

# 데이터 사이언스를 위한 Advanced Python

## 강의 일정 및 내용

월

유용한 Python 테크닉(map, filter 등)

화

데이터베이스(sqlite, Python에서 DB 사용하기)

수

Data manipulation with Pandas

목

Basic Statistical Analysis with Python and Pandas

금

Visualization with Python matplotlib

데이터 사이언스를 위한 Advanced Python

# Jupyter Notebook, Design, and List

2019.09.23

# Jupyter Notebook

## 셀 단위 코드 관리

- Jupyter Notebook 환경에서 개발하면 코드를 셀 단위로 관리할 수 있다
  - 이는 커다란 장점이자 단점
- **셀(Cell)**은 코드를 입력할 수 있는 부분과 코드의 실행 결과를 나타내는 출력 창의 두 가지로 구성된다.
- 코드의 실행은 셀(Cell) 단위로 일어나지만, **실행 환경은 문서 단위에서 공유된다.**
- 따라서 Jupyter Notebook을 이용해 파이썬 코드를 작성할 때는 셀(Cell)을 잘 관리하여야 한다.

# Jupyter Notebook

## 쓸만한 Magic Functions - ?, %whos

- 객체(Object)의 뒤에 물음표를 붙여 실행하면 객체의 정보를 확인할 수 있다.

```
Type:          int
String form: 3
Docstring:
int(x=0) -> integer
int(x, base=10) -> integer
```

- %whos를 이용하면 현재 메모리에 할당된 객체 정보를 확인할 수 있다.

```
[ ] %whos
```

Variable	Type	Data/Info
a	int	3

# Jupyter Notebook

## 쓸만한 Magic Functions - %time, %timeit

- %time [script]
  - 주어진 script를 한 번 실행했을 때 걸린 시간을 계산
- %timeit [script]
  - 주어진 script를 여러 번 실행한 후 걸린 시간을 계산
- 여러 줄의 코드에 대한 실행 시간을 확인하고 싶으면 %%time, %%timeit 사용

```
[34] %time x = [i % 3 for i in range(1000)].
```

```
[> CPU times: user 63 µs, sys: 1 µs, total: 64 µs  
Wall time: 66.5 µs
```

```
[35] %timeit x = [i % 3 for i in range(1000)].
```

```
[> 10000 loops, best of 3: 48.7 µs per loop
```

# Jupyter Notebook

## 쓸만한 Magic Functions - %%writefile, %run

- %%writefile [filename]
  - 해당 셀의 내용을 지정된 파일 이름을 가진 외부 파일로 저장
- %run [filename]
  - 지정된 파일 이름을 가진 파이썬 스크립트를 저장

```
[ ] %%writefile test.py  
  
    message = "Hello how are you?"  
    for word in message.split():  
        print (word)
```

☞ Overwriting test.py

```
[ ] %run test.py
```

# Design

## 프로그램을 작성하기에 앞서 고민할 것

- 주어진 문제를 해결하기 위해 어떤 기능이 필요한가
- 예상되는 문제점은 없는가 (예외처리)
- 간단한 예제를 이용해 검증하기

# Design

## 문제1. 3의 배수, 5의 배수의 합 구하기

- 10 미만의 자연수에서 3 또는 5의 배수는 [3, 5, 6, 9]이며, 이들의 총합은 23 이다.
- 1000 미만의 자연수에서 3 또는 5의 배수의 합을 구하여라.



# Design

## 문제1. 3의 배수, 5의 배수의 합 구하기

- 10 미만의 자연수에서 3 또는 5의 배수는 [3, 5, 6, 9]이며, 이들의 총합은 23 이다.
- 1000 미만의 자연수에서 3 또는 5의 배수의 합을 구하여라.

```
# use for loop
s = 0
for ... in ...:
    ...
print (s)
```

```
# use set
set3 = set(...)
set5 = set(...)
s = ...
print (s)
```

# Design

## 문제2. 승률 올리기

- 당신은 윈도우 운영체제에 포함되어 있는 카드게임을 좋아한다.
- 처음에는 지는 경우가 있었는데, 점점 연습을 함에 따라 필승법을 발견하였고 매번 승리를 하게 되었다. 카드게임을 하면 플레이어에 대한 정보가 다음과 같이 주어지는데, 당신은 표시되고 있는 승률을 1%이상 올리기 위해선 **최소한 몇 번의 연승이 필요한지** 의구심이 들었다.
- 플레이 횟수 :  $N$
- 승리 횟수 :  $M(Z\%)$  이때,  $Z$ 는 소수점을 제외(버림)한 승률이다.
- $N, M$ 이 주어졌을 때,  $Z$ 를 올리기 위한 최소한의 연승횟수를 구하는 프로그램을 작성하라.

# Design

## 문제2. 승률 올리기

- $N, M$ 이 주어졌을 때,  $Z$ 를 올리기 위한 최소한의 연승횟수를 구하는 프로그램을 작성하라.

```
# use for loop
N = 1000
M = 500
Z = int(100 * M / N)

...

print ( "최소한 {}번의 연승이 필요합니다.".format(n) )
```

```
# use mathematics
```

# Design

## 문제2. 승률 올리기

- $N, M$ 이 주어졌을 때,  $Z$ 를 올리기 위한 최소한의 연승횟수를 구하는 프로그램을 작성하라.

$$100 \times \frac{M}{N} = Z + \alpha \quad (0 \leq \alpha < 1)$$

$$100 \times \frac{M + n}{N + n} \geq Z + 1$$

$$100 \times (M + n) \geq (Z + 1) \cdot (N + n)$$

$$100M + 100n \geq (Z + 1) \cdot N + (Z + 1) \cdot n$$

$$(99 - Z) \cdot n \geq (Z + 1) \cdot N - 100M$$

$$n \geq \frac{(Z + 1) \cdot N - 100M}{99 - Z}$$

# Design

## 문제2. 승률 올리기

- $N, M$ 이 주어졌을 때,  $Z$ 를 올리기 위한 최소한의 연승횟수를 구하는 프로그램을 작성하라.
- for loop를 사용한 경우와 수학적 접근 방식을 사용한 두 경우를 각각 함수로 작성하라.

```
# use for loop
def howManyWinsLoop(M, N):
    ...
    return n

# use mathematics
def howManyWinsMath(M, N):
    ...
    return n
```

# Design

## 문제2. 승률 올리기

- 다음의 경우에 대해 두 함수의 수행 시간을 비교해보자.
  - CASE 1:  $M=8, N=10$
  - CASE 2:  $M=47, N=47$
  - CASE 3:  $M=0, N=99000$
  - CASE 4:  $M=470\_000\_000, N=1\_000\_000\_000$

```
# use for loop
%timeit howManyWinsLoop(8, 10)
```

```
# use mathematics
%timeit howManyWinsMath(8, 10)
```

# Design

## 문제3. 규몬 수학 영어 채점 아르바이트

- 당신은 규몬 수학 영어 채점 아르바이트를 하고 있다. 규몬 수학 영어는 초등학교 저학년 학생들을 대상으로 수학과 영어를 동시에 가르치려는 목적의 학습지이며, 당신이 하는 일은 **사칙연산 문제 학습지를 채점**하는 일이다.
- 다른 유사 학습지와 다르게 규몬 수학 영어는 사칙 연산 문제를 독특하게 낸다.
- 가령 " $1 + 2 =$ "와 같은 문제의 경우 숫자를 영문으로 풀어 씀으로써 " $one + two =$ "로 문제가 출제된다. 또한 답 역시 아라비아 숫자가 아닌 영문으로 풀어 쓴 수로 답을 적어야 한다. 즉, 위의 예시의 경우 "three" 라고 답해야 한다.

# Design

## 문제3. 규몬 수학 영어 채점 아르바이트

- 처음으로 채점할 답지를 받은 당신은 채점을 할 때 답안에 오자가 있을 경우 가차 없이 이를 오답으로 채점했다. 예를 들어 "four"를 "fuor" 로 잘못 쓴 것을 오답으로 채점하였다.
- 하지만 채점 결과를 확인한 많은 학부모의 항의로 인해 당신은 **절충**해서 다음과 같이 채점을 하려고 한다. 만약 답이 'seven'일 경우 적은 알파벳의 문자의 수가 동일하고(여기서는 's' 1개, 'e' 2개, 'v' 1개, 'n' 1개), 순서가 뒤섞여 있는 경우까지는 정답으로 간주하기로 했다.



# Design

## 문제3. 규몬 수학 영어 채점 아르바이트

- 조건

- 입력된 수식에 대해 답이 올바르게 기재 되어 있으면 “Yes”를, 그렇지 않을 경우에는 “No”를 출력한다. 만약 연산의 결과가 0보다 작거나 10보다 클 경우에는 무조건 오답으로 간주한다.

```
def gradeIt(sentence):  
    ...  
    return "Yes" if correct else "No"
```

# Design

## 문제3. 규몬 수학 영어 채점 아르바이트

- 예시

```
def gradeIt(sentence):  
    ...  
    return "Yes" if correct else "No"  
  
gradeIt("one + two = three")  
> "Yes"  
gradeIt("one + two = treeh")  
> "Yes"  
gradeIt("one + two = thrre")  
> "No"  
gradeIt("ten + two = three")  
> "No"
```

# List

## List의 중요성

- 데이터를 주로 테이블(엑셀 시트) 형태로 처리
- 테이블은 일종의 리스트의 “모음”

	state	color	food	age	height	score
Jane	NY	blue	Steak	30	165	4.6
Niko	TX	green	Lamb	2	70	8.3
Aaron	FL	red	Mango	12	120	9.0
Penelope	AL	white	Apple	4	80	3.3
Dean	AK	gray	Cheese	32	180	1.8
Christina	TX	black	Melon	33	172	9.5
Cornelia	TX	red	Beans	69	150	2.2

# List

## List 자료형 복습

- List 선언 : `[element1, ... ]` # 대괄호 사용
- List에 원소 추가하기 : `LIST.append(element)`
  - in-place 메소드이므로 리스트를 먼저 선언해야 함
- 두 List 합치기 : `LIST1.extend(LIST2)`
- List 안의 원소 정렬하기
  - `LIST.sort()` # in-place, reverse parameter
  - `sorted()` # # reverse parameter

# List

## List 자료형 활용

- *enumerate(LIST)* # index parameter
- List에 있는 원소들을 순회하면 연산하기
  - index를 이용하는 접근 방식
  - index를 사용하지 않는 접근 방식

	0	1	2	3	4	5	6
<i>colors</i>	"yellow"	"white"	"red"	"plum"	"pink"	"blue"	"black"

# List

## List 자료형 활용

- List에 있는 원소들을 순회하면 연산하기
  - index를 이용하는 접근 방식
  - index를 사용하지 않는 접근 방식

```
[ ] for i in range(0, len(colors)):  
    print (colors[i])
```

```
[> yellow  
white  
red  
plum  
pink  
green  
blue  
black
```

```
[ ] for color in colors:  
    print (color)
```

```
[> yellow  
white  
red  
plum  
pink  
green  
blue  
black
```

# List

## List 자료형 활용

- `enumerate(LIST)` # index parameter
  - `enumerate()` 함수를 이용하면 인덱스를 Python스럽게 사용할 수 있다.

```
[ ] for i, color in enumerate(colors):  
    print ("{}: {}".format(i, color))
```

```
[ ] 0: yellow  
    1: white  
    2: red  
    3: plum  
    4: pink  
    5: green  
    6: blue  
    7: black
```

# List

## List 자료형 활용

- `enumerate(LIST)` # index parameter
- 실습
  - 문자열을 담고 있는 리스트 `colors` 에서 “p”로 시작하는 원소들의 index를 출력하라.

```
colors = ['red', 'blue', 'green', 'black', 'white']
colors.append("pink")
colors.extend(["yellow", "plum"])

res = []
for i, color in enumerate(colors):
    ...

print (res)
> [5, 7]
```



# List

## List 자료형 활용

- 실습

- 색깔에 대한 선호도를 조사하여 *colors*와 *counts*의 두 리스트에 동일한 순서로 저장하였다. (즉, “yellow”를 좋아한다고 응답한 사람은 54명) 두 리스트를 선언하고 색깔별로 좋아하는 사람의 숫자를 출력하는 코드를 작성하여라.

<i>colors</i>	“yellow”	“white”	“red”	“plum”	“pink”	“blue”	“black”
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
<i>counts</i>	54	31	100	87	32	98	5

# List

## List 자료형 활용

- 실습

- 색깔에 대한 선호도를 조사하여 *colors*와 *counts*의 두 리스트에 동일한 순서로 저장하였다. (즉, “yellow”를 좋아한다고 응답한 사람은 54명) 두 리스트를 선언하고 색깔별로 좋아하는 사람의 숫자를 출력하는 코드를 작성하여라.

```
[3] colors = ["yellow", "white", "red", "plum", "pink", "blue", "black"]
    counts = [54, 31, 100, 87, 32, 98, 5]

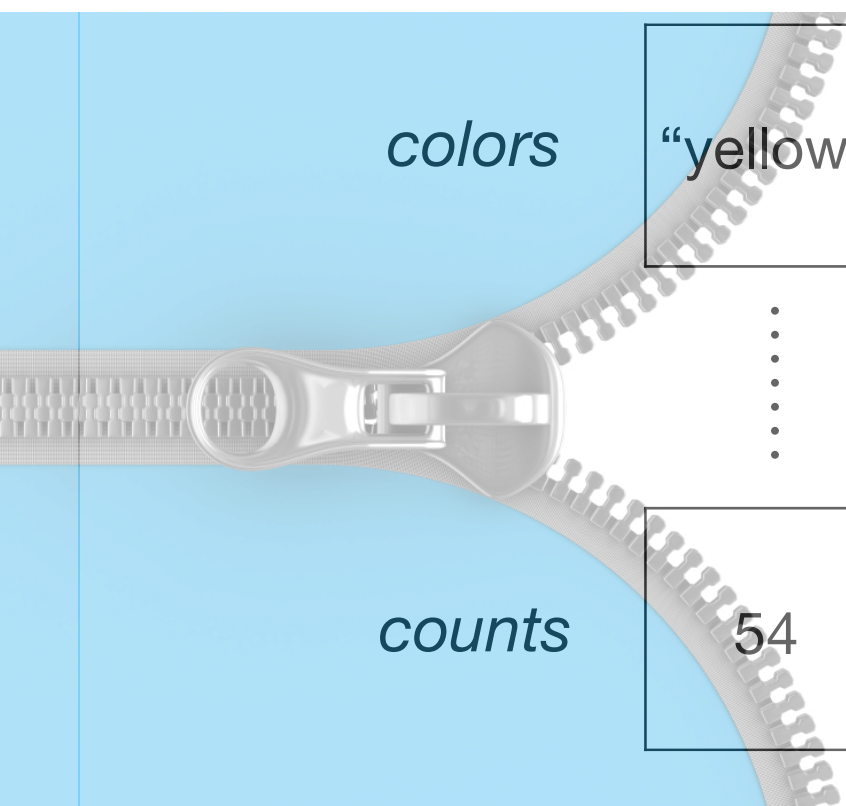
    ...
```

```
[> 색깔 yellow을 좋아하는 사람은 54명입니다.
    색깔 white을 좋아하는 사람은 31명입니다.
    색깔 red을 좋아하는 사람은 100명입니다.
    색깔 plum을 좋아하는 사람은 87명입니다.
    색깔 pink을 좋아하는 사람은 32명입니다.
    색깔 blue을 좋아하는 사람은 98명입니다.
    색깔 black을 좋아하는 사람은 5명입니다.
```

# List

## List 자료형 활용

- *zip(\*iterable)*
  - 2개 이상의 순회형 자료를 받아 순서대로 묶어주기
- 실습
  - 앞의 예제를 *zip()* 함수를 이용해 다시 작성해보자



<i>colors</i>	"yellow"	"white"	"red"	"plum"	"pink"	"blue"	"black"
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
<i>counts</i>	54	31	100	87	32	98	5

# List

## map 활용하기

- *map(function, iterable)*
- 함수(f)와 순회형(iterable) 자료형을 입력으로 받아 iterable의 원소 각각에 함수 f를 적용한 결과를 반환한다.
- 실습 - map을 사용하지 않았을 때
  - 다음과 같이 숫자(1, 3)와 문자열로 작성된 숫자("2")가 공존하는 리스트를 받아 리스트 안의 모든 자료를 숫자로 변환하는 프로그램을 작성해보자.

```
aList = [1, "2", 3, "4", "5"]  
...  
bList = [1, 2, 3, 4, 5]
```

# List

## map 활용하기

- *map(function, iterable)*
- 실습
  - 다음과 같이 숫자(1, 3)와 문자열로 작성된 숫자("2")가 공존하는 리스트를 받아 리스트 안의 모든 자료를 숫자로 변환하는 프로그램을 작성해보자.

```
aList = [1, "2", 3, "4", "5"]  
# 개별 원소별 변환  
def toNumeric(element):  
    if type(element) == str:  
        return int(element)  
    elif type(element) == int:  
        return element  
    else:  
        return None  
  
list(map(toNumeric, aList))
```

# List

## map 활용하기

- *map(function, iterable)*
- 문제
  - 문자열을 입력받아 문자열의 소문자는 대문자로, 대문자는 소문자로 변환하는 코드를 map()을 이용해 작성해보자.

```
my_string = "TeSt"
list(map(..., my_string))
> ['t', 'E', 's', 't']
list(map(..., "tEst function"))
> ['T', 'e', 's', 'T', ' ', 'f', 'u', 'n', 'c', 't', 'i', 'o', 'n']
```

# List

## filter 활용하기

- *filter(f, iterable)*
- 함수(*f*)와 순회형(*iterable*) 자료형을 입력으로 받아 *iterable*의 원소 중에 함수 *f*의 결과가 참(*True*)인 원소만 반환한다.
- 실습 - **filter**를 사용하지 않았을 때
  - 다음과 같이 임의의 숫자 리스트를 생성하고 0보다 큰 숫자만 출력하는 코드를 작성해보자.

```
import random

rList = random.sample(range(-100,101), 10)
print (rList)
> [-43, -87, -1, 16, -51, 24, -20, 80, -82, -11]
```

# List

## filter 활용하기

- *filter(f, iterable)*
- 실습
  - 다음과 같이 임의의 숫자 리스트를 생성하고 0보다 큰 숫자만 출력하는 코드를 작성해보자.

```
import random

rList = random.sample(range(-100,101), 10)
print (rList)
> [-43, -87, -1, 16, -51, 24, -20, 80, -82, -11]
list(filter(lambda element: element > 0, rList))
```



# List

## filter 활용하기

- *filter(f, iterable)*
- 문제
  - 0 이상 200 이하의 숫자 중 임의로 20개를 뽑아 리스트에 저장하고 이 리스트 중에서 3의 배수만 뽑아내는 코드를 *filter()*를 이용해 작성해라.

# Design

## 성능에 대한 고민

- 파이썬은 속도가 느린 편에 속하는 언어이다.
- 그렇다고 하여도 만약 내 코드가 **매우 느리다**면, 다른 방법을 생각해볼 필요
  - 자료형?
  - 연산 방법?

```
import random
pool = range(0, 10_000_000)
sample1 = random.sample(pool, 50_000)
sample2 = random.sample(pool, 50_000)

sample_1L = list(sample1)
sample_2L = list(sample2)
sample_1T = tuple(sample1)
sample_2T = tuple(sample2)
sample_1S = set(sample1)
sample_2S = set(sample2)
```

# Design

## 성능에 대한 고민

- 실습

- 각각 5만개의 숫자를 포함하는 두 리스트에 공통으로 존재하는 숫자의 개수를 세어보자.
  - 자료형: 리스트, 튜플, 셋
  - 연산: in, search

```
%%time

# 후보 1: List, in
count = 0
for e1 in sample_1L:
    if e1 in sample_2L:
        count += 1
else:
    print ("{:},}개의 공통 원소가 있습
```

```
%%time

# 후보 2: List, search
count = 0
for e1 in sample_1L:
    for e2 in sample_2L:
        if e1 == e2:
            count += 1
            break
else:
    print ("{:},}개의 공통 원소가 있습니다."
```

# Numpy

## 보다 편리한 array 연산

- numpy에는 array 연산을 더 편하게 할 수 있는 메소드가 구현되어 있다
- 향상된 성능
  - list vs. numpy.array

```
import numpy as np

my_array = np.arange(1000)
my_list = list(range(1000))

print (type(my_array), type(my_list))
```

```
%timeit [i**2 for i in my_list]
```

```
%timeit my_array**2
```

# Numpy

## 보다 편리한 array 연산

- numpy에는 array 연산을 더 편하게 할 수 있는 메소드가 구현되어 있다
- 깔끔한 연산: between arrays
  - *numpy.add()*, *numpy.subtract()*, *numpy.multiply()*, *numpy.divide()*

*array1*

3	5	-2	9	1	0	-11
---	---	----	---	---	---	-----

$+$ ,  $-$ ,  $\times$ ,  $\div$

*array2*

54	3	10	7	2	9	5
----	---	----	---	---	---	---