

python 실습

머신러닝 기반 빅데이터 엔지니어링 과정
빅데이터 X Campus (단국대학교)

2018.08

데이터사이언스 학과 이성신 석사과정

- 하나의 문제 혹은 특정 문제에 대한 해결책을 제시한 명령문들의 조합
- 해결하고자 하는 문제의 조건 혹은 입력값들에 대한 명확한 결과를 도출해야하며, 명령문들이 무한히 수행되지 않고 유한 번 수행한 후 마칠 수 있는 순서들로 만들어져야 함
- 즉, 주어진 문제를 해결하는 **단계별 절차(procedure)**를 말함. 위의 정의에서 '유한' 이라는 의미는 영원히 수행되지 않고 반드시 종료돼야 함을 뜻함

- 차를 만드는 알고리즘

1. 티백을 컵에 넣는다
2. 주전자에 물을 채운다
3. 주전자를 가스레인지에 올리고 물을 끓인다
4. 끓인 물을 용량에 맞게 컵에 따른다
5. 우유를 컵에 따른다
6. 설탕을 컵에 따른다
7. 차를 젖는다
8. 차를 마신다

- 차를 만드는 알고리즘

1. 티백을 컵에 넣는다
2. 주전자에 물을 채운다
3. 주전자를 가스레인지에 올리고 물을 끓인다
4. 끓인 물을 용량에 맞게 컵에 따른다
5. 우유를 컵에 따른다
6. 설탕을 컵에 따른다
7. 차를 짓는다
8. 차를 마신다

알고리즘은 명확한 절차들로 구성되어 있음!! 각 단계는 정해진 절차대로 구성되어야 하지만 몇몇 단계는 순서가 바뀔수도 있음

- 알고리즘 속성

- **입력값(input)**: 알고리즘은 입력 집합으로부터 유효한 입력값을 받아야 함
- **출력값(output)**: 입력값에 대한 출력값을 산출해야함. 출력값은 문제에 대한 해결책이어야 함
- **유한성(finiteness)**: 입력값에 대해 알고리즘은 유한한 단계 이후에 종료해야 함
- **명확성(definiteness)**: 알고리즘의 모든 단계는 명확히 정의되어야 함
- **효과성(effectiveness)**: 알고리즘의 각 단계는 유한한 시간 안에 정확하게 수행될 수 있을 정도로 충분히 단순해야 함

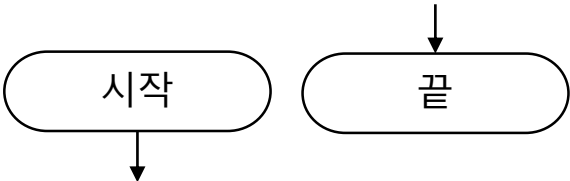
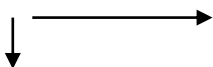
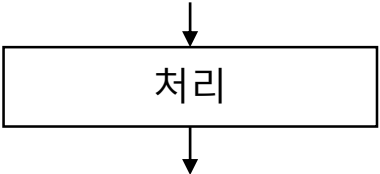
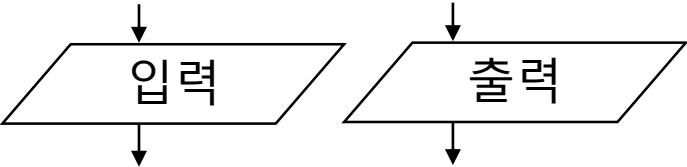

알고리즘 작성의 주요 3단계

- 알고리즘 작성의 주요 3단계
 - 데이터의 **입력**, 데이터의 **처리**, 결과 **출력**
- 예시
 - 세 숫자의 평균을 구하는 컴퓨터 프로그램
 1. 사용자로부터 세가지 숫자를 입력 받는다(**입력**)
 2. 세 숫자의 합을 계산하고, 합을 3으로 나눈다(**처리**)
 3. 결과를 화면에 출력한다(**출력**)

순서도(flowchart)

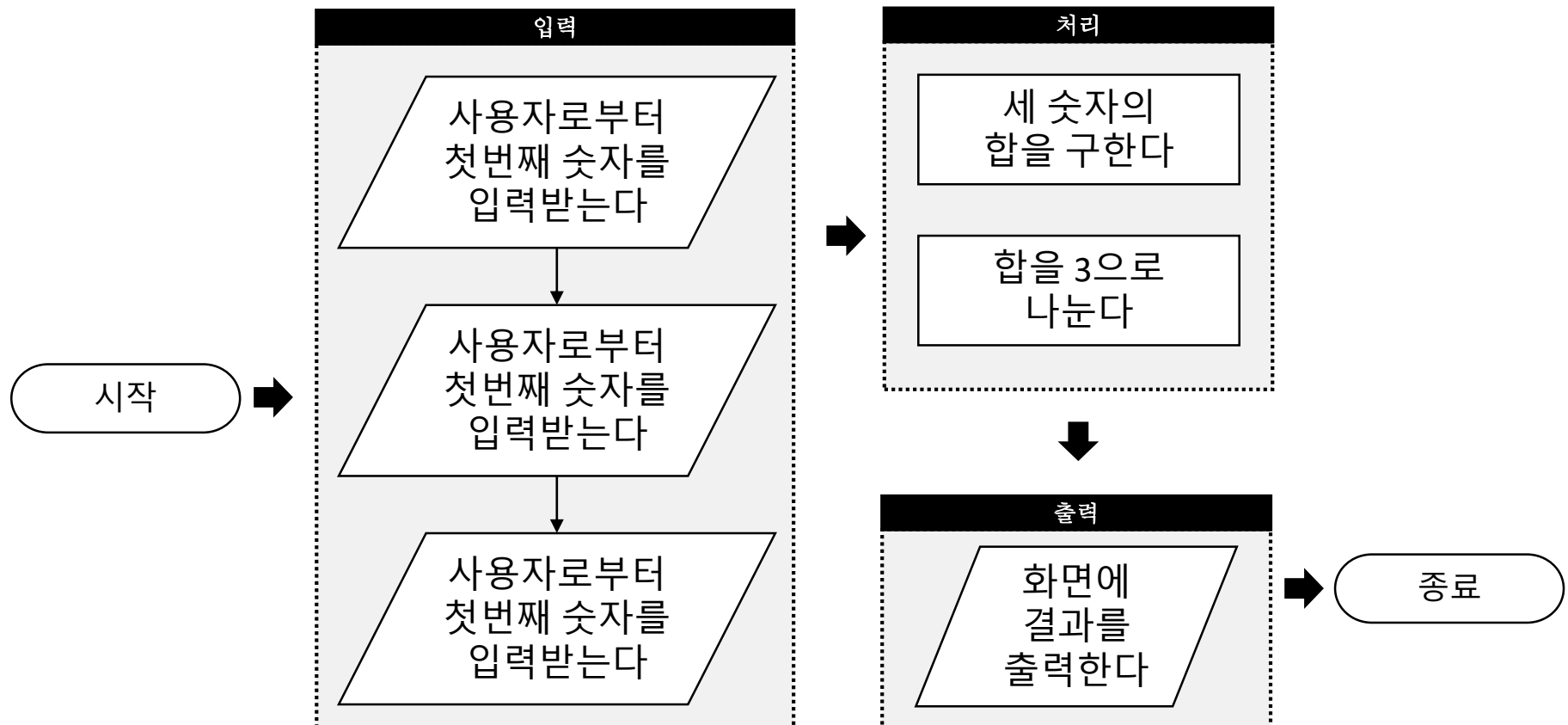
- 순서도는 알고리즘을 종이 혹은 화면에 도식적으로 표현하는 방법으로, 알고리즘의 수행 흐름을 시각적으로 표현함.
- 순서도를 통해 알고리즘의 시작부터 종료까지 각 단계의 수행 흐름이 어떻게 진행되는지를 시각적으로 확인할 수 있음

순서도 기호 및 기능

| 기호 | 설명 |
|---|--|
|  | 시작/종료: 알고리즘의 시작과 종료를 표현. 일반적으로 시작 기호에 한 개의 출구가 있고, 종료기호에 한 개의 입구가 있다. |
|  | 화살표: 수행 흐름을 보여줌. |
|  | 처리: 처리 혹은 산술적 연산을 표현. 처리 기호는 한 개의 입구와 한 개의 출구를 가짐 |
|  | 압력/출력: 데이터의 입력과 출력을 표현. 대부분의 경우 데이터는 키보드로부터 입력을 받으며, 결과를 화면에 출력. |
|  | 결정: 결정이 이루어지는 지점을 나타냄. |

순서도 예시

- 세 개의 숫자를 입력으로 받고 이들 숫자들의 평균값을 계산하여 화면에 출력하는 알고리즘 표현

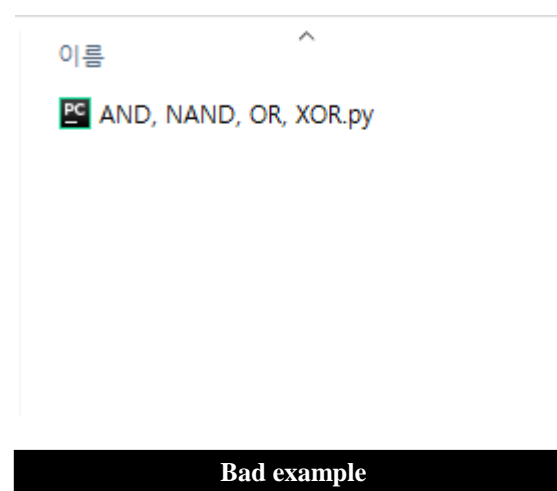
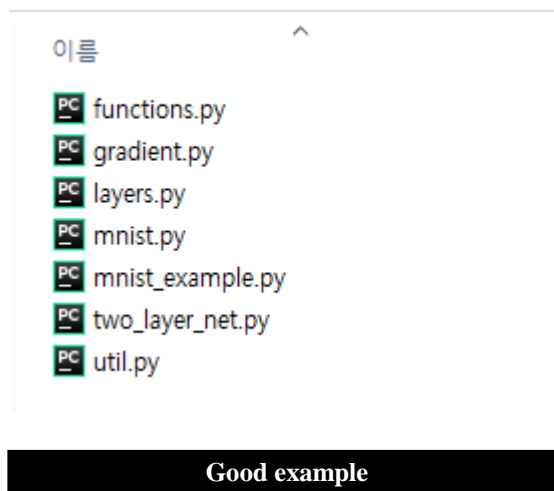


구문오류와 논리오류의 차이점?

- 구문오류(syntax error)
 - 키워드 오타, 문장 부호 생략, 중괄호 생략, 닫는 괄호 생략 등으로 발생하는 오류를 지칭
 - pycharm 또는 Jupyter Notebook 등 다양한 개발 프로그램은 이러한 구문 오류들을 자동으로 감지하고 구문 오류가 발생한 부분을 표시해줌
- 논리오류(logic error)
 - 프로그래머가 예상한 대로 수행되지 않는 것을 의미
 - 논리 오류가 있더라도 파이썬에서 어떠한 경고 메시지도 출력되지 않음
 - 즉 논리 오류가 있더라도 프로그램은 정상적으로 컴파일되고 수행되지만 예상하지 못한 수행 결과를 얻을 수 있음

파이썬 스타일 가이드: PEP8

- PEP8
 - 파이썬 창시자 Guido van Rossum이 작성한 파이썬 스타일 가이드
 - 참고: <https://www.python.org/dev/peps/pep-0008/>
- 대표적인 스타일 가이드
 - 모듈 이름은 only 소문자



파이썬 스타일 가이드: PEP8

- 대표적인 스타일 가이드

- 클래스 이름: PythonTutorial

단어의 시작알파벳만 대문자로 쓰고 나머지는 소문자로 꼭 이어서 작성

```
class PythonTutorial(object):
    def __init__(self):
        print("good example")
```

```
class pythonTutorial(object):
    def __init__(self):
        print("bad example1")

class python_tutorial(object):
    def __init__(self):
        print("bad example2")
```

- 함수 이름: snake_case

모든 단어는 소문자로!, 단어와 단어 사이는 언더바(_)로 분리

```
def python_function_name(arg1, arg2):
    print("good example")
```

```
def PythonFunctionName(arg1, arg2):
    print("bad example1")

def pythonFunctionName(arg1, arg2):
    print("bad example2")
```

파이썬 스타일 가이드: PEP8

- 대표적인 스타일 가이드
 - Boolean 자료형을 비교할 때 == 연산자 와 is 연산자를 사용하지 말것
 - 아래 출력결과 세가지 조건문 모두 기능은 동일하게 작동
- 그럼 왜 사용하지 말라는 것일까?
 - 가독성 있고 직관적으로 작성하기 위해서
 - is 연산자는 뒤쪽!

```

1  def main():
2      greeting = True
3
4      if greeting:
5          print("no operation")
6      if greeting == True:
7          print("'==' operation")
8      if greeting is True:
9          print("'is' operation")
    
```

```

1  main()
    
```

```

no operation
'==' operation
'is' operation
    
```

Equality(동등성) vs Identity (동일성)

- 파이썬에서 값을 비교할 때 어떤 형식으로 사용하라고 PEP8 권고문에 명시되어 있음
- 규칙
 1. True / False 를 비교할 때는 if 문에 변수만 사용
 2. 동등성을 검증할 때는 = 나 != 같은 연산자를 사용
 3. 동일성을 검증할 때는 is 나 is not 을 사용
- 구체적인 내용은 아래 링크 참조
 - 링크: <https://www.python.org/dev/peps/pep-0008/#programming-recommendations>

- 왜 다르게 표시될까?
 - is 는 동일성을 비교하는 연산자라서, 출력 결과가 다른 이유는
 - x, y, z 의 변수가 서로 다른 메모리 영역에 할당되기 때문

```
1 print("==compare identity==")
2 print(999 is 999)
3 x= 999; y=999
4 print(x is y)
5 z=999;
6 print(x is z)
```

```
==compare identity==
True
False
False
```

- 왜 다르게 표시될까?
 - 파이썬에서는 객체의 identity 를 검증해주는 id라는 내장함수가 존재

```
1 print("==compare identity==")
2 print(999 is 999)
3 x= 999; y=999
4 print(x is y)
5 z=999;
6 print(x is z)
7
8 print(id(x))
9 print(id(y))
10 print(id(z))
```

==compare identity==

True

False

False

2207985402064

2207985402512

2207985402352

같은 값이어도 서로 다른
identity 를 보유하고 있는 것으로 확인

- Google Python Style Guide
 - PEP8과 거의 유사함
 - PEP8과 비교해서 가장 다른점은 “Docstring”을 작성하는 방법
 - 구글 프로젝트(Tensorflow) 때문에 가장 많이 접하게 되는 스타일
- 예시

```
def add_two_numbers(number1, number2):  
    return number1 + number2
```

```
def add_two_numbers(number1, number2):  
    """Returns the sum of two numbers  
  
    Args:  
        number1 (int): First number to add  
        number2 (int): Second number to add  
  
    Returns:  
        int: Sum of `number1` and `number2`  
    """
```

Docstring

- Google Python Style Guide
 - PEP8과 거의 유사함
 - PEP8과 비교해서 가장 다른점은 “Docstring”을 작성하는 방법
 - 구글 프로젝트(Tensorflow) 때문에 가장 많이 접하게 되는 스타일
- 예시

```
def add_two_numbers(number1, number2):  
    return number1 + number2
```

Docstring 시작

```
def add_two_numbers(number1, number2):  
    """Returns the sum of two numbers  
  
    Args:  
        number1 (int): First number to add  
        number2 (int): Second number to add  
  
    Returns:  
        int: Sum of `number1` and `number2`  
    """
```

Docstring 끝

print 명령문의 다양한 출력 방법

- python의 print 명령문은 출력되는 항목들 사이에 공백을 자동으로 추가

```
1 print("아침", "점심", "저녁")
```

아침 점심 저녁

- print() 함수 내에 있는 string 들 사이 공백은 어떻게 띄워서 출력할까?

```
1 print("아침", "점심", "저녁")  
2 print("아침", "점심", "저녁")
```

아침 점심 저녁

아침 점심 저녁

print 명령문의 다양한 출력 방법

- python의 print 명령문은 출력되는 항목들 사이에 공백을 자동으로 추가

```
1 print("아침", "점심", "저녁")
```

아침 점심 저녁

- print() 함수 내에 있는 string 인자 사이 공백은 어떻게 띄워서 출력할까?

```
1 print("아침", "점심", "저녁")
2 print("아침", "점심", "저녁")
```

아침 점심 저녁
아침 점심 저녁

print 명령문의 다양한 출력 방법

- python의 print 명령문은 출력되는 항목들 사이에 공백을 자동으로 추가

```
1 print("아침", "점심", "저녁")
```

아침 점심 저녁

- print() 함수 내에 있는 string 들 사이 공백은 어떻게 띄워서 출력할까?

```
1 print("아침", "점심", "저녁")
2 print("아침", "점심", "저녁")
```

아침 점심 저녁
아침 점심 저녁

print 명령문의 다양한 출력 방법

- print() 함수내에 sep 옵션을 사용
 - sep='?' : ? 에 들어가는 형태에 따라 print의 출력결과가 다른것을 확인할 수 있음

```

1 print("아침", "점심", "저녁")
2 print("아침", "점심", "저녁", sep=' ')
3 print("아침", "점심", "저녁", sep='@')
    
```

아침 점심 저녁

아침 점심 저녁

아침@점심@저녁

print 명령문의 다양한 출력 방법

- print의 출력결과를 개행없이 한줄로 표현하고 싶을땐 어떻게?
 - print() 함수내에서 end 옵션을 추가
 - end 옵션은 print 함수 마지막에 추가될 문자열을 바꿀수 있음

```

1 print("아침", "점심", "저녁", end='')
2 print("아침", "점심", "저녁", sep=' ', end='')
3 print("아침", "점심", "저녁", sep='@', end='')
    
```

아침 점심 저녁아침 점심 저녁아침@점심@저녁

사용자로부터 입력을 받을 수 있는 명령어

```

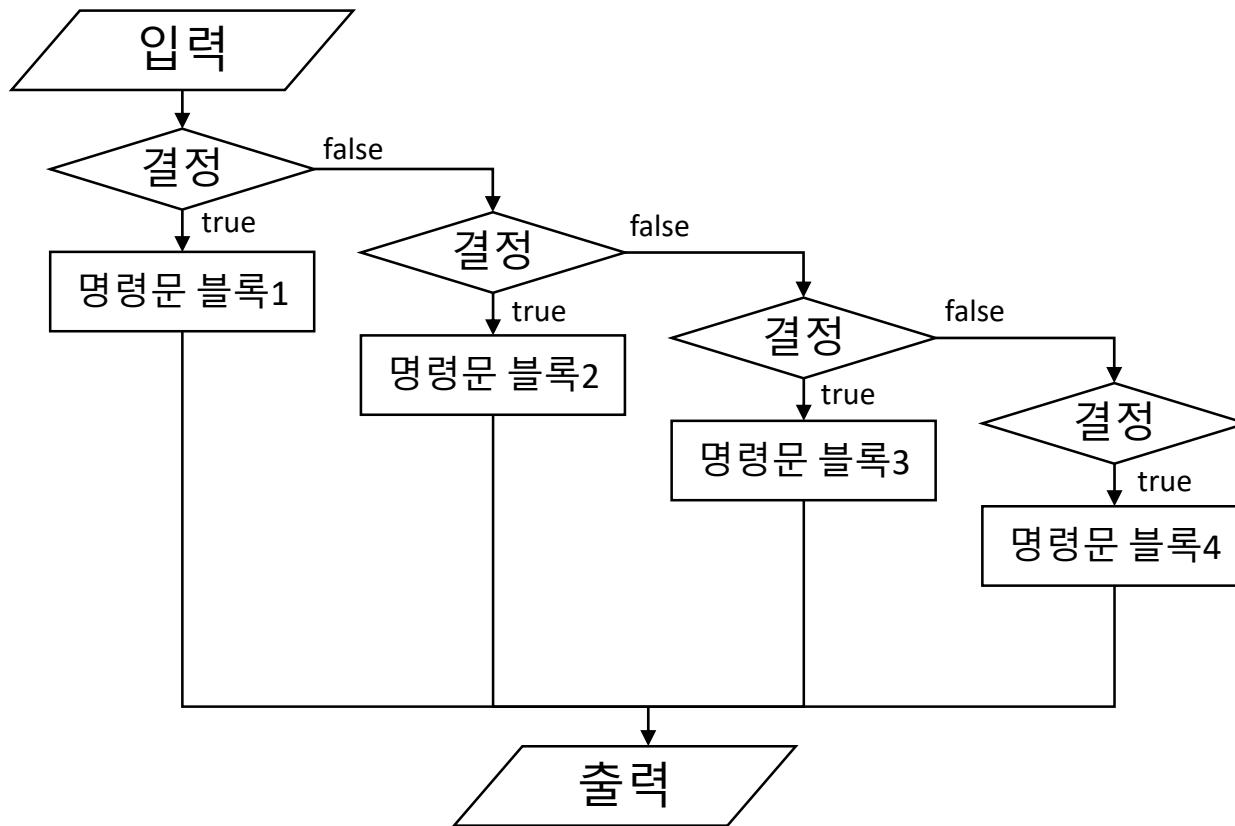
1 # prompt 화면에서 데이터를 입력받을
2 var_name_str = input()
3
4 # input 값으로 받는 데이터의 형식은 string 임
5 print("var_name_str's type:{}".format(type(var_name_str)))
6
7 # 정수 입력받기
8 var_name_int = int(input())
9 print("var_name_int's type:{}".format(type(var_name_int)))
10 print("var_name_int:{}".format(var_name_int))
11
12 # 실수 입력받기
13 var_name_float = float(input())
14 print("var_name_int's type:{}".format(type(var_name_float)))
15 print("var_name_int:{}".format(var_name_float))
16
17 # 여러 수 입력받기
18 num1, num2, num3 = input().split()
19 print("num1:{}, num2:{}, num3:{}".format(num1, num2, num3))
20 print("num1's type:{}, num2's type:{}, num3's type:{}".format(type(num1), type(num2), type(num3)))

```

10
var_name_str's type:<class 'str'>
10
var_name_int's type:<class 'int'>
var_name_int:10
10
var_name_int's type:<class 'float'>
var_name_int:10.0
10 20 30
num1:10, num2:20, num3:30
num1's type:<class 'str'>, num2's type:<class 'str'>, num3's type:<class 'str'>

제어문: if – 다중 택일 결정 구조

- 다중-택일 결정 구조는 다음 순서도와 같이 여러 개의 선택 경로 중에 하나를 선택하는 구조이다



제어문: if – 다중 택일 결정 구조

- 만약 불리언 식1의 결과가 False이면 불리언 식2가 평가된다.
- 불리언 식2의 결과가 True이면 명령문 블록 2를 수행하고, 나머지 다중-택일 결정 구조의 명령문 블록은 건너뛴다.
- 이러한 과정은 불리언 식의 결과가 True 이거나 하위에 더 이상의 불리언 식이 존재하지 않을때 까지 반복된다.

제어문: if – 다중 택일 결정 구조

- 예시

```

1 name = input("이름이 무엇인가요?")
2
3 if "최상일" == name:
4     print("교수님 이시군요!")
5 elif "사나" == name:
6     print("트와이스 군요!")
7 elif "그루트" == name:
8     print("I am Groot!!")
9 else:
10    print("죄송하지만 나는 당신을 모릅니다.")
    
```

이름이 무엇인가요?그루트
I am Groot!!

하나의 불리언 식만 출력

- python의 range() 함수는 연속적인 정수를 생성하는데 사용
- 이 함수를 for 명령문과 함께 사용하면 다음과 같이 확장할 수 있음

```
1 for var in range([initial_value], final_value, [step]):  
2     명령문 블록
```

- initial_value는 연속적인 정수의 시작값을 나타냄. 이 인자는 옵션. 생략하면 기본값은 0임.
- final_value는 연속적인 정수의 마지막을 나타낸다. 그러나 final_value는 포함되지 않음.
- step은 연속한 두 정수 사이의 차이를 나타낸다. 이 인자는 옵션이다. 생략하면 기본값은 1임.

- 예시

```
1 print("0~10까지 순서대로 증가")
2 for i in range(0, 11, 1):
3     print("{}".format(i), end=' ')
4 print()
5 print("10~0까지 순서대로 감소")
6 for i in range(10, 0, -1):
7     print("{}".format(i), end=' ')
```

0~10까지 순서대로 증가
0 1 2 3 4 5 6 7 8 9 10
10~0까지 순서대로 감소
10 9 8 7 6 5 4 3 2 1

for-루프 주의사항

```
1 for var in range([initial_value], final_value, [step]):
2     명령문 블록
```

- 규칙1: step 변수는 0값을 가지지 않아야 한다.
- 규칙2: initial_value가 final_value 보다 작고 step이 음수이면, 루프는 0회의 반복을 수행한다. 아래 예제는 화면에 아무것도 출력하지 않는다.

```
1 for i in range(5, 9, -1):
2     print("i:{}".format(i), end='')
```

- 규칙3: initial_value가 final_value보다 크고, step이 양수이면, 루프는 0회의 반복을 수행한다. 아래 예제는 화면에 아무것도 출력하지 않는다.

```
1 for i in range(10, 6):
2     print("i:{}".format(i), end='')
```

- 첫번째 방법: break 문, 파이썬기초 강의자료 84페이지

```
1 s = "I have a dream"
2
3 letter = input("검색할 영문자를 입력해주세요:")
4
5 found = False
6 for a in s:
7     if letter == a:
8         found = True
9         break
10
11 if found == True:
12     print("문자 {} 를 찾았습니다.".format(letter))
```

검색할 영문자를 입력해주세요:h
문자 h 를 찾았습니다.

- 두번째 방법: 플래그(flag)사용

```
1 s = "I have a dream"
2
3 letter = input("검색할 영문자를 입력해주세요:")
4 found = False
5 i = 0
6 while i <= len(s) - 1 and False == found:
7     if s[i] == letter:
8         found = True
9         i = i + 1
10 if found == True:
11     print("문자 {} 를 찾았습니다.".format(letter))
```

검색할 영문자를 입력해주세요:h
문자 h 를 찾았습니다.

루프 벗어나기

- 세번째 방법 – 루프를 사용하지 않기
 - python 만의 강력한 기능

```

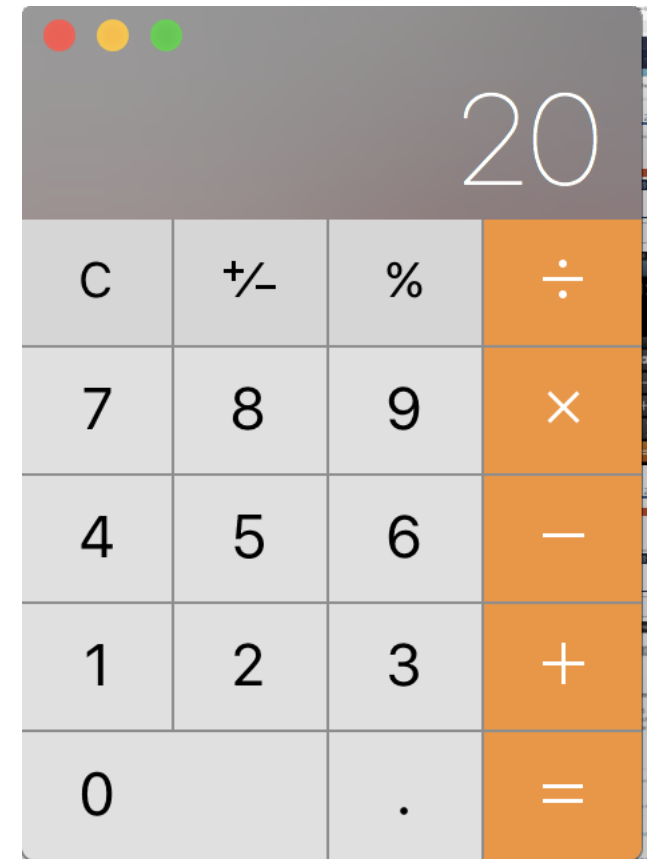
1  s = "I have a dream"
2
3  letter = input("검색할 영문자를 입력해주세요:")
4
5  if letter in s:
6      print("문자 {} 를 찾았습니다.".format(letter))
    
```

검색할 영문자를 입력해주세요:h
문자 h 를 찾았습니다.

문제1 – 제어문 문제

- 계산기에서 연산기능을 모두 구현해보기
- 알고리즘 3단계
 - 입력: 숫자, 연산자, 숫자를 입력으로 받기
 - 처리: 계산기 기능 구현
 - 출력: 계산기와 동일한 결과 출력
 - 출력예시

숫자1, 연산자, 숫자2 를 입력해주세요: 1 + 3
1 + 3 = 4



문제2 – 반복문 문제

- 구구단 출력기 만들어보기
- 알고리즘 3단계
 - 입력: 출력을 원하는 구구단 '수' (n)를 입력
 - 처리: 입력받은 수와 구구단 계산
 - 출력: 구구단 출력
 - 출력예시

출력을 원하는 구구단수를 입력해주세요:2

$$2 \times 1 = 2$$

$$2 \times 2 = 4$$

$$2 \times 3 = 6$$

$$2 \times 4 = 8$$

$$2 \times 5 = 10$$

$$2 \times 6 = 12$$

$$2 \times 7 = 14$$

$$2 \times 8 = 16$$

$$2 \times 9 = 18$$

문제2 - 응용

- 구구단 출력기 응용
- 알고리즘 3단계
 - 입력: 두개의 수를 입력받아서 해당 수 사이의 모든 구구단을 출력
 - 처리: 입력받은 수와 구구단 계산
 - 출력: 구구단 출력
 - 출력예시

출력을 원하는 구구단수를 입력해주세요: 2 5

| | | | | | | | | |
|------------------|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|
| $2 \times 1 = 2$ | $2 \times 2 = 4$ | $2 \times 3 = 6$ | $2 \times 4 = 8$ | $2 \times 5 = 10$ | $2 \times 6 = 12$ | $2 \times 7 = 14$ | $2 \times 8 = 16$ | $2 \times 9 = 18$ |
| $3 \times 1 = 3$ | $3 \times 2 = 6$ | $3 \times 3 = 9$ | $3 \times 4 = 12$ | $3 \times 5 = 15$ | $3 \times 6 = 18$ | $3 \times 7 = 21$ | $3 \times 8 = 24$ | $3 \times 9 = 27$ |
| $4 \times 1 = 4$ | $4 \times 2 = 8$ | $4 \times 3 = 12$ | $4 \times 4 = 16$ | $4 \times 5 = 20$ | $4 \times 6 = 24$ | $4 \times 7 = 28$ | $4 \times 8 = 32$ | $4 \times 9 = 36$ |
| $5 \times 1 = 5$ | $5 \times 2 = 10$ | $5 \times 3 = 15$ | $5 \times 4 = 20$ | $5 \times 5 = 25$ | $5 \times 6 = 30$ | $5 \times 7 = 35$ | $5 \times 8 = 40$ | $5 \times 9 = 45$ |

문제3 – 반복문 문제

- for문을 이용한 별찍기문제 – 삼각형
- 알고리즘 3단계
 - 입력: 출력을 원하는 삼각형의 높이값을 입력
 - 처리: 입력으로 받은 숫자만큼 “*” 을 이용하여 삼각형을 만듦
 - 출력: 삼각형 출력
 - 출력예시

```
5
  *
 ***
*****
*****
*****
```

- 7

*

리스트 - 1차원 리스트 만들기

- 첫번째 방법
 - size 개수만큼의 요소를 가지는 리스트를 생성하기

```

1 size = 4
2 list_name = [None]*size
3
4 list_name[0] = 'A'
5 list_name[1] = 'B'
6 list_name[2] = 'C'
7 list_name[3] = 'D'
8
9 list_name
    
```

['A', 'B', 'C', 'D']

- 상수값 인덱스 대신, 변수나 표현식을 사용할 수 있음

```

1 size = 4
2 k = 0
3 list_name2 = [None]*size
4
5 list_name2[k] = 'A'
6 list_name2[k+1] = 'B'
7 list_name2[k+2] = 'C'
8 list_name2[k+3] = 'D'
9
10 list_name2
    
```

['A', 'B', 'C', 'D']

리스트 - 1차원 리스트 만들기

- 두번째 방법
 - 리스트를 생성하고 생성된 리스트에 값을 저장하는 방법(교재 49쪽)

```

1 list1 = [] # 빈 리스트
2 list2 = [1,2,3] # 정수형을 요소값으로 가진 리스트
3 list3 = ['dankook', 'MLPA', 'AI'] # 문자열을 요소값으로 가진 리스트
4 list3 = [1,2, 'dankook', 'MLPA'] # 정수형 및 문자열을 같이 가진 리스트
5 list4 = [1,2, 'dankook', [1,2,3]] # 정수형, 문자열, 리스트를 가진 리스트
    
```

- 세번째 방법 ← 몇 개의 원소를 저장할지 모를때, 주로 이 방법을 사용
 - 리스트 내장 함수인 append(객체값)을 사용하는 방법(교재 53쪽)

```

1 list_test = [1,2,3,4]
2 list_test.append(5)
3 print("list_test.append(5):{}".format(list_test))
4
5 list_test.extend([6,7,8])
6 print("list_test.extend([6,7,8]):{}".format(list_test))
7
8 list_test.insert(0,0)
9 print("list_test.insert(0,0):{}".format(list_test))
10
11 list_test.clear()
12 print("list_test.clear(): {}".format(list_test))

list_test.append(5):[1, 2, 3, 4, 5]
list_test.extend([6,7,8]):[1, 2, 3, 4, 5, 6, 7, 8]
list_test.insert(0,0):[0, 1, 2, 3, 4, 5, 6, 7, 8]
list_test.clear(): []
    
```

1차원 리스트에 반복문 적용

- 역순으로 단어 출력하기
 - 슬라이싱(slicing) 기법을 사용하여 역순으로 출력

```

1 words = [None] * 5
2 for i in range(5):
3     words[i] = input()
4
5 print("=====")
6 for word in words[::-1]:
7     print(word)
    
```

```

A
B
C
D
E
=====
E
D
C
B
A
    
```


문제4 - 정렬문제

- 역순으로 양수 출력하기
- 알고리즘 3단계
 - 입력: 사용자로부터 10개의 숫자를 입력받기
 - 처리: 양수인지 음수인지 판단
 - 출력: 양수값을 내림차순으로 출력
 - 출력예시

입력데이터

-10

9

-7

4

100

=====

출력데이터

100


9

4

2차원 리스트

- 2차원 리스트를 행 단위로 처리하는 방법

| | | | | |
|---|---|----|----|----|
| 2 | 3 | 5 | 2 | 9 |
| 9 | 8 | 3 | 14 | 12 |
| 5 | 2 | 15 | 20 | 9 |
| 7 | 8 | 3 | 5 | 6 |



| |
|----|
| 21 |
| 46 |
| 51 |
| 29 |

- 보조 리스트 만들기
 - math 모듈을 import
 - math 클래스 내에 있는 fsum 함수를 사용하여 입력 list의 합을 구하는 방법

```
1 import math
2 matrix = [[2,3,5,2,9], [9,8,3,14,12], [5,2,15,20,9], [7,8,3,5,6]]
3 total = []
4
5 for row in matrix:
6     print("row:{}'s sum = {}".format(row, math.fsum(row)))
7     total.append(math.fsum(row))
8 print()
9 print("total:{}".format(total))
```

```
row:[2, 3, 5, 2, 9]'s sum = 21.0
row:[9, 8, 3, 14, 12]'s sum = 46.0
row:[5, 2, 15, 20, 9]'s sum = 51.0
row:[7, 8, 3, 5, 6]'s sum = 29.0
```

```
total:[21.0, 46.0, 51.0, 29.0]
```

- 핸드폰에서 전화번호 조회하기
- 기능
 - 사람의 이름과 그 사람에 관련된 전화번호를 추가
 - 사람 이름을 입력하면 그 이름과 관련된 전화번호를 알려줌

```
1 # dictionary 정의
2 phone_book = {}
3 phone_book['jenny'] = 3180053657
4 phone_book['python'] = 1093217490
5
6 # python의 전화번호
7 print("{} ".format(phone_book['python']))
```

1093217490

- 웹사이트를 접속하고 싶을때
 - 인터넷 창에 주소를 입력하면 이 주소는 내부적으로 IP주소로 바뀌어서 실행
 - 이렇게 웹 주소에 대해 IP 주소를 할당하는 작업을 DNS(Domain Name Service)작업이라고 함

GOOGLE.COM → 74.125.239.133
FACEBOOK.COM → 173.252.128.6
SCRIBD.COM → 23.235.47.175

- 언제 이 자료구조를 사용하면 좋을까?
 - 딕셔너리는 A라는 것과 B라는것을 연관시키고자 할 때
 - 무엇인가를 찾아보고자 할때

문제5 – 자료구조 응용

- 5개의 과목에 대한 성적을 받은 4명의 학생이 있다고 가정해 보자. 이때 모든 과목의 학생성적을 입력받아 학생별 평균을 계산하고 85점보다 높은 학생의 이름을 출력하는 프로그램을 작성하여라.
- 알고리즘 3단계
 - 입력: 사용자로부터 ‘이름’ 과 ‘과목5개의 점수’ 를 입력받음
 - 입력은 다음과 같이 입력
 - 철수: 90 100 75 65 85
 - 영희: 93 97 88 79 86
 - 길동: 88 98 95 82 76
 - 영심: 85 91 78 73 63
 - 처리: 각 학생별로 평균을 구한후에, 평균85점보다 높은 학생의 이름을 찾음
 - 출력: 학생의 이름을 출력
 - 힌트: 리스트 + 딕셔너리

- 출력예시

몇명의 학생들의 성적을 입력하실건가요?4

이름을 입력해주세요:철수

과목 점수를 입력해주세요(각 과목의 점수는 띄어쓰기로 구분): 90 100 75 65 85

이름을 입력해주세요:영희

과목 점수를 입력해주세요(각 과목의 점수는 띄어쓰기로 구분): 93 97 88 79 86

이름을 입력해주세요:길동

과목 점수를 입력해주세요(각 과목의 점수는 띄어쓰기로 구분): 88 98 95 82 76

이름을 입력해주세요:영심

과목 점수를 입력해주세요(각 과목의 점수는 띄어쓰기로 구분): 85 91 78 73 63

평균 85점 이상인 사람은

영희 길동 입니다.

출력결과



- 클래스를 이용하여 은행 입출금 프로그램 만들기
 - 계좌 클래스를 관리하는 은행에서는 계좌가 총 몇 개 개설되었는지 알고 싶을 것입니다. 개설된 계좌 수를 클래스 멤버로 두어 인스턴스를 만들 때마다 하나씩 늘리면 쉽게 알 수 있겠지요. 하지만 계좌 이용 고객은 개설된 계좌의 수를 인스턴스 멤버로 가질 필요가 없습니다. 고객의 이름이나 잔액처럼 객체마다 다른 데이터만 가지면 됩니다.
- 알고리즘 3단계
 - 입력: 객체생성
 - greg 와 john 인스턴스 생성
 - greg는 5000원을 갖고 있고, john 은 1000원을 갖고 있음
 - 처리: 인스턴스 메서드를 호출할 때마다, 멤버 변수값을 수정 및 업데이트
 - 출력: 인스턴스 메서드 호출한 이후 결과값

- 출력 예시

```
user: greg, balance: 5000  
user: John, balance: 1000
```

```
deposit  
user: greg, balance: 5500
```

```
withdraw  
withdraw money:1500
```

```
class number  
The number of accounts:2
```

```
message passing  
user: greg, balance: 4000  
user: John, balance: 1000
```

```
transfer succeeded  
user: greg, balance: 2000  
user: John, balance: 3000
```

문제7 – 클래스 상속문제

- 클래스 계층 구조를 설계하는 예제
- 클래스 계층을 설계할 때는 다음 두 가지를 고려해야 함.
 1. 공통 부분을 기본 클래스로 묶는다. 이렇게 하면 코드를 재사용할 수 있음.
 2. 부모가 추상 클래스인 경우를 제외하고, 파생 클래스에서 기본 클래스의 여러 메서드를 오버라이딩 한다면 파생 클래스는 만들지 않는 것이 좋음.
 - 추상클래스는 공통 특성을 가진 부모클래스를 만들고 해당 클래스는 객체 인스턴스를 생성할 수 없게 하는 클래스를 의미함.

문제7 – 클래스 상속문제

- 게임 캐릭터 클래스를 만든다고 가정.
- 게임에 등장하는 캐릭터는 플레이어와 몬스터임.
- 모든 캐릭터(추상 클래스)는 다음과 같은 특성을 가짐
 - 인스턴스 멤버: 이름(name), 체력(hp), 공격력(power)
 - 인스턴스 메서드: 공격(attack) 할수 있고 공격당하면 피해(get_damage)를 입음.
 - 인스턴스 메서드는 모두 추상 메서드로 구현
- 추상 클래스를 정의할때는 `from abc import *` 모듈이 필요

문제7 – 클래스 상속문제

- Character 추상 클래스

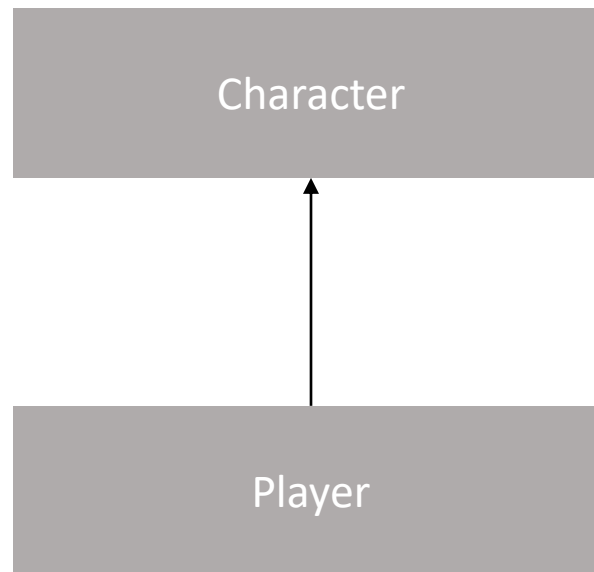
```
1  from abc import *
2
3  class Character(metaclass = ABCMeta):
4      def __init__(self, name, hp, power):
5          self.name = name
6          self.HP = hp
7          self.power = power
8
9      # 추상 메서드
10     # 파생 클래스는 반드시 attack()과 get_damage() 메서드를 오버라이딩해야 함.
11     @abstractmethod
12     def attack(self, other, attack_kind):
13         pass
14
15     @abstractmethod
16     def get_damage(self, power, attack_kind):
17         pass
18
19     def __str__(self):
20         return '{} : {}'.format(self.name, self.HP)
21
```

문제7 – 클래스 상속문제

- Player 클래스 만들기
- Player는 다음과 같은 특성이 있음
 - 추가되는 멤버: 플레이어는 다양한 공격 종류를 담을 수 있는 기술 목록 (Skills)이 있음.
 - attack(): 플레이어는 공격할 때 공격 종류가 기술 목록 안에 있다면 상대 몬스터에게 피해를 입힐 수 있음
 - get_damage(): 플레이어가 피해를 입을 때 몬스터의 공격 종류가 플레이어의 기술 목록에 있다면 몬스터의 공격력이 반감되어 hp가 공격력의 반절만 깎임.

문제7 – 클래스 상속문제

- Character 클래스를 상속받아, Player 클래스를 선언해보세요
- Character 클래스 디자인



문제7 – 클래스 상속문제

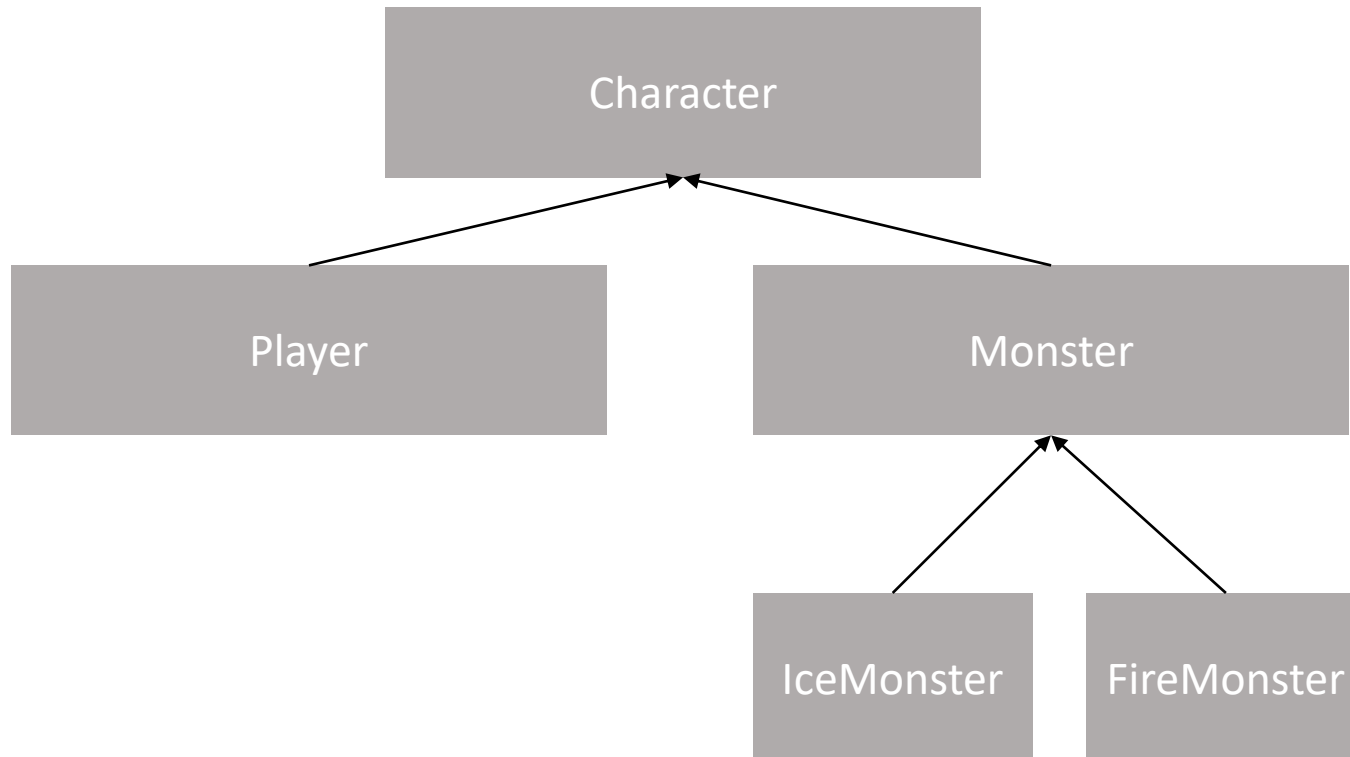
- Monster, IceMonster, FireMonster 클래스 만들기
- 몬스터에는 불과 얼음 몬스터가 있으며 다음과 같은 특징이 있음
 - 추가되는 멤버: 공격 종류(attack_kind)를 가짐. 불 몬스터는 Fire, 아이스 몬스터는 Ice 를 가짐
 - 공통 메서드: 두 몬스터는 같은 행동을 함
 - attack: 공격 종류가 몬스터의 속성과 같다면 공격한다.
 - get_damage: 몬스터는 자신과 속성이 같은 공격을 당하면 체력이 오히려 공격력 만큼 증가한다. 그렇지 않으면 체력이 공격력만큼 감소한다.
 - 서로 다른 메서드: 불 몬스터는 얼음 몬스터는 하지 않는 특별한 행동 (fireball, 실제로는 구현하지 않고 fireball만 출력)을 한다.

문제7 – 클래스 상속문제

- Monster 클래스는 Character 클래스와 멤버도 겹치고 fireball 메서드를 제외하고 겹치기 때문에 Character 클래스를 상속받아 만든다.
- 또한 IceMonster 나 FireMonster 는 공통된 부분이 많으므로 이 부분을 묶어 기본 클래스인 Monster 클래스에 둬

문제7 – 클래스 상속문제

- Character 클래스 디자인



문제7 – 클래스 상속문제

- Monster 클래스를 상속받아, IceMonster와 FireMonster 클래스를 선언해 보세요

```
1 class Monster(Character):
2     def __init__(self, name, hp, power):
3         super().__init__(name, hp, power)
4         self.attack_kind = 'None'
5
6     def attack(self, other, attack_kind):
7         if self.attack_kind == attack_kind:
8             other.get_damage(self.power, attack_kind)
9
10    # 플레이어가 불 공격을 할 때
11    # 공격받는 객체가 얼음 몬스터라면 체력이 깎이고
12    # 공격받는 객체가 불 몬스터라면 체력이 늘어난다.
13    def get_damage(self, power, attack_kind):
14        if self.attack_kind == attack_kind:
15            self.HP = self.HP + power
16        else:
17            self.HP = self.HP - power
18
19    def get_attack_kind(self):
20        return self.attack_kind
```

문제7 – 클래스 상속문제

- 게임 실행 예

```
1 player = Player('sword master', 100, 30, 'ICE')
2 monsters = []
3 monsters.append(IceMonster())
4 monsters.append(FireMonster())
5
6 for monster in monsters:
7     print(monster)
8
9 for monster in monsters:
10     # 공격을 받는 몬스터 객체가
11     # 어떤 몬스터인지에 따라 호출되는 메서드가 달라지고
12     # 그에 따라 결과도 달라짐
13     player.attack(monster, 'ICE')
14
15 print('after the player attacked')
16
17 for monster in monsters:
18     print(monster)
19 print('')
20
21 for monster in monsters:
22     monster.attack(player, monster.get_attack_kind())
23 print("after monsters attacked")
24 print(player)
```

```
Ice monster : 50
Fire monster : 50
after the player attacked
Ice monster : 80
Fire monster : 20
```

```
after monsters attacked
sword master : 75
```

문제8 - 파일 읽고 쓰기

- test.txt 파일에 “Weather is too hot to study !!” 라는 문자열을 저장한 후 다시 파일을 읽어서 출력하는 프로그램을 작성해보세요.

```

1  f1 = open("test.txt", 'w')
2  f1.write("Weather is too hot to study !!")
3
4  f2 = open("test.txt", 'r')
5  print(f2.read())
6

```

- !!? 출력이 안되고 있는데 왜 그런걸까요? “Weather is too hot to study !!” 문자열이 나오기 위해선 어떻게 해야될까요?

문제9 - 파일저장

- 문제2에서 풀었던 구구단의 출력결과를 99dan.txt 파일에 저장하는 프로그램을 작성해보세요.
- 단! 프로그램을 다시 실행하더라도 기존에 작성한 내용은 유지하고 새로 입력한 내용이 추가되어야 합니다.

• 알고리즘 3단계

- 입력: 출력을 원하는 구구단 '수' (n)를 입력
- 처리: 입력받은 수와 구구단 계산
- 출력: 구구단의 출력결과를 99dan.txt 파일에 저장
 - 출력예시

| | | | | | | | | |
|-----------|------------|------------|------------|------------|------------|------------|------------|------------|
| 2 x 1 = 2 | 2 x 2 = 4 | 2 x 3 = 6 | 2 x 4 = 8 | 2 x 5 = 10 | 2 x 6 = 12 | 2 x 7 = 14 | 2 x 8 = 16 | 2 x 9 = 18 |
| 3 x 1 = 3 | 3 x 2 = 6 | 3 x 3 = 9 | 3 x 4 = 12 | 3 x 5 = 15 | 3 x 6 = 18 | 3 x 7 = 21 | 3 x 8 = 24 | 3 x 9 = 27 |
| 4 x 1 = 4 | 4 x 2 = 8 | 4 x 3 = 12 | 4 x 4 = 16 | 4 x 5 = 20 | 4 x 6 = 24 | 4 x 7 = 28 | 4 x 8 = 32 | 4 x 9 = 36 |
| 2 x 1 = 2 | 2 x 2 = 4 | 2 x 3 = 6 | 2 x 4 = 8 | 2 x 5 = 10 | 2 x 6 = 12 | 2 x 7 = 14 | 2 x 8 = 16 | 2 x 9 = 18 |
| 3 x 1 = 3 | 3 x 2 = 6 | 3 x 3 = 9 | 3 x 4 = 12 | 3 x 5 = 15 | 3 x 6 = 18 | 3 x 7 = 21 | 3 x 8 = 24 | 3 x 9 = 27 |
| 4 x 1 = 4 | 4 x 2 = 8 | 4 x 3 = 12 | 4 x 4 = 16 | 4 x 5 = 20 | 4 x 6 = 24 | 4 x 7 = 28 | 4 x 8 = 32 | 4 x 9 = 36 |
| 5 x 1 = 5 | 5 x 2 = 10 | 5 x 3 = 15 | 5 x 4 = 20 | 5 x 5 = 25 | 5 x 6 = 30 | 5 x 7 = 35 | 5 x 8 = 40 | 5 x 9 = 45 |
| 6 x 1 = 6 | 6 x 2 = 12 | 6 x 3 = 18 | 6 x 4 = 24 | 6 x 5 = 30 | 6 x 6 = 36 | 6 x 7 = 42 | 6 x 8 = 48 | 6 x 9 = 54 |
| 7 x 1 = 7 | 7 x 2 = 14 | 7 x 3 = 21 | 7 x 4 = 28 | 7 x 5 = 35 | 7 x 6 = 42 | 7 x 7 = 49 | 7 x 8 = 56 | 7 x 9 = 63 |
| 8 x 1 = 8 | 8 x 2 = 16 | 8 x 3 = 24 | 8 x 4 = 32 | 8 x 5 = 40 | 8 x 6 = 48 | 8 x 7 = 56 | 8 x 8 = 64 | 8 x 9 = 72 |

문제10 – 평균값 구하기

- sample.txt 파일에 저장된 숫자들의 평균을 구한후 result.txt 파일에 쓰는 프로그램을 작성해보세요
- 평균을 구하기 전에, 메모장을 열어서 다음과 같이 입력해주세요

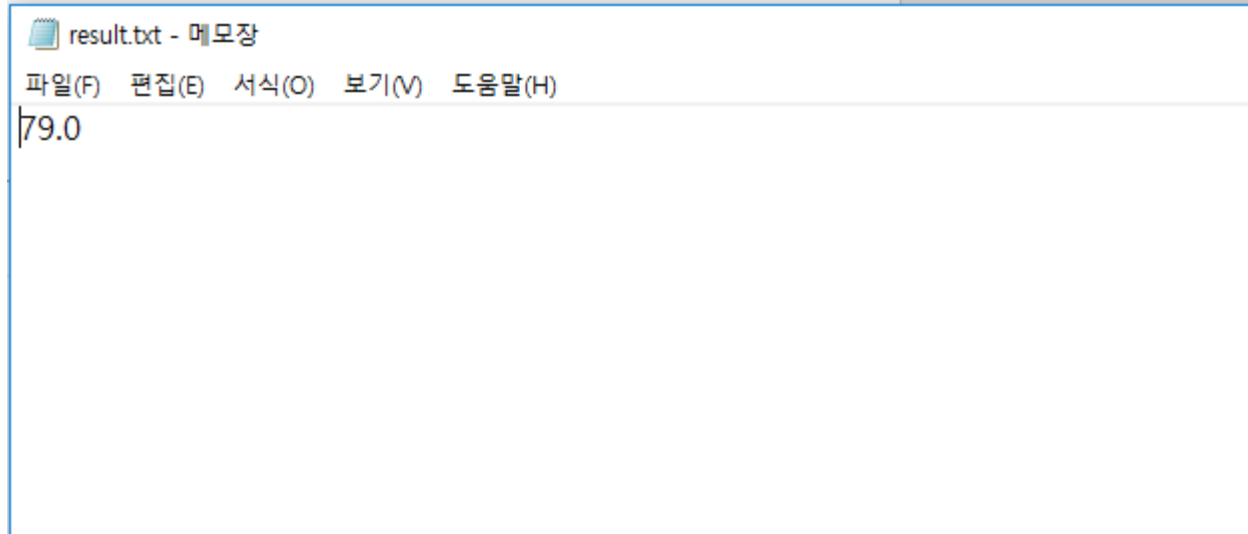
sample.txt - 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

70
60
55
75
95
90
80
80
85
100

문제10 – 평균값 구하기

- 주의: write(값) 에서 값의 자료형태는 string 이어야함
- 알고리즘 3단계
 - 입력: sample.txt 파일을 읽어오기
 - 처리: 읽어온 숫자의 평균을 구하기
 - 출력: 평균값을 result.txt 파일에 저장하기
 - 출력예시



폴더 내 파일들 불러오기

- glob 라는 패키지를 사용하여 특정 폴더 내 원하는 파일들을 불러오기
- 사용방법
 - import glob
 - paths = glob.glob('folder_path')
 - 주의사항: 윈도우의 경우 r'folder_path' , folder_path 앞에 r 을 붙여줘야함.(유니 코드 변환방식 때문)
 - ubuntu 나 macOS 는 r없이 'folder_path'만 적어주면 됨

폴더 내 파일들 불러오기

- 실행예시
- 폴더 내에 있는 txt 파일들이 불러와지는 것을 볼 수 있음
- txt 파일 뿐만 아니라 다른 확장자의 파일형식도 불러올 수 있음

```

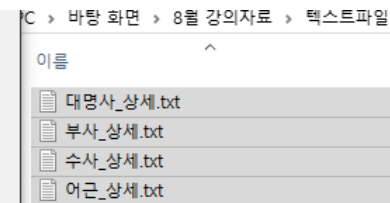
1 dic = {}
2 key_list = {'부사': 'AD', '대명사': 'NNP', '수사': 'NU', '어근': 'XR'}
3 import glob
4 paths = glob.glob(r'C:\Users\User\Desktop\강의자료\텍스트파일\*.*.txt')
5
6 for path in paths:
7     print("path:{}".format(path))
8     key = None
9     postagname = path.split('_')[0].split('###')[-1]
10    print("postagname:{}".format(postagname))
11
12    key_name = key_list[postagname]
13    print("key_name:{}".format(key_name))
14
15    with open(path, "r", encoding='utf-8-sig') as f:
16        lines = f.read().split()
17        for line in lines:
18            if not key_name in dic.keys():
19                dic[key_name] = [line]
20            else:
21                dic[key_name].append(line)

```

```

path: C:\Users\User\Desktop\강의자료\텍스트파일\대명사_상세.txt
postagname: 대명사
key_name: NNP
path: C:\Users\User\Desktop\강의자료\텍스트파일\부사_상세.txt
postagname: 부사
key_name: AD
path: C:\Users\User\Desktop\강의자료\텍스트파일\수사_상세.txt
postagname: 수사
key_name: NU
path: C:\Users\User\Desktop\강의자료\텍스트파일\어근_상세.txt
postagname: 어근
key_name: XR

```

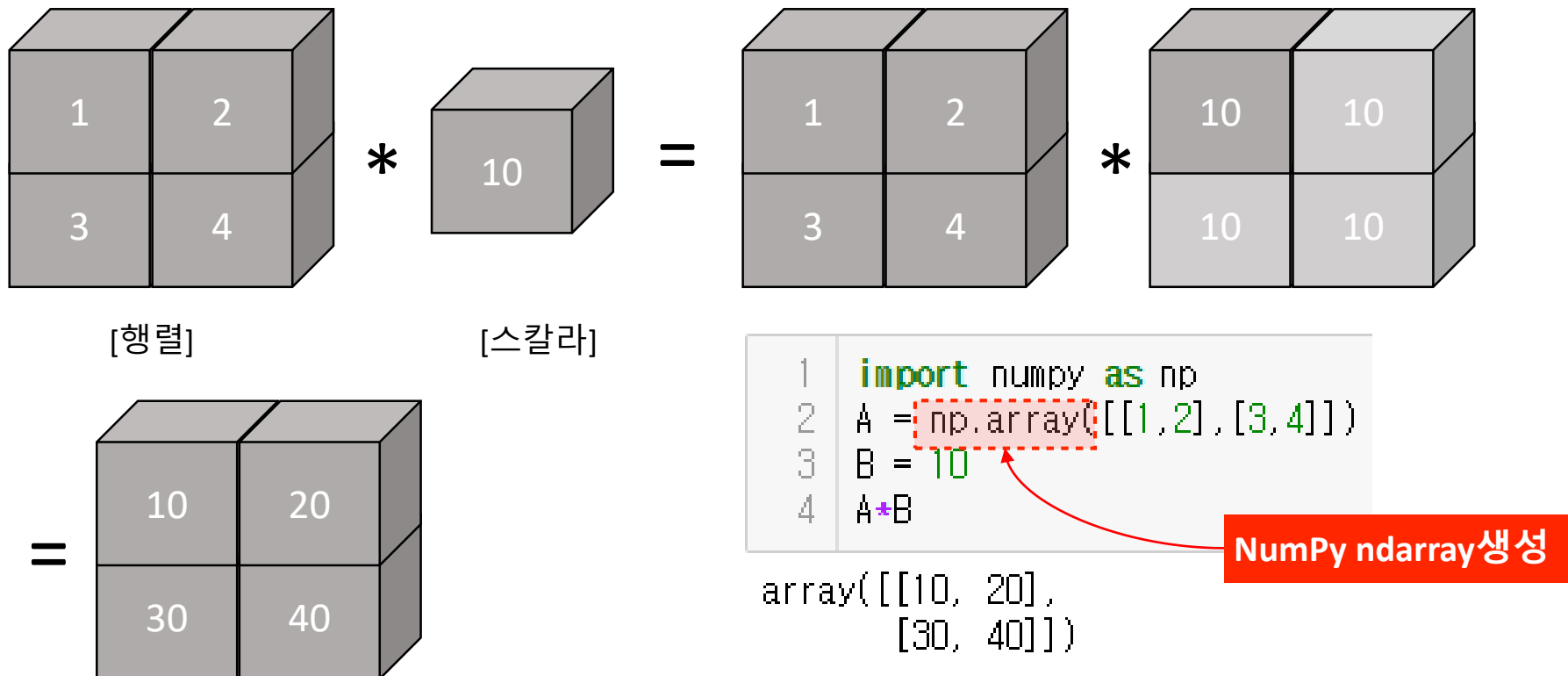


NumPy ndarray: 다차원 배열 객체

- ndarray는 같은 종류의 데이터를 담을 수 있는 다차원 배열 객체
- 브로드캐스트(broadcast) 지원

브로드캐스트(Broadcast)

- 넘파이에서는 형상이 다른 배열끼리도 계산할 수 있음
- 브로드캐스트란 아래 그림과 같이 스칼라값이 2x2 행렬로 확대된 후 연산이 이루어지는 것을 의미함.



브로드캐스트(Broadcast)

- 배열은 for반복문을 작성하지 않고 데이터를 일괄처리 할 수 있기 때문에 매우 중요함!!
- 이를 벡터화라고 하는데, 같은 크기의 배열 간 산술연산은 배열의 각 요소 단위로 적용됨

```
1 arr = np.array([[1,2,3],[4,5,6]])
2 arr
```

```
array([[1, 2, 3],
       [4, 5, 6]])
```

```
1 arr * arr
```

```
array([[ 1,  4,  9],
       [16, 25, 36]])
```

```
1 arr - arr
```

```
array([[0, 0, 0],
       [0, 0, 0]])
```

NumPy ndarray

- np.array 는 새로운 배열을 생성하기 위한 여러 함수를 가지고 있음
 - 예1) zeros와 one는 주어진 길이나 모양에 각각 0과 1이 들어있는 배열 생성
 - 예2) empty함수는 초기화되지 않은 배열을 생성
- 위와 같은 메서드를 사용해서 다차원 배열을 생성하려면 원하는 형태의 튜플을 넘기면 됨

| 함수 | 설명 |
|-------------------|--|
| array | 입력 데이터(리스트, 튜플, 배열 등)를 ndarray로 변환하며 dtype이 명시되지 않을 경우 자료형을 추론하여 저장. 기본적으로 데이터는 복사됨 |
| asarray | 입력 데이터를 ndarray로 변환하지만 입력 데이터가 이미 ndarray일 경우, 복사x |
| arange | 내장 range함수와 유사하지만 리스트 대신 ndarray를 반환 |
| ones, ones_like | ones는 주어진 dtype과 주어진 모양을 가지는 배열을 생성하고 내용을 모두 1로 초기화. ones_like 는 주어진 배열과 동일한 모양과 dtype을 가지는 배열을 새로 생성하며 내용을 모두 1로 초기화함 |
| zeros, zeros_like | ones, ones_like와 같지만 내용을 0으로 채움 |
| empty, empty_like | 메모리를 할당하여 새로운 배열을 생성하고 내용을 모두 0으로 채움 |
| eye, identity | NxN 크기의 단위 행렬을 생성한다. (좌상단에서 우하단을 잇는 대각선은 1로 채워지고 나머지는 0으로 채워진다.) |

배열 전치와 축 바꾸기

- 배열 전치
 - 데이터를 복사하지 않고 데이터 모양이 바뀐 뷰를 반환하는 기능
 - ndarray는 transpose 메서드와 T라는 이름의 특수한 속성을 가지고 있음

```

1 arr = np.arange(15).reshape((3,5))
2 print("arr")
3 print(arr)
4
5 transposed_arr = arr.T
6 print("transposed arr")
7 print(transposed_arr, end='\n\t')
```

arr

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]]
```

transposed arr

```
[[ 0  5 10]
 [ 1  6 11]
 [ 2  7 12]
 [ 3  8 13]
 [ 4  9 14]]
```

배열 전치와 축 바꾸기

- 배열 전치는 행렬 계산을 할 때 주로 사용
- NumPy ndarray의 행렬 내적은 `np.dot()`을 이용해서 구할 수 있음

```
1 print("np.dot(transposed_arr, arr)")
2 print(np.dot(transposed_arr, arr))
```

```
np.dot(arr, transposed_arr)
[[125 140 155 170 185]
 [140 158 176 194 212]
 [155 176 197 218 239]
 [170 194 218 242 266]
 [185 212 239 266 293]]
```


- numpy의 random 모듈은 파이썬 내장 random 함수를 보강하여 다양한 종류의 확률분포로부터 효과적으로 표본 값을 생성하는 데 주로 사용
- 파이썬 내장 random 함수와 비교했을때 약 20배 이상 빠른걸 볼 수 있음

```
1 from random import normalvariate
2 N = 1000000
3
4 %timeit samples = [normalvariate(0,1) for _ in range(N)]
5
6 %timeit np.random.normal(size=N)
```

574 ms \pm 2.78 ms per loop (mean \pm std. dev. of 7 runs, 1 loop each)

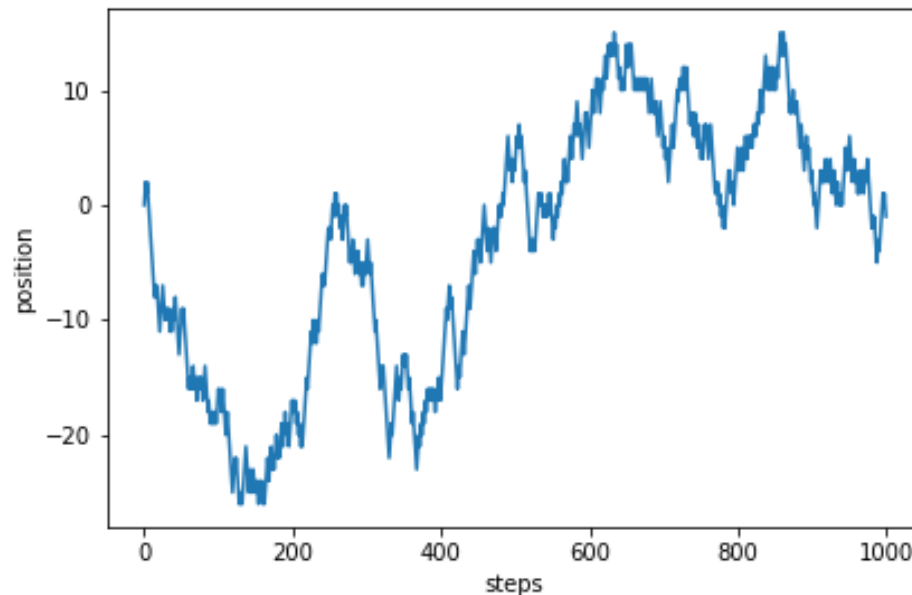
23.5 ms \pm 59.6 μ s per loop (mean \pm std. dev. of 7 runs, 10 loops each)

- 일부 numpy random 함수

| 함수 | 설명 |
|-------------|------------------------------------|
| seed | 난수 발생기의 시드를 지정한다 |
| permutation | 순서를 임의로 바꾸거나 임의의 순열을 반환한다 |
| shuffle | 리스트나 배열의 순서를 뒤섞는다 |
| rand | 균등분포에서 표본을 추출한다 |
| randint | 주어진 최소/최대 범위 안에서 임의의 난수를 추출한다 |
| randn | 표준편차가 1이고 평균 값이 0인 정규분포에서 표본을 추출한다 |
| binomial | 이항분포에서 표본을 추출한다 |
| normal | 정규분포에서 표본을 추출한다 |
| beta | 베타분포에서 표본을 추출한다 |
| chisquare | 카이제곱분포에서 표본을 추출한다 |
| gamma | 감마분포에서 표본을 추출한다 |
| uniform | 균등(0,1)분포에서 표본을 추출한다 |

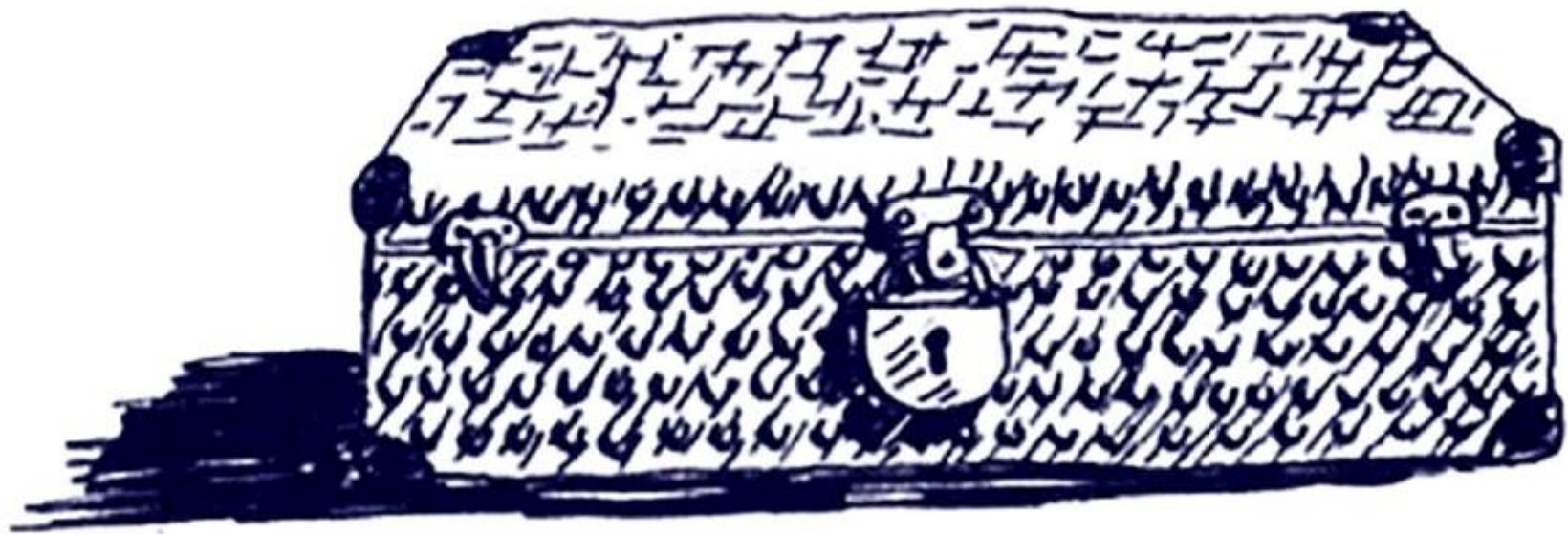
문제11 – 계단 오르내리기

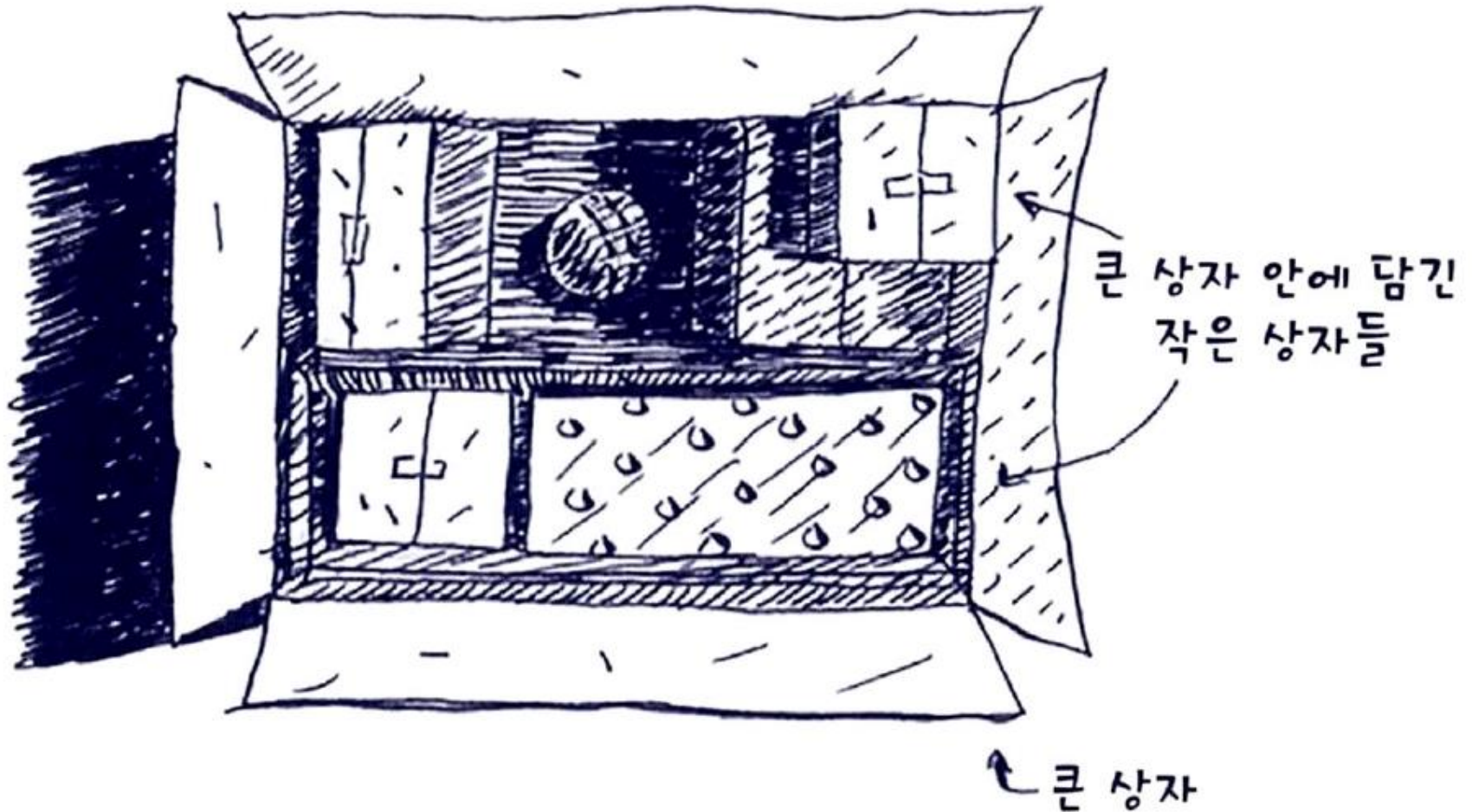
- 배열연산의 활용을 보여주는 어플리케이션
- 처음 계단(0의 위치)에서 random 값이 0일경우 -1칸, 1일경우 1칸 올라간다고 가정
- 힌트: random함수를 사용, matplotlib 를 사용해서 결과확인
 - 출력예시



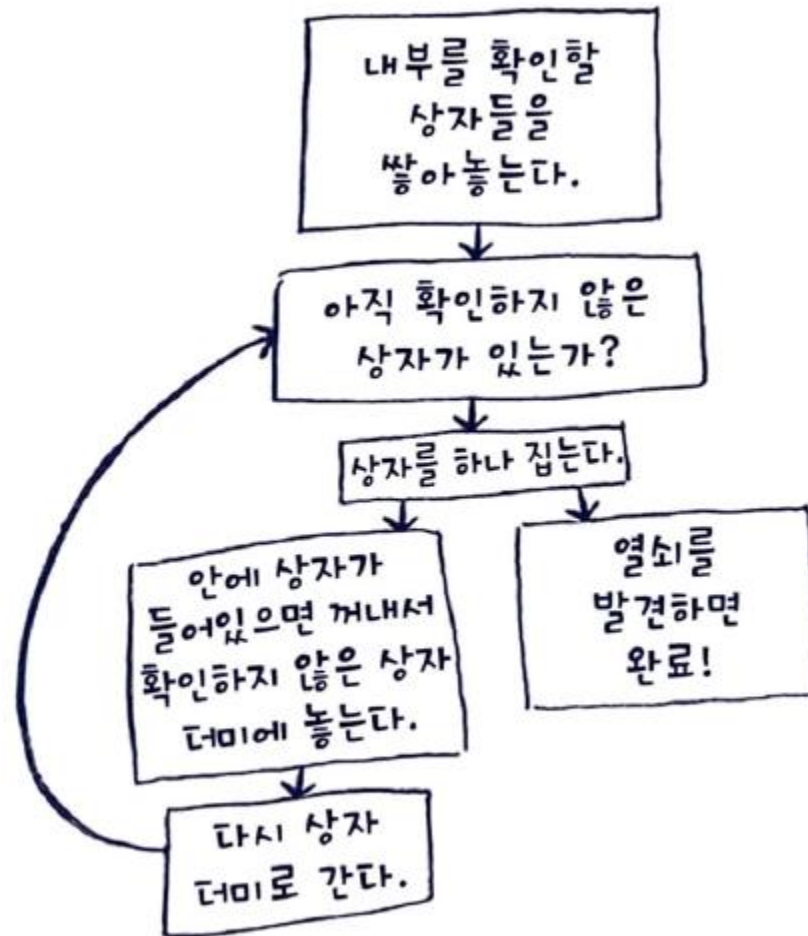
문제12

- NumPy random 함수를 사용하여 배열을 생성하고,
- 함수의 인자로 NumPy array를 받아서
- 함수 내부에서 평균을 구한후 결과값을 return 해보기

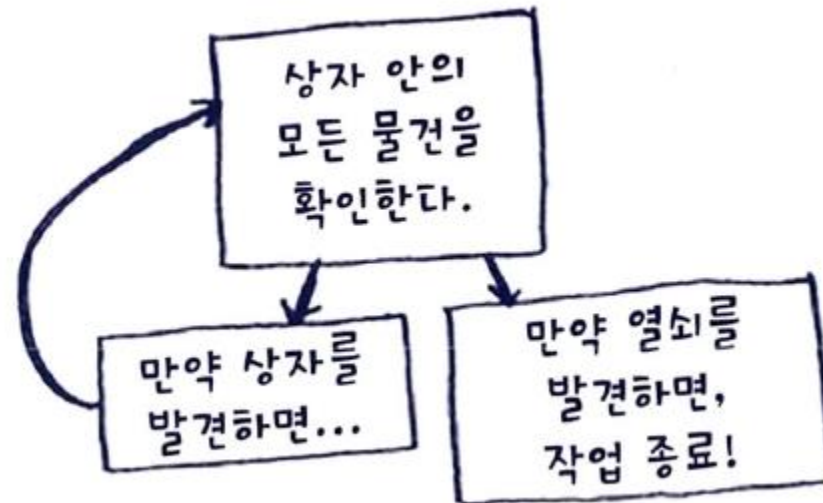




재귀 - 첫번째



재귀 - 두번째



- 두번째 방법은 재귀(recursion)를 사용.
- 재귀란 함수가 자기 자신을 호출하는 것을 말함.
- 재귀를 쓴다고 성능이 더 나아지지는 않습니다. 사실 반복문이 더 성능이 좋은 경우가 많음.
- BUT!! "프로그램에 반복문을 사용하면 프로그램의 성능을 향상시킬 수 있지만 재귀를 사용하면 프로그래머의 능력을 향상시킬 수 있습니다. 상황 따라 적절한 방법을 골라 사용하세요" by 레이 캐드웰(Leigh Caldwell)

- 재귀 함수를 만들 때는 언제 재귀를 멈출지 알려줘야 합니다.
- 그래서 모든 재귀 함수는 **기본 단계(base case)**와 **재귀 단계(recursive case)**라는 두 부분으로 나누어져 있습니다.
 - **기본 단계:** 함수가 자기 자신을 다시 호출하지 않는 경우, 즉 무한 반복으로 빠져들지 않게 하는 부분입니다.
 - **재귀 단계:** 함수가 자기 자신을 호출하는 부분입니다.

- 호출스택
 - 호출스택은 프로그램에서 중요한 개념
- 예시
 - 바비큐를 만든다고 상상해보세요.
 - 바비큐를 준비하기 위한 일들을 접착식 메모지에 적어 놓습니다.
 - 배열과 리스트 그리고 할 일 목록을 공부했을 때를 떠올려 보세요.
 - 어떤 항목이든 마음대로 삭제할 수 있었습니다.
 - 접착식 메모지는 더 단순합니다.
 - 새 항목을 추가할 때는 기존의 목록 위에 덧붙입니다.
 - 항목을 읽을 때는 가장 위에 있는 항목만 읽고 떼어낼 수 있습니다. 그러니까 푸시 Push(삽입)와 팝Pop(떼어 내고 읽기) 두가지 일만 할 수 있는것이죠.

- 이러한 자료구조를 스택(Stack)이라고 함.



스택에서 할 일을
“팝(POP)”한다.



할 일은 “음식 준비”이다.
빵과 햄버거를 준비하고
케이크를 구워야 한다.



이 일들을 메모지에 적어
다시 스택에
“푸시(PUSH)”한다.

- 호출스택

```
def greet(name):  
    print "hello, " + name + "!"  
    greet2(name)  
    print "getting ready to say bye..."  
    bye()  
  
    def greet2(name):  
        print "how are you, " + name + "?"  
  
    def bye():  
        print "ok bye!"
```

- 첫번째 greet("maggie")라고 명령을 했다고 가정
- 컴퓨터는 이 함수 호출을 위해 메모리 상자를 하나 할당



- 이제 메모리를 사용. name이라는 변수의 값이 "maggie"가 되었으므로 이 값을 메모리에 저장



- 여러분이 함수를 호출할 때마다 컴퓨터는 호출에 사용된 변수의 값을 모두 이런식으로 저장
- 그 다음에는 “hello, maggie!” 라고 프린트한 후, greet2(“maggie”) 명령으로 다른 함수를 호출. 이번에도 컴퓨터는 함수 호출에 필요한 또 다른 메모리 상자를 할당



- 컴퓨터는 이런 메모리 상자를 스택으로 사용
- 두번째 상자는 첫번째 상자 위에 올려진 상태
- 이제 how are you, maggie? 라고 프린트하고, 함수 호출 상태에서 반환 (return)하여 돌아와야 함.
- 함수가 반환되면 가장 위에 있는 상자는 팝(Pop)연산으로 인해 없어짐



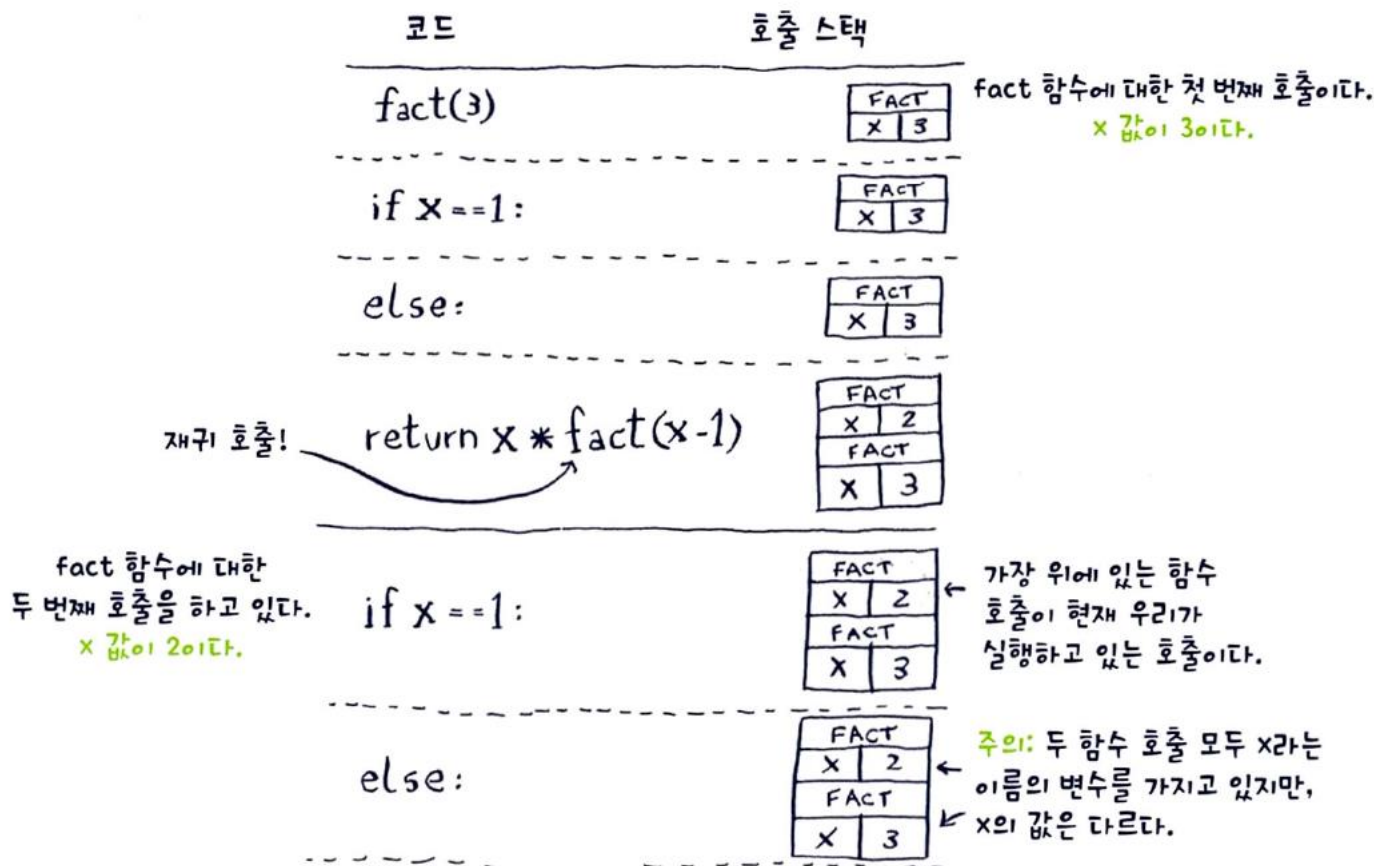
- 이제 스택에서 가장 위에 있는 함수는 greet 함수가 되었음.
- 즉 , greet 함수로 다시 돌아왔다는 뜻.
- 사실 greet2 함수를 호출하였을 때 greet 함수는 아직 완전히 실행되지 않은 상태였습니다. 여러분이 어떤 함수를 호출하여 완전히 실행을 완료하기 전이라도 그 함수를 잠시 멈추고 다른 함수를 호출할 수 있습니다. 중지된 함수의 변수 값들은 모두 메모리에 저장되어 있습니다.

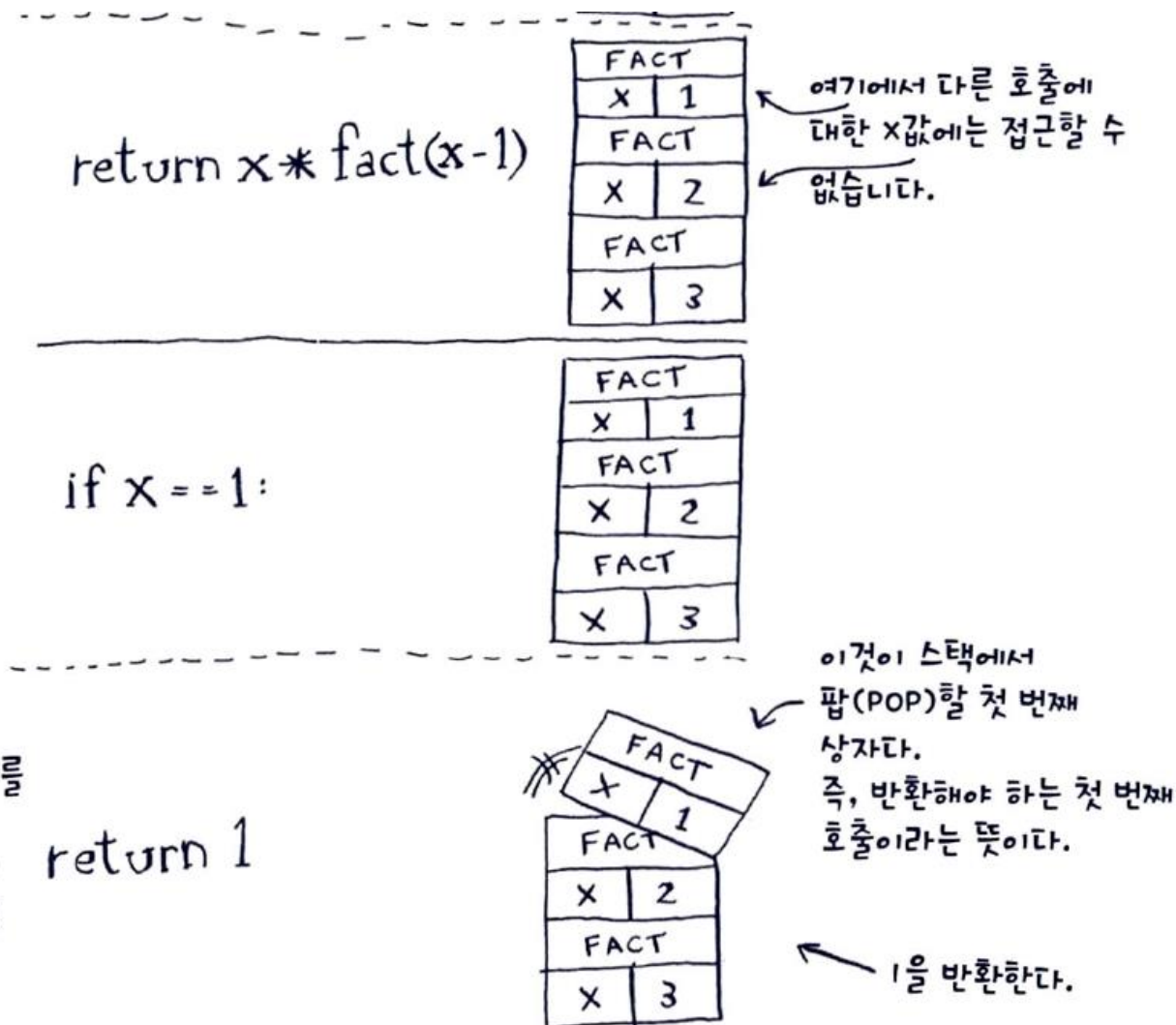
- greet2 함수의 실행을 완료하고 나면 greet 함수로 돌아가 멈추었던 위치에서 다시 실행할 수 있습니다.
- 이번에는 getting ready to say bye ... 라고 프린트한 다음에 bye 함수를 실행합니다.
- 새로운 함수를 위한 메모리 상자가 스택 위에 더해집니다. 이제 ok bye! 라고 프린트한 다음 다시 함수 호출로 부터 돌아옵니다.
- 더 이상 실행할 것이 없으므로 greet 함수에서도 반환하여 돌아옵니다. 이런 방식으로 여러 개의 함수를 호출하면서 함수에 사용되는 변수를 저장하는 스택을 **호출 스택** 이라고 합니다.

- 예시 – 팩토리얼 함수
 - factorial(5)는 5! 라는 뜻
 - $5! = 5 \times 4 \times 3 \times 2 \times 1$ 로 정의되는 값
- 스택에서 가장 위에 있는 상자가 호출하고 있는 fact함수를 의미한다는 것!!

```
def fact(x):  
    if x == 1:  
        return 1  
    else:  
        return x * fact(x-1)
```

- 예시
 - 3!, factorial(3)의 호출스택을 그려보기





이제 우리는 fact 함수를 세 번 호출했다. 하지만 아직 함수 호출 하나도 끝내지 못했다!

이것이 방금
반환한 함수
호출이다.

`return x * fact(x-1)`

x 값이 2이다.

| FACT | |
|------|---|
| X | 2 |
| FACT | |
| X | 3 |

← 2를 반환한다.

`return x * fact(x-1)`

x 값이 3이다.

이 호출은 2를
반환한다.

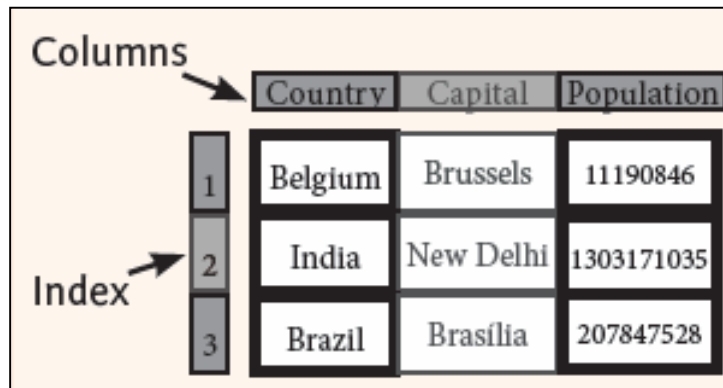
| FACT | |
|------|---|
| X | 3 |

← 6을 반환한다.

문제13 – 피보나치 수열, 재귀응용

- 입력한 숫자까지의 피보나치 수열값을 출력해보세요
- 알고리즘 3단계
 - 입력: 숫자입력
 - 처리: 입력한 숫자까지의 피보나치 수열값을 출력
 - 출력: 피보나치 수열값
 - 입력: 5
 - 출력: 1 1 2 3 5

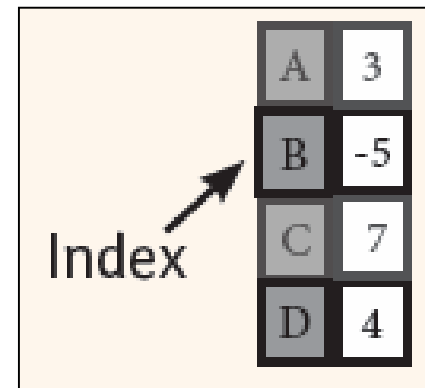
- Numpy를 기반으로 하는 효과적인 데이터분석을 지원하는 모듈
- 데이터 구조는 DataFrame, Series을 효율적으로 구현함
 - DataFrame : 2차원의 라벨화된 데이터 구조로서 데이터의 종류가 다양함
 - Series : 인덱스 정보를 가지고 있는 1차원 배열(series = index +value)



The diagram shows a DataFrame with three columns labeled 'Country', 'Capital', and 'Population'. The rows are indexed 1, 2, and 3. An arrow labeled 'Columns' points to the header row, and an arrow labeled 'Index' points to the index column.

| | Country | Capital | Population |
|---|---------|-----------|------------|
| 1 | Belgium | Brussels | 11190846 |
| 2 | India | New Delhi | 1303171035 |
| 3 | Brazil | Brasília | 207847528 |

DataFrame



The diagram shows a Series with four elements labeled A, B, C, and D. The values are 3, -5, 7, and 4. An arrow labeled 'Index' points to the index column.

| | A | B | C | D |
|-------|---|----|---|---|
| Index | 3 | -5 | 7 | 4 |

Series

- 사용방법
 - `import pandas as pd`
- pandas에 대해서 알아보려면 Series와 DataFrame 이 두가지 자료구조에 익숙해질 필요가 있음
 - 따라서 Series와 DataFrame은 네임스페이스로 import 하여 편리하게 사용
 - `from pandas import Series, DataFrame`
- 예시

```
1 import pandas as pd
2 from pandas import Series, DataFrame
```

pandas - DataFrame

- DataFrame 객체는 다양한 방법으로 생성할 수 있지만 가장 흔하게 사용되는 방법은 같은 길이의 리스트에 담긴 사전을 이용하거나 NumPy 배열을 이용하는 방법
- column 은 자동으로 정렬이 되서 대입됨

```
1 data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada'] ,
2         'year': [2000, 2001, 2002, 2001, 2002],
3         'pop': [1.5, 1.7, 3.6, 2.4, 2.9]}
4 frame = DataFrame(data)
5 frame
```

| | pop | state | year |
|---|-----|--------|------|
| 0 | 1.5 | Ohio | 2000 |
| 1 | 1.7 | Ohio | 2001 |
| 2 | 3.6 | Ohio | 2002 |
| 3 | 2.4 | Nevada | 2001 |
| 4 | 2.9 | Nevada | 2002 |

pandas - DataFrame

- 원하는 순서대로 columns를 지정하면 원하는 순서를 가진 DataFrame 객체가 생성

```
1 data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada'],
2         'year': [2000, 2001, 2002, 2001, 2002],
3         'pop': [1.5, 1.7, 3.6, 2.4, 2.9]}
4 frame = DataFrame(data, columns=['year', 'state', 'pop'])
5 frame
```

| | year | state | pop |
|---|------|--------|-----|
| 0 | 2000 | Ohio | 1.5 |
| 1 | 2001 | Ohio | 1.7 |
| 2 | 2002 | Ohio | 3.6 |
| 3 | 2001 | Nevada | 2.4 |
| 4 | 2002 | Nevada | 2.9 |

pandas - DataFrame

- DataFrame을 만드는 dictionary에 없는 column 값을 DataFrame의 columns 으로 만들면 NaN(Not a Number)값이 저장

```

1 data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada'] ,
2         'year': [2000, 2001, 2002, 2001, 2002],
3         'pop': [1.5, 1.7, 3.6, 2.4, 2.9]}
4 frame = DataFrame(data, columns=['year', 'state', 'pop', 'debt'])
5 frame

```

| | year | state | pop | debt |
|---|------|--------|-----|------|
| 0 | 2000 | Ohio | 1.5 | NaN |
| 1 | 2001 | Ohio | 1.7 | NaN |
| 2 | 2002 | Ohio | 3.6 | NaN |
| 3 | 2001 | Nevada | 2.4 | NaN |
| 4 | 2002 | Nevada | 2.9 | NaN |

pandas - DataFrame

- DataFrame 에 있는 값을 접근하는 방법
 - column 값 접근: 사전형식, 속성형식
 - row 값 접근: ix 메소드

| | year | state | pop | debt |
|---|------|--------|-----|------|
| 0 | 2000 | Ohio | 1.5 | NaN |
| 1 | 2001 | Ohio | 1.7 | NaN |
| 2 | 2002 | Ohio | 3.6 | NaN |
| 3 | 2001 | Nevada | 2.4 | NaN |
| 4 | 2002 | Nevada | 2.9 | NaN |

DataFrame

```
1 frame['state']
0    Ohio
1    Ohio
2    Ohio
3  Nevada
4  Nevada
Name: state, dtype: object
```

```
1 frame.state
0    Ohio
1    Ohio
2    Ohio
3  Nevada
4  Nevada
Name: state, dtype: object
```

column 값들 확인

```
1 frame.ix[3]
year      2001
state     Nevada
pop        2.4
debt       NaN
Name: 3, dtype: object
```

3번째 인덱스 위치의 row값들 확인

pandas - DataFrame

- DataFrame의 column은 대입이 가능함.
 - 스칼라값이나 배열의 값을 대입할 수 있음

```
1 frame['debt']=16.5
2 frame
```

| | year | state | pop | debt |
|---|------|--------|-----|------|
| 0 | 2000 | Ohio | 1.5 | 16.5 |
| 1 | 2001 | Ohio | 1.7 | 16.5 |
| 2 | 2002 | Ohio | 3.6 | 16.5 |
| 3 | 2001 | Nevada | 2.4 | 16.5 |
| 4 | 2002 | Nevada | 2.9 | 16.5 |

스칼라값 대입

```
1 frame['debt'] = np.arange(5.)
2 frame
```

| | year | state | pop | debt |
|---|------|--------|-----|------|
| 0 | 2000 | Ohio | 1.5 | 0.0 |
| 1 | 2001 | Ohio | 1.7 | 1.0 |
| 2 | 2002 | Ohio | 3.6 | 2.0 |
| 3 | 2001 | Nevada | 2.4 | 3.0 |
| 4 | 2002 | Nevada | 2.9 | 4.0 |

배열 대입

- DataFrame의 column은 삭제가 가능함.
- del 예약어를 통해서 삭제

```
1 del frame['debt']  
2 frame
```

| | year | state | pop |
|---|------|--------|-----|
| 0 | 2000 | Ohio | 1.5 |
| 1 | 2001 | Ohio | 1.7 |
| 2 | 2002 | Ohio | 3.6 |
| 3 | 2001 | Nevada | 2.4 |
| 4 | 2002 | Nevada | 2.9 |

- DataFrame 합치기

```
1 columns= ['name','age','gender','job']
2 user1 = DataFrame([[ 'alice',19,'F','student'],
3                     [ 'john',26,'M','student']], columns=columns)
4 user2 = DataFrame([[ 'eric', 22, 'M','student'],
5                     [ 'paul', 58, 'F','manager']], columns=columns)
6 user3 = DataFrame(dict(name=[ 'peter','julie'],
7                         age=[33,44],
8                         gender=[ 'M','F'],
9                         job=[ 'engineer','scientist'])))
```

1 user1

| | name | age | gender | job |
|---|-------|-----|--------|---------|
| 0 | alice | 19 | F | student |
| 1 | john | 26 | M | student |

1 user2

| | name | age | gender | job |
|---|------|-----|--------|---------|
| 0 | eric | 22 | M | student |
| 1 | paul | 58 | F | manager |

1 user3

| | age | gender | job | name |
|---|-----|--------|-----------|-------|
| 0 | 33 | M | engineer | peter |
| 1 | 44 | F | scientist | julie |

pandas – Join DataFrame

- user1 에 user2를 덧붙이는 append()

| | |
|---|-------|
| 1 | user1 |
|---|-------|

| | name | age | gender | job |
|---|-------|-----|--------|---------|
| 0 | alice | 19 | F | student |
| 1 | john | 26 | M | student |

| | |
|---|-------|
| 1 | user2 |
|---|-------|

| | name | age | gender | job |
|---|------|-----|--------|---------|
| 0 | eric | 22 | M | student |
| 1 | paul | 58 | F | manager |

| | name | age | gender | job |
|---|-------|-----|--------|---------|
| 0 | alice | 19 | F | student |
| 1 | john | 26 | M | student |
| 0 | eric | 22 | M | student |
| 1 | paul | 58 | F | manager |

- DataFrame을 row 방향으로 연결하는 concatenate()

| | |
|---|-------|
| 1 | user3 |
|---|-------|

| | age | gender | job | name |
|---|-----|--------|-----------|-------|
| 0 | 33 | M | engineer | peter |
| 1 | 44 | F | scientist | julie |

| | age | gender | job | name |
|---|-----|--------|-----------|-------|
| 0 | 19 | F | student | alice |
| 1 | 26 | M | student | john |
| 0 | 22 | M | student | eric |
| 1 | 58 | F | manager | paul |
| 0 | 33 | M | engineer | peter |
| 1 | 44 | F | scientist | julie |

pandas – Join DataFrame

- DataFrame을 row 방향으로 연결하는 concatenate()
 - user1 , user2, user3을 pandas의 concat 함수를 사용하여서 users 를 만듦

```
1 users = pd.concat([user1, user2, user3])
2 users
```

| | age | gender | job | name |
|---|-----|--------|-----------|-------|
| 0 | 19 | F | student | alice |
| 1 | 26 | M | student | john |
| 0 | 22 | M | student | eric |
| 1 | 58 | F | manager | paul |
| 0 | 33 | M | engineer | peter |
| 1 | 44 | F | scientist | julie |

pandas - merge

- DataFrame의 key값을 사용해서 데이터 집합의 Row를 합침 = Merge

```
1 columns= ['name', 'age', 'gender', 'job']
2 user1 = DataFrame([[ 'alice', 19, 'F', 'student'],
3                     [ 'john', 26, 'M', 'student']], columns=columns)
4 user2 = DataFrame([[ 'eric', 22, 'M', 'student'],
5                     [ 'paul', 58, 'F', 'manager']], columns=columns)
6 user3 = DataFrame(dict(name=[ 'peter', 'julie'],
7                         age=[ 33, 44],
8                         gender=[ 'M', 'F'],
9                         job=[ 'engineer', 'scientist'])))
10
11 users = pd.concat([user1, user2, user3])
12
13 user4 = DataFrame(dict(name=[ 'alice', 'john', 'eric', 'julie'],
14                         height=[ 165, 180, 175, 171]))
15 merge_integer = pd.merge(users, user4, on="name")
16 merge_integer
```

| | age | gender | job | name | height |
|---|-----|--------|-----------|-------|--------|
| 0 | 19 | F | student | alice | 165 |
| 1 | 26 | M | student | john | 180 |
| 2 | 22 | M | student | eric | 175 |
| 3 | 44 | F | scientist | julie | 171 |

on 키워드 위에 key 값을 적어주면, 그 값을 기준으로 row를 합침

pandas - groupby

- groupby
 - “job”을 기준으로 groupby한 내용을 grp와 data 변수에 저장

grp변수에는 groupby의 key 값이 들어가고, data에는 해당 위치의 row값이 할당됨

| | age | gender | job | name |
|---|-----|--------|-----------|-------|
| 0 | 19 | F | student | alice |
| 1 | 26 | M | student | john |
| 0 | 22 | M | student | eric |
| 1 | 58 | F | manager | paul |
| 0 | 33 | M | engineer | peter |
| 1 | 44 | F | scientist | julie |

```
1 for grp, data in users.groupby("job"):
2     print("{} {}".format(grp, data))
```

```
engineer, age gender job name
0 33 M engineer peter
manager, age gender job name
1 58 F manager paul
scientist, age gender job name
1 44 F scientist julie
student, age gender job name
0 19 F student alice
1 26 M student john
0 22 M student eric
```

pandas - summarizing

- columns 을 기준으로 중심 경향, 분산, 데이터 집합의 분포 형태를 요약

```
1 users.describe(include='all')
```

| | age | gender | job | name |
|--------|-----------|--------|---------|------|
| count | 6.000000 | 6 | 6 | 6 |
| unique | NaN | 2 | 4 | 6 |
| top | NaN | M | student | eric |
| freq | NaN | 3 | 3 | 1 |
| mean | 33.666667 | NaN | NaN | NaN |
| std | 14.895189 | NaN | NaN | NaN |
| min | 19.000000 | NaN | NaN | NaN |
| 25% | 23.000000 | NaN | NaN | NaN |
| 50% | 29.500000 | NaN | NaN | NaN |
| 75% | 41.250000 | NaN | NaN | NaN |
| max | 58.000000 | NaN | NaN | NaN |

pandas – filtering

- DataFrame[@]: @ 안에 불리언 식을 표현할 수 있음.
 - users의 DataFrame에서 age columns 값들중 20 미만인 경우에만 출력
 - users[users.age<20] 에 값들중.job의 column 값을 출력

```
1 users[users.age < 20]
```

| | age | gender | job | name |
|---|-----|--------|---------|-------|
| 0 | 19 | F | student | alice |

```
1 young_bool = users.age < 20
2 users[young_bool]
```

| | age | gender | job | name |
|---|-----|--------|---------|-------|
| 0 | 19 | F | student | alice |

```
1 users[users.age < 20].job
```

```
0    student
Name: job, dtype: object
```

pandas – advanced filtering

- DataFrame[@]: @안에는 2개 이상의 불리언식을 작성할 수 있음

```
1 users[(users.age > 20) & (users.gender=='M')]
```

| | age | gender | job | name |
|---|-----|--------|----------|-------|
| 1 | 26 | M | student | john |
| 0 | 22 | M | student | eric |
| 0 | 33 | M | engineer | peter |

pandas – sorting

- `sort_values()` 함수를 사용하여 pandas 데이터를 정렬
- 이때 `by` 매개변수의 값을 기준으로 pandas 데이터를 정렬하게 됨

```
1 users
```

| | age | gender | job | name |
|---|-----|--------|-----------|-------|
| 0 | 19 | F | student | alice |
| 1 | 26 | M | student | john |
| 0 | 22 | M | student | eric |
| 1 | 58 | F | manager | paul |
| 0 | 33 | M | engineer | peter |
| 1 | 44 | F | scientist | julie |

정렬 전

```
1 # Sort
2 users.sort_values(by='age')
```

| | age | gender | job | name |
|---|-----|--------|-----------|-------|
| 0 | 19 | F | student | alice |
| 0 | 22 | M | student | eric |
| 1 | 26 | M | student | john |
| 0 | 33 | M | engineer | peter |
| 1 | 44 | F | scientist | julie |
| 1 | 58 | F | manager | paul |

정렬 후

pandas – 누락된 데이터 처리하기

- pandas에서 누락된 데이터는 실수든 아니든 모두 NaN(Not a Number)으로 취급
- NaN 처리 메서드

| 인자 | 설명 |
|---------|---|
| dropna | 누락된 데이터가 있는 축(로우, 칼럼)을 제외. 어느정도의 누락 데이터까지 용인할 것인지를 지정할 수 있음 |
| fillna | 누락된 데이터를 대신할 값을 채움 |
| isnull | 누락되거나 NA인 값을 알려주는 불리언 값이 저장된, 같은 형의 객체를 반환 |
| notnull | isnull과는 반대되는 메서드 |

pandas – 누락된 데이터 처리하기

- dropna 예시

```
1 from pandas import DataFrame
2 from numpy import nan as NA
3 data = DataFrame([[1., 6.5, 3.],
4                   [1., NA, NA],
5                   [NA, NA, NA],
6                   [NA, 6.4, 3.]])
7 data
```

| | 0 | 1 | 2 |
|---|-----|-----|-----|
| 0 | 1.0 | 6.5 | 3.0 |
| 1 | 1.0 | NaN | NaN |
| 2 | NaN | NaN | NaN |
| 3 | NaN | 6.4 | 3.0 |

```
1 # dropna
2 cleaned = data.dropna()
3 cleaned
```

| | 0 | 1 | 2 |
|---|-----|-----|-----|
| 0 | 1.0 | 6.5 | 3.0 |

```
1 cleaned_how_all = data.dropna(how='all')
2 cleaned_how_all
```

| | 0 | 1 | 2 |
|---|-----|-----|-----|
| 0 | 1.0 | 6.5 | 3.0 |
| 1 | 1.0 | NaN | NaN |
| 3 | NaN | 6.4 | 3.0 |

pandas – 누락된 데이터 처리하기

- dropna 예시

```
1 from pandas import DataFrame
2 from numpy import nan as NA
3 data = DataFrame([[1., 6.5, 3.],
4                   [1., NA, NA],
5                   [NA, NA, NA],
6                   [NA, 6.4, 3.]])
7 data
```

| | 0 | 1 | 2 |
|---|-----|-----|-----|
| 0 | 1.0 | 6.5 | 3.0 |
| 1 | 1.0 | NaN | NaN |
| 2 | NaN | NaN | NaN |
| 3 | NaN | 6.4 | 3.0 |

```
1 # dropna
2 cleaned = data.dropna()
3 cleaned
```

| | 0 | 1 | 2 |
|---|-----|-----|-----|
| 0 | 1.0 | 6.5 | 3.0 |

```
1 cleaned_how_all = data.dropna(how='all')
2 cleaned_how_all
```

| | 0 | 1 | 2 |
|---|-----|-----|-----|
| 0 | 1.0 | 6.5 | 3.0 |
| 1 | 1.0 | NaN | NaN |
| 3 | NaN | 6.4 | 3.0 |

how 매개변수를 사용하여
어느 정도의 누락 데이터까
지 용인할 것이지를 지정할
수 있음

pandas – 누락된 값 채우기

- fillna 예시
 - NaN값을 특정값으로 전부 채우는 방법
 - 열마다 서로다른 값을 채우는 방법

```

1 from pandas import DataFrame
2 from numpy import nan as NA
3 data = DataFrame([[1., 6.5, 3.],
4                   [1., NA, NA],
5                   [NA, NA, NA],
6                   [NA, 6.4, 3.]])
7 data

```

| | 0 | 1 | 2 |
|---|-----|-----|-----|
| 0 | 1.0 | 6.5 | 3.0 |
| 1 | 1.0 | NaN | NaN |
| 2 | NaN | NaN | NaN |
| 3 | NaN | 6.4 | 3.0 |

```

1 filled_data1 = data.fillna(0)
2 filled_data1

```

| | 0 | 1 | 2 |
|---|-----|-----|-----|
| 0 | 1.0 | 6.5 | 3.0 |
| 1 | 1.0 | 0.0 | 0.0 |
| 2 | 0.0 | 0.0 | 0.0 |
| 3 | 0.0 | 6.4 | 3.0 |

```

1 filled_data = data.fillna({0:0.5, 1:1, 2:1.5})
2 filled_data

```

| | 0 | 1 | 2 |
|---|-----|-----|-----|
| 0 | 1.0 | 6.5 | 3.0 |
| 1 | 1.0 | 1.0 | 1.5 |
| 2 | 0.5 | 1.0 | 1.5 |
| 3 | 0.5 | 6.4 | 3.0 |

pandas – 중복 제거하기

- 여러가지 이유로 DataFrame에서 중복된 Row값을 발견할 수 있음
 - `uplicated()`함수는 DataFrame에서 key값이 중복된 위치를 Boolean 자료형으로 반환해줌
 - `drop_duplicates()`함수는 DataFrame에 key값이 중복됐다면 처음 나온값을 기준으로 이 후에 나오는 값은 제거

```
1 from pandas import DataFrame
2 data = DataFrame({'k1':['one'] * 3 + ['two']*4,
3                   'k2':[1,1,2,3,3,4,4]})
4 data
```

| | k1 | k2 |
|---|-----|----|
| 0 | one | 1 |
| 1 | one | 1 |
| 2 | one | 2 |
| 3 | two | 3 |
| 4 | two | 3 |
| 5 | two | 4 |
| 6 | two | 4 |

```
1 data.duplicated()
0    False
1     True
2    False
3    False
4     True
5    False
6     True
dtype: bool
```

```
1 drop_data = data.drop_duplicates()
2 drop_data
```

| | k1 | k2 |
|---|-----|----|
| 0 | one | 1 |
| 2 | one | 2 |
| 3 | two | 3 |
| 5 | two | 4 |

pandas – 중복 제거하기

- 여러가지 이유로 DataFrame에서 중복된 Row값을 발견할 수 있음
 - `uplicated()`함수는 DataFrame에서 key값이 중복된 위치를 Boolean 자료형으로 반환해줌
 - `drop_duplicates()`함수는 DataFrame에 key값이 중복됐다면 처음 나온값을 기준으로 이 후에 나오는 값은 제거

```
1 from pandas import DataFrame
2 data = DataFrame({'k1':['one'] * 3 + ['two']*4,
3                   'k2':[1,1,2,3,3,4,4]})
4 data
```

| | k1 | k2 |
|---|-----|----|
| 0 | one | 1 |
| 1 | one | 1 |
| 2 | one | 2 |
| 3 | two | 3 |
| 4 | two | 3 |
| 5 | two | 4 |
| 6 | two | 4 |

```
1 data.duplicated()
```

drop_duplicates함수의 조건으로 keep='last'를 할 경우 중복된값중 마지막으로 발견된 값을 기준으로 이전값은 모두 제거함

```
1 drop_data = data.drop_duplicates()
2 drop_data
```

| | k1 | k2 |
|---|-----|----|
| 0 | one | 1 |
| 2 | one | 2 |
| 3 | two | 3 |
| 5 | two | 4 |

pandas – 파일 I/O, csv 파일 읽어오기

- pandas는 표 형식의 자료를 DataFrame 객체로 읽어오는 몇가지 기능을 제공
- read_csv 함수와 read_table 함수를 주로 사용
 - read_csv: 파일, URL 또는 파일과 유사한 객체로부터 구분된 데이터를 읽어올때 사용. 데이터 구분자는 쉼표(,)를 기본으로 함
 - read_table: 파일, URL 또는 파일과 유사한 객체로부터 구분된 데이터를 읽어올때 사용. 데이터 구분자는 탭(\t)를 기본으로 함

pandas – 파일 I/O, csv 파일 읽어오기

- csv 파일 읽어오는 방법 – read_csv()
 - 첫번째: 패키지 import >> import pandas as pd
 - 두번째: df = pd.read_csv('데이터셋 이름.csv')

| | | | | | |
|---|------------------------------|--|--|--|--|
| 1 | import pandas as pd | | | | |
| 2 | df = pd.read_csv('test.csv') | | | | |
| 3 | df | | | | |

| | 이름 | 나이 | 키 | 몸무게 |
|---|-----|----|-----|-----|
| 0 | 홍길동 | 20 | 180 | 80 |
| 1 | 짱구 | 25 | 175 | 75 |
| 2 | 불닭 | 29 | 170 | 70 |
| 3 | 아이폰 | 36 | 165 | 65 |

| | A | B | C | D |
|----|-----|----|-----|-----|
| 1 | 이름 | 나이 | 키 | 몸무게 |
| 2 | 홍길동 | 20 | 180 | 80 |
| 3 | 짱구 | 25 | 175 | 75 |
| 4 | 불닭 | 29 | 170 | 70 |
| 5 | 아이폰 | 36 | 165 | 65 |
| 6 | | | | |
| 7 | | | | |
| 8 | | | | |
| 9 | | | | |
| 10 | | | | |

pandas – 파일 I/O, csv 파일 읽어오기

- csv 파일 읽어오는 방법 – read_csv()
 - 파일의 column 명을 수동으로 정해줄 수 있음

```
1 df3 = pd.read_csv('test.csv', names=['a', 'b', 'c', 'd'])
2 df3
```

| | a | b | c | d |
|---|-----|----|-----|-----|
| 0 | 이름 | 나이 | 키 | 몸무게 |
| 1 | 홍길동 | 20 | 180 | 80 |
| 2 | 짱구 | 25 | 175 | 75 |
| 3 | 불닭 | 29 | 170 | 70 |
| 4 | 아이폰 | 36 | 165 | 65 |

| 클립보드 | | 글꼴 | | |
|------|-----|----|-----|-----|
| A1 | : | ✕ | ✓ | f* |
| | A | B | C | D |
| 1 | 이름 | 나이 | 키 | 몸무게 |
| 2 | 홍길동 | 20 | 180 | 80 |
| 3 | 짱구 | 25 | 175 | 75 |
| 4 | 불닭 | 29 | 170 | 70 |
| 5 | 아이폰 | 36 | 165 | 65 |
| 6 | | | | |
| 7 | | | | |

pandas – 파일 I/O, csv 파일 읽어오기

- read_csv()함수를 통해 데이터를 불러올때 주의할점
 - read_csv()함수는 기본적으로 데이터의 첫번째 row값은 column의 이름으로 생각함
 - 아래 출력결과에서도 볼 수 있듯이 첫번째 row값이 DataFrame의 column 이름으로 대입된것을 볼 수 있음

| | |
|---|--------------------------------|
| 1 | df2 = pd.read_csv('test2.csv') |
| 2 | df2 |

| | 홍길동 | 20 | 180 | 80 |
|---|-----|----|-----|----|
| 0 | 짱구 | 25 | 175 | 75 |
| 1 | 불닭 | 29 | 170 | 70 |
| 2 | 아이폰 | 36 | 165 | 65 |
| 3 | 갤럭시 | 22 | 160 | 60 |

| | A | B | C | D |
|---|-----|----|-----|----|
| 1 | 홍길동 | 20 | 180 | 80 |
| 2 | 짱구 | 25 | 175 | 75 |
| 3 | 불닭 | 29 | 170 | 70 |
| 4 | 아이폰 | 36 | 165 | 65 |
| 5 | 갤럭시 | 22 | 160 | 60 |
| 6 | | | | |
| 7 | | | | |
| 8 | | | | |

pandas – 파일 I/O, csv 파일 읽어오기

- read_csv() 함수를 통해 데이터를 불러올때 주의할점
 - 따라서 이런 경우에는 read_csv(header=None) 옵션을 통해서 첫번째 데이터가 header가 아닌 데이터 라는 것을 알려줘야함

```

1 df2 = pd.read_csv('test2.csv', header=None)
2 df2
    
```

| | 0 | 1 | 2 | 3 |
|---|-----|----|-----|----|
| 0 | 홍길동 | 20 | 180 | 80 |
| 1 | 짱구 | 25 | 175 | 75 |
| 2 | 불닭 | 29 | 170 | 70 |
| 3 | 아이폰 | 36 | 165 | 65 |
| 4 | 갤럭시 | 22 | 160 | 60 |

| | A | B | C | D |
|---|-----|----|-----|----|
| 1 | 홍길동 | 20 | 180 | 80 |
| 2 | 짱구 | 25 | 175 | 75 |
| 3 | 불닭 | 29 | 170 | 70 |
| 4 | 아이폰 | 36 | 165 | 65 |
| 5 | 갤럭시 | 22 | 160 | 60 |
| 6 | | | | |
| 7 | | | | |
| 8 | | | | |

pandas – 파일 I/O, csv 파일 읽어오기

- 읽어온 파일에 데이터 추가하기
 - .loc[@] 함수를 사용하여 DataFrame의 @번째 row위치에 값 추가

```
1 import pandas as pd
2 df = pd.read_csv('test.csv')
3 df
```

| | 이름 | 나이 | 키 | 몸무게 |
|---|-----|----|-----|-----|
| 0 | 홍길동 | 20 | 180 | 80 |
| 1 | 짱구 | 25 | 175 | 75 |
| 2 | 불닭 | 29 | 170 | 70 |
| 3 | 아이폰 | 36 | 165 | 65 |

기존

```
1 df.loc[4] = ['콘트라베이스', 29, 175, 74]
2 df
```

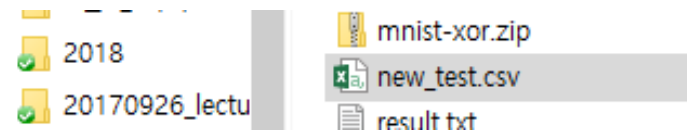
| | 이름 | 나이 | 키 | 몸무게 |
|---|--------|----|-----|-----|
| 0 | 홍길동 | 20 | 180 | 80 |
| 1 | 짱구 | 25 | 175 | 75 |
| 2 | 불닭 | 29 | 170 | 70 |
| 3 | 아이폰 | 36 | 165 | 65 |
| 4 | 콘트라베이스 | 29 | 175 | 74 |

추가

pandas – 파일 I/O, csv 파일 읽어오기

- 수정된 데이터 저장하기
 - `.to_csv('저장할 파일 이름')` 함수를 사용
 - 한글인 경우 `encoding='csv949'`를 꼭 추가해야지만 데이터가 온전하게 저장됨

```
1 df.to_csv('new_test.csv', encoding='ms949')
```



pandas – 파일 I/O, csv 파일 읽어오기

- 수정된 데이터 저장하기
 - .to_csv('저장할 파일 이름') 함수를 사용
 - 한글인 경우 encoding='csv949'를 꼭 추가해야지만 데이터가 온전하게 저장됨

encoding
적용 전

| 이름 | 나이 | 키 | 몸무게 |
|--------|----|-----|-----|
| 홍길동 | 20 | 180 | 80 |
| 짱구 | 25 | 175 | 75 |
| 불닭 | 29 | 170 | 70 |
| 아이폰 | 36 | 165 | 65 |
| 콘트라베이스 | 29 | 175 | 74 |

encoding
적용 후

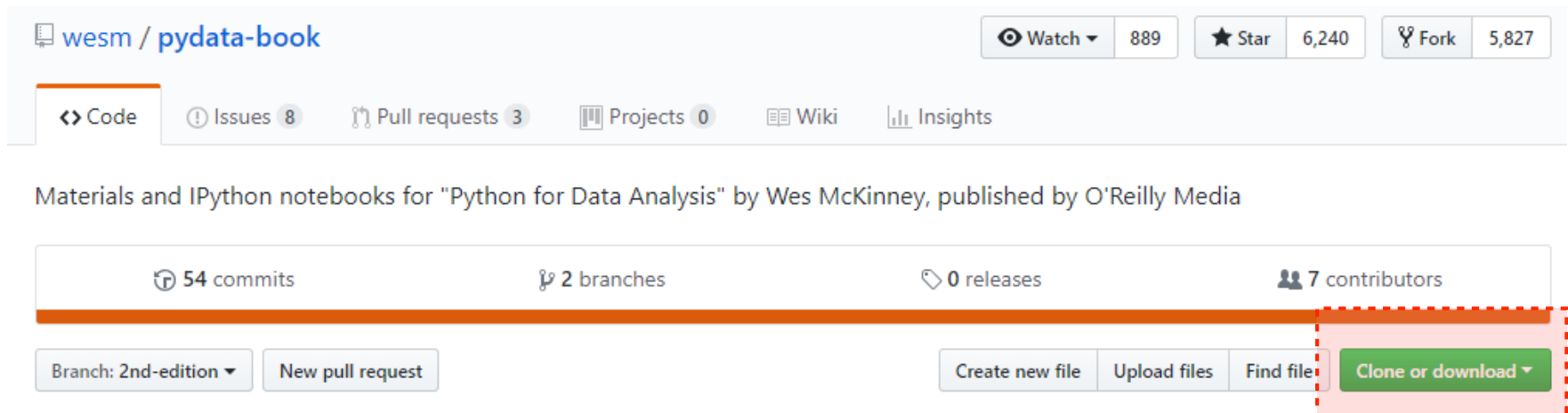
| 이름 | 나이 | 키 | 몸무게 |
|--------|----|-----|-----|
| 홍길동 | 20 | 180 | 80 |
| 짱구 | 25 | 175 | 75 |
| 불닭 | 29 | 170 | 70 |
| 아이폰 | 36 | 165 | 65 |
| 콘트라베이스 | 29 | 175 | 74 |

pandas – 파일 I/O, json 파일 읽어오기

- json 파일 읽어오기
- json 파일이란? 속성-값 쌍으로 이루어진 데이터 오브젝트를 전달하기 위해 인간이 읽을 수 있는 텍스트를 사용하는 개방형 표준 포맷

pandas – 파일 I/O, json 파일 읽어오기

- python 내장모듈인 json 사용
- 웹사이트 <https://github.com/wesm/pydata-book> 에서 Clone or download



wesm / pydata-book

Watch 889 Star 6,240 Fork 5,827

Code Issues 8 Pull requests 3 Projects 0 Wiki Insights

Materials and IPython notebooks for "Python for Data Analysis" by Wes McKinney, published by O'Reilly Media

54 commits 2 branches 0 releases 7 contributors

Branch: 2nd-edition New pull request Create new file Upload files Find file Clone or download

pandas – 파일 I/O, json 파일 읽어오기

- python 내장모듈인 json 사용
- json 데이터를 읽어오는 방법
 - `df = json.load(open('데이터셋 저장위치'))`
 - 윈도우의 경우 '데이터셋저장위치' 앞에 꼭 `r`을 붙여주기

```
1 import json
2 db = json.load(open(r'C:\Users\user\Desktop\8월 강의자료\datasets\usda_food\database.json'))
3 db
```

```
[{'description': 'Cheese, caraway',
  'group': 'Dairy and Egg Products',
  'id': 1008,
  'manufacturer': '',
  'nutrients': [{'description': 'Protein',
    'group': 'Composition',
    'units': 'g',
    'value': 25.18},
    {'description': 'Total lipid (fat)',
    'group': 'Composition',
    'units': 'g',
    'value': 29.2},
    {'description': 'Carbohydrate, by difference',
    'group': 'Composition',
    'units': 'g',
    'value': 3.06},
```

- 데이터에 접근하기

- `json.load(open())` 함수는 json 형태의 데이터를 불러와서 list에 값들을 저장
- dictionary 형태의 구조가 list의 indexing 위치에 저장되어 있음

```
0  {'description': 'Cheese, caraway',  
   'group': 'Dairy and Egg Products',  
   'id': 1008,  
   'manufacturer': '',  
   'nutrients': [{'description': 'Protein',  
                  'group': 'Composition',  
                  'units': 'g',  
                  'value': 25.18},  
1  {'description': 'Total lipid (fat)',  
   'group': 'Composition',  
   'units': 'g',  
   'value': 29.2},  
2  {'description': 'Carbohydrate, by difference',  
   'group': 'Composition',  
   'units': 'g',  
   'value': 3.06}]
```

pandas – 파일 I/O, json 파일 읽어오기

- 데이터에 접근하기
 - 특정 위치에 있는 값의 key값을 알고 싶을때는 key() 함수를 사용
 - 특정 위치에 있는 값을 접근할때는 list indexing 과 dictionary key 값을 대응 시키면됨

```
1 db[0].keys()
```

```
dict_keys(['id', 'description', 'tags', 'manufacturer', 'group', 'portions', 'nutrients'])
```

```
1 # db의 0번째 위치에서 nutrients key값의 0번째 값을 확인
2 db[0]['nutrients'][0]
```

```
{'description': 'Protein',
 'group': 'Composition',
 'units': 'g',
 'value': 25.18}
```

- MovieLens의 영화 평점 데이터 분석
 - 약 6,000여 명의 사용자들로부터 수집한 4,000여 편의 영화에 대한 백만 개의 영화 평점을 담고 있음
 - 이 데이터셋은 평점, 사용자정보, 영화정보 이 세가지 테이블로 나뉘어 있음

- 데이터셋 불러오기

```
1 import pandas as pd
2 import os
3 encoding= 'latin1'
4
5 # 데이터셋 저장위치
6 upath = os.path.expanduser(r'C:\Users\User\Desktop\강의자료\datasets\movielens\users.dat')
7 rpath = os.path.expanduser(r'C:\Users\User\Desktop\강의자료\datasets\movielens\ratings.dat')
8 mpath = os.path.expanduser(r'C:\Users\User\Desktop\강의자료\datasets\movielens\movies.dat')
9
10 # 데이터셋 columns 명 선언
11 unames = ['user_id', 'gender', 'age', 'occupation', 'zip']
12 rnames = ['user_id', 'movie_id', 'rating', 'timestamp']
13 mnames = ['movie_id', 'title', 'genres']
14
15 # 데이터셋 불러오기
16 users = pd.read_csv(upath, sep='::', header=None, names=unames, encoding=encoding)
17 ratings = pd.read_csv(rpath, sep='::', header=None, names=rnames, encoding=encoding)
18 movies = pd.read_csv(mpath, sep='::', header=None, names=mnames, encoding=encoding)
```

- 데이터셋 확인
 - 리스트 분할문법을 사용하여 첫5개의 행을 출력해보기

```
1 users[:5]
```

| | user_id | gender | age | occupation | zip |
|---|---------|--------|-----|------------|-------|
| 0 | 1 | F | 1 | 10 | 48067 |
| 1 | 2 | M | 56 | 16 | 70072 |
| 2 | 3 | M | 25 | 15 | 55117 |
| 3 | 4 | M | 45 | 7 | 02460 |
| 4 | 5 | M | 25 | 20 | 55455 |

```
1 ratings[:5]
```

| | user_id | movie_id | rating | timestamp |
|---|---------|----------|--------|-----------|
| 0 | 1 | 1193 | 5 | 978300760 |
| 1 | 1 | 661 | 3 | 978302109 |
| 2 | 1 | 914 | 3 | 978301968 |
| 3 | 1 | 3408 | 4 | 978300275 |
| 4 | 1 | 2355 | 5 | 978824291 |

```
1 movies[:5]
```

| | movie_id | title | genres |
|---|----------|------------------------------------|------------------------------|
| 0 | 1 | Toy Story (1995) | Animation Children's Comedy |
| 1 | 2 | Jumanji (1995) | Adventure Children's Fantasy |
| 2 | 3 | Grumpier Old Men (1995) | Comedy Romance |
| 3 | 4 | Waiting to Exhale (1995) | Comedy Drama |
| 4 | 5 | Father of the Bride Part II (1995) | Comedy |

예시 - MoiveLens

- 나이와 성별에 따른 영화 평균 평점을 계산해보기
 - pandas의 merge 함수를 이용해 ratings 테이블과 users 테이블을 병합
 - 그 결과 다시 movies 테이블과 병합
 - pandas는 병합하려는 두 테이블에서 **중복되는 열의 이름을 키**로 사용

```
1 users[:5]
```

| | user_id | gender | age | occupation | zip |
|---|---------|--------|-----|------------|-------|
| 0 | 1 | F | 1 | 10 | 48067 |
| 1 | 2 | M | 56 | 16 | 70072 |
| 2 | 3 | M | 25 | 15 | 55117 |
| 3 | 4 | M | 45 | 7 | 02460 |
| 4 | 5 | M | 25 | 20 | 55455 |

```
1 ratings[:5]
```

| | user_id | movie_id | rating | timestamp |
|---|---------|----------|--------|-----------|
| 0 | 1 | 1193 | 5 | 978300760 |
| 1 | 1 | 661 | 3 | 978302109 |
| 2 | 1 | 914 | 3 | 978301968 |
| 3 | 1 | 3408 | 4 | 978300275 |
| 4 | 1 | 2355 | 5 | 978824291 |



```
1 data1 = pd.merge(users, ratings)
2 data1[:5]
```

| | user_id | gender | age | occupation | zip | movie_id | rating | timestamp |
|---|---------|--------|-----|------------|-------|----------|--------|-----------|
| 0 | 1 | F | 1 | 10 | 48067 | 1193 | 5 | 978300760 |
| 1 | 1 | F | 1 | 10 | 48067 | 661 | 3 | 978302109 |
| 2 | 1 | F | 1 | 10 | 48067 | 914 | 3 | 978301968 |
| 3 | 1 | F | 1 | 10 | 48067 | 3408 | 4 | 978300275 |
| 4 | 1 | F | 1 | 10 | 48067 | 2355 | 5 | 978824291 |

예시 - MoiveLens

- 나이와 성별에 따른 영화 평균 평점을 계산해보기
 - pandas의 merge 함수를 이용해 ratings 테이블과 users 테이블을 병합
 - 그 결과 다시 movies 테이블과 병합
 - pandas는 병합하려는 두 테이블에서 **중복되는 열의 이름을 키**로 사용

```
1 data1 = pd.merge(users, ratings)
2 data1[:5]
```

| | user_id | gender | age | occupation | zip | movie_id | rating | timestamp |
|---|---------|--------|-----|------------|-------|----------|--------|-----------|
| 0 | 1 | F | 1 | 10 | 48067 | 1193 | 5 | 978300760 |
| 1 | 1 | F | 1 | 10 | 48067 | 661 | 3 | 978302109 |
| 2 | 1 | F | 1 | 10 | 48067 | 914 | 3 | 978302109 |
| 3 | 1 | F | 1 | 10 | 48067 | 3408 | 4 | 978302109 |
| 4 | 1 | F | 1 | 10 | 48067 | 2355 | 5 | 978302109 |

```
1 movies[:5]
```

| | movie_id | title | genres |
|---|----------|------------------------------------|------------------------------|
| 0 | 1 | Toy Story (1995) | Animation Children's Comedy |
| 1 | 2 | Jumanji (1995) | Adventure Children's Fantasy |
| 2 | 3 | Grumpier Old Men (1995) | Comedy Romance |
| 3 | 4 | Waiting to Exhale (1995) | Comedy Drama |
| 4 | 5 | Father of the Bride Part II (1995) | Comedy |

예시 - MoiveLens

```
1 data2 = pd.merge(data1, movies)
2 data2[:5]
```

| | user_id | gender | age | occupation | zip | movie_id | rating | timestamp | title | genres |
|---|---------|--------|-----|------------|-------|----------|--------|-----------|--|--------|
| 0 | 1 | F | 1 | 10 | 48067 | 1193 | 5 | 978300760 | One Flew Over the Cuckoo's Nest (1975) | Drama |
| 1 | 2 | M | 56 | 16 | 70072 | 1193 | 5 | 978298413 | One Flew Over the Cuckoo's Nest (1975) | Drama |
| 2 | 12 | M | 25 | 12 | 32793 | 1193 | 4 | 978220179 | One Flew Over the Cuckoo's Nest (1975) | Drama |
| 3 | 15 | M | 25 | 7 | 22903 | 1193 | 4 | 978199279 | One Flew Over the Cuckoo's Nest (1975) | Drama |
| 4 | 17 | M | 50 | 1 | 95350 | 1193 | 5 | 978158471 | One Flew Over the Cuckoo's Nest (1975) | Drama |

예시 - MoiveLens

- 한꺼번에 merge 하기

```
1 data2 = pd.merge(data1, movies)
2 data2[:5]
```

| | user_id | gender | age | occupation | zip | movie_id | rating | timestamp | title | genres |
|---|---------|--------|-----|------------|-------|----------|--------|-----------|--|--------|
| 0 | 1 | F | 1 | 10 | 48067 | 1193 | 5 | 978300760 | One Flew Over the Cuckoo's Nest (1975) | Drama |
| 1 | 2 | M | 56 | 16 | 70072 | 1193 | 5 | 978298413 | One Flew Over the Cuckoo's Nest (1975) | Drama |
| 2 | 12 | M | 25 | 12 | 32793 | 1193 | 4 | 978220179 | One Flew Over the Cuckoo's Nest (1975) | Drama |
| 3 | 15 | M | 25 | 7 | 22903 | 1193 | 4 | 978199279 | One Flew Over the Cuckoo's Nest (1975) | Drama |
| 4 | 17 | M | 50 | 1 | 95350 | 1193 | 5 | 978158471 | One Flew Over the Cuckoo's Nest (1975) | Drama |

```
1 data = pd.merge(pd.merge(ratings, users), movies)
2 data[:5]
```

| | user_id | movie_id | rating | timestamp | gender | age | occupation | zip | title | genres |
|---|---------|----------|--------|-----------|--------|-----|------------|-------|--|--------|
| 0 | 1 | 1193 | 5 | 978300760 | F | 1 | 10 | 48067 | One Flew Over the Cuckoo's Nest (1975) | Drama |
| 1 | 2 | 1193 | 5 | 978298413 | M | 56 | 16 | 70072 | One Flew Over the Cuckoo's Nest (1975) | Drama |
| 2 | 12 | 1193 | 4 | 978220179 | M | 25 | 12 | 32793 | One Flew Over the Cuckoo's Nest (1975) | Drama |
| 3 | 15 | 1193 | 4 | 978199279 | M | 25 | 7 | 22903 | One Flew Over the Cuckoo's Nest (1975) | Drama |
| 4 | 17 | 1193 | 5 | 978158471 | M | 50 | 1 | 95350 | One Flew Over the Cuckoo's Nest (1975) | Drama |

- 나이와 성별에 따른 영화 평균 평점을 계산해보기
 - pivot_table 메서드를 사용하면, 성별에 따른 각 영화의 평균 평점을 구할 수 있음
 - pivot_table(DataFrame의 column, index의 형태, 출력하고자 하는 columns, aggfunc='mean'(평균옵션))

예시 - MoiveLens

```
2 data[:5]
```

| | user_id | movie_id | rating | timestamp | gender | age | occupation | zip | title | genres |
|---|---------|----------|--------|-----------|--------|-----|------------|-------|--|--------|
| 0 | 1 | 1193 | 5 | 978300760 | F | 1 | 10 | 48067 | One Flew Over the Cuckoo's Nest (1975) | Drama |
| 1 | 2 | 1193 | 5 | 978298413 | M | 56 | 16 | 70072 | One Flew Over the Cuckoo's Nest (1975) | Drama |
| 2 | 12 | 1193 | 4 | 978220179 | M | 25 | 12 | 32793 | One Flew Over the Cuckoo's Nest (1975) | Drama |
| 3 | 15 | 1193 | 4 | 978199279 | M | 25 | 7 | 22903 | One Flew Over the Cuckoo's Nest (1975) | Drama |
| 4 | 17 | 1193 | 5 | 978158471 | M | 50 | 1 | 95350 | One Flew Over the Cuckoo's Nest (1975) | Drama |

```
1 mean_ratings = data.pivot_table('rating', index='title', columns='gender', aggfunc='mean')
2 mean_ratings[:5]
```

| | gender | |
|-------------------------------|----------|----------|
| | F | M |
| title | | |
| \$1,000,000 Duck (1971) | 3.375000 | 2.761905 |
| 'Night Mother (1986) | 3.388889 | 3.352941 |
| 'Til There Was You (1997) | 2.675676 | 2.733333 |
| 'burbs, The (1989) | 2.793478 | 2.962085 |
| ...And Justice for All (1979) | 3.828571 | 3.689024 |

rating column 값에 대한 평균
(aggfunc='mean')을 구함

예시 - MoiveLens

```
2 data[:5]
```

| | user_id | movie_id | rating | timestamp | gender | age | occupation | zip | title | genres |
|---|---------|----------|--------|-----------|--------|-----|------------|-------|--|--------|
| 0 | 1 | 1193 | 5 | 978300760 | F | 1 | 10 | 48067 | One Flew Over the Cuckoo's Nest (1975) | Drama |
| 1 | 2 | 1193 | 5 | 978298413 | M | 56 | 16 | 70072 | One Flew Over the Cuckoo's Nest (1975) | Drama |
| 2 | 12 | 1193 | 4 | 978220179 | M | 25 | 12 | 32793 | One Flew Over the Cuckoo's Nest (1975) | Drama |
| 3 | 15 | 1193 | 4 | 978199279 | M | 25 | 7 | 22903 | One Flew Over the Cuckoo's Nest (1975) | Drama |
| 4 | 17 | 1193 | 5 | 978158471 | M | 50 | 1 | 95350 | One Flew Over the Cuckoo's Nest (1975) | Drama |

```
1 mean_ratings = data.pivot_table('rating', index='title', columns='gender', aggfunc='mean')
2 mean_ratings[:5]
```

| | gender | |
|-------------------------------|----------|----------|
| | F | M |
| title | | |
| \$1,000,000 Duck (1971) | 3.375000 | 2.761905 |
| 'Night Mother (1986) | 3.388889 | 3.352941 |
| 'Til There Was You (1997) | 2.675676 | 2.733333 |
| 'burbs, The (1989) | 2.793478 | 2.962085 |
| ...And Justice for All (1979) | 3.828571 | 3.689024 |

index는 영화 이름

예시 - MoiveLens

```
2 data[:5]
```

| | user_id | movie_id | rating | timestamp | gender | age | occupation | zip | title | genres |
|---|---------|----------|--------|-----------|--------|-----|------------|-------|--|--------|
| 0 | 1 | 1193 | 5 | 978300760 | F | 1 | 10 | 48067 | One Flew Over the Cuckoo's Nest (1975) | Drama |
| 1 | 2 | 1193 | 5 | 978298413 | M | 56 | 16 | 70072 | One Flew Over the Cuckoo's Nest (1975) | Drama |
| 2 | 12 | 1193 | 4 | 978220179 | M | 25 | 12 | 32793 | One Flew Over the Cuckoo's Nest (1975) | Drama |
| 3 | 15 | 1193 | 4 | 978199279 | M | 25 | 7 | 22903 | One Flew Over the Cuckoo's Nest (1975) | Drama |
| 4 | 17 | 1193 | 5 | 978158471 | M | 50 | 1 | 95350 | One Flew Over the Cuckoo's Nest (1975) | Drama |

```
1 mean_ratings = data.pivot_table('rating', index='title', columns='gender', aggfunc='mean')
2 mean_ratings[:5]
```

| | gender | |
|-------------------------------|----------|----------|
| | F | M |
| \$1,000,000 Duck (1971) | 3.375000 | 2.761905 |
| 'Night Mother (1986) | 3.388889 | 3.352941 |
| 'Til There Was You (1997) | 2.675676 | 2.733333 |
| 'burbs, The (1989) | 2.793478 | 2.962085 |
| ...And Justice for All (1979) | 3.828571 | 3.689024 |

남자(M)와 여자(F)가 영화
에 대한 평균 평점이 몇점
인지를 출력

예시 - MoiveLens

- 평점 정보가 250건 이상의 영화만 추려보기
- 데이터를 영화 제목으로 그룹화하고(groupby) size()함수를 사용해서 제목별 평점 정보 건수를 Series 객체로 얻어냄

```
1 ratings_by_title = data.groupby('title').size()
2 ratings_by_title[:10]
```

```
title
$1,000,000 Duck (1971)      37
'Night Mother (1986)      70
'Til There Was You (1997)  52
'burbs, The (1989)        303
...And Justice for All (1979) 199
1-900 (1994)              2
10 Things I Hate About You (1999) 700
101 Dalmatians (1961)      565
101 Dalmatians (1996)      364
12 Angry Men (1957)       616
dtype: int64
```

- DataFrame의 index를 대입하는 위치에는 조건식을 대입할 수 있음
- 그 결과 아래와같이 250건 이상의 영화데이터만 출력됨

```
1 active_titles = ratings_by_title.index[ratings_by_title>=250]  
2 active_titles
```

```
Index(['burbs, The (1989)', '10 Things I Hate About You (1999)',  
      '101 Dalmatians (1961)', '101 Dalmatians (1996)', '12 Angry Men (1957)',  
      '13th Warrior, The (1999)', '2 Days in the Valley (1996)',  
      '20,000 Leagues Under the Sea (1954)', '2001: A Space Odyssey (1968)',  
      '2010 (1984)',  
      ...  
      'X-Men (2000)', 'Year of Living Dangerously (1982)',  
      'Yellow Submarine (1968)', 'You've Got Mail (1998)',  
      'Young Frankenstein (1974)', 'Young Guns (1988)',  
      'Young Guns II (1990)', 'Young Sherlock Holmes (1985)',  
      'Zero Effect (1998)', 'eXistenZ (1999)'],  
      dtype='object', name='title', length=1216)
```


- 250건 이상의 평점 정보가 있는 영화에 대한 색인은 mean_ratings 에서 항목을 선택하기 위해서 사용

```
1 mean_ratings.loc[active_titles]
2 mean_ratings[:10]
```

| | gender | F | M |
|-----------------------------------|--------|----------|----------|
| title | | | |
| \$1,000,000 Duck (1971) | | 3.375000 | 2.761905 |
| 'Night Mother (1986) | | 3.388889 | 3.352941 |
| 'Til There Was You (1997) | | 2.675676 | 2.733333 |
| 'burbs, The (1989) | | 2.793478 | 2.962085 |
| ...And Justice for All (1979) | | 3.828571 | 3.689024 |
| 1-900 (1994) | | 2.000000 | 3.000000 |
| 10 Things I Hate About You (1999) | | 3.646552 | 3.311966 |
| 101 Dalmatians (1961) | | 3.791444 | 3.500000 |
| 101 Dalmatians (1996) | | 3.240000 | 2.911215 |
| 12 Angry Men (1957) | | 4.184397 | 4.328421 |

- 여성에게 높은 평점을 받은 영화 목록을 확인하기 위해 F열을 내림차순으로 정렬

```
1 # 여성에게 높은 평점을 받은 영화 목록을 확인하기 위해 F 열을 내림차순으로 정렬
2 top_female_ratings = mean_ratings.sort_values(by='F', ascending=False)
3 top_female_ratings[:10]
```

| | gender | F | M |
|---|--------|-----|----------|
| title | | | |
| Clean Slate (Coup de Torchon) (1981) | | 5.0 | 3.857143 |
| Ballad of Narayama, The (Narayama Bushiko) (1958) | | 5.0 | 3.428571 |
| Raw Deal (1948) | | 5.0 | 3.307692 |
| Bittersweet Motel (2000) | | 5.0 | NaN |
| Skipped Parts (2000) | | 5.0 | 4.000000 |
| Lamerica (1994) | | 5.0 | 4.666667 |
| Gambler, The (A Játékos) (1997) | | 5.0 | 3.166667 |
| Brother, Can You Spare a Dime? (1975) | | 5.0 | 3.642857 |
| Ayn Rand: A Sense of Life (1997) | | 5.0 | 4.000000 |
| 24 7: Twenty Four Seven (1997) | | 5.0 | 3.750000 |

- 남녀간의 호불호가 갈리는 영화를 찾아보기
 - mean_ratings에 평균 평점의 차이를 담을 수 있는 열을 하나 추가
 - 그 열을 기준으로 정렬

```
1 mean_ratings['diff'] = mean_ratings['M'] - mean_ratings['F']
2 sorted_by_diff = mean_ratings.sort_values(by='diff')
3 sorted_by_diff[:10]
```

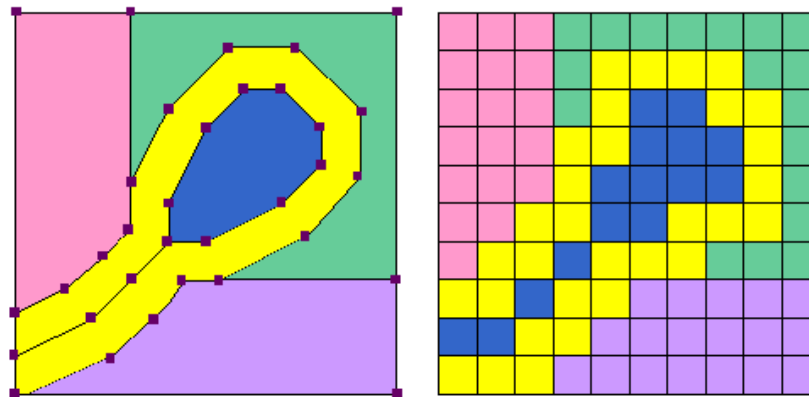
| | gender | F | M | diff |
|-------|---|----------|----------|-----------|
| title | | | | |
| | James Dean Story, The (1957) | 4.000000 | 1.000000 | -3.000000 |
| | Country Life (1994) | 5.000000 | 2.000000 | -3.000000 |
| | Spiders, The (Die Spinnen, 1. Teil: Der Goldene See) (1919) | 4.000000 | 1.000000 | -3.000000 |
| | Babyfever (1994) | 3.666667 | 1.000000 | -2.666667 |
| | Woman of Paris, A (1923) | 5.000000 | 2.428571 | -2.571429 |
| | Cobra (1925) | 4.000000 | 1.500000 | -2.500000 |
| | Other Side of Sunday, The (Søndagsengler) (1996) | 5.000000 | 2.928571 | -2.071429 |
| | To Have, or Not (1995) | 4.000000 | 2.000000 | -2.000000 |
| | For the Moment (1994) | 5.000000 | 3.000000 | -2.000000 |
| | Phat Beach (1996) | 3.000000 | 1.000000 | -2.000000 |

- 성별에 관계없이 영화에 대한 호불호가 극명하게 나뉘는 영화를 찾아 보기
- 호불호는 평점이 분산이나 표준편차를 통해 측정가능

```
1 # 평점이 표준편차
2 ratings_std_by_title = data.groupby('title')['rating'].std()
3 # active_titles만 선택
4 rating_std_by_title = ratings_std_by_title.loc[active_titles]
5 # 내림차순으로 정렬
6 rating_std_by_title.sort_values(ascending=False)[:10]
```

```
title
Dumb & Dumber (1994)          1.321333
Blair Witch Project, The (1999) 1.316368
Natural Born Killers (1994)    1.307198
Tank Girl (1995)              1.277695
Rocky Horror Picture Show, The (1975) 1.260177
Eyes Wide Shut (1999)         1.259624
Evita (1996)                  1.253631
Billy Madison (1995)          1.249970
Fear and Loathing in Las Vegas (1998) 1.246408
Bicentennial Man (1999)        1.245533
Name: rating, dtype: float64
```

- matplotlib은 주로 2D 도표를 위한 데스크톱 패키지
- 3D 도식을 위한 matplotlib3d 및 지도와 투영을 위한 basemap 같은 다양한 확장툴킷도 존재
- 다양한 GUI 백엔드를 지원하고 있으며, PDF, SVG, JPG, PNG, GMP, GIF 등 일반적으로 널리 사용되는 벡터(Vector) 포맷과 래스터(Raster) 포맷으로 도표를 저장가능



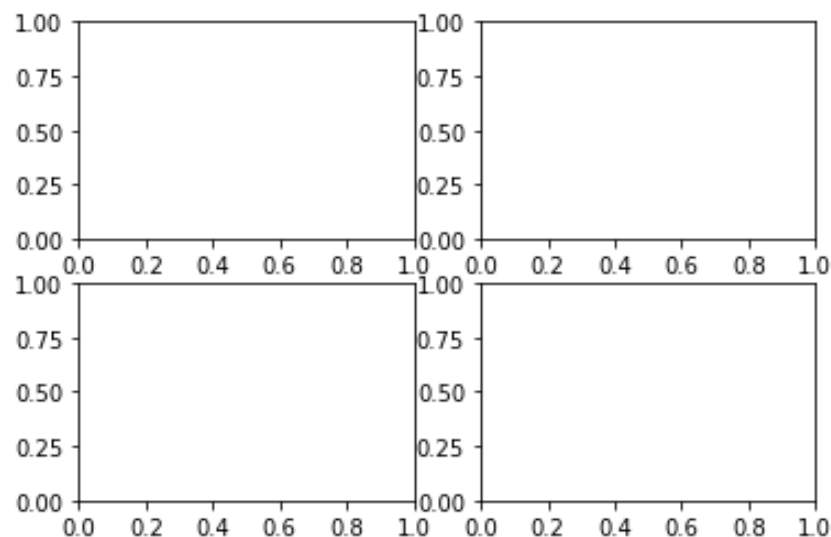
Vector

Raster

- matplotlib API 함수는 matplotlib.pyplot 모듈에 포함되어 있음. 보통 plt 라는 이름으로 import 해서 사용
- 사용방법
 - `import matplotlib.pyplot as plt`

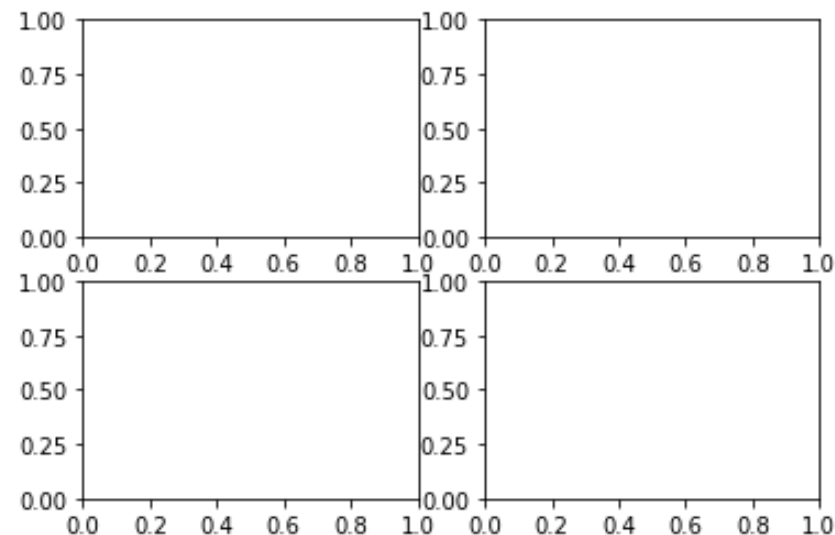
- matplotlib에서 그래프는 Figure 객체 내에 존재
- 그래프를 위한 새로운 Figure는 `plt.figure`를 사용해서 생성 할 수 있음
- 빈 Figure 객체로는 그래프를 만들 수 없으므로 `add_subplot`을 사용해 최소한 하나 이상의 서브플롯을 생성해야 함

```
1 import matplotlib.pyplot as plt
2 %matplotlib inline
3 fig = plt.figure()
4 ax1 = fig.add_subplot(2,2,1)
5 ax2 = fig.add_subplot(2,2,2)
6 ax3 = fig.add_subplot(2,2,3)
7 ax4 = fig.add_subplot(2,2,4)
```



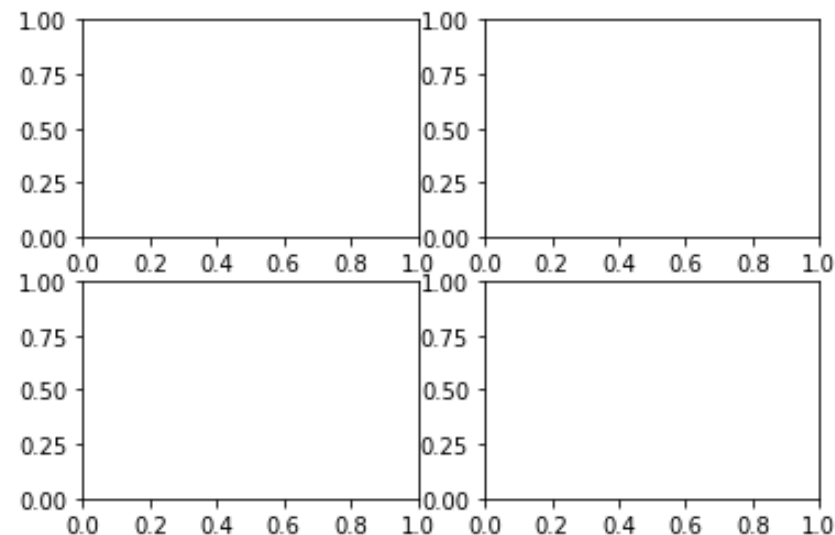
matplotlib.pyplot 패키지 import

```
1 import matplotlib.pyplot as plt
2 %matplotlib inline
3 fig = plt.figure()
4 ax1 = fig.add_subplot(2,2,1)
5 ax2 = fig.add_subplot(2,2,2)
6 ax3 = fig.add_subplot(2,2,3)
7 ax4 = fig.add_subplot(2,2,4)
```



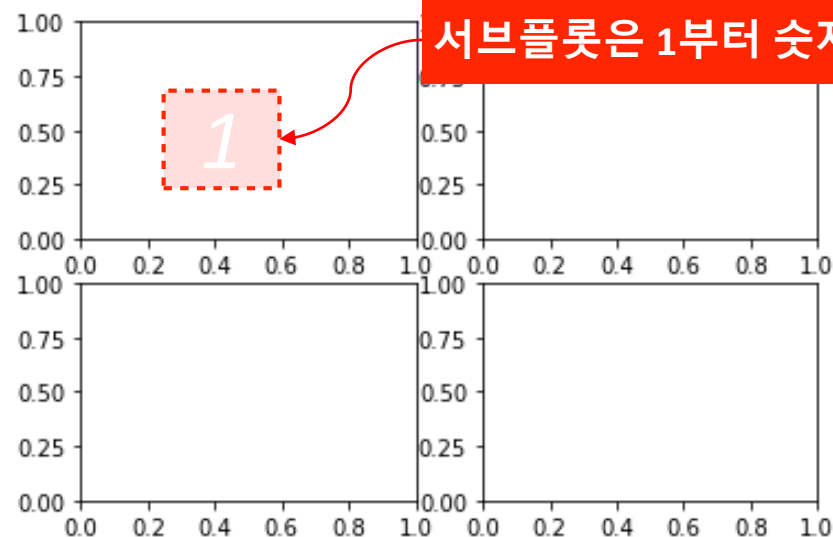
```
1 import matplotlib.pyplot as plt
2 %matplotlib inline
3 fig = plt.figure()
4 ax1 = fig.add_subplot(2,2,1)
5 ax2 = fig.add_subplot(2,2,2)
6 ax3 = fig.add_subplot(2,2,3)
7 ax4 = fig.add_subplot(2,2,4)
```

사용하는 IDE가 Jupyter Notebook 계열이라면 반드시 이 라인을 추가해줘야함



이 코드에서 fig객체는 크기가 2x2 이고 4개의 서브플롯 중에서 첫번째를 선택하겠다는 의미

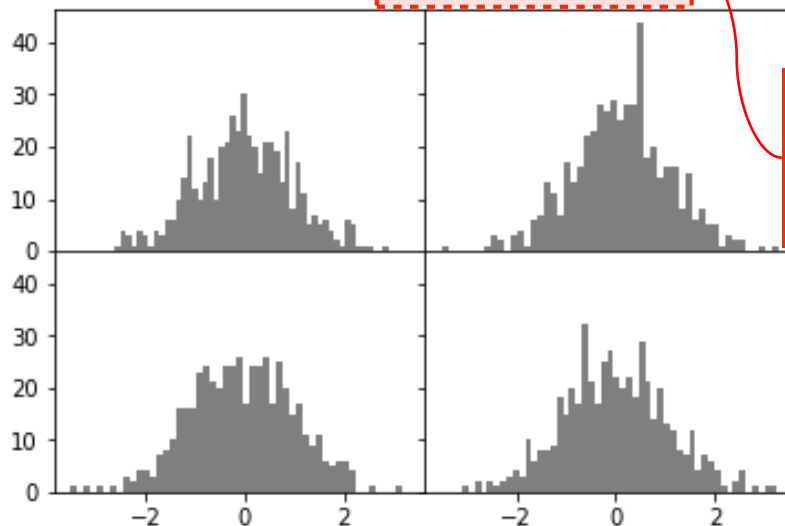
```
1 import matplotlib.pyplot as plt
2 %matplotlib inline
3 fig = plt.figure()
4 ax1 = fig.add_subplot(2,2,1)
5 ax2 = fig.add_subplot(2,2,2)
6 ax3 = fig.add_subplot(2,2,3)
7 ax4 = fig.add_subplot(2,2,4)
```



서브플롯은 1부터 숫자가 매겨짐

- 서브플롯 간 간격 조절하기
 - subplots_adjust() 함수 사용

```
1 import matplotlib.pyplot as plt
2 %matplotlib inline
3 from numpy.random import randn
4
5 fig, axes = plt.subplots(2,2, sharex=True, sharey=True)
6 for i in range(2):
7     for j in range(2):
8         axes[i,j].hist(randn(500), bins=50, color='k', alpha=0.5)
9 plt.subplots_adjust(wspace=0, hspace=0)
```

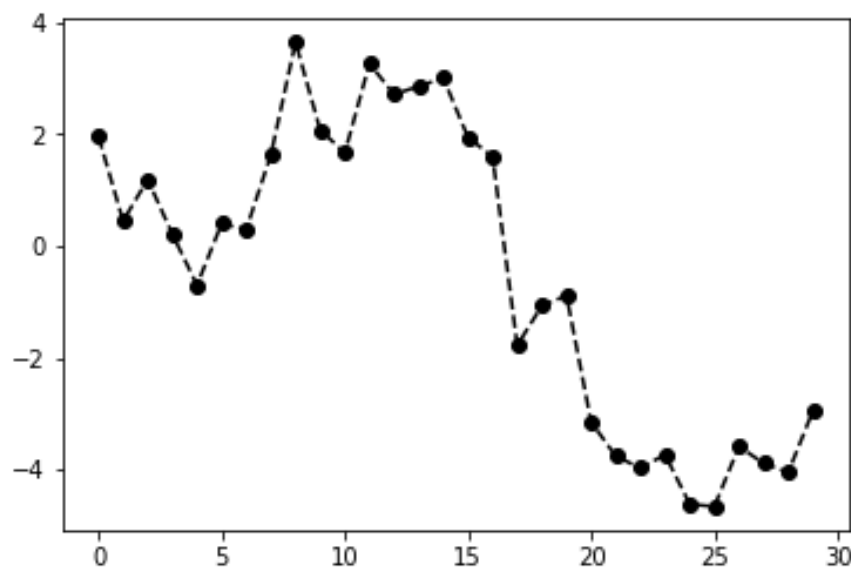


wspace와 hspace는 서브플롯 간 간격을 위해 각각 Figure의 너비와 높이에 대한 비율을 조절

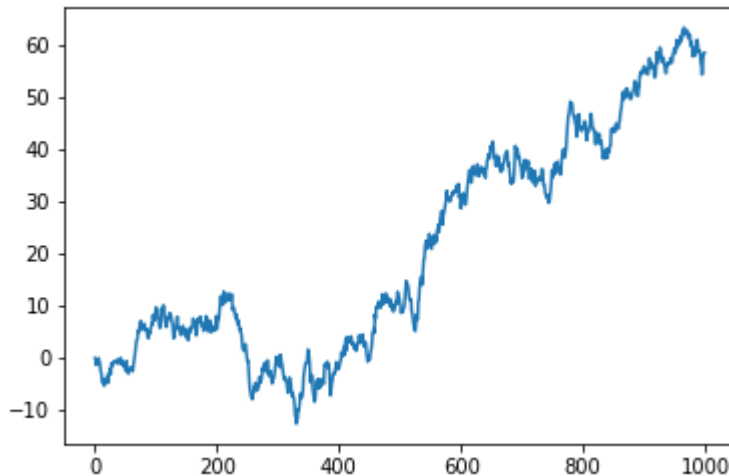
- 색상, 마커, 선 스타일
 - plot()함수는 x와 y 좌표 값이 담긴 배열과 추가적으로 색상과 선 스타일을 나타내는 문자열을 인자로 받음
 - 선 그래프는 특정 지점의 실제 데이터를 돋보이게 하기 위해 마커를 추가하기도 함
 - 마커도 스타일 문자열에 포함시킬 수 있는데, 색상 다음에 마커 스타일이 오고 그 뒤에 선 스타일을 지정함

- 색상, 마커, 선 스타일

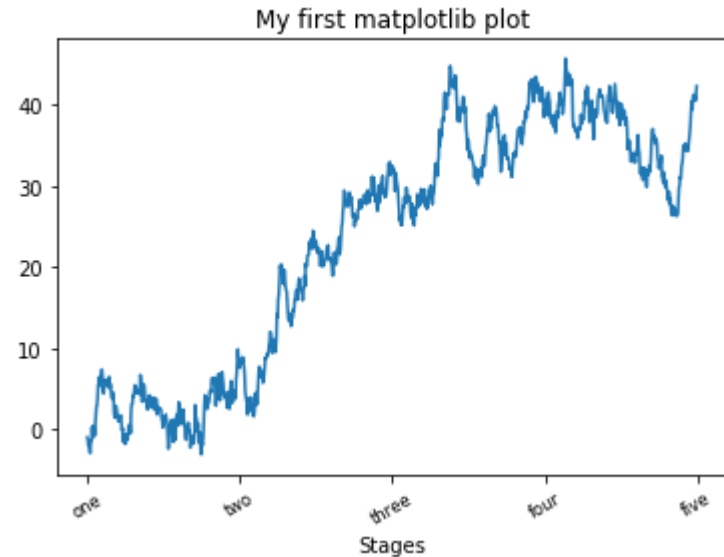
```
1 import matplotlib.pyplot as plt
2 %matplotlib inline
3 from numpy.random import randn
4
5 plt.plot(randn(30).cumsum(), color='k', linestyle='dashed', marker='o')
6 plt.show()
```



- 제목, 축 이름, 눈금, 눈금 이름 설정
 - x축 눈금 변경: `set_xticks` 와 `set_xticklabels` 메서드 사용
 - x축에 대한 이름 지정: `set_xlabel` 메서드 사용
 - 서브플롯 제목 지정: `set_title` 메서드



변경 전

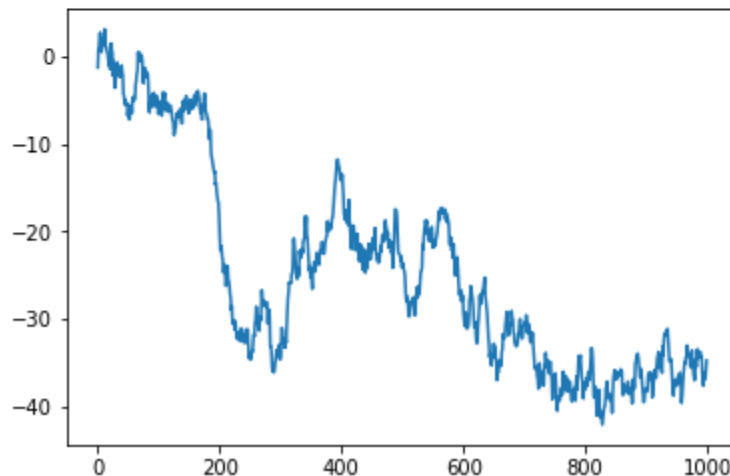


변경 후

- 변경 전

```
1 import matplotlib.pyplot as plt
2 %matplotlib inline
3 from numpy.random import randn
4
5 fig = plt.figure(); ax= fig.add_subplot(1,1,1)
6 ax.plot(randn(1000).cumsum())
7 # ticks = ax.set_xticks([0,250,500,750])
8 # labels = ax.set_xticklabels(['one', 'two', 'three'])
9 # ax.set_title('My first matplotlib plot')
10 # ax.set_xlabel('Stages')
```

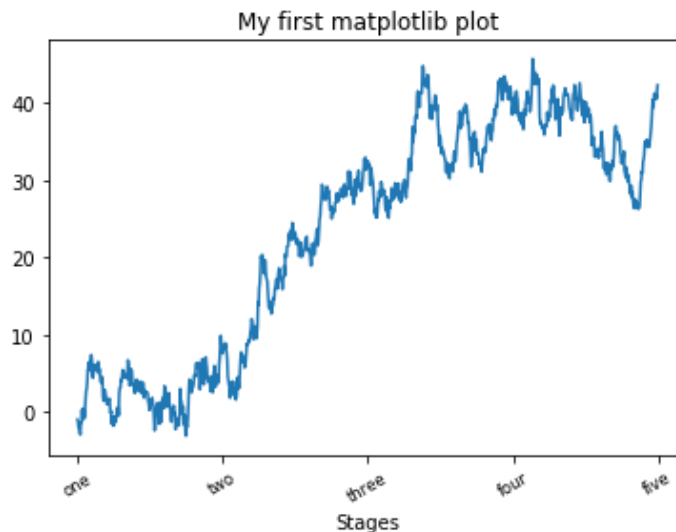
[<matplotlib.lines.Line2D at 0x1d762c6a320>]



- 변경 후

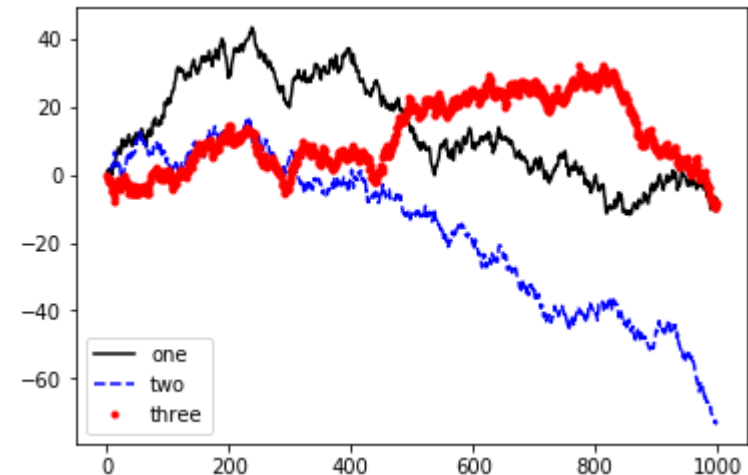
```
1 import matplotlib.pyplot as plt
2 %matplotlib inline
3 from numpy.random import randn
4
5 fig = plt.figure(); ax= fig.add_subplot(1,1,1)
6 ax.plot(randn(1000).cumsum())
7 ticks = ax.set_xticks([0,250,500,750,1000])
8 labels = ax.set_xticklabels(['one','two','three','four','five'], rotation=30, fontsize='small')
9 ax.set_title('My first matplotlib plot')
10 ax.set_xlabel('Stages')
```

Text(0.5,0,'Stages')

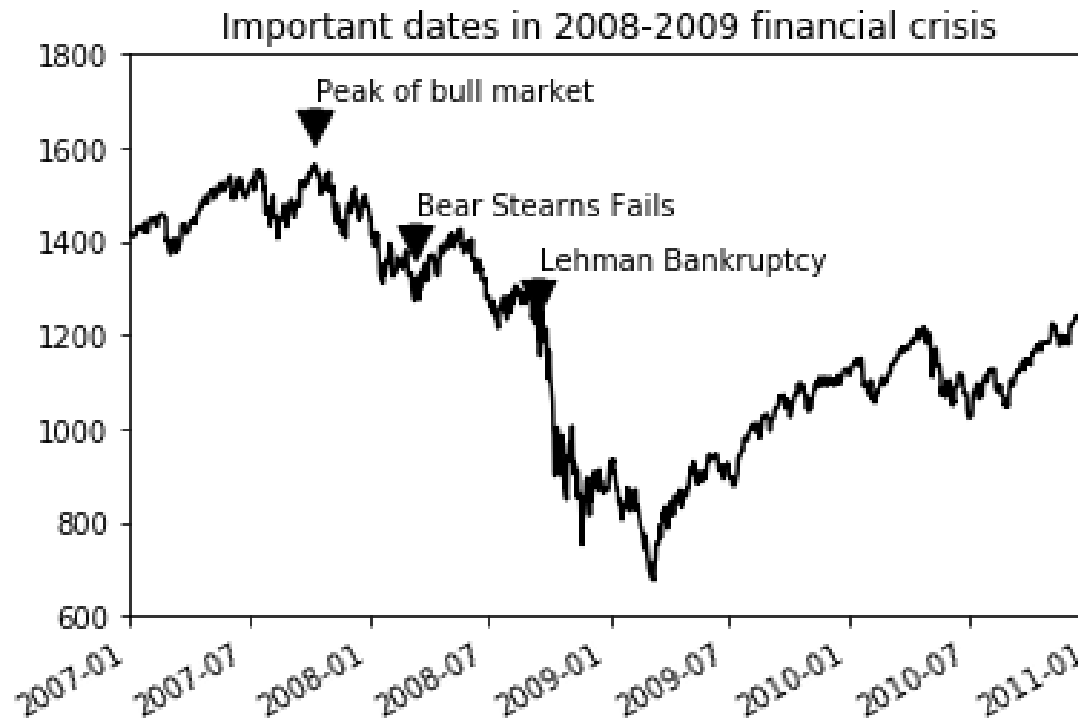


- 범례 추가
 - 각 그래프에 label 인자를 넘기는 것
 - 그 후 legend() 를 실행했을 때 자동으로 범례가 생성

```
1 import matplotlib.pyplot as plt
2 %matplotlib inline
3 from numpy.random import randn
4
5 fig = plt.figure(); ax = fig.add_subplot(1,1,1)
6
7 ax.plot(randn(1000).cumsum(), 'k', label='one')
8 ax.plot(randn(1000).cumsum(), 'k--', label='two')
9 ax.plot(randn(1000).cumsum(), 'k.', label='three')
10 ax.legend(loc='best')
```



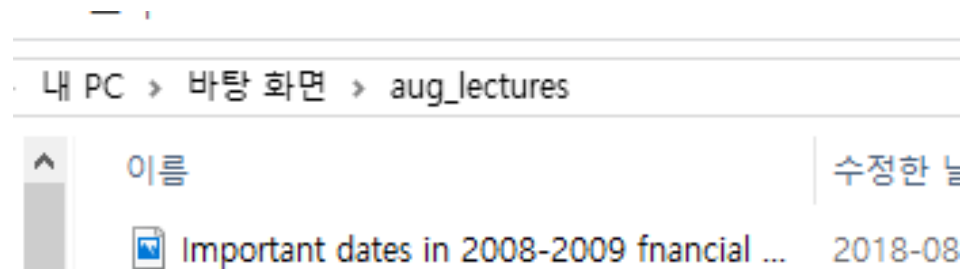
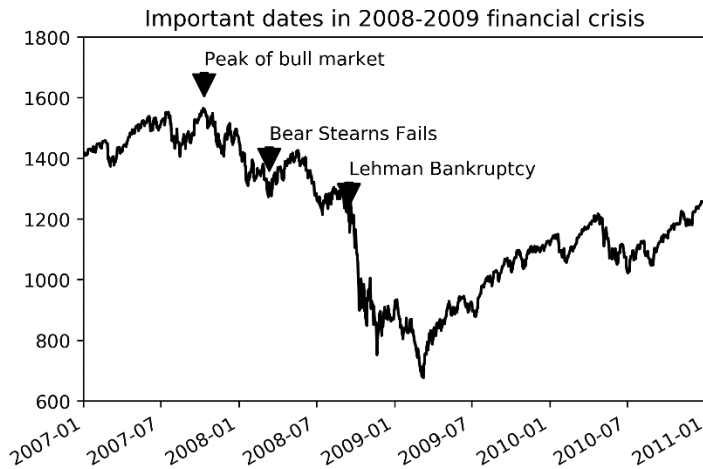
- 주식에 그림 추가
 - 도표에 추가적으로 글자나 화살표 혹은 다른 도형으로 자기만의 주석을 그리고 싶을때
 - 주식과 글자는 text, arrow, annotate 함수를 사용해서 추가할 수 있음



```
1 from datetime import datetime
2 import pandas as pd
3
4 fig = plt.figure()
5 ax = fig.add_subplot(1,1,1)
6
7 data = pd.read_csv(r'C:\Users\User\Downloads\pydata-book-2nd-edition\pydata-book-2nd-edition\examples\spx.csv',
8                   index_col=0, parse_dates=True)
9 spx = data['SPX']
10 spx.plot(ax=ax, style='k-')
11 crisis_data = [
12     (datetime(2007, 10, 11), 'Peak of bull market'),
13     (datetime(2008, 3, 12), 'Bear Stearns Fails'),
14     (datetime(2008, 9, 15), 'Lehman Bankruptcy')
15 ]
16
17 for date, label in crisis_data:
18     ax.annotate(label, xy=(date, spx.asof(date)+50),
19               xytext=(date, spx.asof(date)+200),
20               arrowprops=dict(facecolor='black'),
21               horizontalalignment='left', verticalalignment='top')
22
23 # 2007 ~ 2010 구간 확대
24 ax.set_xlim(['1/1/2007', '1/1/2011'])
25 ax.set_ylim([600, 1800])
26
27 ax.set_title('Important dates in 2008-2009 financial crisis')
```

- 그래프를 파일로 저장
 - savefig() 함수 사용
 - 자주 사용하는 몇가지 중요한 옵션
 - bbox_inches는 실제 Figure 둘레의 공백을 잘라냄
 - dpi 는 인치당 도트 해상도를 조절

```
1 fig.savefig('Important dates in 2008-2009 fnancial crisis.png', dpi=400, bbox_inches='tight')
```



- pandas에서 그래프 그리기
- 요일별 파티 숫자를 뽑고 파티 숫자 대비 팁 비율을 보여주는 막대그래프

```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3 %matplotlib inline
4
5 path = r'C:\Users\User\Downloads\pydata-book-2nd-edition\pydata-book-2nd-edition\examples\tips.csv'
6 tips = pd.read_csv(path)
7 party_counts = pd.crosstab(tips['day'], tips['size'])
8 party_counts
```

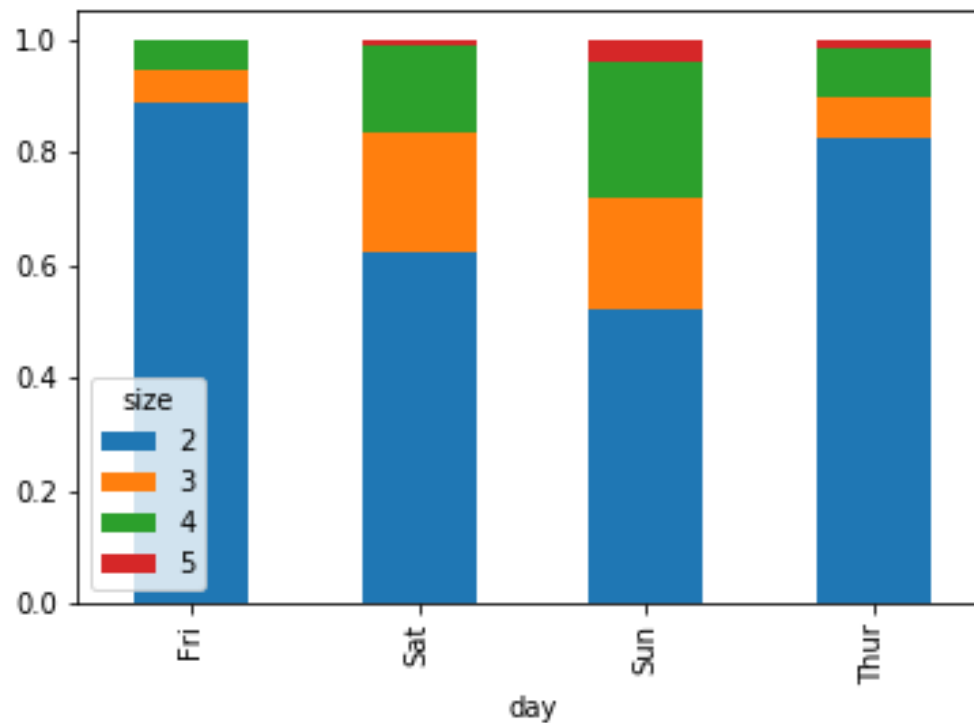
| size | 1 | 2 | 3 | 4 | 5 | 6 |
|------|---|----|----|----|---|---|
| Fri | 1 | 16 | 1 | 1 | 0 | 0 |
| Sat | 2 | 53 | 18 | 13 | 1 | 0 |
| Sun | 0 | 39 | 15 | 18 | 3 | 1 |
| Thur | 1 | 48 | 4 | 5 | 1 | 3 |

```
1 # 혼자거나 6명 규모의 파티는 제외
2 party_counts = party_counts.loc[:, 2:5]
3
4 # 총합이 1이 되도록 정규화
5 party_pcts = party_counts.div(party_counts.sum(1).astype(float), axis=0)
6 party_pcts
```

| size | 2 | 3 | 4 | 5 |
|------|----------|----------|----------|----------|
| day | | | | |
| Fri | 0.888889 | 0.055556 | 0.055556 | 0.000000 |
| Sat | 0.623529 | 0.211765 | 0.152941 | 0.011765 |
| Sun | 0.520000 | 0.200000 | 0.240000 | 0.040000 |
| Thur | 0.827586 | 0.068966 | 0.086207 | 0.017241 |


```
1 party_pcts.plot(kind='bar', stacked=True)
```

<matplotlib.axes._subplots.AxesSubplot at 0x1d763df2f28>



예시 – 신생아 이름 분석

- 미국 사회보장국(Social Security Administration)에서는 1880년부터 현재 까지 가장 빈도가 높은 신생아 이름에 대한 정보를 제공하고 있음
- 이 데이터를 이용해서 여러 가지 분석을 할 수 있음
 - 시대별로 특징 이름이 차지하는 비율 구해 얼마나 흔한 이름인지 파악
 - 이름의 상대 순위 알아보기
 - 각 연도별 가장 인기 있는 이름, 가장 빈도가 높거나 낮은 이름 알아보기
 - 모음, 자음, 길이, 전반적인 다양성, 철자 변화, 첫 글자와 마지막 글자 등 이름 유행 분석
 - 성서에 등장하는 이름, 유명인, 인구통계학적 변화 등 외부 자료를 통한 유행분석

예시 – 신생아 이름 분석

- 다운받은 데이터셋 폴더에서 babynames에 있는 폴더에 있는 .txt 데이터 사용
- 이 데이터셋은 쉼표로 구분되어 있으므로, `pandas.read_csv()` 함수를 사용하여 DataFrame 객체로 데이터를 불러오기

예시 - 신생아 이름 분석

- 데이터 불러오기

```
1 mean_ratings['diff'] = mean_ratings['M'] - mean_ratings['F']  
2 sorted_by_diff = mean_ratings.sort_values(by='diff')  
3 sorted_by_diff[:10]
```

| | gender | F | M | diff |
|---|--------|----------|----------|-----------|
| title | | | | |
| James Dean Story, The (1957) | | 4.000000 | 1.000000 | -3.000000 |
| Country Life (1994) | | 5.000000 | 2.000000 | -3.000000 |
| Spiders, The (Die Spinnen, 1. Teil: Der Goldene See) (1919) | | 4.000000 | 1.000000 | -3.000000 |
| Babyfever (1994) | | 3.666667 | 1.000000 | -2.666667 |
| Woman of Paris, A (1923) | | 5.000000 | 2.428571 | -2.571429 |
| Cobra (1925) | | 4.000000 | 1.500000 | -2.500000 |
| Other Side of Sunday, The (Søndagsengler) (1996) | | 5.000000 | 2.928571 | -2.071429 |
| To Have, or Not (1995) | | 4.000000 | 2.000000 | -2.000000 |
| For the Moment (1994) | | 5.000000 | 3.000000 | -2.000000 |
| Phat Beach (1996) | | 3.000000 | 1.000000 | -2.000000 |

예시 - 신생아 이름 분석

- 자료가 연도별로 나뉘어져 있으니 먼저 모든 데이터를 DataFrame 하나로 취합한 후 year항목을 추가
- 이 작업은 pandas.concat을 이용하면 쉽게 할 수 있음

```

1 years = range(1880, 2011)
2 pieces = []
3 columns = ['name', 'sex', 'births']
4
5 for year in years:
6     path = r'C:\Users\User\Desktop\aug_lectures\datasets\babynames\yob%d.txt' % year
7     frame = pd.read_csv(path, names=columns, header=None)
8     frame['year'] = year
9     pieces.append(frame)
10 # 모두 하나의 DataFrame으로 취합
11 # ignore_index 같은 경우 read_csv 를 통해서 읽어온 원래 데이터의 행 순서는 몰라도 되므로,
12 # concat 메서드에 ignore_index=True 인자전달
13 names = pd.concat(pieces, ignore_index=True)
14 names[10:20]
```

예시 – 신생아 이름 분석

- `pivot_table` 함수를 이용해서 연도나 성별에 따른 데이터를 수집
 - 각 연도별로 남자 아이와 여자 아이의 출생수를 구해보기

```
1 total_births = names.pivot_table('births', index='year', columns='sex', aggfunc=sum)
2 total_births.tail()
```

| sex | F | M |
|------|---------|---------|
| year | | |
| 2006 | 1896468 | 2050234 |
| 2007 | 1916888 | 2069242 |
| 2008 | 1883645 | 2032310 |
| 2009 | 1827643 | 1973359 |
| 2010 | 1759010 | 1898382 |

예시 - 신생아 이름 분석

- prop 열을 추가해서 각 이름이 전체 출생수에서 차지하는 비율을 계산
 - prop값이 0.02라면 100명의 아기 중에 2명의 이름이 같다는 뜻
 - apply() 함수는 커스텀 함수를 사용하기 위해서 사용하는 함수
 - DataFrame이름.apply(커스텀 함수명) 을 하게되면, DataFrame의 이름이 커스텀함수의 매개변수로 들어가서, 처리가 된 후 값을 반환

```
1 def add_prop(group):
2     # integer division floors
3     births = group.births.astype(float)
4     group['prop'] = births / births.sum()
5     return group
6 names = names.groupby(['year', 'sex']).apply(add_prop)
```

```
1 names[:5]
```

| | name | sex | births | year | prop |
|---|-----------|-----|--------|------|----------|
| 0 | Mary | F | 7065 | 1880 | 0.077643 |
| 1 | Anna | F | 2604 | 1880 | 0.028618 |
| 2 | Emma | F | 2003 | 1880 | 0.022013 |
| 3 | Elizabeth | F | 1939 | 1880 | 0.021309 |
| 4 | Minnie | F | 1746 | 1880 | 0.019188 |

예시 - 신생아 이름 분석

- 각 연도별, 성별에 따른 빈도수가 가장 높은 이름 1,000개를 groupby 연산을 사용하여 추출

```
1 import numpy as np
2 def get_top1000(group):
3     return group.sort_values(by='births', ascending=False)[:1000]
4 grouped = names.groupby(['year', 'sex'])
5 top1000 = grouped.apply(get_top1000)
6 top1000.index = np.arange(len(top1000))
7
8 top1000[:1000]
```

| | name | sex | births | year | prop |
|---|-----------|-----|--------|------|----------|
| 0 | Mary | F | 7065 | 1880 | 0.077643 |
| 1 | Anna | F | 2604 | 1880 | 0.028618 |
| 2 | Emma | F | 2003 | 1880 | 0.022013 |
| 3 | Elizabeth | F | 1939 | 1880 | 0.021309 |
| 4 | Minnie | F | 1746 | 1880 | 0.019188 |
| 5 | Margaret | F | 1578 | 1880 | 0.017342 |
| 6 | Ida | F | 1472 | 1880 | 0.016177 |
| 7 | Alice | F | 1414 | 1880 | 0.015540 |

예시 – 신생아 이름 분석

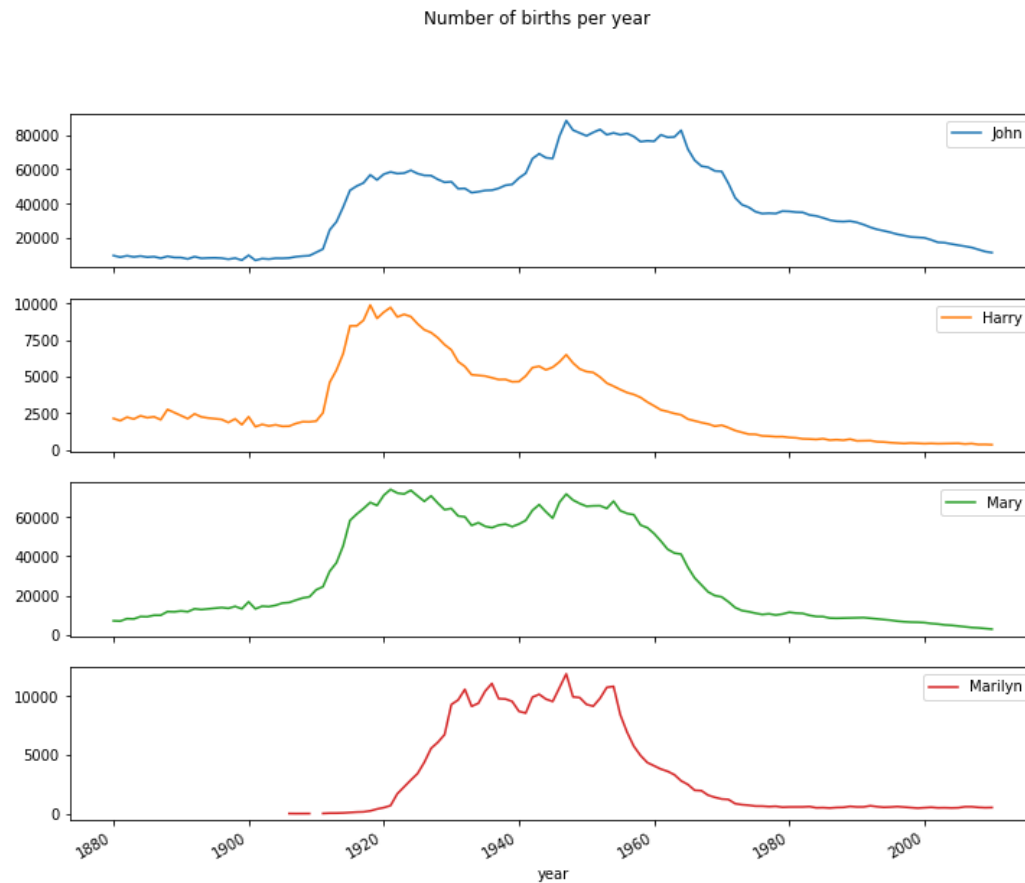
- 이름 유행 분석
 - 상위 1000명의 데이터를 남자아이와 여자아이로 분리
 - 연도와 이름에 대한 전체 출생수를 피벗 테이블로 만들기
 - DataFrame의 plot 메서드를 사용해서 몇몇 이름의 추이를 그래프로 그리기

```

1  #1. 상위 1000명의 데이터를 남자와 여자아이로 분리
2  boys = top1000[top1000.sex == 'M']
3  girls = top1000[top1000.sex == 'F']
4
5  #2. pivot_table 만들기
6  total_births = top1000.pivot_table('births', index='year', columns='name', aggfunc=sum)
7
8  #3. DataFrame의 plot 메서드를 사용해서 몇몇 이름의 추이를 그래프로 그려보기
9  subset = total_births[['John', 'Harry', 'Mary', 'Marilyn']]
10 subset.plot(subplots=True, figsize=(12,10), grid=False,
11              title='Number of births per year')
12
    
```

예시 – 신생아 이름 분석

- 이름 유행 분석



예시 - 신생아 이름 분석

- 위의 그림을 통해서 부모가 아이의 이름을 지을 때 흔한 이름은 기피하는 것으로 해석할 수 있음
- 위와 같은 해석은 데이터에서 살펴볼 수 있으며 확인도 가능
- 인기 있는 이름 1000개가 전체 출생수에서 차지하는 비율을 그래프로 그려보기

```
1 table = top1000.pivot_table('prop', index='year', columns='sex', aggfunc=sum)
2 table.plot(title='Sum of table1000.prop by year ans sex',
3           yticks=np.linspace(0, 1.2, 13), xticks=range(1880, 2020, 10))
4 df = boys[boys.year == 2010]
5 df
```

