

파이썬 라이브러리를 활용한 데이터 분석 1장

파이썬, 최고의 인기 언어

- **접착제로 쓰는 파이썬**
 - C, C++, Fortran 등의 기존 언어와 통합이 쉽다
 - **Glue code, 접착제 코드**
- **이제 파이썬 하나만 된다**
 - 웹 구축
 - 일반 응용프로그램
 - 데이터과학
 - 딥러닝
- **개발 시간이 단축**
 - 실행 시간이 느려도 이익
 - **CPU시간 비용보다 사람의 비용이 더 크다**

필수 파이썬 라이브러리

- **Numpy**
 - Numerical python
 - 빠르고 효율적인 다차원 객체
 - 배열 원소를 다루거나 배열 간의 수학 계산을 수행하는 함수
- **Pandas**
 - Panel data & Python data analysis
 - 구조화된 데이터나 표 형식의 데이터를 쉽게 다루도록 설계된 고수준의 자료구조와 함수 제공
 - 표 형태의 행과 열의 구조인 데이터프레임(dataframe)
 - 1차원 배열 객체인 시리즈(series)
- **Matplotlib**
 - 그래프나 2차원 데이터 시각화
- **Jupyter**
- **Scipy**
 - 과학 계산 컴퓨터 영역의 여러 기본 문제를 다루는 패키지 모음
- **Scikit-learn**
 - 데이터 처리 및 머신 러닝 라이브러리

파이썬 라이브러리를 활용한 데이터 분석 2장

현재와 미래

- 데이터 분석

- Pandas가 2008년에 시작되어 2010년 정도 릴리즈
- 이 책의 초판 2012년: 초기
- 2017년
 - 데이터 과학, 데이터 분석, 머신 러닝, 딥러닝 붐
 - 자료도 많음

- 저자의 데이터 분석을 생산적으로 하려면

- 파이썬은 수단
 - 고급 문법이 필요한 것은 아니다
- Ipython과 jupyter notebook 권장

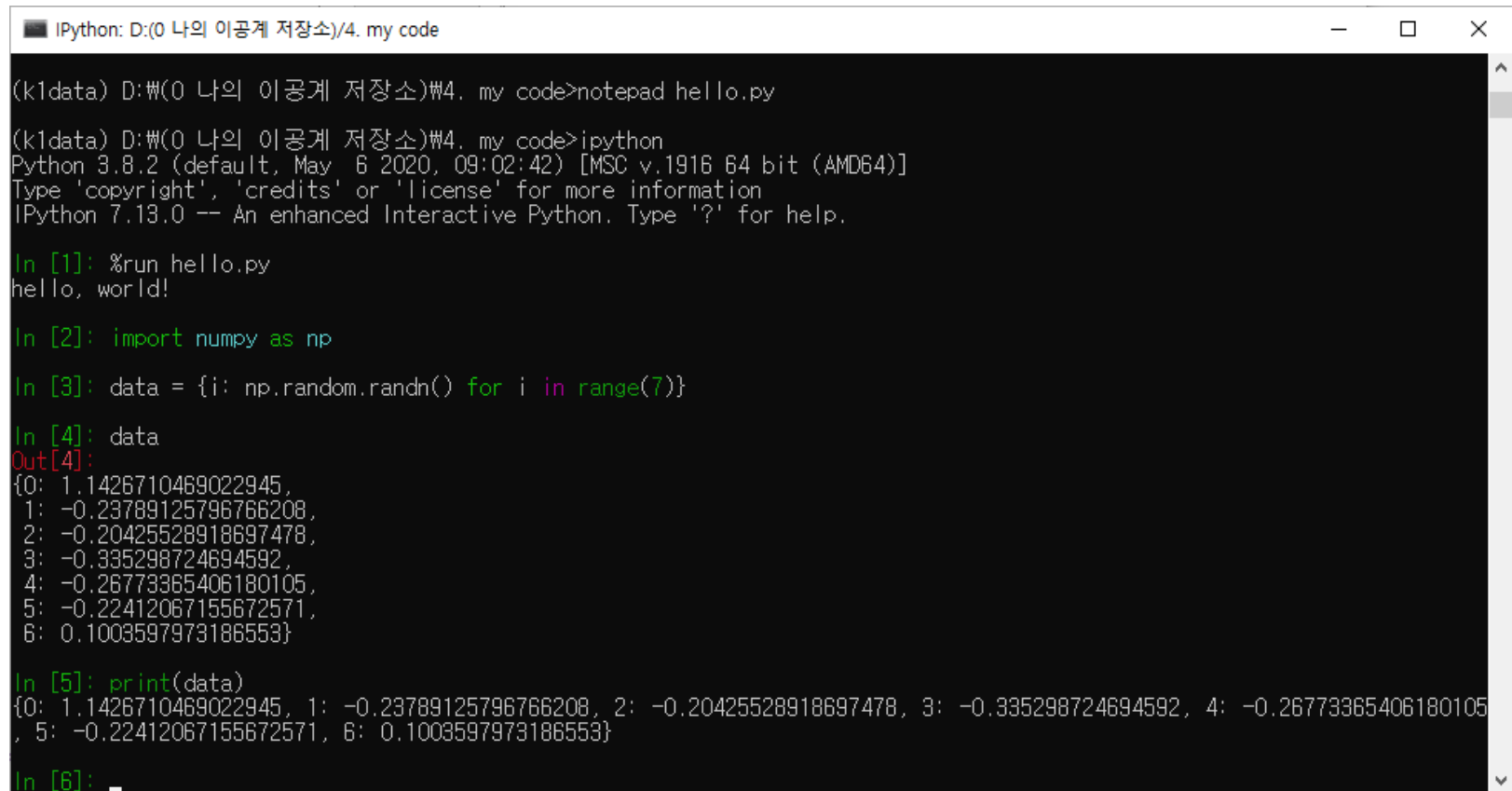
- 미래, 아니 지금

- 데이터 분석과 딥러닝
 - 누구나가 알아야할 보편적인 지식

Ipython 실행(1)

• 방법1

- 아나콘다 프롬프트에서 직접 ipython으로 실행



```
IPython: D:(0 나의 이공계 저장소)/4. my code
(k1data) D:\(0 나의 이공계 저장소)\4. my code>notepad hello.py
(k1data) D:\(0 나의 이공계 저장소)\4. my code>ipython
Python 3.8.2 (default, May 6 2020, 09:02:42) [MSC v.1916 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 7.13.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: %run hello.py
hello, world!

In [2]: import numpy as np

In [3]: data = {i: np.random.randn() for i in range(7)}

In [4]: data
Out[4]:
{0: 1.1426710469022945,
 1: -0.23789125796766208,
 2: -0.20425528918697478,
 3: -0.335298724694592,
 4: -0.26773365406180105,
 5: -0.22412067155672571,
 6: 0.1003597973186553}

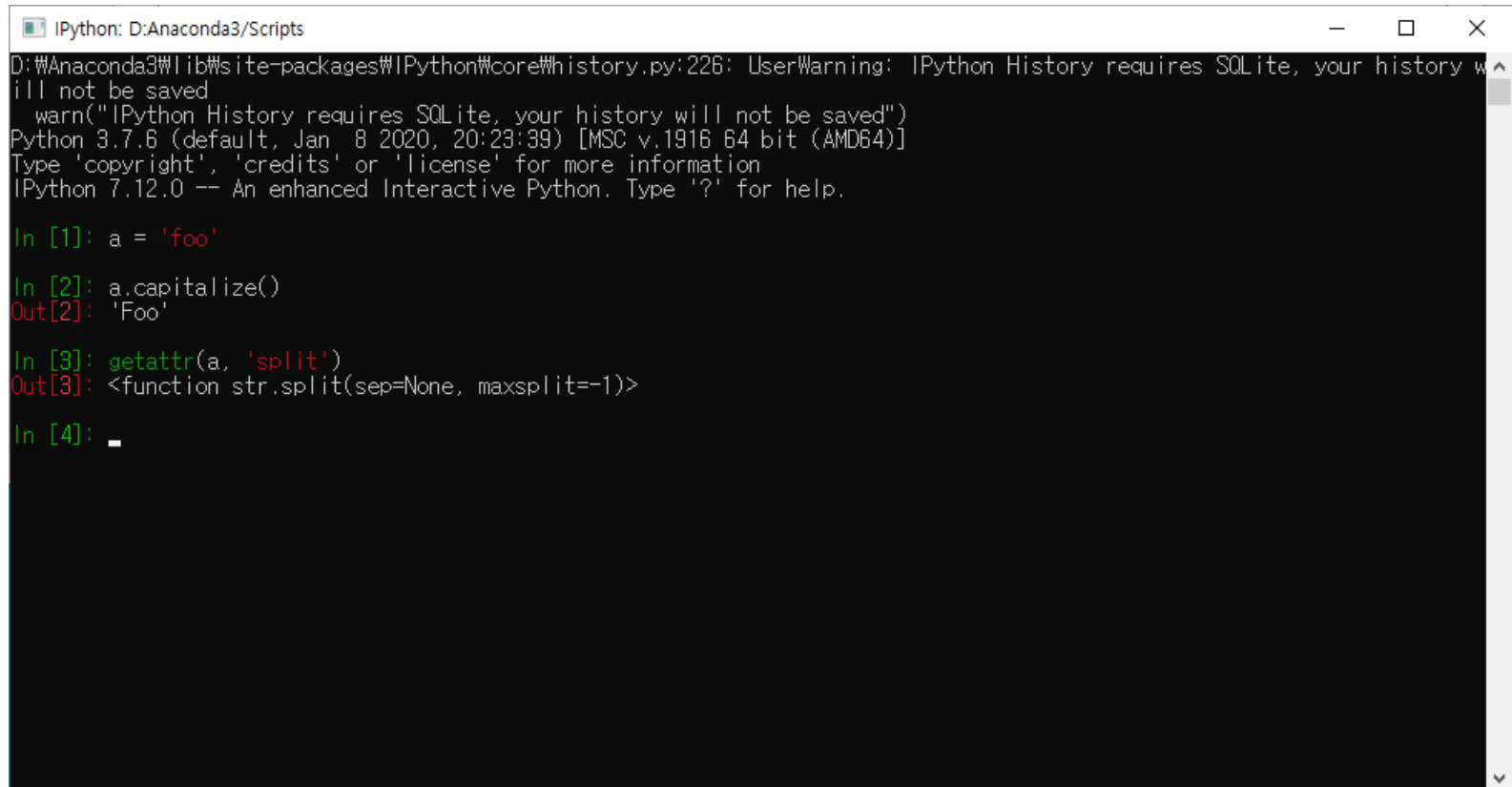
In [5]: print(data)
{0: 1.1426710469022945, 1: -0.23789125796766208, 2: -0.20425528918697478, 3: -0.335298724694592, 4: -0.26773365406180105,
 5: -0.22412067155672571, 6: 0.1003597973186553}

In [6]:
```

Ipython 실행(2)

- 방법2

- Anaconda3/Scripts/ipython.exe 더블 클릭으로 실행



```
IPython: D:\Anaconda3\Scripts
D:\Anaconda3\lib\site-packages\IPython\core\history.py:226: UserWarning: IPython History requires SQLite, your history will not be saved
  warn("IPython History requires SQLite, your history will not be saved")
Python 3.7.6 (default, Jan  8 2020, 20:23:39) [MSC v.1916 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 7.12.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: a = 'foo'

In [2]: a.capitalize()
Out[2]: 'Foo'

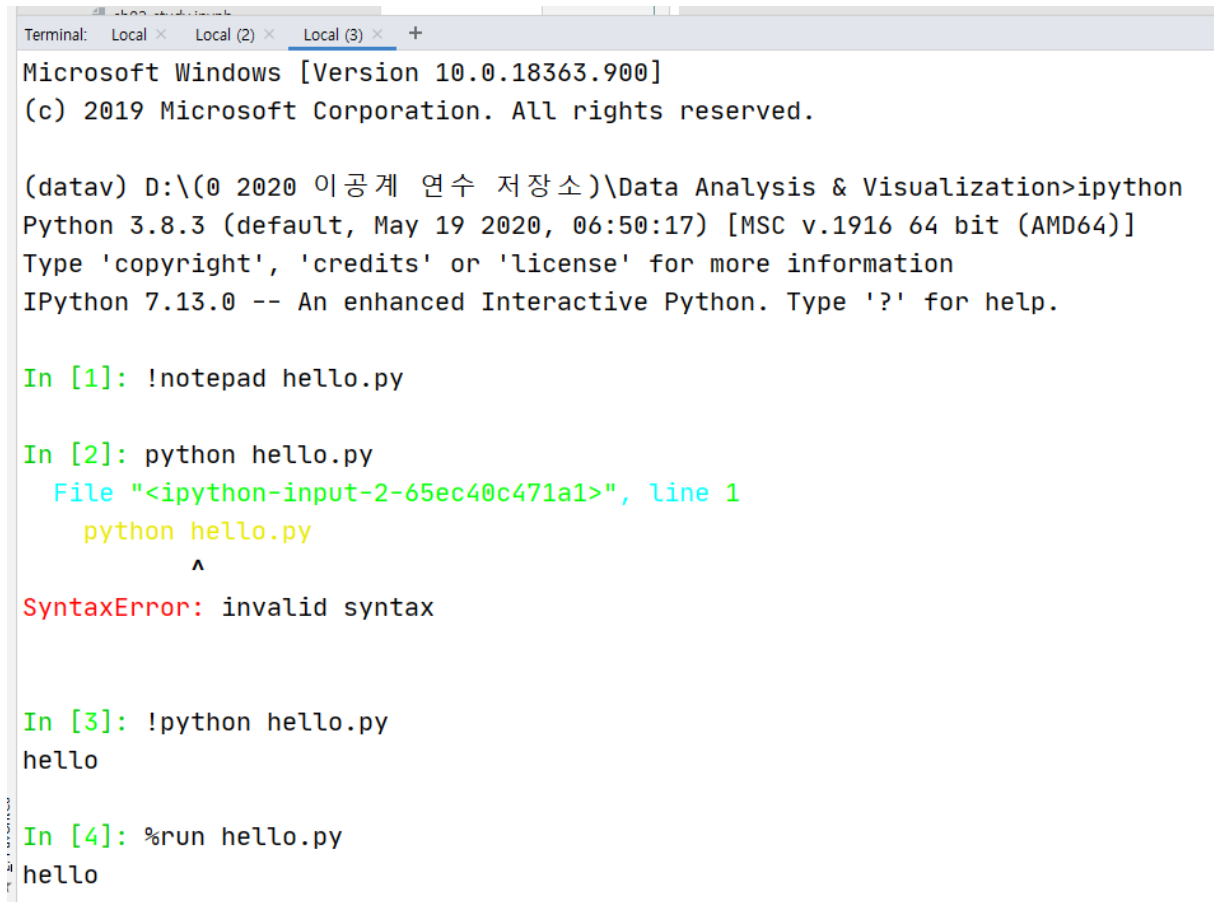
In [3]: getattr(a, 'split')
Out[3]: <function str.split(sep=None, maxsplit=-1)>

In [4]: _
```

Ipython 실행(3)

- 방법3

- 파이참 터미널에서 직접 ipython 실행



```
Terminal: Local x Local (2) x Local (3) x +
Microsoft Windows [Version 10.0.18363.900]
(c) 2019 Microsoft Corporation. All rights reserved.

(datav) D:\(0 2020 이공계 연수 저장소)\Data Analysis & Visualization>ipython
Python 3.8.3 (default, May 19 2020, 06:50:17) [MSC v.1916 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 7.13.0 -- An enhanced Interactive Python. Type '?' for help.

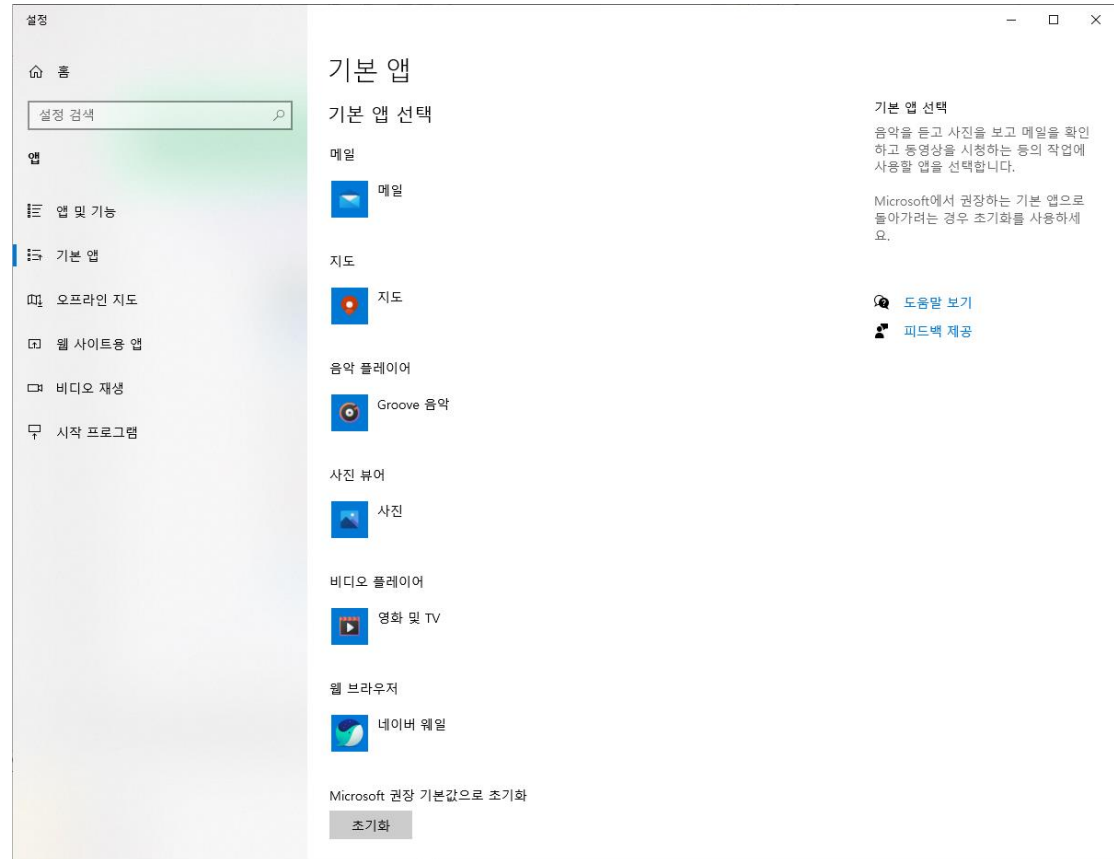
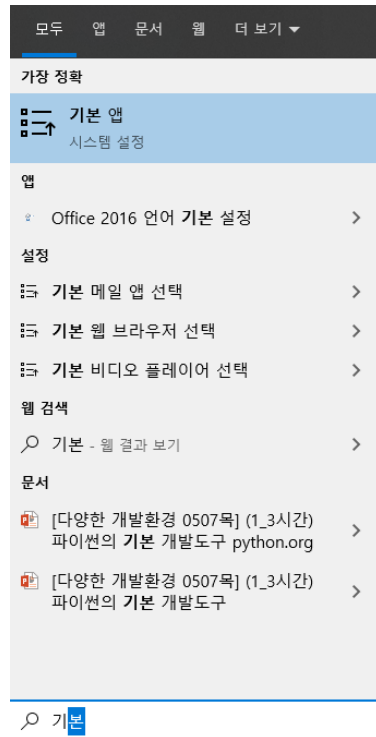
In [1]: !notepad hello.py

In [2]: python hello.py
File "<ipython-input-2-65ec40c471a1>", line 1
python hello.py
      ^
SyntaxError: invalid syntax

In [3]: !python hello.py
hello

In [4]: %run hello.py
hello
```


- 기본 웹 브라우저 설정
 - 크롬이나 웨일로 지정



주피터 노트북 실행(2)

- 개요

- 코드, 텍스트, 데이터 시각화를 비롯한 다른 출력을 대화형으로 구성할 수 있는 대화형 문서 형식
 - 코드
 - 결과
 - 결과 그림
 - 문서
- lpython 커널을 사용

- 명령어로 실행

- jupyter notebook
- python -m notebook

자기관찰(introspection), p51

- 문장 뒤에 ?
 - Shift + enter
 - **print?**
 - 정의되어 있는 문서(doc string) 출력

```

IPython: D:(0 나의 이공계 저장소)/4. my code
Built-in mutable sequence.

If no argument is given, the constructor creates a new empty list.
The argument must be an iterable if specified.

In [16]:
In [16]: b?
Type:      list
String form: [1, 2, 3]
Length:    3
Docstring:
Built-in mutable sequence.

If no argument is given, the constructor creates a new empty list.
The argument must be an iterable if specified.

In [17]: print?
Docstring:
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments:
file: a file-like object (stream); defaults to the current sys.stdout.
sep:  string inserted between values, default a space.
end:  string appended after the last value, default a newline.
flush: whether to forcibly flush the stream.
Type:      builtin_function_or_method

In [18]: _
  
```

클립보드 또는 블록 코드 실행하기

- 다른 곳의 코드 복사 후
 - %paste
 - 복사된 내용이 블록으로 실행됨
- 블록을 코딩하여 실행하고자 하는 경우
 - %cpaste
 - : 이후 줄마다 계속 코딩 이후, 필요하면 ctrl + v 붙여넣기도 가능
 - 마지막 문장에서 –
 - 실행 됨

```

IPython: D:\0 나의 이공계 저장소\4. my code

In [38]: %paste
print('hello, world!')
## -- End pasted text --
hello, world!

In [39]: %cpaste
Pasting code; enter '--' alone on the line to stop or use Ctrl-D.
:a = 10
:b = 2
:c = pow(b, a)
:--

In [40]: a
Out[40]: 10

In [41]: b
Out[41]: 2

In [42]: c
Out[42]: 1024

In [43]: _

```

매직 명령어

- %로 시작
 - %timeit
 - %automagic
 - %를 안 써도 가능하도록, 토글
 - %magic
 - %quickref

IPython: D:\0 나의 이공계 저장소\4. my code

```
In [52]: a = np.random.randn(100, 100)
In [53]: %timeit np.dot(a, a)
15.6 µs ± 840 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
In [54]: %timeit np.dot(a, a)
15.8 µs ± 2.08 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
In [55]: %timeit np.dot(a, a)
17 µs ± 2.61 µs per loop (mean ± std. dev. of 7 runs, 100000 loops each)
In [56]: %automagic
Automagic is OFF, % prefix IS needed for line magics.
In [57]: %automagic
Automagic is ON, % prefix IS NOT needed for line magics.
In [58]: timeit np.dot(a, a)
15.5 µs ± 575 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

IPython: D:\0 나의 이공계 저장소\4. my code

```
In [59]: magic
IPython's 'magic' functions
=====
The magic function system provides a series of functions which allow you to control the behavior of IPython itself, plus a lot of system-type features. There are two kinds of magics, line-oriented and cell-oriented.
Line magics are prefixed with the % character and work much like OS command-line calls: they get as an argument the rest of the line, where arguments are passed without parentheses or quotes. For example, this will time the given statement::
    %timeit range(1000)
Cell magics are prefixed with a double %, and they are functions that get as an argument not only the rest of the line, but also the lines below it in a separate argument. These magics are called with two arguments: the rest of the call line and the body of the cell, consisting of the lines below the first. For example::
    %%timeit x = numpy.random.randn((100, 100))
    numpy.linalg.svd(x)
will time the execution of the numpy svd routine, running the assignment of x as part of the setup phase, which is not timed.
In a line-oriented client (the terminal or Qt console IPython), starting a new input with %% will automatically enter cell mode, and IPython will continue
In [60]: quickref
IPython -- An enhanced Interactive Python -- Quick Reference Card
=====
obj?, obj??      : Get help, or more help for object (also works as ?obj, ??obj).
?foo.*abc*       : List names in 'foo' containing 'abc' in them.
%magic           : Information about IPython's 'magic' % functions.
Magic functions are prefixed by % or %% and typically take their arguments without parentheses, quotes or even commas for convenience. Line magics take a single % and cell magics are prefixed with two %%.
Example magic function calls:
%alias d ls -F    : 'd' is now an alias for 'ls -F'
alias d ls -F    : Works if 'alias' not a python name
alist = %alias    : Get list of aliases to 'alist'
cd /usr/share     : Obvious, cd -<tab> to choose from visited dirs.
%cd??            : See help AND source for magic %cd
%timeit x=10      : time the 'x=10' statement with high precision.
%%timeit x=2**100 : time 'x=2**100' with a setup of 'x=2**100'; setup code is not counted. This is an example of a cell magic.
x**100
System commands:
!cp a.txt b/      : System command escape, calls os.system()
cp a.txt b/       : after %rehashx, most system commands work without !
cp ${f}.txt $bar  : Variable expansion in magics and system commands
files = !ls /usr  : Capture system command output
files.s, files.l, files.n: 'a b c', ['a','b','c'], 'a#b#c'
History:
_, _ii, _iii     : Previous, next previous, next next previous input
_!4, _!h[2:5]    : Input history line 4, lines 2-4
exec _!81        : Execute input history line #81 again
%rep 81          : Edit input history line #81
%_               : previous, next previous, next next previous output
%_               : Directory history
%_               : Output history
%hist            : Command history of current session.
%hist -a foo     : Search command history of (almost) all sessions for 'foo'.
%hist -a         : Command history of (almost) all sessions.
%hist 1/2-8      : Command history containing lines 2-8 of session 1.
%hist 1/ ~2/     : Command history of session 1 and 2 sessions before current.
%hist -8/1--6/5  : Command history from line 1 of 8 sessions ago to line 5 of 6 sessions ago.
%edit 0/         : Open editor to execute code with history of current session.
Autocall:
```

자주 사용하는 매직 명령어 p58

Table 2-1. Standard IPython keyboard shortcuts

Keyboard shortcut	Description
Ctrl-P or up-arrow	Search backward in command history for commands starting with currently entered text
Ctrl-N or down-arrow	Search forward in command history for commands starting with currently entered text
Ctrl-R	Readline-style reverse history search (partial matching)
Ctrl-Shift-V	Paste text from clipboard
Ctrl-C	Interrupt currently executing code
Ctrl-A	Move cursor to beginning of line
Ctrl-E	Move cursor to end of line
Ctrl-K	Delete text from cursor until end of line
Ctrl-U	Discard all text on current line
Ctrl-F	Move cursor forward one character
Ctrl-B	Move cursor back one character
Ctrl-L	Clear screen

Ipython에서 matplotlib 사용(노트북은 상관 없음)

- 다음 코드
 - `import matplotlib.pyplot as plt`
 - `plt.plot(range(5))`
 - `plt.show()`
 - 그림 윈도우가 표시된 후 제어가 콘솔로 안 넘어 옴
 - 달아야 다시 제어권이 넘어 옴
- 불편 해소
 - `%matplotlib`
- 다음으로 그려지며, 계속 콘솔 이용 가능
 - `import matplotlib.pyplot as plt`
 - `plt.plot(range(5))`

파이썬 기초

Duck typing 정의

- 객체의 변수 및 메소드의 집합이 객체의 타입을 결정하는 것을 말함
 - 만약 어떤 새가 오리처럼 걷고, 헤엄치고, 깉깉거리는 소리를 낸다면 나는 그 새를 오리라고 부를 것
 - 사람이 오리처럼 행동하면 오리로 봐도 무방하다라는게 덕 타이핑(Duck Typing)
 - 타입을 미리 정하는게 아니라 실행이 되었을 때 해당 메소드(method)를 확인해 타입을 정함
- 장점
 - 타입에 대해 매우 자유롭다.
 - 런타임 데이터를 기반으로 한 기능과 자료형을 창출하는 것
- 단점
 - 런타임 자료형 오류가 발생 가능
 - 값은 예상치 못한 유형이 있을 수 있고, 그 자료형에 대한 무의미한 작업이 적용
 - 이런 오류가 프로그래밍 구문에서 오랜 시간 후에 발생 가능

Duck typing 활용

- 객체의 자료정보보다 어떤 메소드나 행동을 지원 하느냐에 관심
- 객체의 타입보다 객체가 사용되는 양상이 더 중요
 - 덕 타이핑이 없는 프로그래밍 언어
 - 오리 타입의 객체를 인자로 받아 객체의 걸기 메소드와 꺾꺾거리기 메소드를 차례로 호출하는 함수를 만들 수 있다.
 - 반면에, 같은 함수를 덕 타이핑이 지원되는 언어
 - 인자로 받는 객체의 타입을 검사하지 않도록 만들 수 있다.
 - 걸기 메소드나 꺾꺾거리기 메소드를 호출 할 시점에서
 - 객체에 두 메소드가 없다면 런타임 에러가 발생
 - 두 메소드가 제대로 구현되어 있다면 함수는 정상적으로 작동

Duck typing 샘플 코드

```
In [44]: class Duck:
          def quack(self):
              print("꽹꽹!")
          def feathers(self):
              print("오리에게 흰색, 회색 깃털이 있습니다.")

          class Person:
              def quack(self):
                  print("이 사람이 오리를 흉내내네요.")
              def feathers(self):
                  print("사람은 바닥에서 깃털을 주어서 보여 줍니다.")

          def in_the_forest(duck):
              duck.quack()
              duck.feathers()

          def game():
              donald = Duck()
              john = Person()
              in_the_forest(donald)
              in_the_forest(john)

          game()
```

꽹꽹!
오리에게 흰색, 회색 깃털이 있습니다.
이 사람이 오리를 흉내내네요.
사람은 바닥에서 깃털을 주어서 보여 줍니다.

Iterator

- **iterable 객체 - 반복 가능한 객체**
 - 대표적으로 iterable한 타입
 - **list, dict, set, str, bytes, tuple, range**
- **iterator 객체**
 - 값을 차례대로 꺼낼 수 있는 객체
 - Iterator
 - **iterable한 객체를 내장함수 iter()로 생성**
 - **Iterable 객체의 메소드 __iter__로 객체를 생성**

모듈 импорт

- 간단히 파이썬 코드가 담긴 *.py 파일
- as 사용
 - 다른 이름으로 사용
- 주피터 노트북
 - 코드로 변환
 - y
 - 마크다운으로 변환
 - m

Imports

```
In [45]: !notepad some_module.py
```

```
In [ ]: ```python
# some_module.py
PI = 3.14159

def f(x):
    return x + 2

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

```
In [47]: import some_module
result = some_module.f(5)
result
```

```
Out[47]: 7
```

```
In [48]: pi = some_module.PI
pi
```

```
Out[48]: 3.14159
```

```
In [50]: from some_module import f, g, PI
result = g(5, PI)
result
```

```
Out[50]: 8.14159
```

```
In [51]: import some_module as sm
from some_module import PI as pi, g as gf

r1 = sm.f(pi)
r2 = gf(6, pi)
```

# 파이썬 코드 메모리 보이기

- [Pythontutur.com](https://pythontutorial.com)

# 스칼라형

- 단일 값의 자료형

*Table 2-4. Standard Python scalar types*

| Type  | Description                                                                             |
|-------|-----------------------------------------------------------------------------------------|
| None  | The Python “null” value (only one instance of the None object exists)                   |
| str   | String type; holds Unicode (UTF-8 encoded) strings                                      |
| bytes | Raw ASCII bytes (or Unicode encoded as bytes)                                           |
| float | Double-precision (64-bit) floating-point number (note there is no separate double type) |
| bool  | A True or False value                                                                   |
| int   | Arbitrary precision signed integer                                                      |

- 문자열

- 일련의 유니코드문자, 순차적인 자료형
- r(raw string) 표기, ₩ 그대로 ₩로 인식
  - `r'this₩has'`

# 바이트와 유니코드

- **bytes**
  - byte의 배열, 지정한 인코딩 방식의 텍스트 문자열
    - `'python'.encode('utf8')`
      - `b'python'`
  - 아스키 상수 문자 표현만 가능
    - `b'파이썬'`
      - `SyntaxError: bytes can only contain ASCII literal characters.`
- **string과 bytes 사이의 변환**
  - string => bytes
    - `str.encode('utf8')`
  - bytes => string
    - `bytes.decode('utf8')`
- **인코딩 방식**
  - 유니코드를 바이트 배열로 만드는 방법
  - utf8
    - UTF-8은 유니코드를 위한 가변 길이 문자 인코딩 방식 중 하나
    - Universal Coded Character Set + Transformation Format – 8-bit 의 약자
    - 유니코드 한 문자를 나타내기 위해 1바이트에서 4바이트 까지를 사용
  - utf16
    - 16-bit Unicode Transformation Format
    - 유니코드 문자 인코딩 방식의 하나



# 날짜와 시간

- **내장 모듈 datetime**
  - `From datetime import datetime, date, time`

# 삼항 표현식

- if-else 구문 한 줄로

*value = true-expr if cond else false-expr*

if *cond*:  
    *value = true-expr*  
else  
    *value = false-expr*

```
In [40]: x = 5
 'Non-negative' if x >= 0 else 'Negative'
Out [40]: 'Non-negative'
```

# 넘파이 설정

- **난수와 실수의 정확도**

- `import numpy as np`
- `np.random.seed(12345)`
  - **난수를 발생시키기 위한 초기 값 지정**
    - 이후 난수가 동일하게 발생
- `np.set_printoptions(precision=4, suppress=True)`
  - **precision=4: 소수점 이하 반올림해 4개 표시**
  - **`np.array(3.123456)`**
    - `array(3.1235)`
  - **suppress=True**
    - 가능한 e-04와 같은 scientific notation을 제거하고 싶으면

- **Alt + Enter**

- 현재 셀 실행 후, 다음 셀 삽입