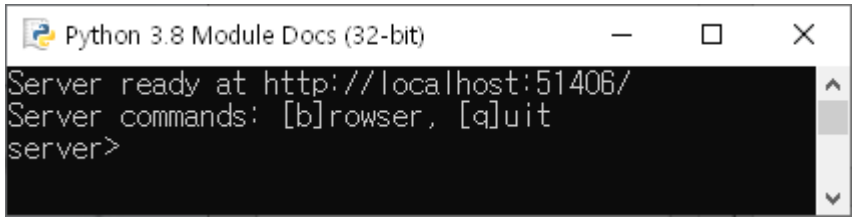
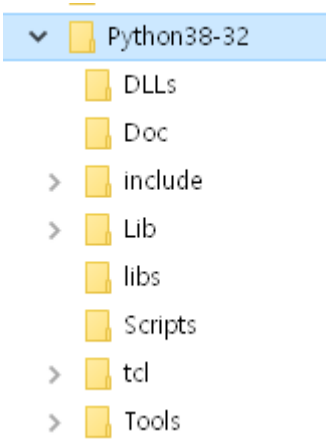
Module Docs
설치 모듈 문서

서버 프로그램, 브라우저에서 조회

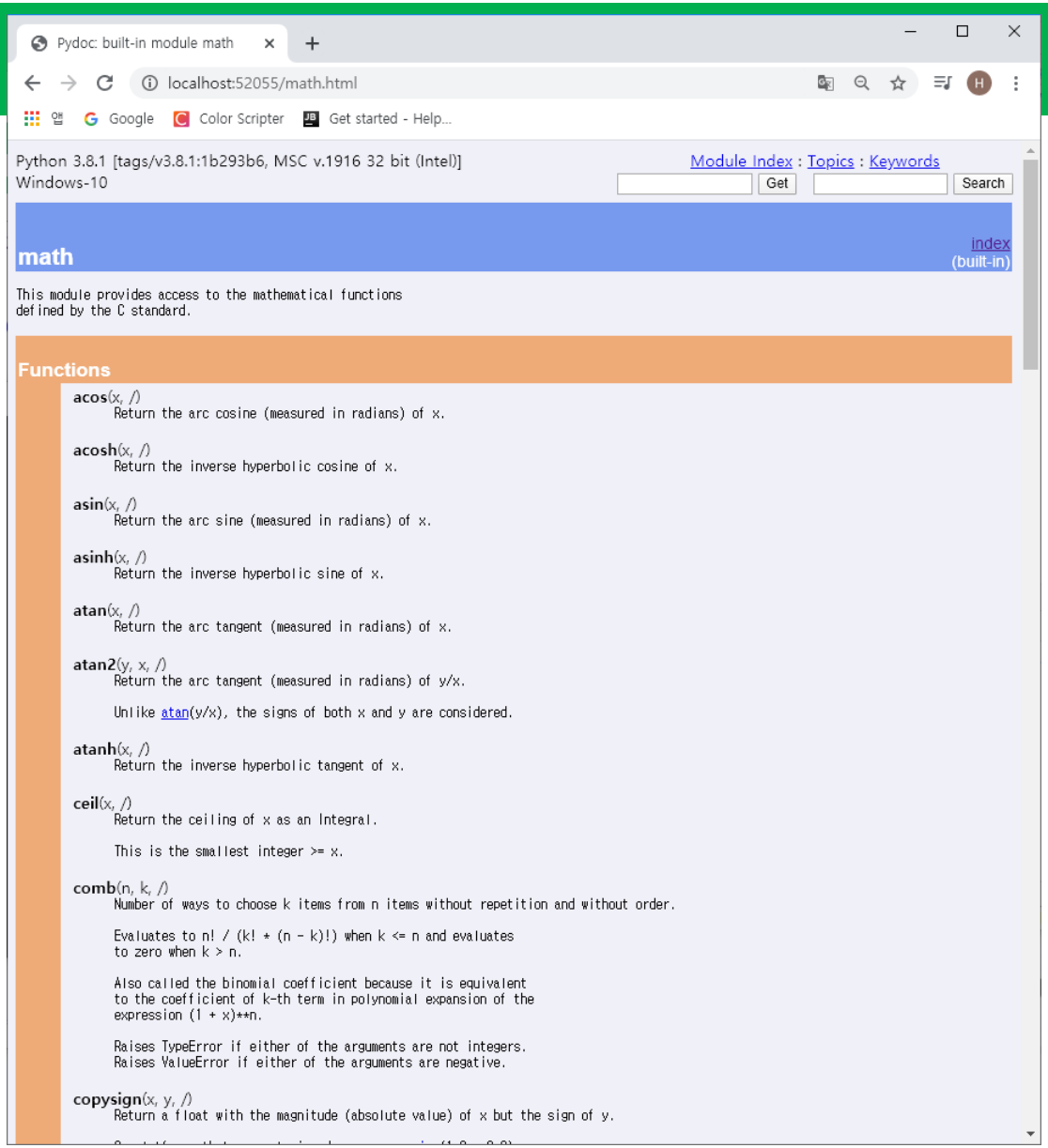


- **Built-in Modules**
  - 빌트인 모듈
- **D:₩Python38-32₩DLLs**
- **D:₩Python38-32₩lib**
  - 표준 모듈과 패키지
- **D:₩Python38-32₩lib₩site-packages**
  - 설치된 외부 모듈



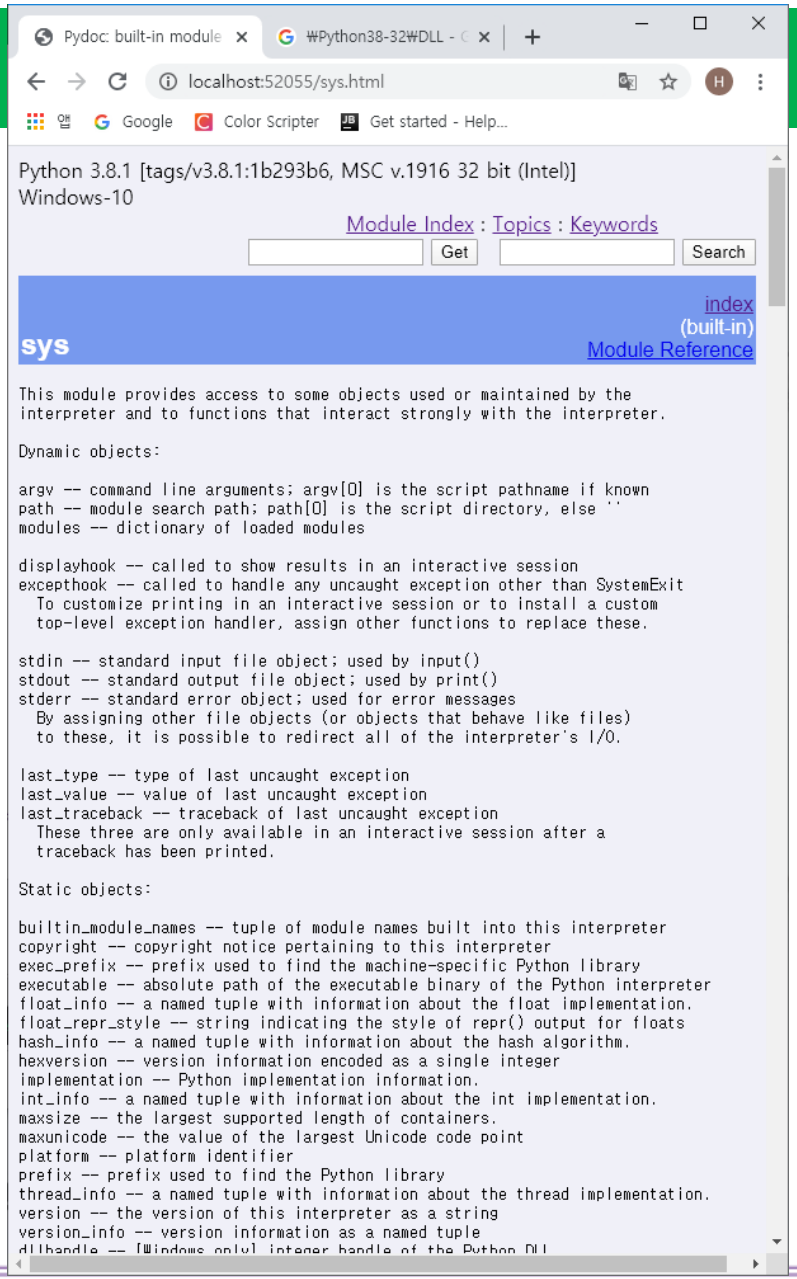
주요 빌트인 모듈

- **math**



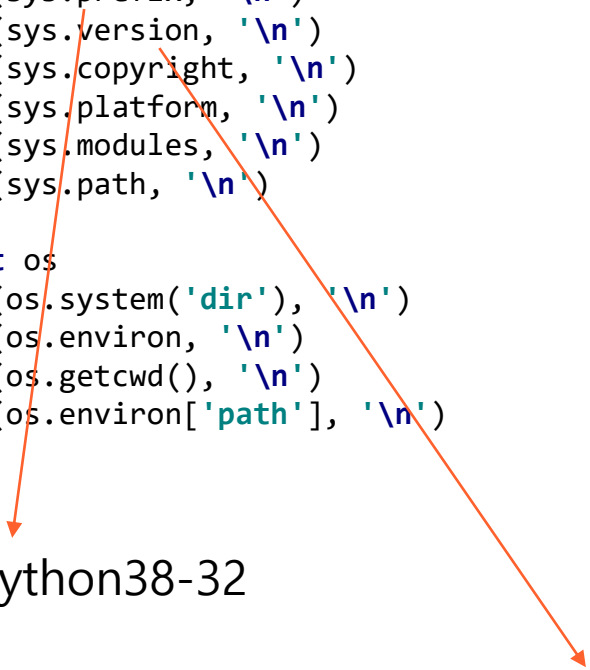
주요 빌트인 모듈

- **sys**


Pydoc: built-in module
localhost:52055/sys.html
Google
Color Scripter
Get started - Help...
Python 3.8.1 [tags/v3.8.1:1b293b6, MSC v.1916 32 bit (Intel)]
Windows-10
Module Index : Topics : Keywords
Get
Search
index
(built-in)
sys
Module Reference
This module provides access to some objects used or maintained by the
interpreter and to functions that interact strongly with the interpreter.
Dynamic objects:
argv -- command line arguments; argv[0] is the script pathname if known
path -- module search path; path[0] is the script directory, else ''
modules -- dictionary of loaded modules
displayhook -- called to show results in an interactive session
excepthook -- called to handle any uncaught exception other than SystemExit
To customize printing in an interactive session or to install a custom
top-level exception handler, assign other functions to replace these.
stdin -- standard input file object; used by input()
stdout -- standard output file object; used by print()
stderr -- standard error object; used for error messages
By assigning other file objects (or objects that behave like files)
to these, it is possible to redirect all of the interpreter's I/O.
last_type -- type of last uncaught exception
last_value -- value of last uncaught exception
last_traceback -- traceback of last uncaught exception
These three are only available in an interactive session after a
traceback has been printed.
Static objects:
builtin_module_names -- tuple of module names built into this interpreter
copyright -- copyright notice pertaining to this interpreter
exec_prefix -- prefix used to find the machine-specific Python library
executable -- absolute path of the executable binary of the Python interpreter
float_info -- a named tuple with information about the float implementation.
float_repr_style -- string indicating the style of repr() output for floats
hash_info -- a named tuple with information about the hash algorithm.
hexversion -- version information encoded as a single integer
implementation -- Python implementation information.
int_info -- a named tuple with information about the int implementation.
maxsize -- the largest supported length of containers.
maxunicode -- the value of the largest Unicode code point
platform -- platform identifier
prefix -- prefix used to find the Python library
thread_info -- a named tuple with information about the thread implementation.
version -- the version of this interpreter as a string
version_info -- version information as a named tuple

모듈 sys os 관련 코드

```
import sys
print(sys.prefix, '\n')
print(sys.version, '\n')
print(sys.copyright, '\n')
print(sys.platform, '\n')
print(sys.modules, '\n')
print(sys.path, '\n')

import os
print(os.system('dir'), '\n')
print(os.environ, '\n')
print(os.getcwd(), '\n')
print(os.environ['path'], '\n')
```

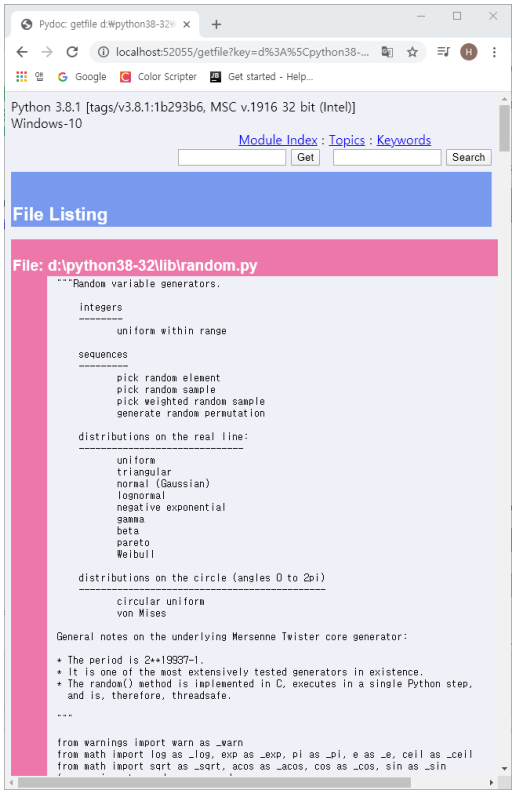


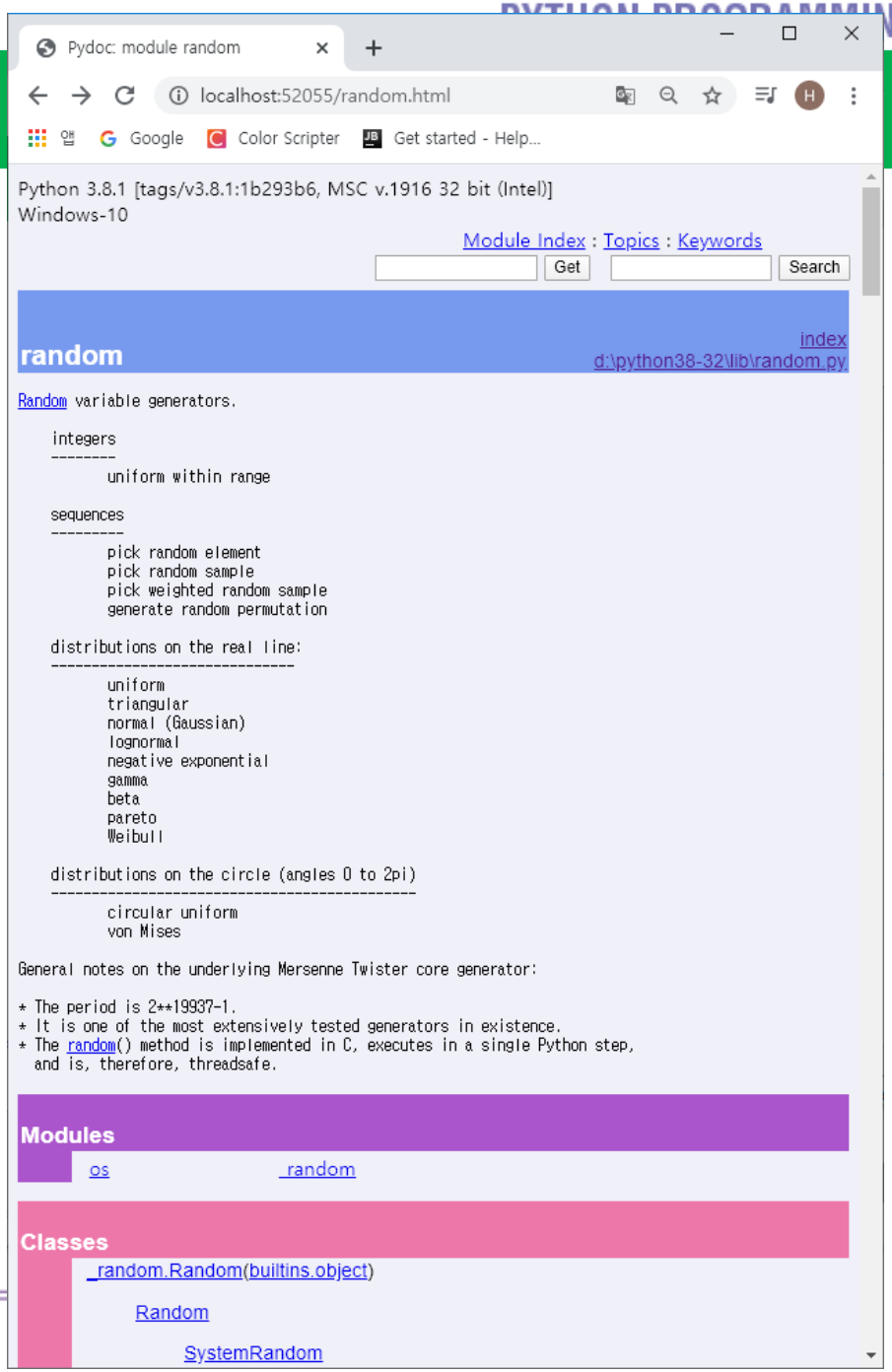
D:₩Python38-32

3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916 32 bit (Intel)]

주요 표준 모듈 random

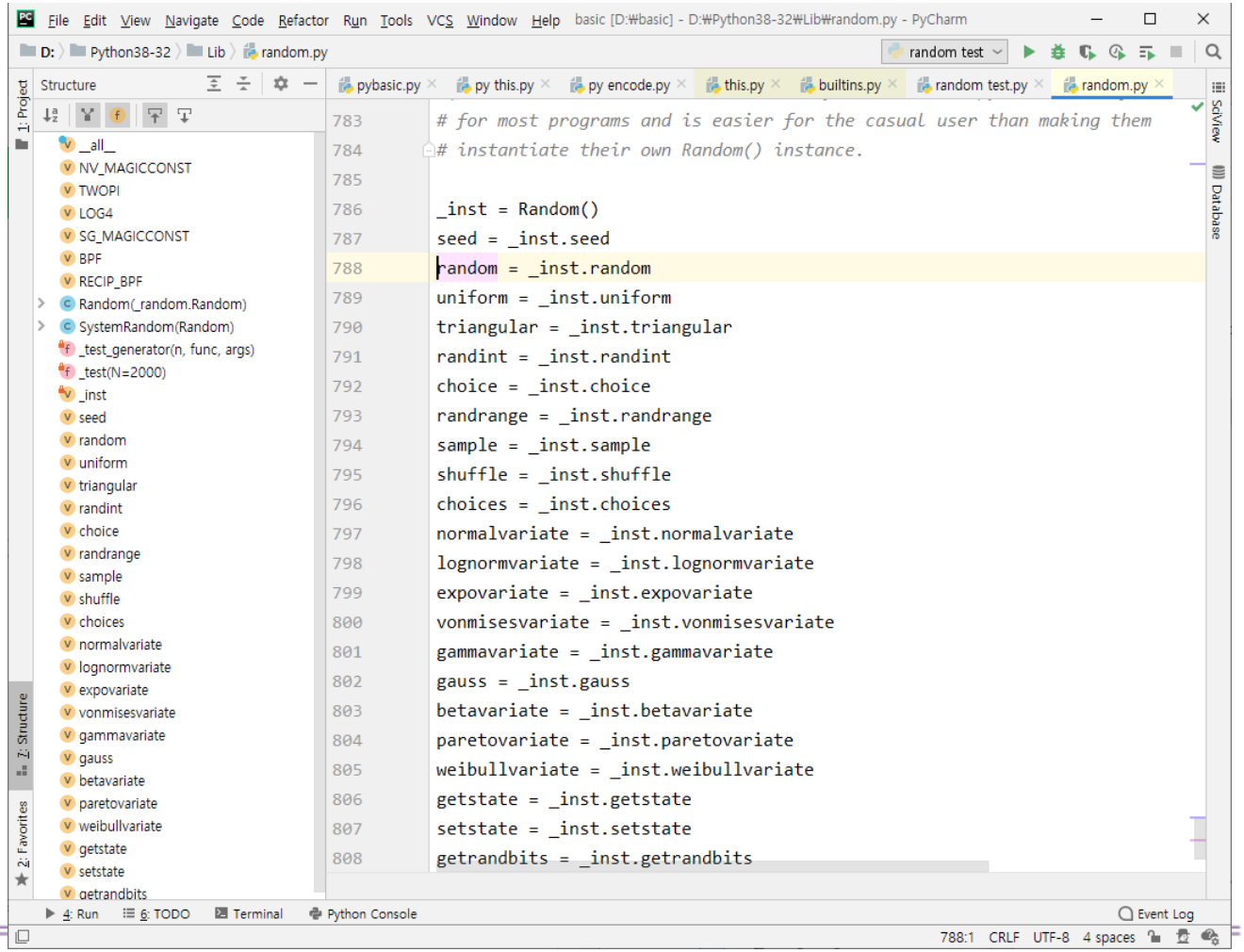
- **D:₩Python38-32₩lib**
  - random
    - **파일 random.py**





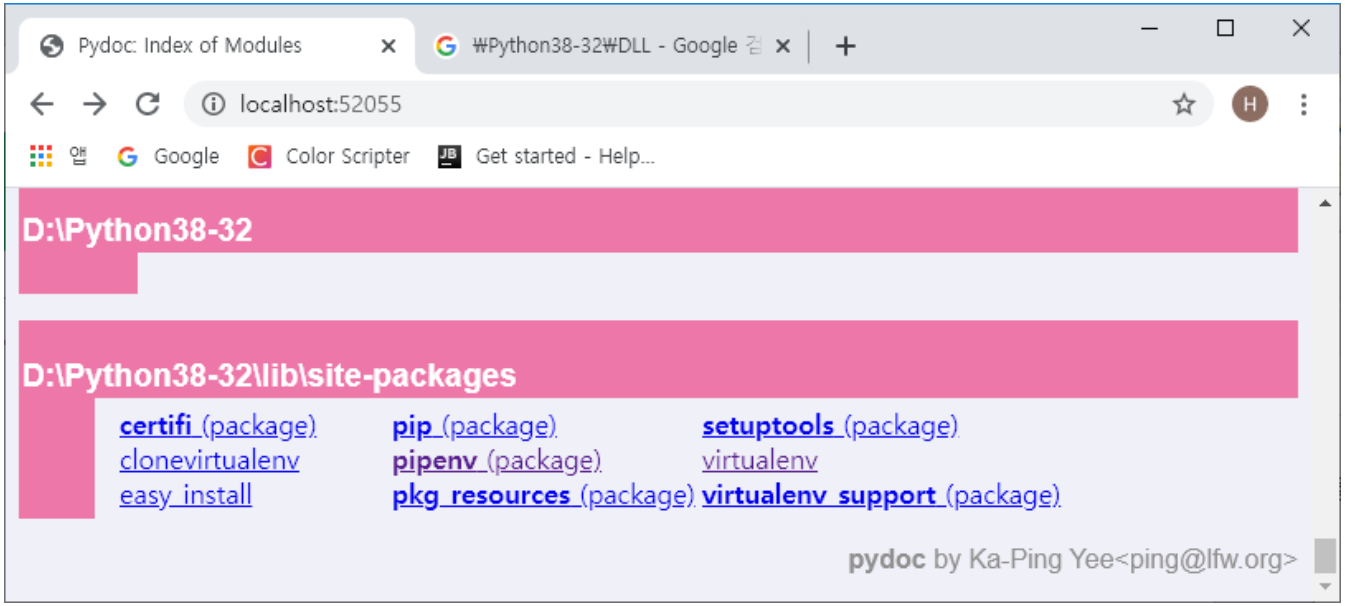
파일 random.py

- 파이참에서 직접 보기



설치된 외부 모듈

- **D:₩Python38-32₩lib₩site-packages**



Python
Cheat Sheet

Beginner's Python Cheat Sheet

Variables and Strings

Variables are used to store values. A string is a series of characters, surrounded by single or double quotes.

Hello world

```
print("Hello world!")
```

Hello world with a variable

```
msg = "Hello world!"
print(msg)
```

Concatenation (combining strings)

```
first_name = 'albert'
last_name = 'einstein'
full_name = first_name + ' ' + last_name
print(full_name)
```

Lists

A list stores a series of items in a particular order. You access items using an index, or within a loop.

Make a list

```
bikes = ['trek', 'redline', 'giant']
```

Get the first item in a list

```
first_bike = bikes[0]
```

Get the last item in a list

```
last_bike = bikes[-1]
```

Looping through a list

```
for bike in bikes:
    print(bike)
```

Adding items to a list

```
bikes = []
bikes.append('trek')
bikes.append('redline')
bikes.append('giant')
```

Making numerical lists

```
squares = []
for x in range(1, 11):
    squares.append(x**2)
```

Lists (cont.)

List comprehensions

```
squares = [x**2 for x in range(1, 11)]
```

Slicing a list

```
finishers = ['sam', 'bob', 'ada', 'bea']
first_two = finishers[:2]
```

Copying a list

```
copy_of_bikes = bikes[:]
```

Tuples

Tuples are similar to lists, but the items in a tuple can't be modified.

Making a tuple

```
dimensions = (1920, 1080)
```

If statements

If statements are used to test for particular conditions and respond appropriately.

Conditional tests

```
equals            x == 42
not equal         x != 42
greater than      x > 42
  or equal to     x >= 42
less than         x < 42
  or equal to     x <= 42
```

Conditional test with lists

```
'trek' in bikes
'surly' not in bikes
```

Assigning boolean values

```
game_active = True
can_edit = False
```

A simple if test

```
if age >= 18:
    print("You can vote!")
```

If-elif-else statements

```
if age < 4:
    ticket_price = 0
elif age < 18:
    ticket_price = 10
else:
    ticket_price = 15
```

Dictionaries

Dictionaries store connections between pieces of information. Each item in a dictionary is a key-value pair.

A simple dictionary

```
alien = {'color': 'green', 'points': 5}
```

Accessing a value

```
print("The alien's color is " + alien['color'])
```

Adding a new key-value pair

```
alien['x_position'] = 0
```

Looping through all key-value pairs

```
fav_numbers = {'eric': 17, 'ever': 4}
for name, number in fav_numbers.items():
    print(name + ' loves ' + str(number))
```

Looping through all keys

```
fav_numbers = {'eric': 17, 'ever': 4}
for name in fav_numbers.keys():
    print(name + ' loves a number')
```

Looping through all the values

```
fav_numbers = {'eric': 17, 'ever': 4}
for number in fav_numbers.values():
    print(str(number) + ' is a favorite')
```

User input

Your programs can prompt the user for input. All input is stored as a string.

Prompting for a value

```
name = input("What's your name? ")
print("Hello, " + name + "!")
```

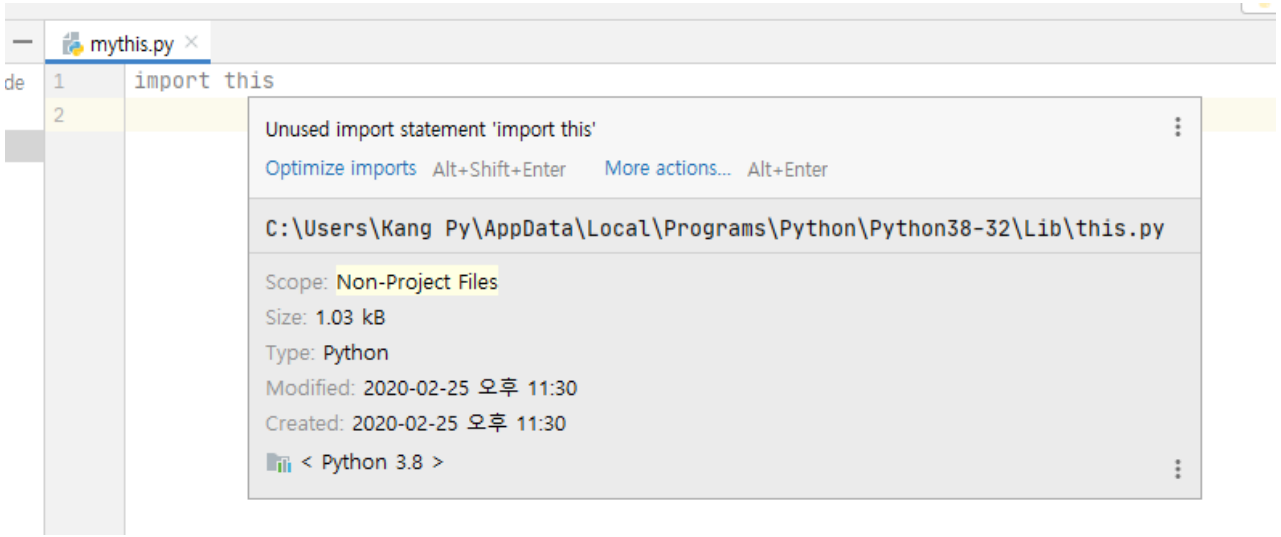
Prompting for numerical input

```
age = input("How old are you? ")
age = int(age)

pi = input("What's the value of pi? ")
pi = float(pi)
```

파이썬의 특징: 스트링, 리스트, 딕셔너리, 컴프리헨션

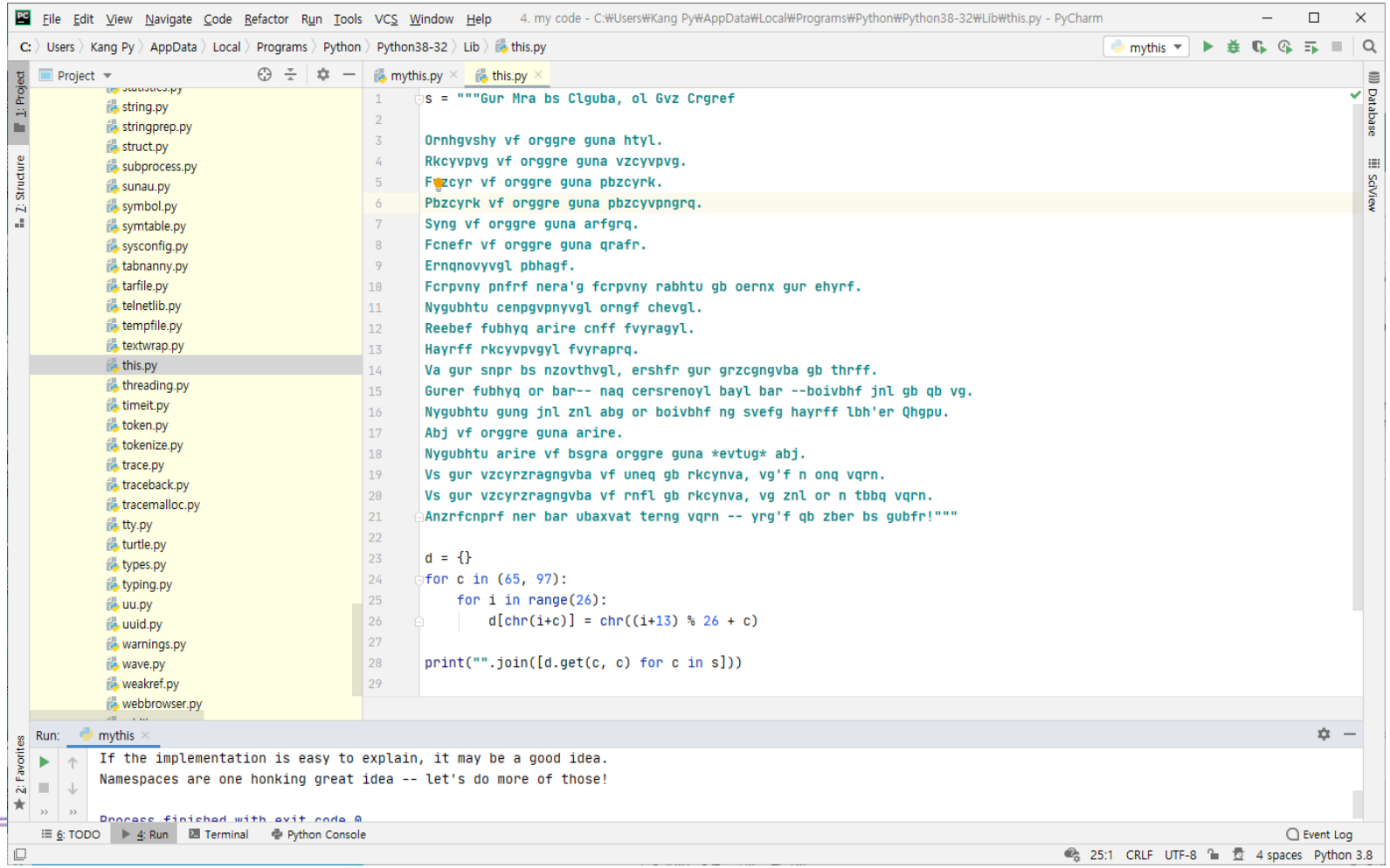
import this

- 실행



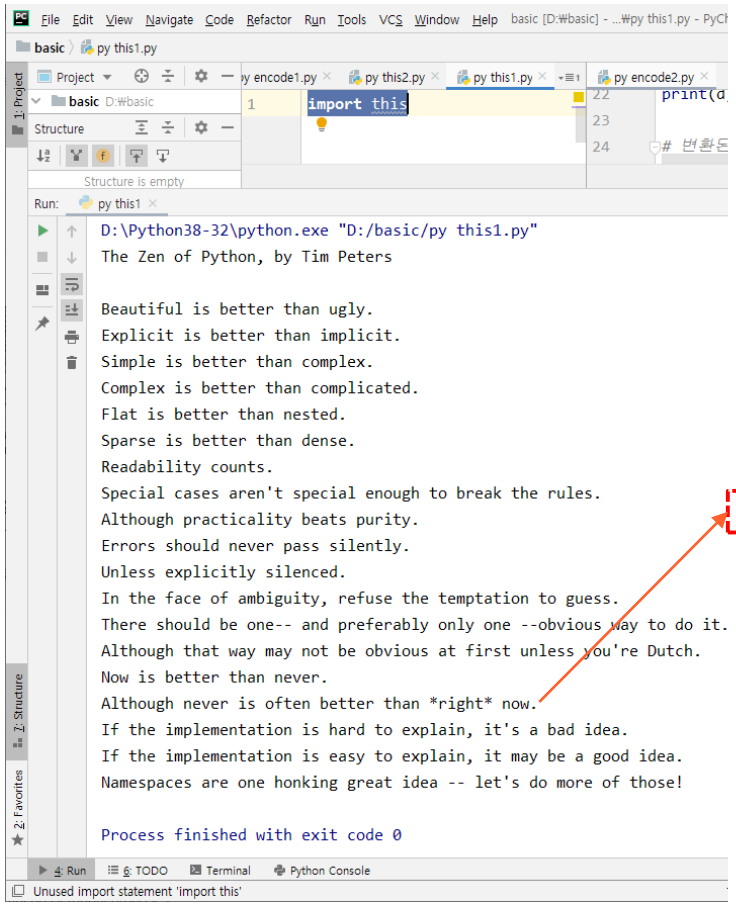
파일 this.py

- 파일
  - C:₩Users₩사용자₩AppData₩Local₩Programs₩Python₩Python38-32₩Lib₩this.py



import this

- 파이썬 철학




```
s = """Gur Mra bs Clguba, ol Gvz Crgref

Ornhgvshy vf orggre guna htyl.
Rkcyvpvg vf orggre guna vzcyvpvg.
Fvzcyr vf orggre guna pbzcyrk.
Pbzcyrk vf orggre guna pbzcyvpngrq.
Syng vf orggre guna arfgrq.
Fcnefr vf orggre guna qrafr.
Ernqnovyvgl pbhagf.
Fcrpvny pnfrf nera'g fcrpvny rabhtu gb oernx gur ehyrf.
Nygubhtu cenpgvpnyvgl orngf chevgl.
Reebef fubhyq arire cnff fvyragyl.
Hayrff rkcyvpvgyl fvyraprq.
Va gur snpr bs nzovthvgl, ershfr gur grzcgngvba gb thrff.
Gurer fubhyq or bar-- naq cersrenoyl bayl bar --boivbhf jnl gb qb vg.
Nygubhtu gung jnl znl abg or boivbhf ng svefg hayrff lbh'er Qhgpu.
Abj vf orggre guna arire.
Nygubhtu arire vf bsgra orggre guna *evtug* abj.
Vs gur vzcyrzragngvba vf uneq gb rkcynva, vg'f n onq vqrn.
Vs gur vzcyrzragngvba vf rnfl gb rkcynva, vg znl or n tbbq vqrn.
Anzrfcnprf ner bar ubaxvat terng vqrn -- yrg'f qb zber bs gubfr!"""

d = {}
for c in (65, 97):
    for i in range(26):
        d[chr(i+c)] = chr((i+13) % 26 + c)

print("".join([d.get(c, c) for c in s]))

type(d)
```

문자열 str.join()

- **s.join(<iterable>)**
  - 반복 가능한 문자열을 연결
  - 문자열 s를 이터러블의 항목 사이에 붙인 문자열 반환

```
print('a,b,c'.split(','))

print('this is my string'.split())

s = "this is my string"
print(s.split(maxsplit=1))

print('a' + 'b' + 'c')
print('do' * 2)

strings = ['do', 're', 'mi']
print(','.join(strings))

print('->'.join('string'))

print(''.join(['p', 'y', 't', 'h', 'o', 'n']))
print(', '.join(['foo', 'bar', 'baz', 'qux']))

print(list('corge'))
print(':'.join(list('corge')))
print(':'.join('corge'))

print('---'.join(['foo', str(23), 'bar']))
```

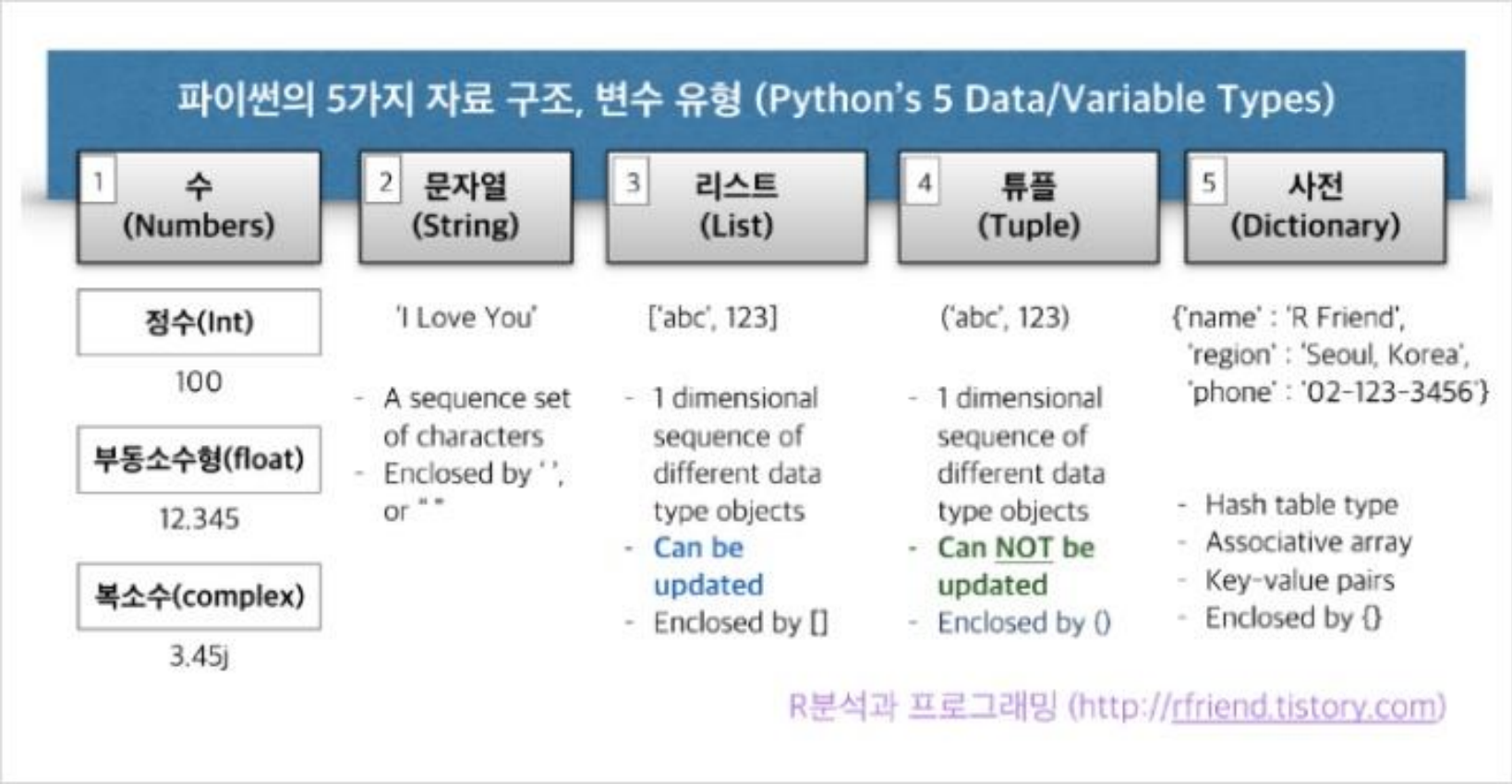
문자열, 컴프리헨션

- 문자열
  - join()
- 리스트
- 컴프리헨션

```
D:₩Python38-32₩python.exe "D:/basic/py encode1.py"
Programming is an Art!
Uwtlwfrrnsl%nx%fs%Fwy&
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q',
'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
['U', 'w', 't', 'l', 'w', 'f', 'r', 'r', 'n', 's', 'l', '%', 'n', 'x', '%', 'f', 's',
'%', 'F', 'w', 'y', '&']
python
python
p-y-t-h-o-n
Uwtlwfrrnsl%nx%fs%Fwy&
Uwtlwfrrnsl%nx%fs%Fwy&
Programming is an Art!

Process finished with exit code 0
```

```python
s = 'Programming is an Art!'
print(s)

# 문자에 각각 5를 더하여 문자열을 출력
key = 5
for c in s:
    print(chr(ord(c) + key), end='')
print()

# 컴프리헨션 이해
print([i for i in range(10)])
print([chr(i) for i in range(ord('a'), ord('a') + 26)])

# 문자에 각각 5를 더한 문자 리스트를 출력
t = [chr(ord(c) + key) for c in s]
print(t)

# 문자열 str.join() 함수 이해
print(''.join('python'))
print(''.join(list('python')))
print('-'.join(list('python')))

# 문자에 각각 5를 더한 문자열 출력
e = '';
e = e.join([chr(ord(c) + key) for c in s])
print(e)

# 문자에 각각 5를 더한 문자열을 한 줄에 출력
print(''.join([chr(ord(c) + key) for c in s]))

e = '';
e = e.join([chr(ord(c) + key) for c in s])

# 비뀐 문자열을 다시 반대로 문자에 각각 5를 뺀 문자열을 한 줄에 출력
key = -key
print(''.join([chr(ord(c) + key) for c in e]))
```

파이썬 주요 자료


파이썬의 5가지 자료 구조, 변수 유형 (Python's 5 Data/Variable Types)
1 수 (Numbers)
2 문자열 (String)
3 리스트 (List)
4 튜플 (Tuple)
5 사전 (Dictionary)
정수(Int)
100
부동소수형(float)
12.345
복소수(complex)
3.45j
'I Love You'
- A sequence set of characters
- Enclosed by ' ', or " "
['abc', 123]
- 1 dimensional sequence of different data type objects
- Can be updated
- Enclosed by []
('abc', 123)
- 1 dimensional sequence of different data type objects
- Can NOT be updated
- Enclosed by ()
{'name' : 'R Friend', 'region' : 'Seoul, Korea', 'phone' : '02-123-3456'}
- Hash table type
- Associative array
- Key-value pairs
- Enclosed by {}
R분석과 프로그래밍 (http://rfriend.tistory.com)

딕셔너리 생성

The bracket method

```
data = {
      "beer_data": beers,
      "brewery_data": breweries
}
```

Key Value

The dict( ) method, option 1

```
data2 = dict(
      beer_data=beers,
      brewery_data=breweries
)
```

Key Value

The dict( ) method, option 2

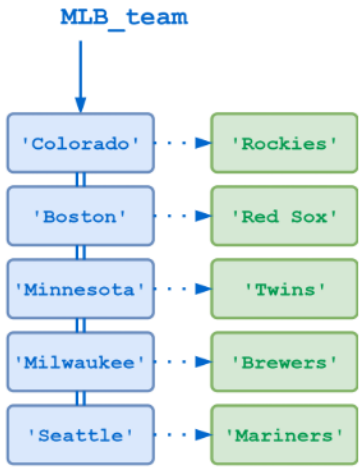
```
tuple_list = [
("brewery_data", breweries),
("beer_data", beers)
]
```

Key Value

```
data3 = dict(tuple_list)
```

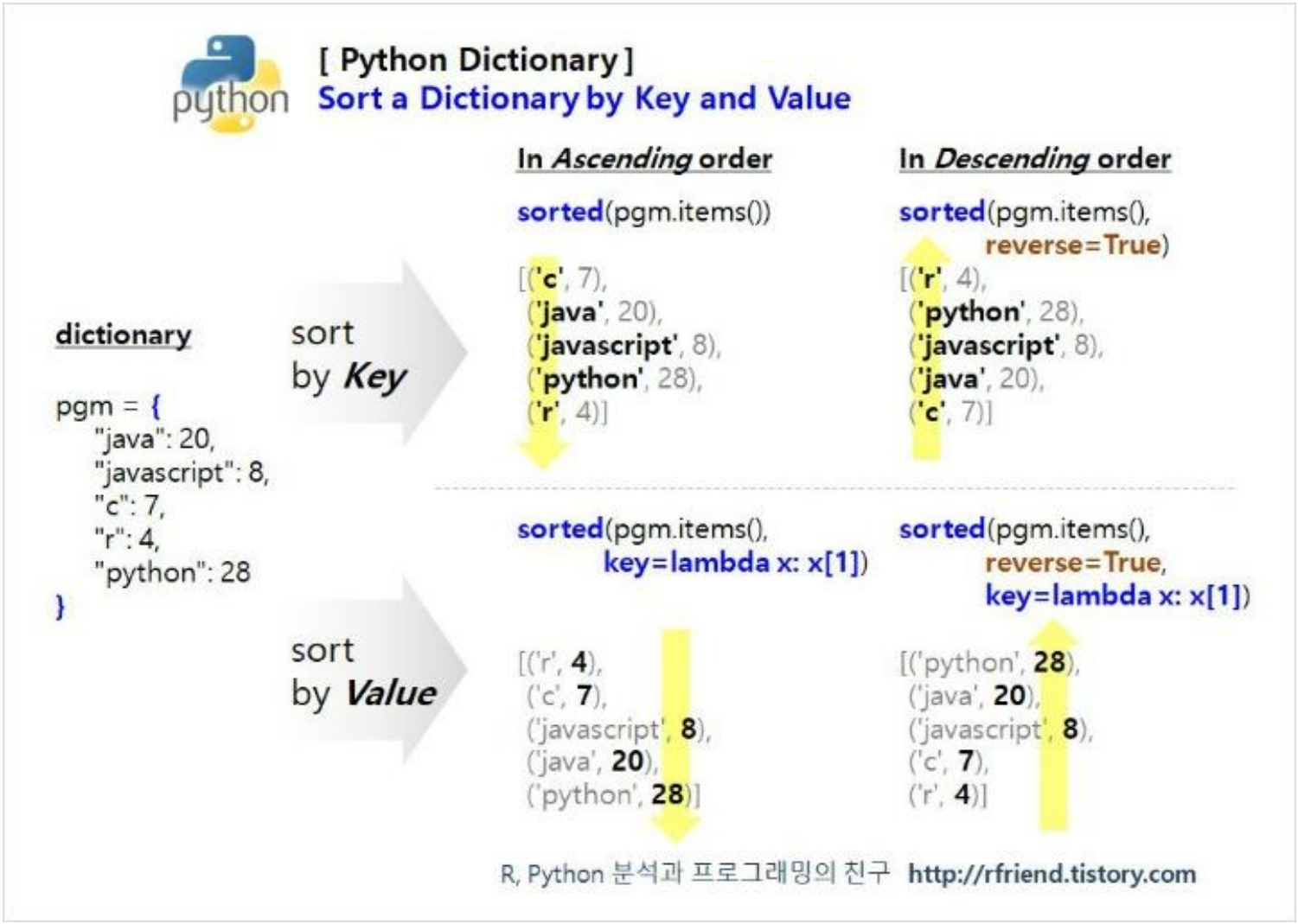
Python >>>

```
>>> MLB_team = {
...     'Colorado' : 'Rockies',
...     'Boston'   : 'Red Sox',
...     'Minnesota': 'Twins',
...     'Milwaukee': 'Brewers',
...     'Seattle'  : 'Mariners'
... }
```



Dictionary Mapping Location to MLB Team

딕셔너리의 정렬


[Python Dictionary]
Sort a Dictionary by Key and Value
dictionary
pgm = {
"java": 20,
"javascript": 8,
"c": 7,
"r": 4,
"python": 28
}
sort by Key
In Ascending order
sorted(pgm.items())
[('c', 7),
('java', 20),
('javascript', 8),
('python', 28),
('r', 4)]
In Descending order
sorted(pgm.items(), reverse=True)
[('r', 4),
('python', 28),
('javascript', 8),
('java', 20),
('c', 7)]
sort by Value
sorted(pgm.items(), key=lambda x: x[1])
[('r', 4),
('c', 7),
('javascript', 8),
('java', 20),
('python', 28)]
sorted(pgm.items(), reverse=True, key=lambda x: x[1])
[('python', 28),
('java', 20),
('javascript', 8),
('c', 7),
('r', 4)]
R, Python 분석과 프로그래밍의 친구 http://rfriend.tistory.com

딕셔너리를 통한 인코딩

- **리스트 딕셔너리**
  - 원소 쉽게 생성

```
D:₩Python38-32₩python.exe "D:/basic/py encode2.py"
Programming is Art!
{0: '1', 1: '2', 2: '3', 3: '4', 4: '5', 5: '6', 6: '7', 7: '8', 8: '9', 9: '10'}
{'A': 'A', 'B': 'B', 'C': 'C', 'D': 'D', 'E': 'E', 'F': 'F', 'G': 'G', 'H': 'H',
'I': 'I', 'J': 'J', 'K': 'K', 'L': 'L', 'M': 'M', 'N': 'N', 'O': 'O', 'P': 'P', 'Q':
'Q', 'R': 'R', 'S': 'S', 'T': 'T', 'U': 'U', 'V': 'V', 'W': 'W', 'X': 'X', 'Y':
'Y', 'Z': 'Z'}
{'A': 'D', 'B': 'E', 'C': 'F', 'D': 'G', 'E': 'H', 'F': 'I', 'G': 'J', 'H': 'K', 'I':
'L', 'J': 'M', 'K': 'N', 'L': 'O', 'M': 'P', 'N': 'Q', 'O': 'R', 'P': 'S', 'Q':
'T', 'R': 'U', 'S': 'V', 'T': 'W', 'U': 'X', 'V': 'Y', 'W': 'Z', 'X': 'A', 'Y':
'B', 'Z': 'C', 'a': 'd', 'b': 'e', 'c': 'f', 'd': 'g', 'e': 'h', 'f': 'i', 'g': 'j', 'h':
'k', 'i': 'l', 'j': 'm', 'k': 'n', 'l': 'o', 'm': 'p', 'n': 'q', 'o': 'r', 'p': 's', 'q':
't', 'r': 'u', 's': 'v', 't': 'w', 'u': 'x', 'v': 'y', 'w': 'z', 'x': 'a', 'y': 'b', 'z':
'c'}
Surjudpplqj lv Duw!
Surjudpplqj lv Duw!
Programming is Art!

Process finished with exit code 0
```

```python
s = 'Programming is Art!'
print(s)

# 딕셔너리 이해
d = {}
for i in range(10):  # 알파벳 인코딩 딕셔너리 생성
  d[i] = str(i+1)
print(d)

d = {}
for i in range(ord('A'), ord('A') + 26):  # 알파벳 인코딩 딕셔너리 생성
  d[chr(i)] = chr(i)
print(d)

# 인코딩을 위한 키, 값의 딕셔너리 생성
key = 3
d = {} # 딕셔너리 만들기
for c in (65, 97): # 알파벳 A, a에서 부터 시작하여
  for i in range(26): # 알파벳 인코딩 딕셔너리 생성
    d[chr(i+c)] = chr((i + key) % 26 + c)

print(d)

# 변환된 인코딩 문자열을 출력
# d.get(c, c): 키 c가 있으면 값, 없으면(알파벳이 아니면) 값은 동일한 c
print("".join([d.get(c, c) for c in s]))

# 인코딩 문자열을 e에 저장
e = '';
e = e.join([d.get(c, c) for c in s])
print(e)

# 다시 인코딩된 문자열을 원래의 문자열로 다시 변환하기 위한 딕셔너리 생성
key = -key
d = {} # 딕셔너리 만들기
for c in (65, 97): # 알파벳 A, a에서 부터 시작하여
  for i in range(26): # 알파벳 인코딩 딕셔너리 생성
    d[chr(i+c)] = chr((i + key) % 26 + c)

# 변환된 인코딩 문자열을 출력
print("".join([d.get(c, c) for c in e]))
```

표준 모듈 this.py의 이해

```
s = """Gur Mra bs Clguba, ol Gvz Crgref

Ornhgvshy vf orggre guna htyl.
Rkcyvpvg vf orggre guna vzcyvpvg.
Fvzcyr vf orggre guna pbzcyrk.
Pbzcyrk vf orggre guna pbzcyvpngrq.
Syng vf orggre guna arfgrq.
Fcnefr vf orggre guna qrafr.
Ernqnovyvgl pbhagf.
Fcrpvny pnfrf nera'g fcrpvny rabhtu gb oernx gur ehyrf.
Nygubhtu cenpgvpnyvgl orngf chevgl.
Reebef fubhyq arire cnff fvyragyl.
Hayrff rkcyvpvgyl fvyraprq.
Va gur snpr bs nzovthvgl, ershfr gur grzcgngvba gb thrff.
Gurer fubhyq or bar-- naq cersrenoyl bayl bar --boivbhf jnl gb qb vg.
Nygubhtu gung jnl znl abg or boivbhf ng svefg hayrff lbh'er Qhgpu.
Abj vf orggre guna arire.
Nygubhtu arire vf bsgra orggre guna *evtug* abj.
Vs gur vzcyrzragngvba vf uneq gb rkcynva, vg'f n onq vqrn.
Vs gur vzcyrzragngvba vf rnfl gb rkcynva, vg znl or n tbbq vqrn.
Anzrfcnprf ner bar ubaxvat terng vqrn -- yrg'f qb zber bs gubfr!"""

d = {}
for c in (65, 97):
    for i in range(26):
        d[chr(i+c)] = chr((i+13) % 26 + c)

print("".join([d.get(c, c) for c in s]))
```

```
D:₩Python38-32₩python.exe "D:/basic/py this1.py"
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

Process finished with exit code 0
```

소스 pybasic.py 이해

```
# import this
s = """The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!"""

d = {}
for c in (65, 97):
    for i in range(26):
        d[chr(i+c)] = chr((i-13) % 26 + c)

print("".join([d.get(c, c) for c in s]))
```

```
D:₩Python38-32₩python.exe D:/basic/pybasic.py
Gur Mra bs Clguba, ol Gvz Crgref

Ornhgvshy vf orggre guna htyl.
Rkcyvpvg vf orggre guna vzcyvpvg.
Fvzcyr vf orggre guna pbzcyrk.
Pbzcyrk vf orggre guna pbzcyvpngrq.
Syng vf orggre guna arfgrq.
Fcnefr vf orggre guna qrafr.
Ernqnovyvgl pbhagf.
Fcrpvny pnfrf nera'g fcrpvny rabhtu gb oernx gur ehyrf.
Nygubhtu cenpgvpnyvgl orngf chevgl.
Reebef fubhyq arire cnff fvyragyl.
Hayrff rkcyvpvgyl fvyraprq.
Va gur snpr bs nzovthvgl, ershfr gur grzcgngvba gb thrff.
Gurer fubhyq or bar-- naq cersrenoyl bayl bar --boivbhf jnl gb qb vg.
Nygubhtu gung jnl znl abg or boivbhf ng svefg hayrff lbh'er Qhgpu.
Abj vf orggre guna arire.
Nygubhtu arire vf bsgra orggre guna *evtug* abj.
Vs gur vzcyrzragngvba vf uneq gb rkcynva, vg'f n onq vqrn.
Vs gur vzcyrzragngvba vf rnfl gb rkcynva, vg znl or n tbbq vqrn.
Anzrfcnprf ner bar ubaxvat terng vqrn -- yrg'f qb zber bs gubfr!

Process finished with exit code 0
```