

금융 데이터
핸들링을 위한 파이썬

금융 데이터
핸들링을 위한 파이썬

금융 데이터 핸들링을 위한 Python

Contents

- ❶ 파이썬
- ❷ 변수와 자료형
- ❸ 제어문
- ❹ 입력과 출력
- ❺ 함수
- ❻ 데이터 핸들링

금융 데이터 핸들링을 위한 Python

Contents

- ❶ 파이썬
- ❷ 변수와 자료형
- ❸ 제어문
- ❹ 입력과 출력
- ❺ 함수
- ❻ 데이터 핸들링

금융 데이터 핸들링을 위한 Python

Python이란

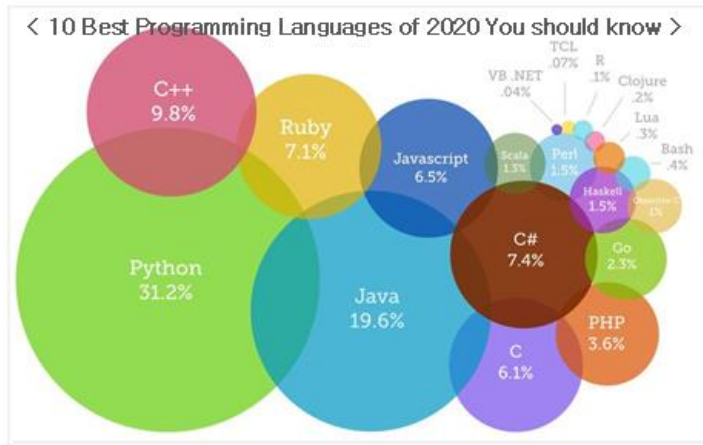
- 파이썬(Python)은 1991년 암스테르담의 귀도 반 로섬(Guido Van Rossum)이 발표한 프로그래밍 언어로서 컴파일 과정없이 명령을 내리면 바로 동작하는 인터프리터 언어(혹은 스크립트 언어)
- 파이썬(Python)은 비단뱀을 뜻하기도 하지만, 실제로는 뱀과는 상관없이 지어진 이름
- 귀도 반 로섬이 영국 방송국인 BBC(British Broadcasting Corporation)에서 1969년에서 1974년 사이에 방영했던 'Monty Python' s Flying Circus' 라는 TV프로그램의 이름을 따서 지었다고 알려져 있음



- 플랫폼이 독립적이며 인터프리터 (interpreter)식, 객체지향적, 동적 타이핑 (dynamically typed) 대화형 언어
- 비영리의 파이썬 소프트웨어 재단이 관리하는 개방형, 공동체 기반 개발 모델
- 파이썬은 무료이며 누구나 다운받아 사용이 가능함
- 윈도우, 리눅스 ,Mac OS X 등 다양한 시스템에서 사용 가능
- PYPL (PopularitY of Programming Language, 구글에서 조사) 프로그래밍 언어 순위 2위
- 구글에서 만들어진 소프트웨어의 50% 이상이 파이썬으로 만들어 짐
- Dropbox, Facebook (Tornado) 개발에 활용됨

금융 데이터 핸들링을 위한 Python

Python 소개



출처 url1 : <https://www.devsaran.com/blog/10-best-programming-languages-2020-you-should-know>
url2 : <http://statisticstimes.com/tech/top-computer-languages.php>

PYPL Index (Worldwide)

July 2020 ▲	Change ▼	Programming language ▼	Share ▼	Trends ▼
1		Python	31.73 %	+3.9 %
2		Java	17.13 %	-2.7 %
3		Javascript	7.98 %	-0.3 %
4		C#	6.67 %	-0.6 %
5	↑	C/C++	5.93 %	+0.1 %
6	↓	PHP	5.64 %	-1.1 %
7		R	4.14 %	+0.3 %
8		Objective-C	2.61 %	-0.1 %
9		Swift	2.29 %	-0.1 %
10	↑	TypeScript	1.91 %	+0.2 %

- 파이썬(Python)은 사회과학자들의 데이터 분석 도구로써 점점 더 인기가 높아지고 있음.
- 유용하고 광범위한 라이브러리들이 개발되고있어 Data Scientist들도 R과 Stata에서 Python으로 많이 옮기고 있는 추세
- 습득이 쉽기 때문에 미국 top 35 대학의 컴퓨터학과에서 처음 가르치는 언어들 중 Python의 비율이 가장 높게 조사됨

금융 데이터 핸들링을 위한 Python

Python 장단점

Python 장점

- 문법이 쉽고 순서가 영어 구문과 유사하여 빠르게 습득 가능
- 풍부한 라이브러리를 보유하고 있어 개발 생산성이 매우 높음
- 멀티패러다임 언어 (절차적 언어 지원, 객체 지향, 함수형)이기
- 때문에 재사용성이 높음
- 다양한 플랫폼에서 사용가능
- 쓰레딩 (threading) 대신 단일 이벤트 루프를 사용해 소수 유닛에서
- 작업하는 비동기식 코드작성에 유리
- 메모리 자동 관리
- 행렬 수학을 처리할 수 있음
- Tab, space가 엄격히 적용되어 가독성이 좋음
- 다른 언어로 쓰인 모듈을 연결하는 glue language로 많이 이용됨

Python 단점

- 인터프리터 기반이기 때문에 빠른 속도로 처리 불가 (C > JAVA > Python)
- 하드웨어를 직접 건드릴어야 하는 프로그램에는 부적합 (C, C++ 이용)
- 2.x 버전과 3.x 버전이 100% 호환되지 않음

금융 데이터 핸들링을 위한 Python

파이썬 개발환경

파이썬 다운로드

Anaconda Installers

Windows 	MacOS 	Linux 	버전 선택
Python 3.7 64-Bit Graphical Installer (466 MB) 32-Bit Graphical Installer (423 MB)	Python 3.7 64-Bit Graphical Installer (442 MB) 64-Bit Command Line Installer (430 MB)	Python 3.7 64-Bit (x86) Installer (522 MB) 64-Bit (Power8 and Power9) Installer (276 MB)	
Python 2.7 64-Bit Graphical Installer (413 MB) 32-Bit Graphical Installer (356 MB)	Python 2.7 64-Bit Graphical Installer (637 MB) 64-Bit Command Line Installer (409 MB)	Python 2.7 64-Bit (x86) Installer (477 MB) 64-Bit (Power8 and Power9) Installer (295 MB)	

- Anaconda 홈페이지에서 다운로드 가능
(<https://www.anaconda.com/distribution/#download-section>)
- 사용하는 운영체제에 맞는 installer 다운로드
- Python 2.7버전과 3.7버전이 있으며, 보통 3.7버전을 다운로드하여 사용함

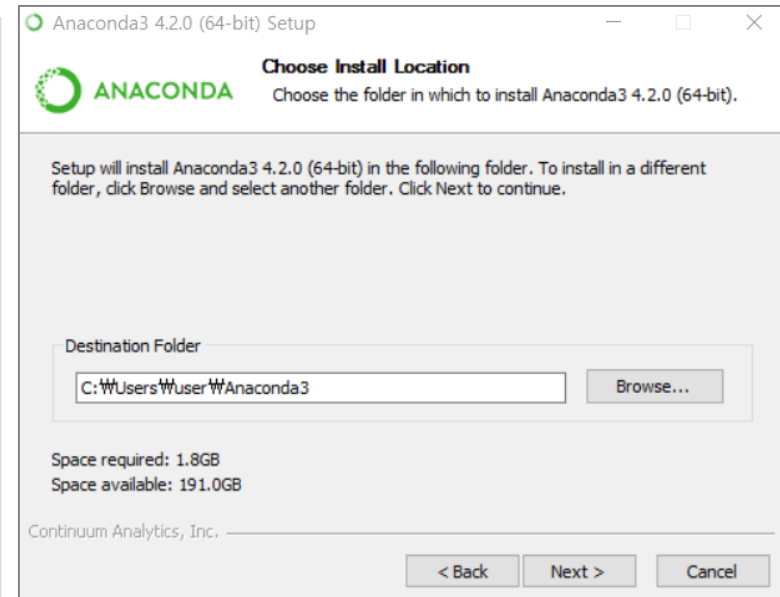
금융 데이터 핸들링을 위한 Python

파이썬 개발환경

파이썬 설치



- 다운받은 Anaconda 설치 파일을 위와 같이 실행



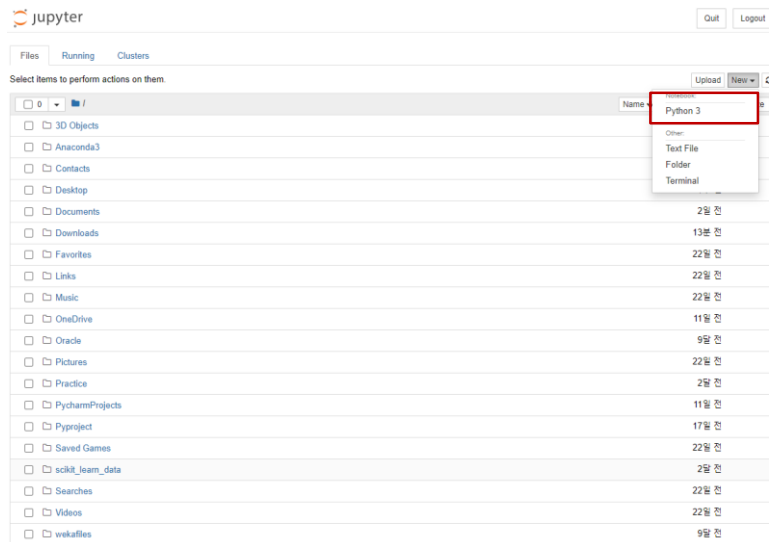
- Anaconda 설치 되는 경로 지정 가능
- Anaconda 설치 시 Python의 환경변수를 설정하지 않아도 됨 (자동설정 가능)

금융 데이터 핸들링을 위한 Python

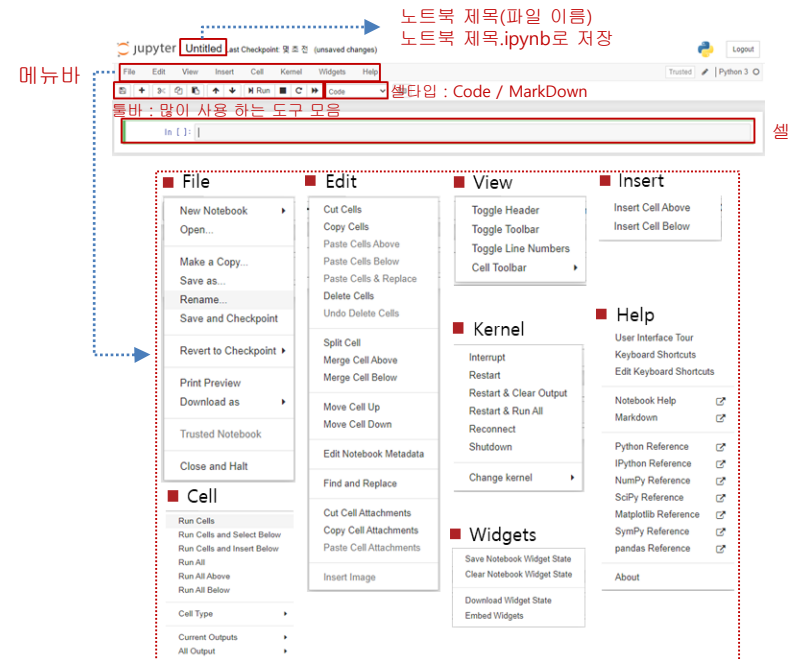
파이썬 개발환경

Jupyter Notebook 사용법

■ Jupyter Notebook에서 새 노트북 생성



■ Jupyter Notebook에서 새 노트북 생성

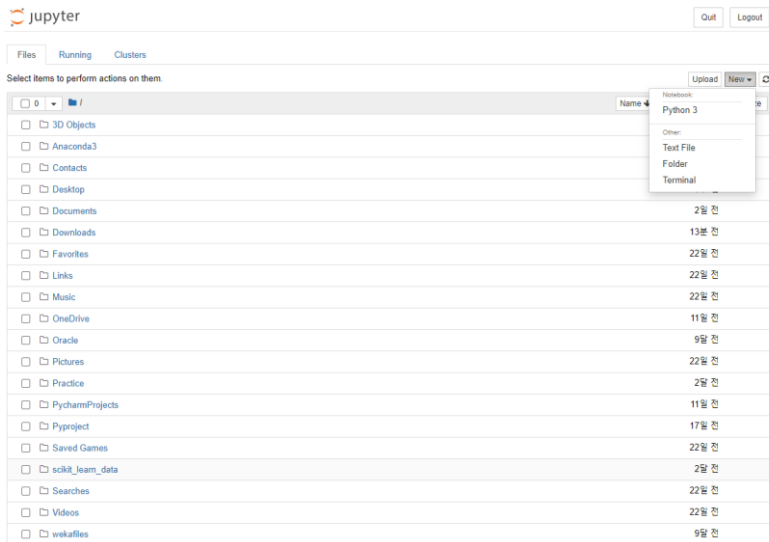


금융 데이터 핸들링을 위한 Python

파이썬 개발환경

Package 설치

■ Anaconda 패키지 위치



■ 'Anaconda 설치된 경로'/Anaconda3/pkgs

■ 패키지 설치하기

- 파이썬 3의 경우 3.4버전 이상부터 파이썬 패키지 매니저인 pip 가 자동으로 설치됨
- cmd창에서 “pip install 패키지명” 의 명령어를 입력하면 해당 패키지가 설치됨
- 명령어에 “python”을 입력 후 “import 패키지명“을 입력 했을 때 오류가 나지 않으면 해당 패키지 설치 성공임

```
Requirement already satisfied: numpy in c:\users\admin\anaconda3\lib\site-packages (1.18.1)

(base) C:\Users\Admin>python
Python 3.7.6 (default, Jan 8 2020, 20:23:39) [MSC v.1916 64 bit (AMD64)]
:: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy
```

금융 데이터 핸들링을 위한 Python

파이썬 개발환경

Packages 불러오기

- `import (package name)` 형태로 패키지를 불러옴
- `import (package name) AS (축약이름)` 형태를 사용해서 축약
- `from`은 패키지들을 포함하고 있는 라이브러리를 호출할 때 쓰임
- `from (library name) import (package name)` 명령어는 (library name)내에 있는 (package name) 패키지 호출

Example

```
import numpy
import pandas as pd
import matplotlib.pyplot as plt
import keras
from keras.layers import Input, Dense
from kersa.models import Model
```

금융 데이터 핸들링을 위한 Python

파이썬 개발환경

작업 공간 설정 및 호출

- 작업 공간의 위치를 알고 싶거나 위치를 변경하기 위해서는 os 패키지의 os.getcwd() 함수와 os.chdir() 함수를 사용
- 코드를 저장하기 위해서는 File → Save로 코드 저장(ctrl + s)
- 코드를 불러오기 위해서는 File → Open로 코드 불러오기(ctrl + o)

작업공간 찾기와 설정

```
import os

os.getcwd()

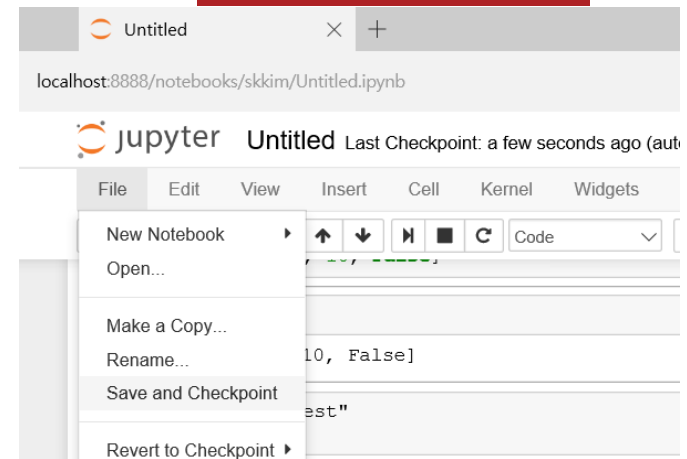
'C:\\\\Users\\\\begas\\\\Pyproject\\\\PSC'

os.chdir('C:\\\\Users\\\\begas\\\\Pyproject')

os.getcwd()

'C:\\\\Users\\\\begas\\\\Pyproject'
```

작업공간 저장 및 호출



금융 데이터 핸들링을 위한 Python

Contents

- 1 파이썬
- 2 변수와 자료형**
- 3 제어문
- 4 입력과 출력
- 5 함수
- 6 데이터 핸들링

금융 데이터 핸들링을 위한 Python

변수와 자료형

변수소개 및 규칙

■ 변수란?

숫자와 같은 자료(data)를 넣을 수 있는 상자

■ 변수명규칙

- 변수명은 문자, 숫자, 밑줄 기호(_)를 이용. 밑줄 기호(_)로 시작하는 변수명은 사용할 수 있지만 특별한 용도에 사용하므로 보통 사용하지 않음.
- 숫자로 시작하는 변수명은 만들 수 없음
 - 5ri 같은 변수명은 만들 수 없음
- 대소문자를 구분함
 - exam과 Exam은 다른 변수
- 공백을 포함할 수 없음
 - my home사용 불가능 → my_home로 사용
- Python의 예약어는 사용할 수 없음
 - and, exec, not, assert, finally, or, break, for, pass, class, from, print, continue, global, raise, def, if, return, del, import, try, elif, in, while, else, is, with, except, lambda, yield

금융 데이터 핸들링을 위한 Python

변수와 자료형

변수 유형

■ 데이터유형

■ 숫자형(Numeric Type)

■ Integer

- ✓ 자료형의 크기는 플랫폼이 32/64 bit인지에 따라 다르지만 Python은 아주 큰 정수를 무한히 큰 정수로 저장할 수 있는 long 자료형으로 변환해줌
- ✓ 형변환 : int()

■ Float

- ✓ 부동소수점 숫자를 나타냄 (Python에는 double형이 없음)
- ✓ Python 3.x 버전에서, 정수 나눗셈은 정수를 반환하지 않고 부동소수점 숫자를 반환
- ✓ 형변환 : float()

■ Complex

- ✓ 복소수의 허수 부분은 j로 나타냄

■ 논리형(Boolean Type)

- ✓ Python에서 논리형 값은 True와 False
- ✓ 주로 if문이나 기타 비교 및 조건식에 사용
- ✓ 형변환 : bool()

■ 문자형(String Type)

- ✓ Python에서는 문자열 처리가 쉬움
- ✓ (' ')나 큰 따옴표 (" ")로 표현가능
- ✓ 여러줄에 걸친 문자열은 3개의 작은 따옴표나 큰 따옴표로 표현
- ✓ 문자열은 변경 불가능하다
- ✓ 형변환 : str()

■ 변수 Type 확인

- ✓ type(변수명)으로 변수 Type 확인

금융 데이터 핸들링을 위한 Python

변수와 자료형

변수 유형

■ 데이터유형

■ 리스트(list)

- ✓ 하나의 집합 단위로 동일한 변수에 순차적으로 저장
- ✓ 배열(array)이라고도 부름
- ✓ []로 선언

■ 튜플(Tuple)

- ✓ 튜플은 1차원의 고정된 크기를 가지는 변경이 불가능한 순차 자료형
- ✓ 쉼표로 구분된 값을 대입하는 것으로 괄호를 사용해서 값을 묶어 줌으로써 중첩된 튜플을 정의할 수 있음
- ✓ 모든 순차 자료형이나 Iterator는 tuple()을 통해 형변환 가능
- ✓ 튜플에 저장된 객체 자체는 변경이 가능하지만 한번 생성이 되면 각 슬롯에 저장된 객체를 변경하는 것은 불가능(Immutable)

■ 집합(Set)

- ✓ 수학의 집합 개념을 파이썬에서 이용할 수 있도록 만든 데이터
- ✓ Index, 순서가 없음
- ✓ 중복이 허용되지 않음
- ✓ Set()함수 또는 중괄호({})로 선언

■ 딕셔너리(Dictionary)

- ✓ Key와 Value로 구성
- ✓ 중괄호({Key1 : Value1, Key2 : Value2})로 선언
- ✓ Key들의 집합은 set 성질을 가짐
- ✓ Key는 중복 불가능

금융 데이터 핸들링을 위한 Python

변수와 자료형

리스트(list) 다루기

In [122]: <code>a=[1,2,3,4]</code>	✓ 리스트 생성
In [123]: <code>a[0] = 100</code> <code>a</code>	✓ 리스트 항목 변경
Out [123]: <code>[100, 2, 3, 4]</code>	
In [124]: <code>a.append(8)</code> <code>a</code>	✓ 리스트 항목 추가
Out [124]: <code>[100, 2, 3, 4, 8]</code>	
In [125]: <code>a.remove(100)</code> <code>a</code>	✓ 리스트 항목 제거
Out [125]: <code>[2, 3, 4, 8]</code>	
In [126]: <code>a.insert(1, 100)</code> <code>a</code>	✓ 리스트 항목 삽입 (index, value)
Out [126]: <code>[2, 100, 3, 4, 8]</code>	
In [127]: <code>a.pop(1)</code>	✓ 항목 추출 후 (1번째 index)
Out [127]: <code>100</code>	
In [128]: <code>a</code>	✓ Pop 사용 후 리스트 항목
Out [128]: <code>[2, 3, 4, 8]</code>	

리스트 데이터 참조하기

■ 리스트[i_start : i_end : i_step]

In [131]: <code>a=[4,6,1,9]</code>	✓ 리스트 생성
In [132]: <code>a[1:3]</code>	
Out [132]: <code>[6, 1]</code>	
In [133]: <code>a[:3]</code>	
Out [133]: <code>[4, 6, 1]</code>	
In [134]: <code>a[:-2]</code>	
Out [134]: <code>[4, 6]</code>	
In [135]: <code>a[:]</code>	
Out [135]: <code>[4, 6, 1, 9]</code>	

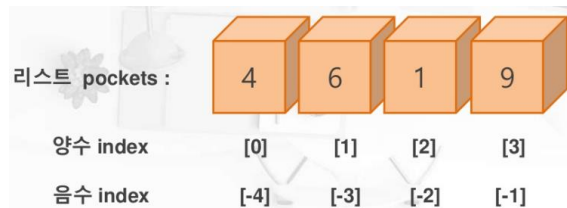
리스트 참조

금융 데이터 핸들링을 위한 Python

변수와 자료형

리스트(list) 다루기

■ 리스트 구조



```
In [142]: a=[1,2,3] ✓ 리스트 생성
In [143]: b=[4,5,6] ✓ 리스트 생성
In [144]: c=a+b ✓ 두 리스트를 합쳐
           c      새로운 리스트 생성
Out[144]: [1, 2, 3, 4, 5, 6]
In [145]: a.extend(b) ✓ 리스트 확장
           a      기존 리스트에
Out[145]: [1, 2, 3, 4, 5, 6] 새로운값 추가
```

```
In [146]: type(a) ✓ 데이터 타입 확인
Out[146]: list
In [147]: a=[1,2,3,4] ✓ 리스트 생성
In [148]: del a[0] ✓ 리스트 항목 제거
           a      0번 항목 제거
Out[148]: [2, 3, 4]
In [149]: del a[1:3] ✓ 리스트 범위 제거
           a      1-3번 항목 제거
Out[149]: [2]
In [150]: del a[:] ✓ 리스트 전체 항목 제거
           a      리스트 공간 존재
Out[150]: []
In [151]: del a ✓ 리스트 변수 제거
           a      a 변수명 제거된 상태
           -----
           NameError                                Traceback (most recent call last)
           <ipython-input-151-ef9d13752aff> in <module>()
           1 del a
           ----> 2 a
           NameError: name 'a' is not defined
           ※ 제거된 변수명을 선언 했기 때문에 Error 발생
```

금융 데이터 핸들링을 위한 Python

변수와 자료형

리스트 메서드

메서드	설명	사용 예
append()	리스트에서 항목 하나를 맨 마지막에 추가	myList.append('Thomas')
insert()	리스트에서 특정 위치에 항목을 삽입	myList.insert(1, 'Paul')
extend()	리스트에서 항목 여러 개를 맨 마지막에 추가	myList.extend(['Laura' , 'Betty'])
remove()	입력값과 첫 번째로 일치하는 항목을 리스트에서 삭제	myList.remove('Laura')
pop()	리스트의 마지막 항목을 제거한 후에 반환	popList = myList.pop()
index()	리스트에서 인자와 일치하는 첫 번째 항목의 위치를 반환	indexList = myList.index('Lisa')
count()	리스트에서 인자와 일치하는 항목의 개수를 반환	countList = myList.count('Lina')
sort()	숫자나 문자열로 구성된 리스트 항목을 순방향으로 정렬	myList.sort()
reverse()	리스트 항목을 끝에서부터 역순으로 정렬	myList.reverse()

금융 데이터 핸들링을 위한 Python

변수와 자료형

튜플(Tuple) 생성 및 다루기

튜플 생성하기

```
In [148]: tuple1 = (1, 2, 3, 4)
Out[148]: (1, 2, 3, 4)
```

✓ 튜플 생성 방법 1

```
In [151]: tuple2 = 1, 2, 3, 4
Out[151]: (1, 2, 3, 4)
```

✓ 튜플 생성 방법 2

```
In [152]: tuple3 = (1, 2, 3), (4, 5)
Out[152]: ((1, 2, 3), (4, 5))
```

✓ 튜플 생성 방법 3

```
In [153]: print(type(tuple1))
           print(type(tuple2))
           print(type(tuple3))
```

✓ 데이터 타입 확인

```
<class 'tuple'>
<class 'tuple'>
<class 'tuple'>
```

1개 인자 튜플 생성

```
In [156]: not_tuple5 = (1)
Out[156]: 1
```

✓ 잘못된 튜플 생성 방법 1

```
In [157]: type(not_tuple5)
Out[157]: int
```

※ 항목을 하나만 튜플 생성시 콤마(,) 필요

✓ 데이터 타입 확인

```
In [158]: tuple5 = (1,)
Out[158]: (1,)
```

✓ 1개 인자 튜플 생성 방법 1

```
In [159]: type(tuple5)
Out[159]: tuple
```

✓ 데이터 타입 확인

```
In [160]: not_tuple6 = 1
Out[160]: 1
```

✓ 잘못된 튜플 생성 방법 2

```
In [161]: type(not_tuple6)
Out[161]: int
```

※ 항목을 하나만 튜플 생성시 콤마(,) 필요

✓ 데이터 타입 확인

```
In [162]: tuple6 = 1,
Out[162]: (1,)
```

✓ 1개 인자 튜플 생성 방법 2

```
In [163]: type(tuple6)
Out[163]: tuple
```

✓ 데이터 타입 확인

금융 데이터 핸들링을 위한 Python

변수와 자료형

튜플(Tuple) 생성 및 다루기

■ 한번 생성된 튜플의 요소를 변경하거나 삭제할 수 없음

튜플 다루기

```
In [180]: tuple7 = ('a', 'a', 'a', 'b', 'b', 'b', 'c', 'd', 'd')  
tuple7  
Out[180]: ('a', 'a', 'a', 'b', 'b', 'b', 'c', 'd', 'd')
```

✓ 튜플 생성

```
In [181]: tuple7[1] = 'e'
```

```
TypeError                                 Traceback (most recent call last)  
<ipython-input-181-8cc354a8d8c3> in <module>  
----> 1 tuple7[1] = 'e'
```

TypeError: 'tuple' object does not support item assignment
※ 튜플의 요소를 변경할 수 없음

```
In [182]: del tuple7[1]
```

```
TypeError                                 Traceback (most recent call last)  
<ipython-input-182-9a39e5aa932e> in <module>  
----> 1 del tuple7[1]
```

TypeError: 'tuple' object doesn't support item deletion
※ 튜플의 요소를 제거할 수 없음

■ 요소를 변경하지 않는 메서드는 튜플에서도 사용 가능

튜플 다루기

✓ Tuple.index()

```
In [183]: tuple7.index('d')  
Out[183]: 7
```

✓ Tuple.count()

```
In [184]: tuple7.count('d')  
Out[184]: 2
```

금융 데이터 핸들링을 위한 Python

변수와 자료형

튜플 메서드

메서드	설명	사용 예
index()	튜플에서 인자와 일치하는 첫 번째 항목의 위치를 반환	myTuple.index('Lisa')
count()	튜플에서 인자와 일치하는 항목의 개수를 반환	myTuple.count('Lina')

금융 데이터 핸들링을 위한 Python

변수와 자료형

집합(Set) 생성 및 다루기

- 집합(Set)는 데이터를 중복해서 사용할 수 없음

집합 생성

```
In [185]: set1 = {1, 2, 3}
          set1a = {1, 2, 3, 3}
          print(set1)
          print(set1a)

{1, 2, 3}
{1, 2, 3}
```

※ 중복값인 3이 제거된 것을 확인

```
In [187]: print(type(set1))
          print(type(set1a))

<class 'set'>
<class 'set'>
```

- 집합(Set)는 수학 집합의 기본이 되는 메서드 사용할 수 있음

집합 다루기

```
In [191]: A = {1, 2, 3, 4, 5}
          B = {4, 5, 6, 7, 8, 9, 10}
```

✓ 집합 생성

```
In [192]: print(A.intersection(B))
          print(A&B)

{4, 5}
{4, 5}
```

✓ 교집합

```
In [193]: print(A.union(B))
          print(A|B)

{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

✓ 합집합

```
In [194]: print(A.difference(B))
          print(A-B)

{1, 2, 3}
{1, 2, 3}
```

✓ 차집합

```
In [195]: print(A.symmetric_difference(B))
          print(A^B)

{1, 2, 3, 6, 7, 8, 9, 10}
{1, 2, 3, 6, 7, 8, 9, 10}
```

✓ 대칭차집합

금융 데이터 핸들링을 위한 Python

변수와 자료형

집합(Set) 메서드

메서드	설명	사용 예
intersection()	교집합(집합 mySet1에 대한 집합 mySet2의 교집합)	mySet1.intersection('mySet2')
union()	합집합(집합 mySet1에 대한 집합 mySet2의 합집합)	mySet1.union('mySet2')
difference()	차집합(집합 mySet1에 대한 집합 mySet2의 차집합)	mySet1.difference('mySet2')
symmetric_difference()	대칭차집합(집합 mySet1에 대한 집합 mySet2의 대칭차집합)	mySet1.symmetric_difference('mySet2')
add()	값 1개 추가하기	mySet1.add('a')
update()	값 여러 개 추가하기	mySet1.update(['a' , 'b' , 'c'])
remove()	특정 값 제거하기	mySet1.remove('a')

금융 데이터 핸들링을 위한 Python

변수와 자료형

딕셔너리(Dictionary) 생성 및 다루기

딕셔너리 생성

✓ 딕셔너리 생성

```
In [229]: country_capital = {"영국" : "런던", "호주" : "멜버른", "덴마크" : "코펜하겐"}  
country_capital
```

```
Out[229]: {'영국': '런던', '호주': '멜버른', '덴마크': '코펜하겐'}
```

```
In [230]: type(country_capital)
```

✓ 데이터 타입 확인

```
Out[230]: dict
```

```
In [231]: country_capital["영국"]
```

✓ 데이터 값 확인

```
Out[231]: '런던'
```

딕셔너리 다루기

✓ 딕셔너리 데이터 추가

```
In [232]: country_capital["독일"] = "베를린"  
country_capital
```

```
Out[232]: {'영국': '런던', '호주': '멜버른', '덴마크': '코펜하겐', '독일': '베를린'}
```

✓ 딕셔너리 데이터 변경

```
In [233]: country_capital["호주"] = "캔버라"  
country_capital
```

```
Out[233]: {'영국': '런던', '호주': '캔버라', '덴마크': '코펜하겐', '독일': '베를린'}
```

✓ 딕셔너리 데이터 제거

```
In [234]: del country_capital["덴마크"]  
country_capital
```

```
Out[234]: {'영국': '런던', '호주': '캔버라', '독일': '베를린'}
```

금융 데이터 핸들링을 위한 Python

변수와 자료형

딕셔너리(Dictionary) 생성 및 다루기

딕셔너리 다루기

- ✓ 딕셔너리 키 출력

```
In [235]: print(country_capital.keys())  
dict_keys(['영국', '호주', '독일'])
```

- ✓ 딕셔너리 값 출력

```
In [237]: print(country_capital.values())  
dict_values(['런던', '캔버라', '베를린'])
```

- ✓ 딕셔너리 키,값 쌍 출력

```
In [239]: country_capital.items()  
Out[239]: dict_items([('영국', '런던'), ('호주', '캔버라'), ('독일', '베를린')])
```

딕셔너리 다루기

- ✓ 딕셔너리 키 리스트 형태로 출력

```
In [238]: list(country_capital.keys())  
Out[238]: ['런던', '캔버라', '베를린']
```

- ✓ 딕셔너리 값 리스트 형태로 출력

```
In [238]: list(country_capital.values())  
Out[238]: ['런던', '캔버라', '베를린']
```

- ✓ 딕셔너리 키,값 쌍 리스트 형태로 출력

```
In [240]: list(country_capital.items())  
Out[240]: [('영국', '런던'), ('호주', '캔버라'), ('독일', '베를린')]
```

금융 데이터 핸들링을 위한 Python

변수와 자료형

딕셔너리(Dictionary) 생성 및 다루기

딕셔너리 다루기

✓ 딕셔너리 생성

```
In [241]: country_capital_add = {"한국" : "서울", "미국" : "뉴욕"}
```

✓ 기존 딕셔너리에 새로운 딕셔너리 추가

```
In [242]: country_capital.update(country_capital_add)
country_capital
Out[242]: {'영국': '런던', '호주': '캔버라', '독일': '베를린', '한국': '서울', '미국': '뉴욕'}
```

✓ 딕셔너리 모든 항목 삭제

```
In [243]: country_capital.clear()
print(country_capital)
type(country_capital)

{}
```

금융 데이터 핸들링을 위한 Python

변수와 자료형

딕셔너리 메서드

메서드	설명	사용 예
keys()	딕셔너리에서 키 전체를 리스트 형태로 반환	myDict.keys()
values()	딕셔너리에서 값 전체를 리스트 형태로 반환	myDict.values()
items()	딕셔너리에서 키와 값 쌍을 (키, 값)처럼 튜플 형태로 반환	myDict.items()
update(dict2)	딕셔너리에 딕셔너리 데이터('dict2') 추가	myDict.update(dict2)
clear()	딕셔너리의 모든 항목 삭제	myDict.clear()

금융 데이터 핸들링을 위한 Python

Contents

- 1 파이썬
- 2 변수와 자료형
- 3 제어문**
- 4 입력과 출력
- 5 함수
- 6 데이터 핸들링

금융 데이터 핸들링을 위한 Python

제어문

제어문소개

■ 제어문이란?

코드의 진행 순서를 바꾸는 구문.

- 조건문 : 조건을 검사해 분기하는 구문
- 반복문 : 어떤 구간이나 조건을 만족하는 동안 코드의 일부분을 반복하는 구문

■ 조건을 판단하기 위한 연산자

■ 비교연산자

연산자	의미	활용 예	설명
==	같다	<code>a == b</code>	a는 b와 같다
!=	같지 않다	<code>a != b</code>	a는 b와 같지 않다
<	작다	<code>a < b</code>	a는 b보다 작다
>	크다	<code>a > b</code>	a는 b보다 크다
<=	작거나 크다	<code>a <= b</code>	a는 b보다 작거나 같다
>=	크거나 작다	<code>a >= b</code>	a는 b보다 크거나 같다

■ 논리연산자

연산자	의미	활용 예	설명
and	논리곱	<code>A and B</code>	A와 B 모두가 참이면 참
or	논리합	<code>A or B</code>	A와 B중 하나라도 참이면 참
not	논리부정	<code>not A</code>	A가 거짓이면 참

■ 참고사항 : 파이썬 구조 결정

- 콜론(:)
- 들여쓰기
 - ✓ 스페이스바를 이용한 공백(빈칸)
 - ✓ 탭(Tab)키를 이용한 탭 입력
 - ✓ 잘못되는 경우 오류 발생

금융 데이터 핸들링을 위한 Python

제어문

조건에 따라 분기하는 if문

■ 단일 조건에 따른 분기(if)

```
if <조건문>:  
    <코드 블록>
```

example

```
In [244]: x = 10  
          if x > 90:  
              print("90보다 크다")
```

※ 출력결과 없음

```
In [245]: if x < 90:  
          print("90보다 작다")
```

90보다 작다

■ 단일 조건 및 그 외 조건에 따른 분기(if ~ else)

```
if <조건문>:  
    <코드 블록1>  
else :  
    <코드 블록2>
```

example

```
In [246]: x = 10  
          if x > 90:  
              print("90보다 크다")  
          else :  
              print("90보다 작다")
```

90보다 작다

금융 데이터 핸들링을 위한 Python

제어문

조건에 따라 분기하는 if문

■ 여러 조건에 따른 분기(if ~ elif ~ else)

```
if <조건문>:  
    <코드 블록>  
elif <조건문2>:  
    <코드 블록>  
...  
else:  
    <코드 블록>
```

example

변수가 90보다 작으면 '90 미만', 90보다 크면 '90초과', 아무것도 해당하지 않으면 '해당없음'을 출력하시오.

```
x = 90  
  
if x < 90 :  
    print('90 미만')  
  
elif x > 90 :  
    print('90 초과')  
  
else:  
    print('해당 없음')
```


금융 데이터 핸들링을 위한 Python

제어문

지정된 범위만큼 반복하는 for문

■ for문의 구조

```
for <반복 변수> in <반복 범위>:  
    <코드 블록>
```

■ 중첩 for 문

```
for <반복 변수 1> in <반복 범위 1>:  
    for <반복 변수 2> in <반복 범위 2>:  
        <코드 블록>
```

example

```
x_list = ['x1', 'x2']  
y_list = ['y1', 'y2']
```

```
for x in x_list:  
    for y in y_list:  
        print(x,y)
```

```
x1 y1  
x1 y2  
x2 y1  
x2 y2
```

금융 데이터 핸들링을 위한 Python

제어문

조건에 따라 반복하는 **while** 문

■ while문의 구조

```
while <조건문> :  
    <코드 블록>
```

■ 무한 반복 while 문 : 무제한 반복해야하는 경우

```
while True :  
    <코드 블록>
```

example

```
i = 0  
sum = 0  
  
while (sum < 20):  
    i = i + 1  
    sum = sum + i  
  
print(i)  
print(sum)  
  
6  
21
```

금융 데이터 핸들링을 위한 Python

제어문

반복문을 제어하는 **break**와 **continue**

- 반복문을 빠져나오는 **break**
: 반복문 안에서 **break**를 만나면 반복문을 빠져나옵니다.

```
while <조건문> :  
    <코드 블록>  
    if <조건문> :  
        break  
    <코드 블록>
```

- 다음 반복을 실행하는 **continue**
: 반복문 안에서 **continue**를 만나면 반복문의 처음으로 돌아가서 다음 반복을 진행합니다

```
while <조건문> :  
    <코드 블록>  
    if <조건문> :  
        continue  
    <코드 블록>
```

example

✓ break

```
i = 0  
  
while (i < 5):  
    i = i + 1  
    if i == 3:  
        break  
    print(i)
```

1
2

✓ continue

```
i = 0  
  
while (i < 5):  
    i = i + 1  
    if i == 3:  
        continue  
    print(i)
```

1
2
4
5

금융 데이터 핸들링을 위한 Python

Contents

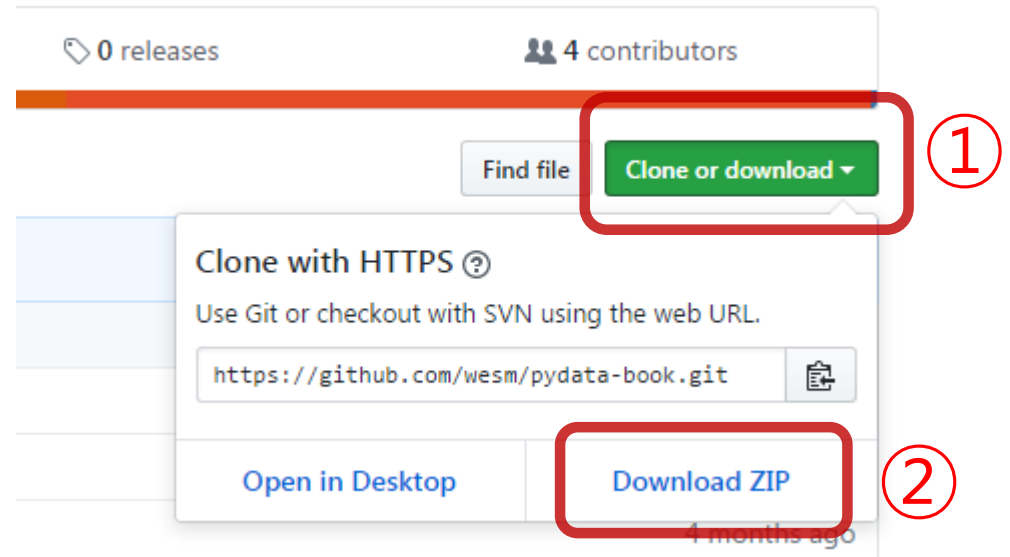
- 1 파이썬
- 2 변수와 자료형
- 3 제어문
- 4 입력과 출력**
- 5 함수
- 6 데이터 핸들링

금융 데이터 핸들링을 위한 Python

입력과 출력

예제 데이터 다운받기

- <https://github.com/wesm/pydata-book> 접속
- zip 파일로 내려받기
- Python에서, 데이터를 불러올 때 경로를 지정해 줘야 되지만 현재 작업하는 폴더에 데이터 파일이 있으면 파일명만 입력해서 불러올 수 있다



금융 데이터 핸들링을 위한 Python

입력과 출력

파일 로딩 함수

함수	설명
read_csv	파일, URL 또는 파일과 유사한 객체로부터 구분된 데이터를 읽어 옴 데이터 구분자는 쉼표(,)를 기본으로 함
read_table	파일, URL 또는 파일과 유사한 객체로부터 구분된 데이터를 읽어 옴 데이터 구분자는 탭('wt')을 기본으로 함
read_fwf	고정폭 칼럼 형식에서 데이터를 읽어 옴 (구분자가 없는 데이터)

파일 로딩함수의 옵션

- 색인 : 반환하는 DataFrame에서 하나 이상의 칼럼을 색인으로 지정할 수 있음
- 파일이나 사용자로부터 칼럼의 이름을 받거나
- 아무것도 받지 않을 수 있음
- 자료형 추론과 데이터 변환 : 사용자가 정의 값 변환과 비어 있는 값을 위한 사용자 리스트를 포함
- 날짜 분석 : 여러 칼럼에 걸쳐 있는 날짜와 시간 정보를 하나의 칼럼에 조합해서 결과에 반영
- 반복 : 여러 파일에 걸쳐 있는 자료를 반복적으로 읽어올 수 있음
- 정제되지 않은 데이터 처리 : 로우나 꼬리말, 주석 건너뛰기 또는 천 단위마다 쉼표로 구분된 숫자 같은 사소한 일을 처리해줌

금융 데이터 핸들링을 위한 Python

입력과 출력

텍스트 파일 이용하는 방법

```
> import pandas as pd
> import os
> dat_path = 'C:/Users/hgpar/Pyproject/DAT'
> os.chdir(dat_path)
> df = pd.read_csv('ex1.csv')
> df
> pd.read_table('ex1.csv', sep=',')
> 불러온 데이터의 변수들의 유형 확인 가능
> df.dtypes
```

```
In [5]: df
```

```
Out[5]:
```

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

```
In [6]: df.dtypes
```

```
Out[6]: a                int64
b                int64
c                int64
d                int64
message          object
dtype: object
```

금융 데이터 핸들링을 위한 Python

입력과 출력

텍스트 파일 이용하는 방법

- > `pd.read_csv('ex2.csv', header=None)`
- > `pd.read_csv('ex2.csv')`
- > `pd.read_csv('ex2.csv',
names=['a', 'b', 'c', 'd', 'message'])`
 - 파일에 칼럼의 이름이 없는 경우도 존재함
=> pandas가 자동으로 칼럼 이름을 생성하도록 하거나
직접 칼럼 이름을 지정할 수 있음

```
In [7]: pd.read_csv('ex2.csv', header=None)
```

```
Out [7]:
```

	0	1	2	3	4
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

```
In [8]: pd.read_csv('ex2.csv')
```

```
Out [8]:
```

	1	2	3	4	hello
0	5	6	7	8	world
1	9	10	11	12	foo

```
In [9]: pd.read_csv('ex2.csv', names=['a', 'b', 'c', 'd', 'message'])
```

```
Out [9]:
```

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

금융 데이터 핸들링을 위한 Python

입력과 출력

텍스트 파일 이용하는 방법

- > `pd.read_csv('ex2.csv', names=['a','b','c','d','message'], index_col='message')`
- `message` 칼럼을 색인으로 하는 `DataFrame`을 반환하기 위해 `index_col` 인자에 네번째 또는 'message'라는 이름을 가진 칼럼을 지정해서 색인으로 만들 수 있음

Out [12]:

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

Out [14]:


	a	b	c	d
message				
hello	1	2	3	4
world	5	6	7	8
foo	9	10	11	12

금융 데이터 핸들링을 위한 Python

입력과 출력

텍스트 파일 이용하는 방법

- > `pd.read_csv('csv_mindex.csv')`
- 계층적 색인을 지정하고 싶을 경우, 칼럼 번호나 이름의 리스트를 넘김
- > `pd.read_csv('csv_mindex.csv', index_col=['key1', 'key2'])`



	key1	key2	value1	value2
0	one	a	1	2
1	one	b	3	4
2	one	c	5	6
3	one	d	7	8
4	two	a	9	10
5	two	b	11	12
6	two	c	13	14
7	two	d	15	16

	key1	key2	value1	value2
one	one	a	1	2
		b	3	4
		c	5	6
		d	7	8
two	two	a	9	10
		b	11	12
		c	13	14
		d	15	16

금융 데이터 핸들링을 위한 Python

입력과 출력

텍스트 파일 이용하는 방법

> list(open('ex3.txt'))

```
In [16]: list(open('ex3.txt'))
Out[16]: ['      A      B      C\n',
'aaa -0.264438 -1.026059 -0.619500\n',
'bbb  0.927272  0.302904 -0.032399\n',
'ccc -0.264273 -0.386314 -0.217601\n',
'ddd -0.871858 -0.348382  1.100491\n']
```

- 고정된 구분자 없이 공백이나 다른 패턴으로 필드를 구분해 놓은 경우, read_table의 구분자로 정규표현식(\\s+)을 사용하여 처리

> result = pd.read_table('ex3.txt', sep='\\s+')

> result

```
Out[19]:
```

	A	B	C
aaa	-0.264438	-1.026059	-0.619500
bbb	0.927272	0.302904	-0.032399
ccc	-0.264273	-0.386314	-0.217601
ddd	-0.871858	-0.348382	1.100491

> result = pd.read_csv('ex5.csv')

> result

```
In [22]: result
```

```
Out[22]:
```

	something	a	b	c	d	message
0	one	1	2	3.0	4	NaN
1	two	5	6	NaN	8	world
2	three	9	10	11.0	12	foo

금융 데이터 핸들링을 위한 Python

입력과 출력

텍스트 파일 이용하는 방법

> result = pd.read_csv('ex5.csv', na_values=4)

> result

In [22]: result

Out [22]:

	something	a	b	c	d	message
0	one	1	2	3.0	NaN	NaN
1	two	5	6	NaN	8.0	world
2	three	9	10	11.0	12.0	foo

- na_values 옵션은 리스트나 문자열 집합을 받아서 누락된 값을 처리

> sentinels = {'message': ['foo', 'NA'], 'something':['two']}

> pd.read_csv('ex5.csv', na_values=sentinels)

In [30]: result

Out [30]:

	something	a	b	c	d	message
0	one	1	2	3.0	4	NaN
1	NaN	5	6	NaN	8	world
2	three	9	10	11.0	12	NaN

- 얼마다 다른 NA 문자를 사전 값으로 넘겨 처리할 수 있음

금융 데이터 핸들링을 위한 Python

입력과 출력

데이터를 텍스트 형식으로 출력

- > data = pd.read_csv('ex5.csv')
- > data
- > data.to_csv('out.csv')
- > pd.read_csv('out.csv')

```
In [18]: data = pd.read_csv('ex5.csv')
data
data.to_csv('out.csv')
```

```
In [19]: pd.read_csv('out.csv')
```

```
Out [19]:
```

	Unnamed: 0	something	a	b	c	d	message
0	0	one	1	2	3.0	4	NaN
1	1	two	5	6	NaN	8	world
2	2	three	9	10	11.0	12	foo

- DataFrame의 to_csv 메서드를 이용하면 쉼표로 구분된 형식으로 데이터를 파일로 쓸 수 있음

- > import sys
- > data.to_csv(sys.stdout, na_rep='NULL')

	Unnamed: 0	something	a	b	c	d	message
0	0	one	1	2	3.0	4	NaN
1	1	two	5	6	NaN	8	world
2	2	three	9	10	11.0	12	foo



```
data.to_csv(sys.stdout, na_rep='NULL')
, something, a, b, c, d, message
0, one, 1, 2, 3.0, 4, NULL
1, two, 5, 6, NULL, 8, world
2, three, 9, 10, 11.0, 12, foo
```

- 결과에서 누락된 값은 비어있는 문자열로 나타나는데, 이를 원하는 값(NULL)으로 지정할 수 있음

금융 데이터 핸들링을 위한 Python

입력과 출력

데이터를 텍스트 형식으로 출력

- > data.to_csv(sys.stdout, index=False, columns=['a', 'b', 'c'])
- a,b,c에 해당하는 값만 출력

```
, something, a, b, c, d, message  
0, one, 1, 2, 3.0, 4,  
1, two, 5, 6, , 8, world  
2, three, 9, 10, 11.0, 12, foo
```

→

```
a, b, c  
1, 2, 3.0  
5, 6,  
9, 10, 11.0
```

- 칼럼의 일부분만 기록하거나 순서를 직접 지정할 수 있음

- > import np
- > dates = pd.date_range('1/1/2000', periods=7)
- > ts = pd.Series(np.arange(7), index=dates)
- > ts.to_csv('tseries.csv')
- > pd.read_csv('tseries.csv')

```
In [23]: import numpy as np  
  
dates = pd.date_range('1/1/2000', periods=7)  
ts = pd.Series(np.arange(7), index=dates)  
ts.to_csv('tseries.csv')  
pd.read_csv('tseries.csv')
```

Out [23]:

	Unnamed: 0	0
0	2000-01-01	0
1	2000-01-02	1
2	2000-01-03	2
3	2000-01-04	3
4	2000-01-05	4
5	2000-01-06	5
6	2000-01-07	6

- Series에서도 to_csv 메서드 사용 가능

금융 데이터 핸들링을 위한 Python

Contents

- 1 파이썬
- 2 변수와 자료형
- 3 제어문
- 4 입력과 출력
- 5 함수**
- 6 데이터 핸들링

금융 데이터 핸들링을 위한 Python

함수정의

Python 코드 및 산출물

- 함수의 3요소: 함수명, 인자값, 반환값

- def 를 통해 함수정의

- return으로 함수의 반환 값을 반환함

```
> def my_function(x,y,z):
```

```
>     if z>2:
```

```
>         return z*(x+y)
```

```
>     else:
```

```
>         return z/(x+y)
```

- 위의 예제는 x,y,z 의 인자를 받아 조건(if,else)에 따라 결과값을 반환하는 my_function을 정의한 것

```
In [252]: def my_function(x,y,z):  
           if z>2:  
               return z*(x+y)  
           else:  
               return z/(x+y)
```

```
In [253]: my_function(1,2,3)
```

```
Out[253]: 9
```

```
In [254]: my_function(1,2,2)
```

```
Out[254]: 0.6666666666666666
```

- 인자 값에 따라 결과가 다르게 나옴

금융 데이터 핸들링을 위한 Python

Contents

- 1 파이썬
- 2 변수와 자료형
- 3 제어문
- 4 입력과 출력
- 5 함수
- 6 데이터 핸들링**

금융 데이터 핸들링을 위한 Python

데이터 핸들링

데이터 합치기

- > import pandas as pd
- > import numpy as np
- > df1 = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'a', 'b'],
 'data1': range(7)})
- > df1
- > df2 = pd.DataFrame({'key': ['a', 'b', 'd'],
 'data2': range(3)})
- > df2
- > pd.merge(df1, df2)
- > pd.merge(df1, df2, on='key')

	data2	key
0	0	a
1	1	b
2	2	d

	data1	key
0	0	b
1	1	b
2	2	a
3	3	c
4	4	a
5	5	a
6	6	b

```
In [28]: pd.merge(df1, df2)
```

```
Out[28]:
```

	data1	key	data2
0	0	b	1
1	1	b	1
2	6	b	1
3	2	a	0
4	4	a	0
5	5	a	0

```
In [29]: pd.merge(df1, df2, on='key')
```

```
Out[29]:
```

	data1	key	data2
0	0	b	1
1	1	b	1
2	6	b	1
3	2	a	0
4	4	a	0
5	5	a	0

- 어떤 칼럼을 기준으로 병합할 것인지 명시하지 않으면 merge함수는 겹치는 칼럼을 키로 사용
- 병합할 칼럼을 key칼럼으로 명시해줌(결과와 위와 동일)

금융 데이터 핸들링을 위한 Python

데이터 핸들링

데이터 합치기

- > df3 = pd.DataFrame({'lkey': ['b', 'b', 'a', 'c', 'a', 'a', 'b'],
 'data1': range(7)})
- > df4 = pd.DataFrame({'rkey': ['a', 'b', 'd'], 'data2':
 range(3)})
- > df3
- > df4

- > pd.merge(df3, df4, left_on = 'lkey', right_on = 'rkey')
- > # 칼럼명이 같지 않다면 각각 지정을 해줌

In [36]: df3

Out[36]:

	data1	lkey
0	0	b
1	1	b
2	2	a
3	3	c
4	4	a
5	5	a
6	6	b

In [37]: df4

Out[37]:

	data2	rkey
0	0	a
1	1	b
2	2	d

In [35]: pd.merge(df3, df4, left_on = 'lkey', right_on = 'rkey')

Out[35]:

	data1	lkey	data2	rkey
0	0	b	1	b
1	1	b	1	b
2	6	b	1	b
3	2	a	0	a
4	4	a	0	a
5	5	a	0	a

- 병합 결과, 'c'와 'd'에 해당하는 값이 빠진 것을 알 수 있음

금융 데이터 핸들링을 위한 Python

데이터 핸들링

데이터 합치기

> `pd.merge(df1, df2, how='outer')`

merge 함수는 기본적으로 내부조인(inner join)을 수행하여 교집합인 결과를 반환

how인자로 'left', 'right', 'outer'를 넘겨서 각각 왼쪽 우선 외부조인, 오른쪽 우선 외부조인, 완전 외부조인을 수행

완전 외부조인은 합집합인 결과를 반환하고 왼쪽 우선 외부조인과 오른쪽 우선 외부조인은 각각 왼쪽, 오른쪽의 모든 로우를 포함하는 결과를 반환함

```
In [38]: pd.merge(df1, df2, how='outer')
```

Out[38]:

	data1	key	data2
0	0.0	b	1.0
1	1.0	b	1.0
2	6.0	b	1.0
3	2.0	a	0.0
4	4.0	a	0.0
5	5.0	a	0.0
6	3.0	c	NaN
7	NaN	d	2.0

- 완전 외부조인을 이용하여 합집합인 결과를 반환함

금융 데이터 핸들링을 위한 Python

데이터 핸들링

데이터 합치기

```
> df1 = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'b'], 'data1':  
    range(6)})  
> df2 = pd.DataFrame({'key': ['a', 'b', 'a', 'b', 'd'],  
    'data2': range(5)})  
> df1  
> df2  
> pd.merge(df1, df2, on='key', how='left')  
> # 다 대 다 병합은 두 로우의 데카르트 곱을 반환
```

```
In [41]: df1
```

```
Out[41]:
```

	data1	key
0	0	b
1	1	b
2	2	a
3	3	c
4	4	a
5	5	b

```
In [42]: df2
```

```
Out[42]:
```

	data2	key
0	0	a
1	1	b
2	2	a
3	3	b
4	4	d

```
In [43]: pd.merge(df1, df2, on='key', how='left')
```

```
Out[43]:
```

	data1	key	data2
0	0	b	1.0
1	0	b	3.0
2	1	b	1.0
3	1	b	3.0
4	2	a	0.0
5	2	a	2.0
6	3	c	NaN
7	4	a	0.0
8	4	a	2.0
9	5	b	1.0
10	5	b	3.0

금융 데이터 핸들링을 위한 Python

데이터 핸들링

데이터 합치기

- > left_df = pd.DataFrame({'key': ['a', 'b', 'a', 'a', 'b', 'c'],
'value': range(6)})
- > right_df = pd.DataFrame({'group_val': [3.5, 7]},
index=['a', 'b'])
- > left_df
- > right_df
- > pd.merge(left_df, right_df, left_on='key',
right_index=True)
- # Merge하려는 키가 DataFrame의 색인일 경우
left_index=True 혹은 right_index=True 옵션을 지정해 해당 색인을 머지 키로 사용할 수 있음

In [22]: left_df

Out [22]:

	key	value
0	a	0
1	b	1
2	a	2
3	a	3
4	b	4
5	c	5

In [23]: right_df

Out [23]:

	group_val
a	3.5
b	7.0

In [24]: pd.merge(left_df, right_df, left_on='key', right_index=True)

Out [24]:

	key	value	group_val
0	a	0	3.5
2	a	2	3.5
3	a	3	3.5
1	b	1	7.0
4	b	4	7.0

- Merge는 기본적으로 해당 머지 키의 교집합을 구함

금융 데이터 핸들링을 위한 Python

데이터 핸들링

데이터 합치기

> `pd.merge(left_df, right_df, left_on='key', right_index=True, how='outer')`

```
In [18]: pd.merge(left_df, right_df, left_on='key', right_index=True, how='outer')
```

Out [18]:

	key	value	group_val
0	a	0	3.5
2	a	2	3.5
3	a	3	3.5
1	b	1	7.0
4	b	4	7.0
5	c	5	NaN

- Merge는 기본적으로 해당 머지 키의 교집합을 구하지만 외부조인을 실행해서 합집합을 구할 수 있음

> `left_df2 = pd.DataFrame({'data': np.arange(5.), 'key1': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada'], 'key2': [2000, 2001, 2002, 2001, 2002]})`

> `right_df2 = pd.DataFrame(np.arange(12).reshape((6, 2)), index=[['Nevada', 'Nevada', 'Ohio', 'Ohio', 'Ohio', 'Ohio'], [2001, 2000, 2000, 2000, 2001, 2002]], columns=['event1', 'event2'])`

	data	key1	key2
0	0.0	ohio	2000
1	1.0	ohio	2001
2	2.0	ohio	2002
3	3.0	Nevada	2001
4	4.0	Nevada	2002

		event1	event2
Nevada	2001	0	1
	2000	2	3
Ohio	2000	4	5
	2000	6	7
	2001	8	9
	2002	10	11

금융 데이터 핸들링을 위한 Python

데이터 핸들링

데이터 합치기

- 계층 색인된 데이터의 Merge
- > `pd.merge(left_df2, right_df2, left_on=['key1', 'key2'], right_index=True)`

	data	key1	key2	event1	event2
0	0.0	Ohio	2000	4	5
0	0.0	Ohio	2000	6	7
1	1.0	Ohio	2001	8	9
2	2.0	Ohio	2002	10	11
3	3.0	Nevada	2001	0	1

- 계층 색인된 데이터의 경우, 리스트로 여러 개의 칼럼을 지정해서 머지해야 함

- > `pd.merge(left_df2, right_df2, left_on=['key1', 'key2'], right_index=True, how='outer')`

	data	key1	key2	event1	event2
0	0.0	Ohio	2000	4.0	5.0
0	0.0	Ohio	2000	6.0	7.0
1	1.0	Ohio	2001	8.0	9.0
2	2.0	Ohio	2002	10.0	11.0
3	3.0	Nevada	2001	0.0	1.0
4	4.0	Nevada	2002	NaN	NaN
4	NaN	Nevada	2000	2.0	3.0

- 외부조인을 실행하여 합집합을 구함

금융 데이터 핸들링을 위한 Python

데이터 핸들링

데이터 합치기

```
> left_df3 = pd.DataFrame([[1., 2.], [3., 4.], [5., 6.]],  
    index=['a', 'c', 'e'], columns=['Ohio', 'Nevada'])  
> right_df3 = pd.DataFrame([[7., 8.], [9., 10.],  
    [11., 12.], [13., 14.]], index=['b', 'c', 'd', 'e'],  
    columns=['Missouri', 'Alabama'])  
> left_df3  
> right_df3  
> pd.merge(left_df3, right_df3, how='outer',  
    left_index=True, right_index=True)
```

In [28]: left_df3

Out [28]:

	Ohio	Nevada
a	1.0	2.0
c	3.0	4.0
e	5.0	6.0

In [29]: right_df3

Out [29]:

	Missouri	Alabama
b	7.0	8.0
c	9.0	10.0
d	11.0	12.0
e	13.0	14.0

In [30]: pd.merge(left_df3, right_df3, how='outer', left_index=True, right_index=True)

Out [30]:

	Ohio	Nevada	Missouri	Alabama
a	1.0	2.0	NaN	NaN
b	NaN	NaN	7.0	8.0
c	3.0	4.0	9.0	10.0
d	NaN	NaN	11.0	12.0
e	5.0	6.0	13.0	14.0

- 양쪽에 공통으로 있는 여러 개의 색인을 merge함

금융 데이터 핸들링을 위한 Python

데이터 핸들링

데이터 합치기

> left2.join(right2, how='outer')

```
In [33]: left_df3.join(right_df3, how='outer')
```

Out [33]:

	Ohio	Nevada	Missouri	Alabama
a	1.0	2.0	NaN	NaN
b	NaN	NaN	7.0	8.0
c	3.0	4.0	9.0	10.0
d	NaN	NaN	11.0	12.0
e	5.0	6.0	13.0	14.0

- 색인으로 머지할 때 DataFrame의 join 메서드를 사용
- join 메서드는 칼럼이 겹치지 않으며 완전히 같거나 유사한 색인 구조를 가진 여러 개의 DataFrame 객체를 병합할 때 사용

- 데이터를 합치는 또 다른 방법으로는 NumPy에서 ndarray를 연결하는 concatenate 함수

> arr = np.arange(12).reshape((3, 4)) #array형식

> arr

> np.concatenate([arr, arr], axis=1)

```
In [79]: arr = np.arange(12).reshape((3, 4))  
arr
```

```
Out[79]: array([[ 0,  1,  2,  3],  
                [ 4,  5,  6,  7],  
                [ 8,  9, 10, 11]])
```

```
In [82]: np.concatenate([arr, arr], axis=1)
```

```
Out[82]: array([[ 0,  1,  2,  3,  0,  1,  2,  3],  
                [ 4,  5,  6,  7,  4,  5,  6,  7],  
                [ 8,  9, 10, 11,  8,  9, 10, 11]])
```

금융 데이터 핸들링을 위한 Python

데이터 핸들링

데이터 합치기

- > `s1 = pd.Series([0, 1], index=['a', 'b'])`
 - > `s2 = pd.Series([2, 3, 4], index=['c', 'd', 'e'])`
 - > `s3 = pd.Series([5, 6], index=['f', 'g'])`
 - > `pd.concat([s1, s2, s3])`
 - > `pd.concat([s1, s2, s3], axis=1)`
- `s1, s2, s3` 세 객체를 리스트로 묶어서 `concat` 함수에 전달하면 값과 색인을 연결
 - `concat` 함수는 `axis = 0`을 기본 값으로 하고 새로운 Series 객체를 생성함
 - 만약 `axis=1` 이라면 결과는 Series가 아니라 DataFrame 이 됨 (`axis=1`은 칼럼을 의미)

```
In [86]: pd.concat([s1, s2, s3])
```

```
Out[86]: a    0  
        b    1  
        c    2  
        d    3  
        e    4  
        f    5  
        g    6  
        dtype: int64
```

```
In [87]: pd.concat([s1, s2, s3], axis=1)
```

```
Out[87]:
```

	0	1	2
a	0.0	NaN	NaN
b	1.0	NaN	NaN
c	NaN	2.0	NaN
d	NaN	3.0	NaN
e	NaN	4.0	NaN
f	NaN	NaN	5.0
g	NaN	NaN	6.0

금융 데이터 핸들링을 위한 Python

데이터 핸들링

데이터 합치기

■ concat 함수 인자

인자	설명
objs	이어 붙일 pandas 객체의 사전이나 리스트 (필수 인자)
axis	이어 붙일 축 방향 기본값은 0
join	조인 방식 'inner'(내부조인, 교집합)와 'outer'(외부조인, 합집합)가 있으며 기본 값은 'outer'
join_axes	합집합/교집합을 수행하는 대신 다른 n-1축으로 사용할 색인을 지정
keys	이어붙일 객체나 이어붙인 축에 대한 계층 색인을 생성하는 데 연관된 값 리스트나 임의의 값이 들어있는 배열, 튜플의 배열 또는 배열의 리스트(levels 옵션에 다차원 배열이 넘어온 경우)가 될 수 있음

인자	설명
levels	계층 색인 레벨로 사용할 색인을 지정 keys가 넘어온 경우 여러 개의 색인을 지정
names	keys나 levels 혹은 둘 다 있을 경우, 생성된 계층 레벨을 위한 이름
verify_integrity	이어붙인 객체에 중복되는 축이 있는지 검사하고 있다면 예외를 발생시킴 기본 값은 False로, 중복을 허용
ignore_index	이어붙인 축의 색인을 유지하지 않고 range(total_length)로 새로운 색인을 생성
levels	계층 색인 레벨로 사용할 색인을 지정 keys가 넘어온 경우 여러 개의 색인을 지정

금융 데이터 핸들링을 위한 Python

데이터 핸들링

데이터 합치기

```
> a = pd.Series([np.nan, 2.5, np.nan, 3.5, 4.5, np.nan],
                 index=['f', 'e', 'd', 'c', 'b', 'a'])
> a
> b = pd.Series(np.arange(len(a),
                           dtype=np.float64),
                 index=['f', 'e', 'd', 'c', 'b', 'a'])
> b
> b[-1] = np.nan
> b
> np.where(pd.isnull(a), b, a)
```

- 두 데이터셋의 색인이 일부 겹치거나 전체가 겹치는 경우, 데이터를 합칠 때 merge나 이어붙이기로는 불가능함
- 벡터화된 if-else 구문을 표현하는 Numpy의 where 함수를 사용

a	b	b[-1]=np.nan b
f NaN	f 0.0	f 0.0
e 2.5	e 1.0	e 1.0
d NaN	d 2.0	d 2.0
c 3.5	c 3.0	c 3.0
b 4.5	b 4.0	b 4.0
a NaN	a 5.0	a NaN
dtype: float64	dtype: float64	dtype: float64

```
In [136]: np.where(pd.isnull(a), b, a)
```

```
Out[136]: array([ 0. ,  2.5,  2. ,  3.5,  4.5, nan])
```

금융 데이터 핸들링을 위한 Python

데이터 핸들링

재형성과 피벗

- 표 형식의 데이터를 재배포하는 연산을 재형성(reshaping) 또는 피벗 연산이라고 함
 - # stack : 데이터의 칼럼을 로우로 피벗 또는 회전
 - # unstack : 로우를 칼럼으로 피벗시킴
-
- ```
> data = pd.DataFrame(np.arange(6).reshape((2, 3)),
index=pd.Index(['Ohio', 'Colorado'],
name='state'),
columns=pd.Index(['one', 'two', 'three'], name='number'))
> data
> result = data.stack()
> result
> result.unstack()
```

```
In [138]: data
```

```
Out[138]:
```

| number   | one | two | three |
|----------|-----|-----|-------|
| state    |     |     |       |
| Ohio     | 0   | 1   | 2     |
| Colorado | 3   | 4   | 5     |

```
In [142]: result = data.stack()
result
```

```
Out[142]: state number
Ohio one 0
 two 1
 three 2
Colorado one 3
 two 4
 three 5
dtype: int32
```

```
In [143]: result.unstack()
```

```
Out[143]:
```

| number   | one | two | three |
|----------|-----|-----|-------|
| state    |     |     |       |
| Ohio     | 0   | 1   | 2     |
| Colorado | 3   | 4   | 5     |

# 금융 데이터 핸들링을 위한 Python

## 데이터 핸들링

### 재형성과 피벗

- > result.unstack(0)
- > result.unstack('state')
- unstack 메서드를 사용하면 앞에서 얻은 계층적 색인을 가진 Series로부터 DataFrame을 얻을 수 있음
- stack과 마찬가지로 보통 가장 안쪽에 있는 것 부터 끄집어내는데, 레벨 이름이나 숫자를 전달해서 끄집어낼 단계를 지정할 수 있음
- 해당 레벨에 있는 모든 값이 하위 그룹에 속하지 않을 경우 unstack을 하게 되면 누락된 데이터가 생길 수 있음을 주의

```
In [144]: result.unstack(0)
```

```
Out [144]:
```

| state  | Ohio | Colorado |
|--------|------|----------|
| number |      |          |
| one    | 0    | 3        |
| two    | 1    | 4        |
| three  | 2    | 5        |

```
In [145]: result.unstack('state')
```

```
Out [145]:
```

| state  | Ohio | Colorado |
|--------|------|----------|
| number |      |          |
| one    | 0    | 3        |
| two    | 1    | 4        |
| three  | 2    | 5        |

# 금융 데이터 핸들링을 위한 Python

## 데이터 핸들링

### 재형성과 피벗

```
> s1 = pd.Series([0, 1, 2, 3], index=['a', 'b', 'c', 'd'])
> s2 = pd.Series([4, 5, 6], index=['c', 'd', 'e'])
> s1
> s2

> data2 = pd.concat([s1, s2], keys=['one', 'two'])
> data2
```

```
In [147]: s1
Out[147]: a 0
 b 1
 c 2
 d 3
 dtype: int64
```

```
In [148]: s2
Out[148]: c 4
 d 5
 e 6
 dtype: int64
```

```
In [149]: data2 = pd.concat([s1, s2], keys=['one', 'two'])
 data2
Out[149]: one a 0
 b 1
 c 2
 d 3
 two c 4
 d 5
 e 6
 dtype: int64
```



# 금융 데이터 핸들링을 위한 Python

## 데이터 핸들링

### 재형성과 피벗

- > data2.unstack()
  - > data2.unstack().stack()
  - > data2.unstack().stack(dropna=False)
- stack 메서드는 누락된 데이터를 자동으로 걸러내기 때문에 연산을 쉽게 원상복구할 수 있음

```
In [150]: data2.unstack()
```

```
Out[150]:
```

|     | a   | b   | c   | d   | e   |
|-----|-----|-----|-----|-----|-----|
| one | 0.0 | 1.0 | 2.0 | 3.0 | NaN |
| two | NaN | NaN | 4.0 | 5.0 | 6.0 |

```
In [151]: data2.unstack().stack()
```

```
Out[151]: one a 0.0
 b 1.0
 c 2.0
 d 3.0
 two c 4.0
 d 5.0
 e 6.0
 dtype: float64
```

```
In [152]: data2.unstack().stack(dropna=False)
```

```
Out[152]: one a 0.0
 b 1.0
 c 2.0
 d 3.0
 e NaN
 two a NaN
 b NaN
 c 4.0
 d 5.0
 e 6.0
 dtype: float64
```

# End of Document