $[nq] = 2$ ,  $g = 0.5$ ,  $X_{(nq)} = 0.5X_{(2)} + 0.5X_{(3)}$
- 순서통계량: 데이터를 크기 순으로 나열
- 24, 45, 23, 21, 37: 원래 데이터
- i* step *i*에서 부터 *n* 까지 값 중, 가장 작은 값이 있는 배열 첨자 *k* search,  
 $i = 1, k = 4$   
 $x_1$  와  $x_4$  값을 switch  
repeat the step until  $i = n - 1$

```

def sort(x):
    n = len(x)
    for i in range(n-1):
        k = i
        for j in range(i+1, n):
            if x[k] > x[j]: k = j

        x[k], x[i] = x[i], x[k]

    return x

import random
x = [random.random() for _ in range(5)]
print(x)
[0.8623, 0.9836, 0.5952, 0.0771, 0.6885]

print(sort(x))
[0.0771, 0.5952, 0.6885, 0.8623, 0.9836]

def quantile(x, q):
    n = len(x)
    nq = n * q
    t = int(nq)
    g = nq - t
    return (1-g) * x[t-1] + g * x[t]  # X[0], X[1], ...

```

```

class Univariate:
    def __init__(self, x):    # 초기자 (initializer)
        self.x = x

    @staticmethod             # static method
    def sort(x):
        n = len(x)
        for i in range(n-1):
            k = i
            for j in range(i+1, n):
                if x[k] > x[j]: k = j

            x[k], x[i] = x[i], x[k]
        return x

    @staticmethod             # static method
    def quantile(x, q):
        n = len(x)
        nq = n * q
        t = int(nq)
        g = nq - t
        return (1-g) * x[t-1] + g * x[t]

```

```

@staticmethod                                # static method
def summary(x):
    n = len(x)
    xbar = 0; s2 = 0

    for i in range(n):
        xbar = xbar + x[i]
        s2 = s2 + x[i] * x[i]

    xbar = xbar/n
    s2 = (s2 - n * xbar * xbar)/(n-1)

    x = Univariate.sort(x)
    median = Univariate.quantile(x, 0.5)
    Q1 = Univariate.quantile(x, 0.25)
    Q3 = Univariate.quantile(x, 0.75)

    return {"Mean":xbar, "Var":s2, "Q1":Q1, \
            "Median":median, "Q3":Q3}

import numpy as np
x = np.random.normal(loc = 170., scale=5.0, size=100)

print(Univariate.summary(x))
{'Mean': 170.09, 'Var': 19.48, 'Q1': 167.33, 'Median': ...}

```

# Histogram

- $[0, 100)$ 를 10개의 등 구간으로 나누었을 때, 56은 몇 번째 구간?

```
if (x < 10) k = 1
elif (x >= 10) & (x < 20)) k = 2
....
```

- We can get the answer 6, directly. How?
- 구간수:  $kkh$ , 구간의 넓이  $D = (\text{최대값} - \text{최소값})/kkh$
- $i$ -번째 구간 =  $[\text{최소값} + (i-1)D, \text{최소값} + i*D)$   
ie., if  $x$ 가  $i$ -번째 구간에 속하면  
 $\text{최소값} + (i-1)D \leq x < \text{최소값} + i * D$ 를 만족
- $i \leq (x - \text{최소값})/D + 1 < i + 1$ 를 만족
- compute  $i = \text{int}((x - \text{최소값})/D) + 1$

# Homework

```
def(x, kkh) # x: 데이터 , kkh 구간 수
  nobs: kkh의 원소를 갖는 벡터 생성하고 각 원소를 0으로 초기화
  xmin, xmax: 최소값과 최대값(sort 함수를 이용)
  D = (xmax - xmin)/kkh

  x_i, i = 0, 1, ... (n-1)에 대해
    k = int((x_i - xmin)/D)
    nobs[k] = nobs[k] + 1

  return nobs
```

# 정규확률지

- A graphical method for testing the normality.
- ordered observations:  $x_{(1)}, x_{(2)}, \dots, x_{(n)}$
- Order statistics from  $N(0, 1)$ :  $z_{(1)}, z_{(2)}, \dots, z_{(n)}$
- rankit:  $m_i = E(z_{(i)}), i = 1, 2, \dots, n$
- $m_i \approx \Phi^{-1} \left( \frac{i-3/8}{n+0.25} \right)$
- 정규확률지: plot  $(m_i, x_{(i)}), i = 1, 2, \dots, n$
- Power transformation:

$$\text{power}(x, \lambda) = \begin{cases} \log(x) & \text{if } \lambda = 0 \\ x^\lambda & \text{if } \lambda \neq 0 \end{cases}$$

# qqplot

```
import math
from scipy.stats import norm
from scipy.stats import chi2
from matplotlib import pyplot as plt

class NormQuantPlot:
    def __init__(self, x, lambdax = 1):
        self.x = x
        self.lambdax = lambdax

    def plot(self):
        y = self.x
        if self.lambdax != 1 :
            y = NormQuantPlot.powerTransform(self.x, self.lambdax)

        NormQuantPlot.qqplot(y, titlestr = 'Normal_Probability_Plot,\n'
                                         + r'$\lambda=\n$' + str(self.lambdax))

    @staticmethod
    def powerTransform(x, lambdax):
        if lambdax == 0:
            return [ math.log(x[i]) for i in range(len(x))]
        else:
            return [ pow(x[i], lambdax) for i in range(len(x))]
```



```

@staticmethod
def qqplot(y, titlestr = 'Normal_Probability_Plot', isShow = True):
    n = len(y)
    y = NormQuantPlot.sortx(y)
    rankits = [norm.ppf(((i+1)-3./8.)/(n+0.25)) for i in range(n)]
    plt.plot(rankits, y, 'ro')    # red circle markers
    plt.title(titlestr)
    plt.xlabel("rankit")
    plt.ylabel("Sample_Quantiles")
    plt.grid(True)
    if isShow: plt.show()

```

```

@staticmethod
def sortx(x):
    n = len(x)
    for i in range(n-1):
        k = i
        for j in range(i+1, n):
            if x[k] > x[j]: k = j
        x[k], x[i] = x[i], x[k]
    return x

```

```

x = [chi2.rvs(df=2) for _ in range(100)]
NormQuantPlot.qqplot(x)    # qqplot without generating NormQuantPlot object
qq = NormQuantPlot(x)      # Generating NormQuantPlot object
qq.plot()                  # plot original data
qq.lambdax = 0              # plot the log transformed data
qq.plot()

```

# UnivariatePlot: NormQuantPlot 상속

```
import math
from scipy.stats import norm
from scipy.stats import chi2
from matplotlib import pyplot as plt

class NormQuantPlot:
    ...
class UnivariatePlot (NormQuantPlot):
    def __init__(self, x):
        self.x = x

    def plot(self, iType = 2, nbin = 0):
        if iType == "qqplot":
            UnivariatePlot.qqplot(self.x)
        elif iType == "hist":
            UnivariatePlot.hist(self.x, nbin = nbin)
        else:
            plt.figure(1)
            plt.subplot(121)
            UnivariatePlot.qqplot(self.x, isShow = False)
            plt.subplot(122)
            UnivariatePlot.hist(self.x, nbin, isShow = False)
            plt.show()
        return
```

```

@staticmethod
def hist(y, nbins = 0, isShow = True):
    if nbins == 0:
        nbins = int(1 + math.log2(len(y)))

    titlestr = 'Histogram, nbins=' + str(nbins)
    histo = plt.hist(y, nbins)
    plt.title(titlestr)
    plt.ylabel('Probability')
    plt.grid(True)
    if isShow: plt.show()

    return histo

x = [norm.rvs()*5 + 170. for _ in range(300)]
UnivariatePlot.qqplot(x)           # qqplot without generating object
UnivariatePlot.hist(x)             # Histogram without generating object
uniplot = UnivariatePlot(x)        # Generating UnivariatePlot object
uniplot.plot("qqplot")              # qqplot of the object
uniplot.plot("hist")                # Histogram of the object
uniplot.plot("hist", nbins = 30)    # Histogram of the object with nbins = 30
uniplot.plot()                      # Both qqplot and hist with default nbins
uniplot.plot(nbins = 15)            # Both qqplot and hist with nbins = 15

```

# Univariate: UnivariatePlot 상속

```
import math
from scipy.stats import norm
from scipy.stats import chi2
from matplotlib import pyplot as plt
class NormQuantPlot:
    ...
class UnivariatePlot (NormQuantPlot):
    ...
class Univariate (UnivariatePlot):
    def __init__(self, x):
        self.x = x

    def summary(self):
        n = len(self.x)
        xbar = 0. : s2 = 0.

        for i in range(n):
            xbar = xbar + self.x[i]
            s2 = s2 + self.x[i] * self.x[i]

        xbar = xbar/n
        s2 = (s2 - n * xbar * xbar)/(n-1)
        stdDev = math.sqrt(s2)
```

```

y = Univariate.sortx(self.x)
median = Univariate.quantile(y, 0.5)
Q1 = Univariate.quantile(y, 0.25)
Q3 = Univariate.quantile(y, 0.75)
d = {"N":n, "Mean": xbar, "Variance":s2, "STD_DEV":stdDev, \
     "Minimum":y[0], "Maximum":y[-1], "25%_Q1":Q1, \
     "Median":median, "75%_Q3":Q3, "Range":y[-1] - y[0] }
keys = list(d.keys())
values = list(d.values())
print("\t\t\tDescriptive Statistics\n")
print(' \t{0:8s}\t{1:10d}\t{2:8s}\t{3:10.3f}'. \
      format("N",n,"Mean",xbar))

for i in range(1,5):
    j = 2 * i
    print("\t{0:8s}\t{1:10.3f}\t{2:8s}\t{3:10.3f}".format\
          (keys[j], values[j], keys[j+1], values[j+1]))

return d

def plot(self, nbins = 0):
    plt.figure(1)
    plt.subplot(121)
    Univariate.qqplot(self.x, isShow = False)
    plt.subplot(122)
    Univariate.hist(self.x, nbins, isShow = False)
    plt.show()
return

```

```

@staticmethod
def quantile(x, q):
    n = len(x)
    nq = n * q
    t = int(nq)
    g = nq - t
    return (1-g) * x[t-1] + g * x[t]

```

```

x = [norm.rvs()*5 + 170. for _ in range(300)]
Univariate.qqplot(x)
uni = Univariate(x)
uni.summary()
uni.plot()
uni.plot(nbin = 30)

```

# Python

---

모듈과 패키지

# 모듈 (module)

- 모듈은 서로 연관된 작업을 하는 코드들의 모임
  - 표준 모듈: 파이썬 패키지에 포함된 모듈
  - 사용자 생성 모듈: 사용자 작성 모듈
  - Third party 모듈: 회사 등이 만들어서 제공되는 모듈
- 모듈의 생성
  - 필요한 변수나 함수를 정의한 파이썬 프로그램을 작성, 확장자는 `py`  
# File: mymodule.py  
  
myPi = 3.141592  
def add(a, b):  
 return(a + b)
  - mymodule.py라는 이름으로 저장



```
>>> import mymodule      # mymodule을 현재의 모듈로 호출
>>> dir(mymodule)        # mymodule에 정의된 것들
['__builtins__', ... , 'add', 'myPi']
>>> mymodule.myPi        # mymodule에 정의된 myPi 호출
3.141592
```

`mymodule.myPi`: 이름공간.속성:

- 이름공간이 주어지지 않은 속성은 **LEGB** 규칙에 따라 탐색
  - 모듈 탐색경로
    - `import`에 의해 가져 온 모듈은 특별히 지정된 폴더에서 찾는다.
- ```
>>> import sys
>>> sys.path
['/home/seung',
 '/opt/anaconda3/bin',
 '/opt/anaconda3/lib/python37.zip',
 ...,
 '/home/seung/.ipython']
```
- 위의 폴더에서 모듈을 찾지 못하면 `ImportError`가 발생
  - `>>> sys.path.append('c:\\myfolder')` # 탐색 경로 추가

# 절대 가져오기

절대 가져오기: `sys.path` 변수에 정해진 순서대로 폴더를 검색

```
>>> import math # ①
>>> from math import sin, cos, pi # ②
>>> from math import * # ③
>>> import numpy as np # ④
>>> from re import sub as substitute # ⑤
>>> from re import sub as sub1, subn as sub2 # ⑥
>>> from tkinter import (Tk, Frame, Entry) # ⑦
```

- ① 모듈 이름을 가져 왔다. `math.sin(math.pi)`
- ② 모듈의 특정 이름만 현재의 이름공간으로 가져 왔다. `sin(pi)`
- ③ 모듈에 정의된 모든 이름을 가져 왔다. 모듈 수준에서만 사용가능  
(함수내에서는 사용 불가능)
- ④ 모듈의 이름이 너무 길 때 사용(①과 동일)
- ⑤ `sub`를 `substitute`라는 이름으로 사용(④와 동일)
- ⑥ ⑤와 동일
- ⑦ ④와 동일하나 여러 줄에 걸쳐 기술 가능

# 패키지(package)

- 모듈을 모아 놓은 단위: 여러 개의 모듈을 계층적인 몇 개의 디렉토리로 분류하여 저장
- 예제: 음성 관련 패키지 구조

```
Speech/                                     # 패키지의 최상위
  __init__.py
  SignalProcessing/                         # 신호처리 하위 패키지
    __init__.py
    LPC.py
    FilterBank.py
  Recognition/                             # 음성인식 하위 패키지
    __init__.py
    Adaptation/                           # Reconition의 하위 패키지
      __init__.py
      ML.py
```

- `__init__.py`
  - 각 디렉토리에는 `__init__.py`가 반드시 있어야 한다.
  - 패키지를 가져 올 때 실행되는 초기화 스크립트
  - `Speech/__init__.py`

```
__all__ = [ 'SignalProcessing', 'Recognition' ] # (1)
__version__ = '2.3'
from . import Recognition # 상대 가져 오기
from . import SignalProcessing
```

(1) `from Speech import *`으로 가져 올 모듈이나 패키지 이름 지정

# Python

---

파이썬 패키지

# 파이썬 패키지

---

- NumPy

- 파이썬 언어를 기반으로 하는 모든 분석용 솔루션의 핵심
- 다차원 배열과 배열을 대상으로 수학적 연산을 수행하는 많은 함수들을 제공

- SciPy

- 선형대수, 희소행렬, 신호 및 이미지 처리, 최적화, 푸리에 변환 등 다양한 종류의 과학용 계산을 위한 함수 제공
- NumPy의 기능을 보완

- Pandas

- 객체 데이터 구조, 데이터 프레임, **Series** 등으로 서로 다른 데이터 타입으로 구성되는 복잡한 테이블과 시계열 데이터를 처리
- 데이터 처리를 위한 필수 패키지

- **matplotlib**
  - 배열로 부터 고품질의 다양한 도표를 작성하는 함수 제공
  - 대화형으로 도표를 시각화하기 위한 라이브러리 제공
- **Gensim**
  - 대형 문자 집합 분석용 오픈소스 패키지
  - 병렬 분산 온라인 알고리즘 사용
- **H2O**
  - 빅데이터 분석을 위한 오픈소스 프레임워크
  - 파이썬, R, 자바 등으로 사용
  - **Hadoop** 클러스터를 쉽게 사용할 수 있도록 하며, 스케일 업과 스케일 다운을 가능하게 한다.
- **XGBoost**
  - 앙상블을 위한 분산형 라이브러리
  - 파이썬, R, 자바 등을 지원
  - **Hadoop**과 **Spark** 클러스트에서 동작

- Scikit-learn

- 파이썬을 사용하는 데이터 연산의 핵심
- 데이터 전처리, 지도 및 비지도 학습, 모델선택 등에 필요한 모든 기능을 제공
- 본 과목의 핵심 패키지

- TensorFlow

- 딥 러닝 뉴럴 네트워크를 위한 오픈소스 라이브러리
- 여러개의 CPU와 GPU에서 일반 연산을 수행하게 하는 CUDA 확장기능을 사용하여 구동될 수 있다



# Python

---

## Useful Packages

# What are Numpy and Numpy arrays?

- Numpy: creating and manipulating numerical data
- It provides
  - extension package to Python for multi-dimensional arrays
  - closer to hardware (efficiency)
  - designed for scientific computation (convenience)
  - array oriented computing

```
>>> import numpy as np
>>> a = np.array([0, 1, 2 ,3])    # np.array(list)
>>> a
array([0, 1, 2, 3])
```

- Why it is useful: Memory-efficient container that provides fast numerical operations.

```
>>> L = range(1000)
>>> %timeit [i**2 for i in L]
148 µs ± 482 ns per loop (mean ± std. ....)
```

```
>>> a = np.arange(1000)
>>> %timeit a**2          # elementwise calculation
808 ns ± 4.28 ns per loop (mean ± std. ....)
```

- Creating arrays

- 1-D

```
>>> a = np.array([0, 1, 2, 3])
>>> a
array([0, 1, 2, 3])
>>> a.ndim
1
>>> a.shape
(4,)
>>> len(a)
4
```

- 2-D, 3-D, ...

```
>>> b = np.array([[0, 1, 2], [3, 4, 5]]) # 2 x 3 array
>>> b
array([[0, 1, 2],
       [3, 4, 5]])
>>> b.ndim
2
>>> b.shape
(2, 3)
>>> len(b) # returns the size of the first dimension
2
>>> c = np.array([[[1], [2]], [[3], [4]]])
>>> c
array([[[1],
       [2]],
       [[3],
       [4]]])
>>> c.shape
(2, 2, 1)
```

# Functions for creating arrays

- Evenly spaced:

```
>>> a = np.arange(10)          # 0 .. n-1 (!)
```

```
>>> a  
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
>>> b = np.arange(1, 9, 2) # start, end (exclusive), step
```

```
>>> b  
array([1, 3, 5, 7])
```

- By number of points:

```
>>> c = np.linspace(0, 1, 6) # start, end, num-points
```

```
>>> c  
array([0. , 0.2, 0.4, 0.6, 0.8, 1. ])
```

```
>>> d = np.linspace(0, 1, 5, endpoint=False)
```

```
>>> d  
array([0. , 0.2, 0.4, 0.6, 0.8])
```

- Common arrays:

```
>>> a = np.ones((2, 2)) # reminder: (2, 2) is a tuple
```

```
>>> a
array([[ 1.,  1.],
       [ 1.,  1.]])
```

```
>>> b = np.zeros((2, 2))
```

```
>>> b
array([[ 0.,  0.],
       [ 0.,  0.]])
```

```
>>> c = np.eye(2)
```

```
>>> c
array([[ 1.,  0.],
       [ 0.,  1.]])
```

```
>>> d = np.diag(np.array([1, 2, 3, 4]))
```

```
>>> d
array([[1, 0, 0, 0],
       [0, 2, 0, 0],
       [0, 0, 3, 0],
       [0, 0, 0, 4]])
```

# Random numbers

---

```
>>> a = np.random.rand(4)    # uniform in [0, 1]
>>> a
array([ 0.95799151, 0.14222247, 0.08777354, 0.51887998])

>>> b = np.random.randn(4)   # Gaussian
>>> b
array([ 0.37544699, -0.11425369, -0.47616538, 1.7966411])

>>> np.random.seed(1234)     # Setting the random seed
```

# Indexing and slicing

- The items of an array can be accessed and assigned to the same way as other Python sequences (e.g. lists)

```
>>> a = np.arange(10)
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
>>> a[0], a[2], a[-1]
(0, 2, 9)
```

```
>>> a[::-1]
array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
```



- For multidimensional arrays, indexes are tuples of integers:

```
>>> a = np.diag(np.arange(2))
```

```
>>> a
array([[0, 0],
       [0, 1]])
```

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

```
>>> a[1, 1] = 10 # second line, second column
```

```
>>> a
array([[ 0,  0 ],
       [ 0, 10]])
```

```
>>> a[1]           # a[1,:] is better
array([0, 10])
```

- In 2D, the first dimension corresponds to rows, the second to columns: `a[i]`, `a[:,j]`: i-th row and j-th column

# Copies and views

- When modifying the view, the original array is modified as well:

```
>>> a = np.arange(10)
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> b = a[::2]
>>> b
array([0, 2, 4, 6, 8])
>>> np.may_share_memory(a, b)
True
>>> b[0] = 12
>>> b
array([12, 2, 4, 6, 8])
>>> a      # (?)
array([12, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

# Copies and views

---

- To force a copy

```
>>> a = np.arange(10)
>>> c = a[::2].copy() # force a copy
>>> c[0] = 12
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

>>> np.may_share_memory(a, c)
False
```

# Fancy indexing

- Numpy arrays can be indexed with slices, but also with boolean or integer arrays (masks) just like R. This method is called fancy indexing. It creates copies not views.
- Using boolean masks

```
>>> a = np.random.random_integers(0, 20, 7)
>>> a
array([14,  4, 11,  6, 15, 11,  3])

>>> (a % 3 == 0)    # tuple
array([False, False, False,  True,  True, False,  True])

>>> mask = (a % 3 == 0)
>>> extract_from_a = a[mask] # or, a[a%3==0]
>>> extract_from_a    # extract a sub-array with the mask
array([ 6, 15,  3])
```

- Indexing with an array of integers

```
>>> a = np.arange(0, 10, 10)
>>> a
array([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

>>> a[[2, 3, 2, 4, 2]] # [2, 3, 2, 4, 2] is a list
array([2, 3, 2, 4, 2])

>>> a[[9, 7]] = -1    # New values can be assigned
>>> a
array([0, 1, 2, 3, 4, 5, 6, -1, 8, -1])
```

# Numerical operations on arrays

- Elementwise operations:
- All arithmetic operates elementwise

```
>>> a = np.array([1, 2, 3, 4]) ; a + 1  
array([2, 3, 4, 5])
```

```
>>> 2**a  
array([ 2, 4, 8, 16])
```

```
>>> b = np.ones(4) + 1 ; a - b  
array([-1., 0., 1., 2.])
```

```
>>> a * b  
array([ 2., 4., 6., 8.])
```

```
>>> j = np.arange(5) ; 2**(j + 1) - j  
array([ 2, 3, 6, 13, 28])
```

- These operations are of course much faster than if you did them in pure python.

- As R, array multiplication is not matrix multiplication:

```
>>> c = np.ones((2, 2))
```

```
>>> c * c  
array([[1., 1.],  
       [1., 1.]])
```

```
>>> c.dot(c)          # matrix multiplication  
array([[2., 2.],  
       [2., 2.]])
```

- Comparison

```
>>> a = np.array([1, 2, 3, 4])
```

```
>>> b = np.array([4, 2, 2, 4])
```

```
>>> a == b  
array([False,  True, False,  True], dtype=bool)
```

```
>>> a > b  
array([False, False,  True, False], dtype=bool)
```

- Array-wise comparison:

```
>>> a = np.array([1, 2, 3, 4])
>>> b = np.array([4, 2, 2, 4])
>>> c = np.array([1, 2, 3, 4])
>>> np.array_equal(a, b)
False
```

```
>>> np.array_equal(a, c)
True
```

- Logical operations:

```
>>> a = np.array([1, 1, 0, 0], dtype=bool)
>>> b = np.array([1, 0, 1, 0], dtype=bool)
>>> np.logical_or(a, b)
array([ True,  True,  True, False], dtype=bool)
```

```
>>> np.logical_and(a, b)
array([ True, False, False, False], dtype=bool)
```



# Reduction

- Computing sums:

```
>>> x = np.array([1, 2, 3, 4])
>>> np.sum(x)          # x.sum()
10
```

```
>>> x = np.array([[1, 1], [2, 2]])
```

```
>>> x.sum(axis=0)      # columns (first dimension)
array([3, 3])
```

```
>>> x[:, 0].sum(), x[:, 1].sum() # same as above
(3, 3)
```

```
>>> x.sum(axis=1)      # rows (second dimension)
array([2, 4])
```

```
>>> x[0, :].sum(), x[1, :].sum() # same as above
(2, 4)
```

- Same idea in higher dimensions:

```
>>> x = np.random.rand(2, 2, 2)
>>> x.sum(axis=2)[0, 1]
1.14764...
```

```
>>> x[0, 1, :].sum()
1.14764...
```

- Other reductions: works the same way (take axis=)
  - min, max, argmax, argmin etc.

- Statistics:

```
>>> x = np.array([1, 2, 3, 1])
>>> y = np.array([[1, 2, 3], [5, 6, 1]])
>>> x.mean()
1.75

>>> np.median(x)
1.5

>>> np.median(y, axis=-1) # last axis
array([ 2., 5.])

>>> x.std()    ## full population standard dev.
0.82915619758884995
```

# Pandas data-frame

- Reading from a CSV file:
- brain\_size.csv

```
"";"Gender";"FSIQ";"VIQ";"PIQ";"Weight";"Height";"MRI_Count"  
"1";"Female";133;132;124;"118";"64.5";816932  
"2";"Male";140;150;124;"."; "72.5";1001121  
----
```

```
>>> import pandas  
>>> data=pandas.read_csv('brain_size.csv',sep=';',na_values=".")  
>>> data
```

|     | Unnamed: 0 | Gender | FSIQ | VIQ | PIQ | Weight | Height | MRI_Count |
|-----|------------|--------|------|-----|-----|--------|--------|-----------|
| 0   | 1          | Female | 133  | 132 | 124 | 118    | 64.5   | 816932    |
| 1   | 2          | Male   | 140  | 150 | 124 | NaN    | 72.5   | 1001121   |
| 2   | 3          | Male   | 139  | 123 | 150 | 143    | 73.3   | 1038437   |
| 3   | 4          | Male   | 133  | 129 | 128 | 172    | 68.8   | 965353    |
| 4   | 5          | Female | 137  | 132 | 134 | 147    | 65.0   | 951545    |
| ... |            |        |      |     |     |        |        |           |

- data: an pandas.DataFrame object
- missing values: NaN

- Creating from arrays:

```
import pandas as pd
import numpy as np
t = np.linspace(-6, 6, 20)    # -6에서 6까지 등 간격으로 20
sin_t = np.sin(t)
cos_t = np.cos(t)
pd.DataFrame({'t': t, 'sin': sin_t, 'cos': cos_t})
```

|     | t         | sin       | cos      |
|-----|-----------|-----------|----------|
| 0   | -6.000000 | 0.279415  | 0.960170 |
| 1   | -5.368421 | 0.792419  | 0.609977 |
| --- |           |           |          |
| 18  | 5.368421  | -0.792419 | 0.609977 |
| 19  | 6.000000  | -0.279415 | 0.960170 |

- A pandas.DataFrame can also be seen as a dictionary of 1D 'series', eg arrays or lists.

# Manipulating data

- pandas.DataFrame은 R의 dataframe과 유사

```
>>> data.shape      # 40 rows and 8 columns
(40, 3)
```

```
>>> data.columns    # It has columns
Index([u'Unnamed: 0', u'Gender', u'FSIQ', u'VIQ', u'PIQ',
       u'Weight', u'Height', u'MRI_Count'], dtype='object')
```

```
>>> print(data['Gender'])    # Column addressed by name
```

```
0      Female
1       Male
2       Male
3       Male
4      Female
...
```

```
>>> data[data['Gender'] == 'Female']['VIQ'].mean()
109.45
```

- For a quick view on a large dataframe, use its describe method: `panda.DataFrame.describe()`

```
>>> data.describe()
```

|       | Unnamed: 0 | FSIQ       | VIQ        | ... | MRI_Count    |
|-------|------------|------------|------------|-----|--------------|
| count | 40.000000  | 40.000000  | 40.000000  | ... | 4.000000e+01 |
| mean  | 20.500000  | 113.450000 | 112.350000 | ... | 9.087550e+05 |
| std   | 11.690452  | 24.082071  | 23.616107  | ... | 7.228205e+04 |
| min   | 1.000000   | 77.000000  | 71.000000  | ... | 7.906190e+05 |
| 25%   | 10.750000  | 89.750000  | 90.000000  | ... | 8.559185e+05 |
| 50%   | 20.500000  | 116.500000 | 113.000000 | ... | 9.053990e+05 |
| 75%   | 30.250000  | 135.500000 | 129.750000 | ... | 9.500780e+05 |
| max   | 40.000000  | 144.000000 | 150.000000 | ... | 1.079549e+06 |

```
[8 rows x 7 columns]
```

- `groupby`: splitting a dataframe on values of categorical variables:

```
>>> groupby_gender = data.groupby('Gender')
```

- `groupby_gender` is a powerful object that exposes many operations on the resulting group of dataframes.

```
>>> groupby_gender.mean()
```

|        | Unnamed: 0 | FSIQ  | VIQ    | PIQ    | Weight | Height | MRI_Count |
|--------|------------|-------|--------|--------|--------|--------|-----------|
| Gender |            |       |        |        |        |        |           |
| Female | 19.65      | 111.9 | 109.45 | 110.45 | 137.20 | 65.76  | 862654.6  |
| Male   | 21.35      | 115.0 | 115.25 | 111.60 | 166.44 | 71.43  | 954855.4  |

```
>>> for gender, value in groupby_gender['VIQ']:
...     print((gender, value.mean()))
('Female', 109.45)
('Male', 115.25)
```

- Box plots of different columns for each gender

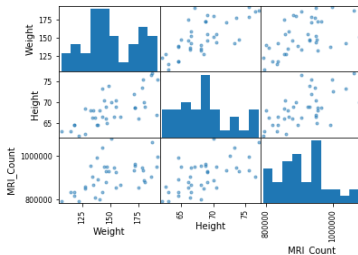
```
>>> groupby_gender.boxplot(column=['FSIQ', 'VIQ', 'PIQ'])
```



# Plotting data

- Pandas comes with some plotting tools (pandas.tools.plotting, using matplotlib behind the scene) to display statistics of the data in dataframes:
- Scatter matrices:

```
>>> from pandas.tools import plotting  
>>> plotting.scatter_matrix(data[['Weight', 'Height', 'MRI_Count']])
```



# Scipy : high-level scientific computing

---

- The scipy package contains various toolboxes dedicated to common issues in scientific computing.
- scipy can be compared to other standard scientific-computing libraries, such as the GSL (GNU Scientific Library for C and C++), or Matlab toolboxes.
- scipy is the core package for scientific routines in Python; it is meant to operate efficiently on numpy arrays, so that numpy and scipy work hand in hand.

# Scipy sub-modules

- scipy is composed of task-specific sub-modules:

---

|                   |                                        |
|-------------------|----------------------------------------|
| scipy.cluster     | Vector quantization / Kmeans           |
| scipy.constants   | Physical and mathematical constants    |
| scipy.fftpack     | Fourier transform                      |
| scipy.integrate   | Integration routines                   |
| scipy.interpolate | Interpolation                          |
| scipy.io Data     | input and output                       |
| scipy.linalg      | Linear algebra routines                |
| scipy.ndimage     | n-dimensional image package            |
| scipy.odr         | Orthogonal distance regression         |
| scipy.optimize    | Optimization                           |
| scipy.signal      | Signal processing                      |
| scipy.sparse      | Sparse matrices                        |
| scipy.spatial     | Spatial data structures and algorithms |
| scipy.special     | Any special mathematical functions     |
| scipy.stats       | Statistics                             |

---

# File input/output: scipy.io

- Loading and saving matlab files:

```
from scipy import io as spio
a = np.ones((3, 3))
spio.savemat('file.mat',{'a': a}) #save mat expects a dic
data = spio.loadmat('file.mat', struct_as_record=True)
data['a']
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
```

- Reading images:

```
from scipy import misc
misc.imread('fname.png')
array(...)

# Matplotlib also has a similar function
import matplotlib.pyplot as plt
plt.imread('fname.png')
array(...)
```

# Linear algebra operations: `scipy.linalg`

- The `scipy.linalg` module provides standard linear algebra operations, relying on an underlying efficient implementation (BLAS, LAPACK).
- The `scipy.linalg.det()` function computes the determinant of a square matrix:

```
>>> from scipy import linalg
>>> arr = np.array([[1, 2], [3, 4]])
>>> linalg.det(arr)
-2.0
>>> arr = np.array([[3, 2], [6, 4]])
>>> linalg.det(arr)
0.0
>>> linalg.det(np.ones((3, 4)))
Traceback (most recent call last):
...
ValueError: expected square matrix
```

- The `scipy.linalg.inv()` function computes the inverse of a square matrix:

```
>>> arr = np.array([[1, 2], [3, 4]])
>>> iarr = linalg.inv(arr)
>>> iarr
array([[ -2. ,  1. ],
       [ 1.5, -0.5]])
>>> np.allclose(np.dot(arr, iarr), np.eye(2))
True
```

# Hypothesis Testing: comparing two groups

- For simple statistical tests, we will use the `scipy.stats`, a sub-module of `scipy`:

```
>>> from scipy import stats
```

- 1-sample t-test: testing the value of a population mean

```
>>> stats.ttest_1samp(data['VIQ'], 0)
Ttest_1sampResult(statistic=30.0880999708, pvalue=1.328919646e-28)
```

- 2-sample t-test: testing for difference across populations

```
>>> female_viq = data[data['Gender'] == 'Female']['VIQ']
>>> male_viq = data[data['Gender'] == 'Male']['VIQ']
>>> stats.ttest_ind(female_viq, male_viq)
Ttest_indResult(statistic=-0.772616172327, pvalue=0.4445287677)
```

- Paired t-test: repeated measurements on the same individuals

```
>>> stats.ttest_rel(data['FSIQ'], data['PIQ'])
Ttest_relResult(statistic=1.7842019405, pvalue=0.082172638183)
```

# Linear models

---

- First, we generate simulated data according to the model:

```
import numpy as np
x = np.linspace(-5, 5, 20)
np.random.seed(1)

# normal distributed noise
y = -5 + 3*x + 4 * np.random.normal(size=x.shape)

# Create a data frame containing all relevant variables
data = pandas.DataFrame({'x': x, 'y': y})
```

- OLS fit:

```
from statsmodels.formula.api import ols
model = ols("y ~ x", data).fit()
print(model.summary())
```



# OLS Regression Results

```

=====
Dep. Variable:          y      R-squared:          0.804
Model:                  OLS    Adj. R-squared:       0.794
Method:                 Least Squares  F-statistic:        74.03
Date:                   Tue, 16 May 2017  Prob (F-statistic): 8.56e-08
Time:                   16:39:44  Log-Likelihood:     -57.988
No. Observations:       20      AIC:                120.0
Df Residuals:           18      BIC:                122.0
Df Model:                1
Covariance Type:        nonrobust
=====

```

# Scikit-learn

---

**Out-of-core learning**

# 코어 메모리 방식

- 코어 메모리 방식
  - 데이터를 행렬 형태로 메모리에 로딩
  - 머신 러닝 알고리즘을 통해 훈련 (learning)
  - 표본추출로 얻은 관측 데이터 (test dataset)를 이용, 해당 알고리즘의 일반화 능력을 테스트
- Problems:
  - 가용 가능한 메모리 (RAM)은 한정: 대략 2GB ~ 8GB이고 대용량의 서버 머신이라고 하더라도 256GB
  - 가상메모리 (virtual memory)를 사용해도 메모리에 load될 수 있는 데이터의 양은 제한적
- 코어 메모리 방식이 상대적으로 처리 속도가 빠르지만 현재의 대용량 데이터의 분석에 적합하지 않는 경우가 많다.

# An example

- pandas dataframe에서 표본평균과 표본분산 계산(without missing and all numeric features)

```
import pandas as pd
import numpy as np

def wrongStat(pdDataFrame):
    varName = pdDataFrame.keys()      # get keys(variable name)
    npArray = pdDataFrame.to_numpy()   # convert to numpy array

    n, k = npArray.shape               # n:# of cases; k:# of features

    xbar = np.zeros(k)                 # initialize for summation
    S2 = np.zeros(k)

    for i in range(n):
        xbar = xbar + npArray[i,:]    # sum

    xbar = xbar / n                    # sample mean
```

```

for i in range(n):
    S2 = S2 + (npArray[i,:] - xbar) ** 2.

S2 = S2/(n-1)                                # sample variance

return(pd.DataFrame({"variable": varName, "xvar": xbar, \
                    "S^2": S2}))

data = pd.read_csv("example.csv", sep=',', \
                    na_values=".", encoding='utf-8')
print(wrongStat(data))

```

|   | variable  | xvar       | S^2          |
|---|-----------|------------|--------------|
| 0 | FSIQ      | 113.450    | 5.799462e+02 |
| 1 | VIQ       | 112.350    | 5.577205e+02 |
| 2 | PIQ       | 111.025    | 5.049481e+02 |
| 3 | MRI_Count | 908755.000 | 5.224695e+09 |

- 계산 상의 문제들

1.  $x^{**2}$ .,  $x^{**2}$ ,  $x * x$ 의 차이:  $x^{**2}$ . inefficient

- $x^{**2}$ .:  $x^2 = \exp(2. \times \log(x))$ 와 같은 방식으로 계산

2. 평균과 S2을 구하기 위해 2번의 loop를 사용하였으나 이를 합칠 수 있어야 한다(n이 커지면 더욱 큰 문제).

```
for i in range(n):
```

```
    xbar = xbar + npArray[i,:]          # sum X  
    S2 = S2 + npArray[i,:] * npArray[i,:] # sum X^2
```

```
xbar = xbar/n                          # sample mean  
S2 = (S2 - n * xbar * xbar)/(n-1)      # sample variance
```

- We already have talked about these problems.
- Main problem: 코어 메모리 방식으로 메모리에 모든 데이터를 load(n이 커지면 계산 불가능, imagine  $n=10e100$ ,  $k=300$ )

# Possible solutions for large scale data

---

1. Increase the available memory
2. Reducing the size of the dataset by picking only a part of the observations(subsampling)
3. Hadoop and Spark와 같은 분산처리 시스템을 이용
4. Out-of-core learning
  - 탑재된 메모리에는 올라 갈 수 없으나 하드 디스크 또는 웹 저장소와 같은 데이터 **storage**에 올라 갈 수 있는 데이터에 대해 적용할 수 있는 방식
  - 데이터 **storage**로 부터 단일 또는 작은 **batch** 단위로 점진적 학습
  - 경사하강법(**gradient descent**)과 같은 점진적 최적화 알고리즘을 이용

# An example of out-of-core learning

```
def outofCoreStat(file):  
    f = open(file, 'r', encoding='utf-8')  
    rdr = csv.reader(f)  
    varName = next(rdr)  # first line: names of feature  
    k = len(varName)  
    xbar = np.zeros(k) ;    S2 = np.zeros(k)  
  
    for n, row in enumerate(rdr):  # enumerate object  
  
        #      change str to float and make numpy array  
  
        xValues = np.array(row, dtype = np.float32)  
        xbar = xbar + xValues          # calculate sum X  
        S2 = S2 + xValues * xValues   # calculate sum X^2  
  
    f.close()  
    n += 1  
    xbar = xbar/n  
    S2 = (S2 - n * xbar * xbar) / (n-1)  
    return(pd.DataFrame({"variable": varName, "xbar": xbar,\n                          "S^2": S2}))
```



```
file = 'example.csv'  
print(outofCoreStat(file))
```

|   | variable  | xbar       | S <sup>2</sup> |
|---|-----------|------------|----------------|
| 0 | FSIQ      | 113.450    | 5.799462e+02   |
| 1 | VIQ       | 112.350    | 5.577205e+02   |
| 2 | PIQ       | 111.025    | 5.049481e+02   |
| 3 | MRI_Count | 908755.000 | 5.224696e+09   |

- Core of data science operations in Python
- Out-of-core 방식의 machine learning method 제공
  - Data processing: `sklearn.preprocessing` and `sklearn.feature_extraction`
  - Model selection and validation: `sklearn.cross_validation`, `sklearn.grid_search`, and `sklearn.metrics`
  - Supervised learning methods: `sklearn.linear_model`
  - Unsupervised learning methods: `sklearn.cluster` and `sklearn.neighbors`
  - etc.

# Scikit-learn

---

## Preparing datasets

# Streaming data from sources

- 데이터 스트림: 데이터를 연속적으로 전송하여 실시간으로 재생하는 일
- 데이터 스트림 예제
  - 환경센서에서 측정하는 온도, 기압, 습도 등
  - GPS 탐지 센서로 기록되는 위치(위도/경도)
  - 인공위성에서 기록되는 이미지 데이터
  - 감시비디오 영상과 음성 기록
  - 웹 트래픽
- UCI Machine Learning Repository
  - UCI(University of California, Irvine)에서 운영하는 머신러닝의 실제 테스트를 위한 데이터의 저장소
  - 다양한 사용 목적을 갖는 475 개의 데이터 세트를 제공

# Out-of-core learning을 위한 스크립트<sup>1</sup>

```
import urllib
import requests, io, os
import numpy as np
import tarfile, zipfile, gzip
def unzip_from_UCI(UCI_url, dest=''):

    # Downloads and unpacks datasets from UCI in zip format

    response = requests.get(UCI_url)
    compressed_file = io.BytesIO(response.content)
    z = zipfile.ZipFile(compressed_file)
    print ('Extracting in %s' % os.getcwd()+os.sep+dest)

    for name in z.namelist():
        if '.csv' in name:
            print ('\tunzipping %s' %name)
            z.extract(name, path=os.getcwd()+os.sep+dest)
```

---

<sup>1</sup>Large Scale Machine Learning with Python 참조

```
def gzip_from_UCI(UCI_url, dest=''):

    # Downloads and unpacks datasets from UCI in gzip format

    response = urllib.request.urlopen(UCI_url)
    compressed_file = io.BytesIO(response.read())
    decompressed_file = gzip.GzipFile(fileobj = compressed_file)
    filename = UCI_url.split('/')[-1][:3]

    with open(os.getcwd()+os.sep+filename, 'wb') as outfile:
        outfile.write(decompressed_file.read())
        print ('File %s decompressed' % filename)
```

- UCI Machine Learning Repository에서 data 파일(zip file)을 다운받아 압축을 풀어 파일로 저장

```
UCI_url = 'https://archive.ics.uci.edu/ml/  
          machine-learning-databases/00275/  
          Bike-Sharing-Dataset.zip'  
unzip_from_UCI(UCI_url, dest='bikesharing')
```

```
UCI_url = 'https://archive.ics.uci.edu/ml/  
          machine-learning-databases/covtype/  
          covtype.data.gz'  
gzip_from_UCI(UCI_url)
```

- requests module: HTTP 요청처리를 위해 사용하는 모듈
  - response = requests.get(url): get 메서드를 호출하여 url에 관련된 정보를 수집
  - response.content: response body as bytes, for non-text requests
- io module: perform stream and buffer operations
  - io.BytesIO(): keep data as bytes in an in-memory buffer
  - io.StringIO(): keep data as string in an in-memory buffer
- zipfile: read and write ZIP archive files
  - zipfile.ZipFile(compressed\_file): zip file 객체 생성
  - .namelist(): zip file 객체내의 파일 이름 리스트
  - .extract(name, path) zip file 객체에서 name file 추출
- os: operating system과 관련된 module
  - os.getcwd(): 현재의 디렉토리를 반환
  - os.sep: 디렉토리 구분자(windows, '\\', unix 계열, '/')



# 파일에서 스트리밍으로 데이터를 가져오기

- Recover the data as a list

```
import os, csv
SEP = ','
local_path = os.getcwd() + os.sep
source = 'bikesharing/hour.csv'
fileName = local_path + source

with open(fileName, 'r') as R:
    iterator = csv.reader(R, delimiter=SEP)
    header = next(iterator)

    for n, row in enumerate(iterator):
        # DATA PROCESSING placeholder"
        # MACHINE LEARNING placeholder"
        pass

    print ('Total_rows: %i ' % (n+1))
    print ('Header: %s ' % ','.join(header))
    print ('Sample_values: %s ' % ','.join(row))
```

- Recover the data as a dictionary

```
import os, csv
SEP = ','
local_path = os.getcwd() + os.sep
source = 'bikesharing/hour.csv'
fileName = local_path + source

with open(fileName, 'r') as R:
    iterator = csv.DictReader(R, delimiter=SEP)
    for n, row in enumerate(iterator):
        # DATA PROCESSING placeholder
        # MACHINE LEARNING placeholder
        pass

print ('Total rows: %i ' % (n+1))
print ('Sample values: %s ' % str(row))

Total rows: 17379
Sample values: OrderedDict([('instant', '17379'), ...])
```

# Using pandas I/O tools

---

- As an alternative to the csv module, we can use pandas' read\_csv function.
- Advantages of using pandas I/O functions
  - Code consistent, that is, you need to redefine the streaming iterator
  - Can access a large number of different formats such as CSV, plain TXT, HDF, JSON, and SQL query for a specific database.
  - The data is streamed into chunks of the desired size as DataFrame data structures
- The iterator is instantiated by specifying a chunk size, that is, the number of rows
- The chunksize parameter is clearly the size of the mini-batch (the chunk retrieved) is strictly connected to your available memory to store and manipulate it in the following preprocessing phase.

```

import pandas as pd
import os
SEP = ','
local_path = os.getcwd() + os.sep
source = 'bikesharing/hour.csv'
fileName = local_path + source
CHUNK_SIZE = 1000

with open(fileName, 'rb') as R:
    iterator = pd.read_csv(R, chunksize=CHUNK_SIZE)
    for n, data_chunk in enumerate(iterator):
        print ('Size of uploaded chunk: %i instances, \
        %i features' %(data_chunk.shape))
        # DATA PROCESSING placeholder
        # MACHINE LEARNING placeholder
        pass

print ('Sample values: \n%s' % str(data_chunk.iloc[0]))

```

# Working with databases

---

- 데이터베이스를 이용하여 대용량 데이터를 스트리밍하면 디스크 공간 및 처리 시간에 유리할 수 있다.
- 파이썬의 표준라이브러리에는 **SQLite3** 데이터베이스 시스템의 처리를 위한 **sqlite3 module**이 포함
- We will cover this topic if time permits.

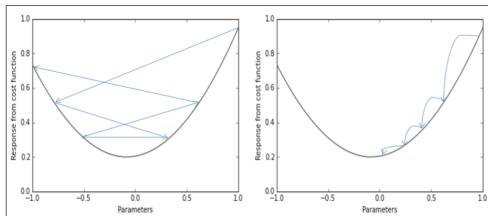
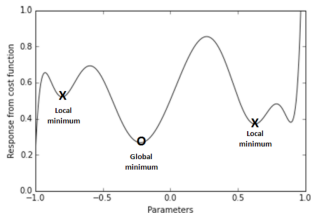
# Scikit-learn

---

## Stochastic learning

# 경사하강법 (gradient descent)

- 최적화 기법: 목표함수의 최대 또는 최소를 구하는 기법
  - 파라미터의 임의 초기값에서 시작하여 점차 최적해로 수렴해 가는 방식.
  - 현재의 위치에서 편미분을 이용하여 내리막(오르막) 방향으로 전진.
  - 평평한 위치(편미분값이 0)인 위치에서 최적해를 구한다.
  - 초기값에 따라 지역 최적해로 수렴할 수도 있다.
  - 최적화의 각 단계에서 파라미터 값을 얼마나 조정해야 하는지를 결정되어야 한다. 보통  $\alpha$ 로



# 확률적 경사하강법 (stochastic gradient descent)

- Out-of-core 방식의 경사하강법
  - 한 번에 하나의 객체를 대상으로 갱신이 실행
  - 객체들이 특정 규칙없이 임의로 선택되면 평균적으로 최적화가 진행
  - 이런 이유로 방향성을 제거해 가능한 무작위로 스트리밍이 되어야 한다
  - 데이터가 독립적이고 동일하게 분포되면 (iid), 전역해로 수렴
- Out-of-core 방식의 learning에서는 원치 않는 종류의 순서 정보가 포함되는 것을 방지하기 위해 데이터를 스트리밍하기 전에 행을 뒤섞는다.



# 회귀모형에서의 경사하강법

- $y_i = \mathbf{x}_i' \boldsymbol{\beta} + \epsilon_i, i = 1, 2, \dots, n$
- minimize  $J(\boldsymbol{\beta}) = \frac{1}{2n} \sum_{i=1} (\mathbf{x}_i' \boldsymbol{\beta} - y_i)^2$  with respect to  $\boldsymbol{\beta}$
- 회귀모형에서는 closed form solution( $\hat{\boldsymbol{\beta}} = (\mathbf{X}'\mathbf{X})^{-1} \mathbf{X}'\mathbf{y}$ )이 존재
- $\partial J(\boldsymbol{\beta}) / \partial \boldsymbol{\beta} = \frac{1}{n} \sum \mathbf{x}_i (\mathbf{x}_i' \boldsymbol{\beta} - y_i)$
- **Batch** 경사하강법
  - 갱신:  $\boldsymbol{\beta}_{k+1} = \boldsymbol{\beta}_k - \alpha \partial [J(\boldsymbol{\beta}) / \partial \boldsymbol{\beta}]_{\boldsymbol{\beta}=\boldsymbol{\beta}_k}$  until no change
- **Stochastic** 경사하강법
  - 갱신:  $\boldsymbol{\beta}_{k+1} = \boldsymbol{\beta}_k - \alpha [\mathbf{x}_k (\mathbf{x}_k' \boldsymbol{\beta} - y_k)]$ ,  $k = 1, 2, \dots, n$

# Ordering of instances

- Out-of-Core 방식의 learning에서는 관찰값의 순서는 random인 것이 중요
- How to shuffle data set
  - In-memory 데이터는 순서를 random화 하기 어렵지 않다.
  - Use Random Access file(Anaconda dose not include this module)
  - Use database system(SQLite3): Python과 비교하여 작은 memory를 사용
  - Mimic random shuffling: memory에 할당 가능한 만큼씩 batch로 데이터를 읽고, 각 batch 내에서 random화 하여 저장

# In-memory shuffling

```
import pandas as pd
import numpy as np
import os

def ram_Shuffle(filename_in, filename_out, header=True, SEP=', '):

    with open(filename_in, 'r') as R:
        data = pd.read_csv(R, sep = SEP)

    with open(filename_out, 'w') as W:
        data.iloc[np.random.permutation(len(data))].to_csv \
            (W, index=False, header = header, sep=SEP)
    # data.reindex(np.random.permutation(len(data))).to_csv(W,...)

    local_path = os.getcwd() + os.sep
    source = 'bikesharing/hour.csv'
    fileName_in = local_path + source
    fileName_out = local_path + 'bikesharing/hour_RamShuffled.csv'

    ram_Shuffle(fileName_in, fileName_out, header=True)
```

# Disk shuffling

```
import pandas as pd
import numpy as np
import os

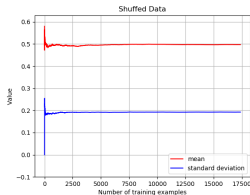
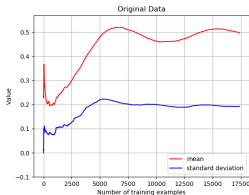
def disk_shuffle(filename_in, filename_out, header=True, \
                  CHUNK_SIZE=10000, SEP=', '):
    with open(filename_out, 'a') as W:
        with open(filename_in, 'r') as R:
            iterator = pd.read_csv(R, chunksize=CHUNK_SIZE)

            for n, df in enumerate(iterator):
                if n==0 and header:
                    df.iloc[np.random.permutation(len(df))].to_csv(W, \
                        index = False, header = True, sep = SEP)
                else :
                    df.iloc[np.random.permutation(len(df))].to_csv(W, \
                        index = False, header = False, sep = SEP)
```

```
SEP = ','  
local_path = os.getcwd() + os.sep  
source = 'bikesharing/hour.csv'  
fileName_in = local_path + source  
fileName_out = local_path + 'bikesharing/shufflehour.csv'  
disk_shuffle(fileName_in, fileName_out, header=True,\  
              CHUNK_SIZE=5000, SEP=', ')
```

# Effect of randomization in order

- 관찰값들의 순서를 random하게 shuffling 한 것에 대한 효과를 보기 위해 각 표본크기 별로 표본평균과 표본 편준편차를 계산
- 표본평균과 표준편차의 계산
  - $S^2 = \frac{1}{n-1}(\sum_{i=1}^n x_i^2 - n\bar{x}^2)$ 에 의한 계산 시간을 줄일 수 있는 반면, 표본분산의 계산은 계산오차가 클 수 있다.
  - Let  $m_n = \frac{1}{n} \sum_{i=1}^n x_i$ ,  $S_n = \sum_{i=1}^n (x_i - m_n)^2$
  - Then,  $m_n = \frac{1}{n}[(n-1)m_{n-1} + x_n]$ ,  $S_n = S_{n-1} + \frac{n-1}{n}(x_n - m_{n-1}^2)$
  - 점화식에 의해  $m_n, S_n, n = 1, 2, \dots$ 를 계산



```

import os, csv, math
import pandas as pd
import matplotlib.pyplot as plt

class StreamStat:
    def __init__(self, fileName, feature, title):
        self.fileName = fileName
        self.feature = feature
        self.title = title

    def meanStd(self):
        running_mean = list()
        running_std = list()
        with open(self.fileName, 'r') as R:
            iterator = csv.DictReader(R, delimiter=',')
            first = next(iterator)
            m = float(first[self.feature])      #  $n = 1$ 
            s = 0.
            running_mean.append(m)
            running_std.append(s)

```

```

    for n, row in enumerate(iterator):
        n = n + 2
        x = float(row[self.feature])
        s = s + (1. - 1./n) * (x - m) * (x - m)
        m = ((n-1) * m + x)/n
        running_mean.append(m)
        running_std.append(math.sqrt(s/(n-1)))

    return(pd.DataFrame({'mean': running_mean, \
                        'std': running_std}))

def plot(self):
    data = StreamStat.meanStd(self)
    minY = min(data.min())
    maxY = max(data.max())
    plt.plot(data['mean'], 'r-', label='mean')
    plt.plot(data['std'], 'b-', label='standard_deviation')
    plt.title(self.title)
    plt.ylim(minY-0.1, maxY+ 0.05)
    plt.xlabel('Number_of_training_examples')
    plt.ylabel('Value')
    plt.legend(loc='lower_right', numpoints= 1)
    plt.grid(True)
    plt.show()

```



```
local_path = os.getcwd() + os.sep
original = local_path + 'bikesharing/hour.csv'
shuffled = local_path + 'bikesharing/hour_RamShuffled.csv'
nonShuff = StreamStat(original, 'temp', "Original_Data")
nonShuff.plot()

shuff = StreamStat(shuffled, 'temp', "Shuffled_Data")
shuff.plot()
```

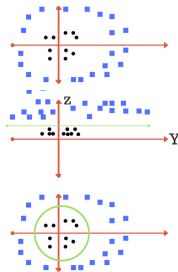
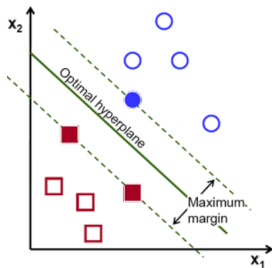
# Scikit-learn

---

## Support vector machine

# Support vector machine: classification

- 분류(classification)와 회귀분석을 위한 지도학습 모형
  - 데이터가 어느 범주에 속할지 판단하는 비확률적 선형, 비선형 분류 모델
  - 비확률적 분류 모델은 데이터가 사상된 공간에서 경계로 표현
  - SVM은 가장 큰 폭의 **margin**을 가진 경계
  - **margin**을 결정해 주는 관찰값(벡터)이 **support vector**
  - 비선형 분류를 위해서는 데이터를 고차원 공간으로 사상하는 작업이 필요



# 선형 Support vector machine

- 학습용 데이터  $\mathcal{D} = \{(\mathbf{x}_i, y_i) | \mathbf{x}_i \in \mathbb{R}^p, y_i \in \{-1, 1\}\}_{i=1}^n$
- class를 분리하는 평면:  $\mathbf{w} \cdot \mathbf{x} - b = 0$  where  $\cdot$ 은 내적
- support vector ( $\mathbf{x}^+, \mathbf{x}^-$ )
  - $\mathbf{x}^+$ : +1 class 데이터 중에서 평면에 가장 가까운 데이터
  - $\mathbf{x}^-$ : -1 class 데이터 중에서 평면에 가장 가까운 데이터
  - support vector 사이의 margin은  $2/\|\mathbf{w}\|$
- support vector를 지나는 평면 사이에는 데이터가 존재하지 않으므로
  - $\mathbf{w} \cdot \mathbf{x}_i - b \geq +1$  for such  $i, y_i = +1$
  - $\mathbf{w} \cdot \mathbf{x}_i - b \leq -1$  for such  $i, y_i = -1$
  - $y_i(\mathbf{w} \cdot \mathbf{x}_i) \geq 1$  for all  $i = 1, \dots, n$
- $\arg \min_{(\mathbf{w}, b)} \|\mathbf{w}\|$  subject to  $y_i(\mathbf{w} \cdot \mathbf{x}_i) \geq 1$  for all  $i = 1, \dots, n$
- or equivalently  $\arg \min_{(\mathbf{w}, b)} \frac{1}{2} \|\mathbf{w}\|^2$  subject to  $y_i(\mathbf{w} \cdot \mathbf{x}_i) \geq 1$  for all  $i = 1, \dots, n$

- Lagrange multiplier method

$$\arg \min_{(\mathbf{w}, b)} \max_{\alpha \geq 0} \left\{ \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^n \alpha_i [y_i(\mathbf{w} \cdot \mathbf{x}_i) - 1] \right\}$$

- To extend SVM to cases in which the data are not linearly separable.
- Soft margin: 잘못 분류된 데이터를 허용
  - $y_i(\mathbf{w} \cdot \mathbf{x}_i) \geq 1 - \xi_i$  for all  $i = 1, \dots, n; \xi_i \geq 0$
  - Put some penalty of  $\xi_i$
  - $\arg \min_{(\mathbf{w}, b, \xi)} \left\{ \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \xi_i \right\}$   
subject to  $y_i(\mathbf{w} \cdot \mathbf{x}_i) \geq 1 - \xi_i$  for all  $i = 1, \dots, n; \xi_i \geq 0$

$$\arg \min_{(\mathbf{w}, b)} \max_{\alpha, \beta} \left\{ \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \xi_i - \sum_{i=1}^n \alpha_i [y_i(\mathbf{w} \cdot \mathbf{x}_i) - 1 + \xi_i] - \sum_{i=1}^n \beta_i \xi_i \right\}$$

- This is somewhat complicated.

- Introduce the hinge loss function,  $\max(0, y_i(\mathbf{w} \cdot \mathbf{x}_i) - 1)$

$$\arg \min_{(\mathbf{w}, b)} \max_{\alpha \geq 0} \left\{ \frac{\lambda}{2} \|\mathbf{w}\|^2 - \frac{1}{n} \sum_{i=1}^n \max[0, 1 - y_i(\mathbf{w} \cdot \mathbf{x}_i)] \right\}$$

- $\lambda$ : regularization parameter
- Often, as in the Scikit-learn,  $\lambda$  is removed and replaced by a misclassification parameter  $C$

$$\arg \min_{(\mathbf{w}, b)} \max_{\alpha \geq 0} \left\{ \frac{1}{2} \|\mathbf{w}\|^2 - C \sum_{i=1}^n \max[0, 1 - y_i(\mathbf{w} \cdot \mathbf{x}_i)] \right\}$$

where  $\lambda = 1/nC$ .

- High value of  $C$ 
  - Impose a high penalty on errors
  - Tend to force the margin to be tighter
  - Take into consideration fewer support vectors
  - Increase bias but reduce variance

# 비선형 Support vector machine

- Kernel: 선형 SVM을 hyperplane에서 훈련시키기 위해서는 kernel에 의한 변환이 필요
  - linear kernel: no transformation
  - polynomial kernel:  $\mathcal{K}(\mathbf{x}, \mathbf{x}_i) = (\mathbf{x} \cdot \mathbf{x}_i)^d$
  - radial basis kernel:  $\mathcal{K}(\mathbf{x}, \mathbf{x}_i) = \exp\left(-\frac{1}{2\sigma^2} \|\mathbf{x} - \mathbf{x}_i\|^2\right)$
  - hyperbolic tangent:  $\mathcal{K}(\mathbf{x}, \mathbf{x}_i) = \tanh(\gamma \mathbf{x} \cdot \mathbf{x}_i + c)$  for some  $\gamma > 0$  and  $c < 0$ , a sigmoid kernel
  - where  $\mathbf{x} \cdot \mathbf{y} = \sum_j x_j y_j$  (dot product) and  $\|\mathbf{x}\| = \sqrt{\sum_j x_j^2}$  (norm)

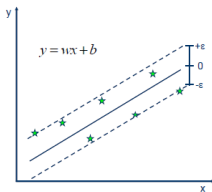
# Support vector regression

- Support vector regression은 SVM에서 유래
- A robust regression analysis
  - hyperplane를 나타내는  $\mathbf{w}$ 를  $\beta$ 로 표시

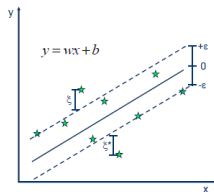
$$\sum_j \beta_j^2 + C \sum_{i=1}^n L_\epsilon(y_i, \hat{y}_i)$$

where  $L$  is a loss function

- Hinge loss:  $L_\epsilon(y_i, \hat{y}_i) = \max(0, 1 - y_i \hat{y}_i)$
- Squared error loss(or L2 loss):  $L_\epsilon(y_i, \hat{y}_i) = \max(0, 1 - y_i \hat{y}_i)^2$



- Solution:  
 $\min \frac{1}{2} \|\mathbf{w}\|^2$
- Constraints:  
 $y_i - wx_i - b \leq \epsilon$   
 $w x_i + b - y_i \leq \epsilon$



- Minimize:  
 $\frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N (\xi_i + \xi_i^*)$
- Constraints:  
 $y_i - wx_i - b \leq \epsilon + \xi_i$   
 $w x_i + b - y_i \leq \epsilon + \xi_i^*$   
 $\xi_i, \xi_i^* \geq 0$