

Python Tutorial 2

String, RegExp, Date, File

Haesun Park, haesunrpark@gmail.com,

String

String Functions

s = 'some string'

s.upper()	대문자로 바꾼 문자열 생성
s.lower()	소문자로 바꾼 문자열 생성
s.capitalize()	첫문자를 대문자로 바꾼 문자열 생성
s.find('a')	'a' 문자열의 위치(index) 찾기
s.count('a')	'a' 문자열 갯수 세기
s.isalpha(), s.isdigit()	영문자, 숫자만인지 확인
s.join([.]), s.split(',')	s 로 합치거나 구분자로 s 나누기
s.replace('a', 'b')	'a'를 'b'로 바꾼 문자열 생성
s.startswith('a')	'a'로 시작하는지
s.endswith('a')	'a'로 끝나는지
s.strip(), s.lstrip(), s.rstrip()	(좌, 우)공백을 제거한 문자열 생성
len(s)	문자열 길이
int(s), float(s)	정수, 실수형으로 변환

s = 'some string'

s.title()	'very good' → 'Very Good'
s.rfind('a')	뒤에서부터 'a'의 위치(index) 찾기
s.zfill(num)	s 왼쪽에 num-len(s)만큼 '0'를 채움
s.center(width)	width 중간에 s 위치시키고 공백 채움
s.ljust(num), rjust(num)	s 왼쪽, 오른쪽에 num-len(s) 만큼 공백
s.encode(enc)	enc(디폴트 utf-8) 타입으로 바이트화

String Format Old

"Year: %d, Month: %02d, Day: %02d" % (2016, 7, 9) → 'Year: 2016, Month: 07, Day: 09'

%[flags][width][.precision]type

flags

0: zero 패딩

-: 좌측 정렬

+: 부호 추가

#: 8진수나 16진수 표기

width

문자열로 표현될 길이

.precision

숫자일 경우 정밀도

type

d, o, x, f, s, %: 10진수, 8진수, 16진수, 실수형, 문자열, %

'%+-10.2f' % 12345

'+12345.00 '

'%#x' % 16

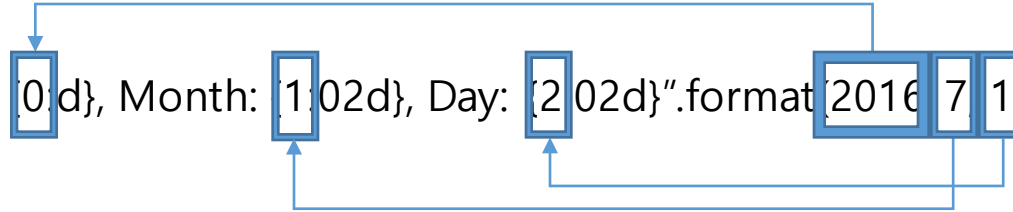
'0x10'

'my name is %s' % 'ricky'

'my name is ricky'

String Format New

"Year: {0:d}, Month: {1:02d}, Day: {2:02d}".format(2016, 7, 1)



→ 'Year: 2016, Month: 07, Day: 01'

"Year: {2:d}, Month: {0:02d}, Day: {1:02d}".format(7, 1, 2016)

→ 'Year: 2016, Month: 07, Day: 01'

"Year: {y:d}, Month: {m:02d}, Day: {d:02d}".format(m=7, d=1, y=2016)

→ 'Year: 2016, Month: 07, Day: 01'

"Year: {y:s}, Month: {m:>2s}, Day: {d:>2s}".format(m='7', d='1', y='2016')

→ 'Year: 2016, Month: 7, Day: 1'

"Year: {y:s}, Month: {m:<2s}, Day: {d:<2s}".format(m='7', d='1', y='2016')

→ 'Year: 2016, Month: 7, Day: 1'

"Year: {y:}, Month: {m:>2}, Day: {d:>2}".format(m='7', d='1', y='2016')

→ 'Year: 2016, Month: 7, Day: 1'

"Year: {y}, Month: {m}, Day: {d}".format(m='7', d='1', y='2016')

→ 'Year: 2016, Month: 7, Day: 1'

"Year: {2}, Month: {0}, Day: {1}".format('7', '1', '2016')

→ 'Year: 2016, Month: 7, Day: 1'

"Year: {}, Month: {}, Day: {}".format('7', '1', 2016)

→ 'Year: 7, Month: 1, Day: 2016'

String Format New

```
animal = ('cat', 'dog')
```

```
'This is {0[0]}'.format(animal)
```

```
'This is {0[1]}'.format(animal)
```

```
'This is cat'
```

```
'This is dog'
```

```
animal = {'cat':3, 'dog':5}
```

```
'Weight is {0[cat]}'.format(animal)
```

```
'Weight is 3'
```

```
'{: ^30}'.format('Menu')
```

```
→ '                Menu                '
```

```
'{: = ^30}'.format('Menu')
```

```
→ '=====Menu===== '
```

```
'Accuracy is {:.2}'.format(55/78)
```

```
→ 'Accuracy is 0.71'
```

```
'Accuracy is {:.2%}'.format(55/78)
```

```
→ 'Accuracy is 70.51%'
```

```
"{{Price is ${:,.}}}".format(3000)
```

```
→ '{Price is $3,000}'
```

String

Demo

jupyter notebook

Regular Expression

정규표현식은 문자열에서 원하는 패턴을 찾거나 다른 문자열로 바꾸는 기능입니다.

```
import re
p = re.compile('\d')
s = p.search('Room No 101')
s.__class__
s.group()
s = p.search('Room No')
s.__class__

s = re.search('\d', 'Room No 101')
s.group()
```

한 패턴을 여러번 사용할 때

→ `_sre.SRE_Match`

→ `'1'`

→ `None`

패턴을 한번만 사용할 때

→ `'1'`

Regular Expression

iPython Magic Keyword %timeit

```
import re
```

```
p = re.compile('\d')
```

```
%timeit -n 100 p.search('Room No 101')
```

→ 100 loops, best of 3: 550 ns per loop

```
%timeit -n 100 re.search('\d', 'Room No 101')
```

→ 100 loops, best of 3: 1.07 μ s per loop

약 두배의 차이가 남

```
%timeit -n (반복횟수) (문장)
```

반복횟수 만큼 실행을 3번 한 후 그 중에 가장 좋은 결과의 평균을 리턴함.

Regular Expression

`search()`

```
m = re.search('\d', 'Room No 123').group()
```

```
m.group()
```

문장의 모든 위치에서 검색

→ '1'

`match()`

```
re.match('\d', 'Room No 123')
```

```
m = re.match('.*(\d)', 'Room No 123')
```

문장의 처음 위치에서 부터 검색

→ None

. 은 \n 을 제외한 모든 문자에 매칭

* 은 없거나 하나 이상인 것에 매칭 (greedy)

괄호는 매칭된 그룹을 구분합니다.

```
m.group(0)
```

```
m.group(1)
```

→ 'Room No 101'

→ '3'

```
re.match('.*?(\d)', 'Room No 123').group(1)
```

→ '1' # ?: non-greedy

Regular Expression

```
re.match('.*?(\\d+)', 'Room No 123').group(1)
```

```
m = re.match('(.*?)(\\d+)', 'Room No 123')
```

```
m.group(1)
```

```
m.group(2)
```

→ '123' # + 하나 또는 그 이상에 매칭

여러개의 괄호를 사용하여 그룹을 생성

→ 'Room No '

→ '123'

[...] 매칭 문자 리스트, [^...] 제외 문자 리스트

```
m = re.match('([a-zA-Z]*?)([0-9]+)', 'Room No 123')
```

→ 'Room No ', '123' # [0-9] == \\d

```
m = re.match('([\\d]*)([0-9]+)', 'Room No 123')
```

→ 'Room No ', '123' # greedy 방지 효과

```
m = re.search('[\\s]+', 'Room No 123')
```

→ 'Room' # \\s 공백문자와 매칭

```
m = re.search('[\\s]+$', 'Room No 123')
```

→ '123' # \$는 문자열 맨 끝을 의미

```
m = re.search('^([\\s]+)', 'Room No 123')
```

→ 'Room' # ^는 문자열 맨 앞을 의미

Regular Expression

```
re.findall('\d', 'Room No 123')
```

모든 패턴을 한꺼번에 찾기

→ ['1', '2', '3'] # 리스트를 반환

```
re.split('\s', 'Room No 123')
```

패턴으로 문자열 나누기

→ ['Room', 'No', '123']

```
re.sub('ws', ' : ', 'Room No 123')
```

일치하는 패턴을 다른 문자열로 바꾸기

→ 'Room : No : 123'

```
re.sub('([^\s]+)\s([^\s]+)', '\2 \1', 'Room No 123')
```

→ 'No Room 123'

Regular Expression

Demo

jupyter notebook

time

```
import time
```

```
time.time()
```

```
tm = time.localtime()
```

```
tm.tm_year
```

```
tm = time.localtime(time.time() - 60*60)
```

```
tm = time.gmtime()
```

```
time.strftime('%Y %m %d')
```

```
time.strftime('%Y %m %d',  
              time.localtime(time.time()-24*60*60))
```

```
time.sleep(1)
```

```
# C 계열 언어의 시간관련 함수 제공
```

```
→ 1468224815.533957 # Timestamp from Epoch
```

```
# C 언어의 struct tm 와 유사한 구조 리턴
```

```
→ time.struct_time(tm_year=2016, tm_mon=7,  
tm_mday=11, tm_hour=17, tm_min=24, tm_sec=21,  
tm_wday=0, tm_yday=193, tm_isdst=0)
```

```
→ 2016
```

```
# 1시간 전의 struct_time 인스턴스 생성
```

```
# GMT 기준의 struct_time 인스턴스 생성
```

```
→ '2016 07 11'
```

```
→ '2016 07 10'
```

```
# block 1 second
```

datetime

```
from datetime import date          # datetime 모듈에서 date 클래스만 импорт
some_day = date(2016, 7, 1)       # date 인스턴스 생성
today = date.today()

today.year                        → 2016   # month, day 속성 있음
today.strftime('%Y-%m-%d')        '2016-07-11'
```



```
from datetime import time          # 시, 분, 초
tm = time(17, 10, 1)              # time 인스턴스 생성
tm.hour                           → 17     # minute, second
tm.strftime('%H:%M:%S')           → '17:10:01'
```

datetime

```
from datetime import datetime
```

```
now = datetime.now()
```

```
now.timestamp()
```

```
now.year, now.month, now.day, now.hour,  
now.minute, now.sec
```

```
now.date()
```

```
now.time()
```

```
now.strftime('%Y-%m-%d %H:%M:%S')
```

```
now.isoformat()
```

```
# datetime 모듈에서 date 클래스만 импорт
```

```
→ datetime.datetime(2016, 7, 12, 10, 46, 22, 486760)
```

```
# time.time() 과 동일. timestamp from epoch
```

```
# datetime.date 오브젝트와 동일
```

```
# datetime.time 오브젝트와 동일
```

```
→ '2016-07-12 10:49:33'
```

```
→ '2016-07-12T10:49:33.766653'
```


time

Demo

jupyter notebook

file

```
open('파일이름', '모드')
```

\r' : 읽기 모드
\w' : 쓰기 모드, 이미 파일이 존재하면 재생성
\a' : 추가 모드, 파일이 없을 경우 새로 생성
\b' : 바이너리 모드
\t' : 텍스트 모드
\+' : 다른 모드에서 파일을 갱신할 수 있음.
 \r+' : 기존 파일에서 읽거나 쓰기
 \w+' : 기존 파일 재 생성후 읽거나 쓰기
 \a+' : 기존 파일에 추가하거나 읽기
디폴트는 '\rt'

```
f = open('myfile.txt', 'r')
```

```
text = f.read(byte=-1)
```

byte 만큼 혹은 파일 전체를 읽음

```
line = f.readline()
```

한 라인씩 읽어 들임

```
f.readlines(hint=-1)
```

hint 라인만큼 혹은 전체를 읽어 리스트로 반환함.

```
f.close()
```

file - seek

```
f = open('myfile.txt', 'w+')
```

```
n = f.write('this is a file')
```

저장된 텍스트 길이가 리턴됨

```
f.seek(0)
```

파일의 맨 처음으로 이동

```
seek(offset, from)
```

offset: 음수는 'b' 모드로 열어야 가능
from: 0(파일시작), 1(현재), 2(파일끝)

```
f = open('myfile.txt', 'rb+')
```

```
f.seek(-10, 2)
```

파일의 끝에서 10바이트 뒤로 이동

```
f.close()
```

```
with open('myfile.txt') as f:
```

```
    for line in f:
```

```
        print(line)
```

with 문은 오브젝트가 블록을 종료할 때 지정된 작업을 수행하도록 합니다. 파일의 경우 with 블록에서 익셉션이 발생하더라도 close()가 되는 것이 보장됩니다.

CSV

```
import csv
```

```
# csv 스탠다드 모듈 임포트
```

```
with open('sample.csv', 'w') as f:
```

```
    writer = csv.writer(f)
```

```
    writer.writerows([['age', 'blood'], [10, 'A'], [20, 'B']])
```

```
# sample.csv
```

```
# age,blood
```

```
# 10,A
```

```
# 20,B
```

```
with open('sample.csv') as f:
```

```
    reader = csv.reader(f)
```

```
    for row in reader:
```

```
        print(row)
```

```
→
```

```
['age', 'blood']
```

```
['10', 'A']
```

```
['20', 'B']
```

```
with open('sample.csv') as f:
```

```
    reader = csv.DictReader(f)
```

```
    for row in reader:
```

```
        print(row['age'], row['blood'])
```

```
→
```

```
10 A
```

```
20 B
```

json

```
import json
```

```
jobj = json.loads('{"age": 10, "blood": "A"}')  
jobj.__class__  
jobj = json.loads(' [{"age": 10, "blood": "A"},  
                    {"age": 20, "blood": "B"} ]')
```

```
jobj.__class__
```

```
json.dumps(jobj)
```

```
jobj = json.load(f)
```

```
json.dump(jobj, f)
```

```
# json.org  
# json 스탠다드 모듈 임포트
```

```
# 딕셔너리나 리스트 반환  
→ dict
```

```
→ list
```

```
# 스트링 리턴  
→ '[{"blood": "B", "age": 10}, {"blood":  
    "B", "age": 20}]'
```

```
# 파일핸들 f에서 Json 스트링을 읽어 파이썬 오브젝  
트로 변경
```

```
# jobj 오브젝트를 직렬화하여 파일핸들 f에 기록
```

ujson

```
import ujson
```

```
# https://github.com/esnme/ultrajson
```

```
# 서드파티 json 패키지
```

```
$ pip install ujson
```

```
# pip로 설치
```

```
jobj = ujson.loads('...')  
ujson.load(f)
```

```
# Json 스트링에서 파이썬 오브젝트로 변환
```

```
# 파일 핸들에서 Json 스트링을 읽어 파이썬 오브젝트로 변환
```

```
jstr = ujson.dumps(jobj)  
ujson.dump(jobj, f)
```

```
# 파이썬 오브젝트를 Json 스트링으로 변환
```

```
# 파이썬 오브젝트를 Json 스트링으로 변환하여 파일로 저장
```

ujson

```
d = { 'first_name': 'Guido',  
      'second_name': 'Rossum',  
      'titles': ['BDFL', 'Developer'], }
```

```
%timeit -n 100 json.dumps(d)  
100 loops, best of 3: 6.07 µs per loop
```

ujson이 대략 5배 이상 빠름

```
%timeit -n 100 ujson.dumps(d)  
100 loops, best of 3: 1.02 µs per loop
```

```
s = json.dumps(d)
```

```
%timeit -n 100 json.loads(s)  
100 loops, best of 3: 5.01 µs per loop
```

ujson 이 약 4배 정도 빠름

```
%timeit -n 100 ujson.loads(s)  
100 loops, best of 3: 1.33 µs per loop
```

file

Demo

jupyter notebook

Setup & Example

Installer

← → ↻ https://www.continuum.io/downloads#_windows

If you don't have time or disk space for the entire distribution, try [Miniconda](#), which contains only conda and Python. Then install just the individual packages you want through the conda command.

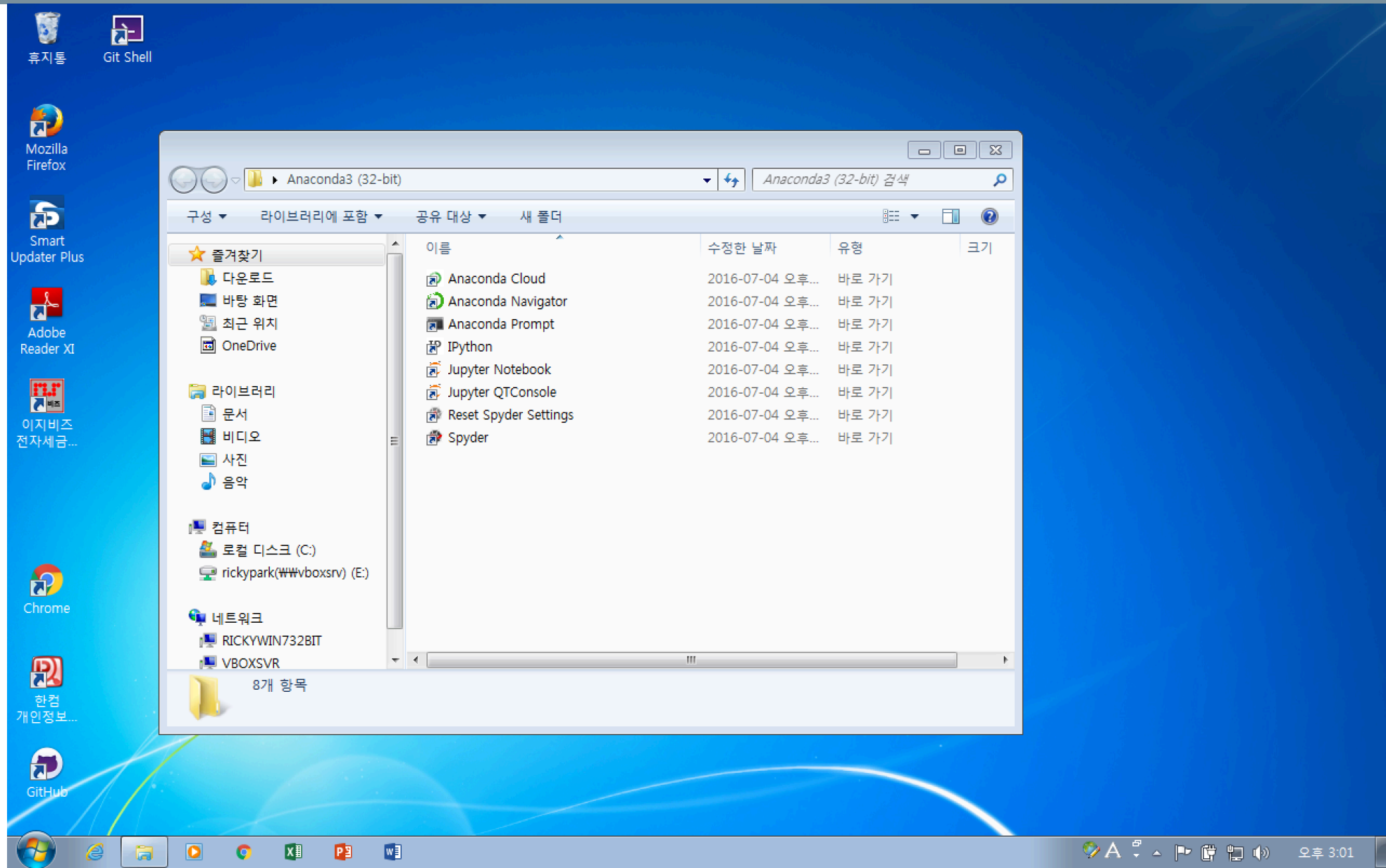
Anaconda for Windows

PYTHON 2.7	PYTHON 3.5
<div>WINDOWS 64-BIT GRAPHICAL INSTALLER</div> <div>340M</div>	<div>WINDOWS 64-BIT GRAPHICAL INSTALLER</div> <div>351M</div>
<div>Windows 32-bit Graphical Installer</div> <div>285M</div>	<div>Windows 32-bit Graphical Installer</div> <div>292M</div>
Behind a firewall? Use these zipped Windows installers .	

Windows Anaconda Installation


1. Download the graphical installer.
2. Optional: [Verify data integrity with SHA-256](#).
3. Double-click the .exe file to install Anaconda and follow the instructions on the screen.


Anaconda




Github Download

GitHub - rickiepark/pytho x

← → ↻  GitHub, Inc. [US] <https://github.com/rickiepark/python-tutorial>

 Personal Open source Business Explore Pricing Blog Support This repository Search Sign in Sign up


 rickiepark / python-tutorial Watch 1 Star 0 Fork 0


[Code](#) Issues 0 Pull requests 0 Pulse Graphs

python code & jupyter notebooks for learning

6 commits 1 branch 0 releases 1 contributor

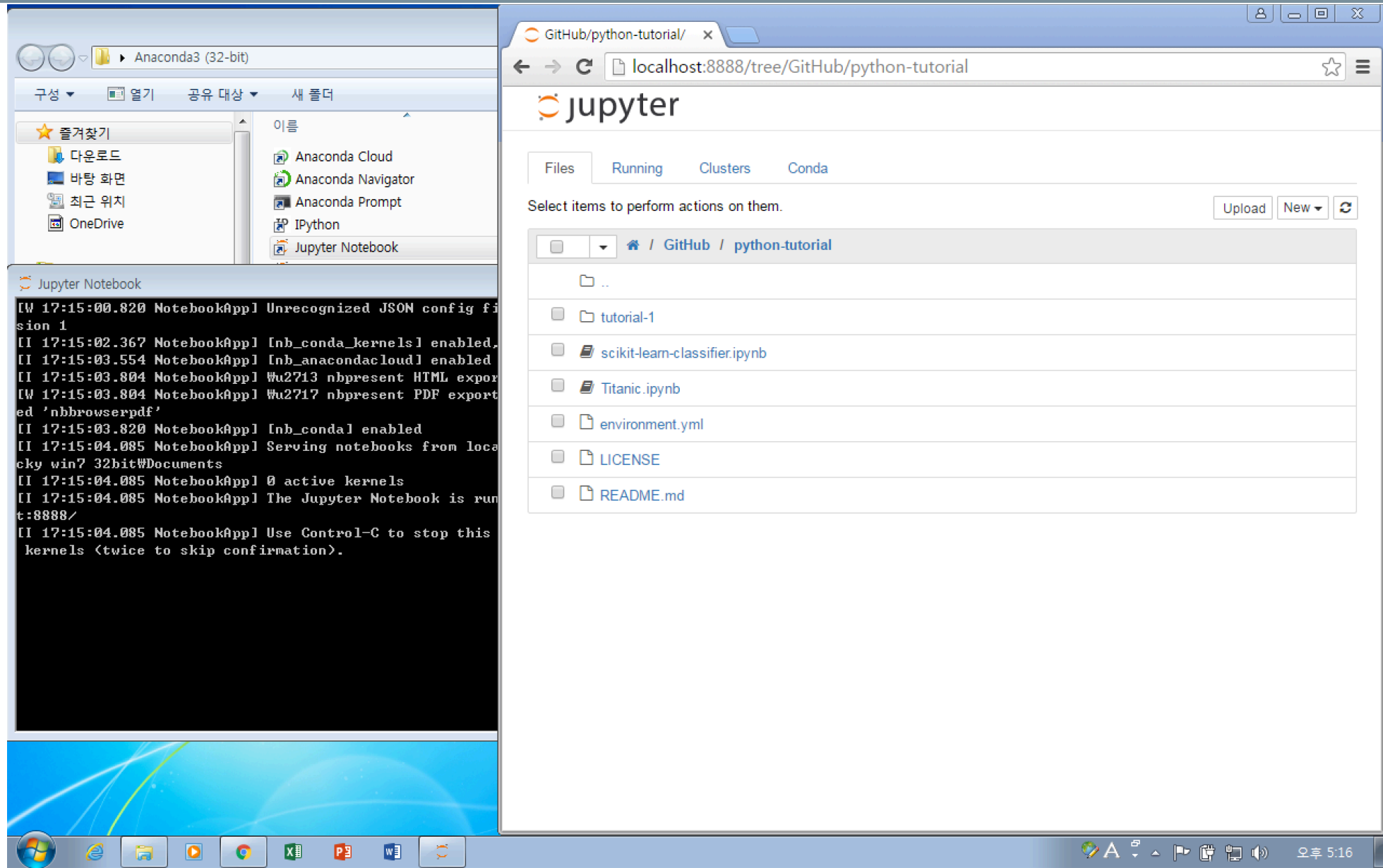
Branch: master New pull request Find file Clone or download

 rickiepark 테스트 코드 정리 Latest commit 8dc6907 7 days ago

 tutorial-1 테스트 코드 정리 7 days ago

내 문서 밑으로 압축 해제

Jupyter notebook



Word Histogram

구텐베르크 프로젝트에 공개된 정글북 소설의 텍스트를 사용하여

- 1) 단어의 횟수를 카운트(히스토그램) 한다.
: re.findall 이용
- 2) 만들어진 히스토그램 정보를 json 형태로 파일로 저장한다.
- 3) 고유한 단어의 수를 출력한다.
- 4) 전체 단어의 수를 출력한다.
- 5) 가장 자주 나타나는 단어 상위 10개를 출력한다.
: list 의 sort 메소드 이용
- 6) 단어의 카운트를 반영하여 랜덤하게 하나를 추출한다.
: random 모듈의 choice 메소드 이용

Q&A

Any Question: haesunrpark@gmail.com

github URL: <https://github.com/rickiepark/python-tutorial>

This slide available at
<http://tensorflowkorea.wordpress.com>