

# Python\_Advanced

류영표

# Python\_advanced

파이썬 내장함수

fluent python

class

# 파이썬 내장함수

# 파이썬 내장함수

- input
- len
- abs
- range
- max/min
- enumerate
- zip
- lambda
- map

# 파이썬 내장함수

- 내장함수 기본제공 함수, 모듈을 임포트하지 않아도 사용할 수 있는 함수들

<b>abs()</b>	<b>dict ()</b>	<b>help ()</b>	<b>min ()</b>	<b>setattr()</b>
all()	<b>dir()</b>	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	<b>enumerate()</b>	<b>input()</b>	oct()	staticmethod
bin()	eval()	<b>int()</b>	<b>open()</b>	<b>str()</b>
<b>bool()</b>	exec()	isinstance()	ord()	<b>sum()</b>
bytearray()	filter()	issubclass()	pow()	super()
bytes()	<b>float()</b>	iter()	<b>print()</b>	<b>tuple()</b>
callable()	format()	<b>len()</b>	property()	<b>type()</b>
chr()	frozenset()	list()	<b>range()</b>	vars()
classmethod()	getattr()	locals()	repr()	<b>zip()</b>
compile()	globals()	<b>map()</b>	reversed()	<b>import()</b>
complex()	hasattr()	<b>max()</b>	round()	
delattr()	hash()	memoryview()	<b>set()</b>	

# 파이썬 내장함수

## 1. input( ) : 외부로부터 입력 받아오기

```
a = input()
```

...

```
[2] print(a)
```

내장함수 알아보기

```
[3] b = input()
```

352

```
[4] print('b의 타입은 {}'.format(type(b)))
```

b의 타입은 <class 'str'>

→사용자가 직접 입력할 수 있음

→입력받은 input은 string

# 파이썬 내장함수

## 2. len( ) : 자료형의 길이 재기

```
[2] print(a)
```

☞ 내장함수 알아보기

```
[3] b = input()
```

☞ 352

```
[8] len(a)
```

☞ 9

```
[9] len(b)
```

☞ 3

## 3. abs( ) : 절대값

```
[10] c = -32
```

```
[11] c
```

☞ -32

```
[12] abs(c)
```

☞ 32

# 파이썬 내장함수

## 4. range( ) : 범위 지정하기

```
[16] for x in range(3) :  
      print(x+1)
```

```
↳ 1  
   2  
   3
```

## 5. max( ) / min( ) : 최대값 / 최소값

```
[17] d = [1, 34, 74, 22, 10, 5820, -123, -47]
```

```
[18] max(d)
```

```
↳ 5820
```

```
[19] min(d)
```

```
↳ -123
```



# 파이썬 내장함수

**pop quiz!** 문제를 먼저 풀어보자. 여기는 동네 유명한 빵집이다. 사람들에게 먼저 온 순서대로 번호표를 나누어주려고 한다. 번호표를 나누어주는 함수를 작성해보자.

- 함수는 사람이름으로 되어 있는 리스트를 받아서 "대기번호 x번 : 사람 이름" 를 화면에 출력하고 (번호표, 사람이름)을 원소로 이루어진 리스트를 반환한다.

- input : 리스트
- output : 리스트

```
▶ people = ['펑수', '뿌로로', '똥딱이', '텔레토비']
```

```
▶ # func1을 만들어주세요
```

```
▶ lines = func1(people)
```

```
대기번호 1번 : 펑수  
대기번호 2번 : 뿌로로  
대기번호 3번 : 똥딱이  
대기번호 4번 : 텔레토비
```

# 파이썬 내장함수

```
[ ] people = ['펍수', '뽀로로', '뚝딱이', '텔레토비']
```

```
[ ] def func1(line) :  
    new_lines = []  
    i = 1  
    for x in line :  
        print("대기번호 %d번 : %s" % (i, x))  
        new_lines.append((i, x))  
        i += 1  
    return new_lines
```

→ 대기 번호를 트래킹하는 변수 i

→ 별도로 업데이트 해야함

```
[ ] lines = func1(people)
```



대기번호 1번 : 펍수

대기번호 2번 : 뽀로로

대기번호 3번 : 뚝딱이


대기번호 4번 : 텔레토비

# 파이썬 내장함수

## 6. enumerate

- 반복가능한 객체의 인덱스와 원소에 함께 접근할 수 있는 함수
- tuple(인덱스, 원소)의 형태로 객체를 반환
- 보통 리스트와 함께 쓰임 (tuple도 가능)

```
[ ] lst = ['a', 'b', 'c']  
    for x in enumerate(lst) :  
        print(x)
```




```
(0, 'a')  
(1, 'b')  
(2, 'c')
```

# 파이썬 내장함수

## 6. enumerate


```
[ ] st = 'abcd'

for x in enumerate(st) :
    print(x)
```

 (0, 'a')  
(1, 'b')  
(2, 'c')  
(3, 'd')


```
[ ] dic = {0 : 'p', 1 : 'b' , 2 : 'd'}

for x in enumerate(dic):
    print(x)
```

 (0, 0)  
(1, 1)  
(2, 2)

```
[ ] se = {'a', 'b', 'c'}

for x in enumerate(se) :
    print(x)
```

 (0, 'c')  
(1, 'a')  
(2, 'b')

**\*주의\***

**set은 순서가 없는 자료형**

# 파이썬 내장함수

앞에서 풀었던 퀴즈를 enumerate를 이용해서 다시 작성해보자  
(빵집 문제)

```
[ ] def func1_with_enumerate(line) :  
    new_lines = []  
    for idx, val in enumerate(line):  
        print("대기번호 %d번 : %s" %(idx+1, val))  
        new_lines.append((idx+1, val))  
    return new_lines
```

```
[ ] lines = func1_with_enumerate(people)
```



대기번호 1번 : 펍수  
대기번호 2번 : 뽀로로  
대기번호 3번 : 뚝딱이  
대기번호 4번 : 텔레토비

# 파이썬 내장함수

## 7. zip

- 반복가능한 객체들을(2개 이상) 병렬적으로 묶어주는 함수
- 각 원소들을 튜플의 형식으로 묶어줌

```
[1] str_list = ['one', 'two', 'three', 'four']  
    num_list = [1, 2, 3, 4]  
  
    for i in zip(num_list, str_list) :  
        print(i)
```

```
↳ (1, 'one')  
   (2, 'two')  
   (3, 'three')  
   (4, 'four')
```

```
[2] s1 = '123'  
    s2 = 'abc'  
    s3 = 'ㄱㄴㄷ'  
    list(zip(s1,s2,s3))
```

```
↳ [('1', 'a', 'ㄱ'), ('2', 'b', 'ㄴ'), ('3', 'c', 'ㄷ')]
```

# 파이썬 내장함수

## 8. lambda

- lambda(람다)는 식 형태로 되어있어서 lambda expression(람다 표현식)이라고도 불림
- 람다는 익명의 함수로서 함수를 간편하게 작성할 수 있게 해줌
- python3에서 사용이 권장 되지는 않지만 머신러닝이나 데이터 분석시 많이 사용됨

**lambda** 매개변수 : 리턴 값

# 파이썬 내장함수

## 8. lambda

```
[ ] def plus_two(num) :  
    return num + 2
```

```
[ ] a = 2  
    b = plus_two(a)  
    print(b)
```



4

```
[ ] lambda x : x + 2
```



```
<function __main__.<lambda>(x)>
```

→ 람다는 함수를 생성함

```
[ ] func2 = lambda x : x + 2
```

```
[ ] c = func2(2)
```

```
[ ] c
```



4



# 파이썬 내장함수

## 9. map

- 리스트, 튜플, 스트링 등 자료형 각각의 원소에 동일한 함수를 적용

**map ( 함수, 자료형 )**

```
items = [1, 2, 3, 4, 5]
```

```
squared = []  
for i in items :  
    squared.append(i*i)
```

```
print(squared)
```

```
[1, 4, 9, 16, 25]
```



```
squared_map = list(map(lambda x : x**2, items))
```

```
print(squared_map)
```

```
[1, 4, 9, 16, 25]
```

**map**

# 연습문제

lambda와 map을 이용하여 items의 요소들을 string(문자)로 바꾸는 것을 짜봅시다.

```
▶ items = [1, 24, 3, 6, 7]
```

```
▶ # 정답을 적어주세요
```

```
▶ print(str_items)
```

```
['1', '24', '3', '6', '7']
```

# 연습문제

1~10까지의 정수를 항목으로 갖는 리스트 객체에서 map 함수와 람다식을 이용해  
항목의 제곱 값을 갖는 리스트를 반환하는 프로그램을 작성하십시오.

Fluent python

# Fluent Python

- Formatting
- List comprehension

# Formatting

## 1. %

코드	설명
%d	int
%f	float
%s	string
%c	문자 1개
%%	% 자체

```
[4] num = 1
```

```
[5] print('문자열 포매팅 하기 예시 %d' %num)
```

```
↳ 문자열 포매팅 하기 예시 1
```

```
[6] ch = '이름'
```

```
[7] print('내 이름은 : %s' %ch)
```

```
↳ 내 이름은 : 이름
```

```
[8] fl = 2.566132
```

```
[9] print('반올림 하기 : %0.1f' %fl)
```

```
↳ 반올림 하기 : 2.6
```

# Formatting

## 1. %

코드	설명
%d	int
%f	float
%s	string
%c	문자 1개
%%	% 자체

괄호로 묶어주면 두 개 이상도 포매팅 할 수 있음

```
[10] a = 3  
      b = 5
```

```
[11] print('%d 와 %d 를 더하면 %d' %(a,b,a+b))
```

```
☞ 3 와 5 를 더하면 8
```

# Formatting

## 2. format

‘{ }’.format(포매팅 할 변수)

```
[12] print('문자열 포매팅 하기 예시 {}'.format(2))
```

↳ 문자열 포매팅 하기 예시 2

```
[14] print('내 이름은 : {}'.format(ch))
```

↳ 내 이름은 : 이름



# Formatting

## 2. format

- 괄호로 묶어주면 두 개 이상도 포매팅 할 수 있음

```
a = '류'  
b = '영표'
```

```
print('성은 {}, 이름은 {}'.format(a,b))
```

성은 류, 이름은 영표

- 인덱스를 통해 순서를 지정할 수도 있음

```
print('성은 {0}, 이름은 {1}'.format(a,b))
```

성은 류, 이름은 영표

```
print('Frist name : {1}, Last name : {0}'.format(a,b))
```

Frist name : 영표, Last name : 류

# Formatting

## 2. format

- 이름을 통해서 포매팅 할 수도 있음

```
[20] print('나는 오늘 {음료}를 {개수} 잔이나 마셨다'.format(음료='커피', 개수=3))
```

```
↳ 나는 오늘 커피를 3 잔이나 마셨다
```

- 이 밖에도 정렬이나 공백 이용하는 등 다양한 포매팅이 가능

# Formatting

## 3. f 문자열 포매팅

f '{ }'

```
[28] year = 2020
```

```
[29] f'올해는 {year}'
```

```
↳ '올해는 2020'
```

# Formatting

## 3. f 문자열 포매팅

- 다른 포매팅과는 달리 표현식을 지원함

```
[39] drink = '커피'  
      nums = 3
```

```
[40] f'나는 오늘 {drink}를 {nums + 1} 잔이나 마셨다'
```

```
↳ '나는 오늘 커피를 4 잔이나 마셨다'
```

```
# 문자열 맨 앞에 f를 붙이고, 출력할 변수, 값을 중괄호 안에 넣습니다.  
s = 'coffee'  
n = 5  
result1 = f'저는 {s}를 좋아합니다. 하루 {n}잔 마세요.'  
print(result1)
```

저는 coffee를 좋아합니다. 하루 5잔 마세요.

# Formatting

```
[37] '나는 오늘 %s 를 %d + 1 잔이나 마셨다' %('커피', 3)
```

```
↳ '나는 오늘 커피 를 3 + 1 잔이나 마셨다'
```

```
[38] '나는 오늘 {음료}를 {개수+1} 잔이나 마셨다'.format(음료='커피', 개수=3)
```

```
↳ -----  
KeyError                                Traceback (most recent call last)  
<ipython-input-38-21356a636e64> in <module>()  
----> 1 '나는 오늘 {음료}를 {개수+1} 잔이나 마셨다'.format(음료='커피', 개수=3)  
  
KeyError: '개수+1'
```

SEARCH STACK OVERFLOW

```
[39] drink = '커피'  
     nums = 3
```

```
[40] f'나는 오늘 {drink}를 {nums + 1} 잔이나 마셨다'
```

```
↳ '나는 오늘 커피를 4 잔이나 마셨다'
```

# List comprehension

- 파이썬만의 독특한 문법으로 간결한 코딩을 할 수 있음

## 1) for 문

0부터 9까지를 순서대로 가지고 있는 리스트를 만드세요.

```
[ ] list_1 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
[ ] list_2 = []  
    for x in range(10) :  
        list_2.append(x)  
    print(list_2)
```




```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

# List comprehension

1) for 문

0부터 9까지를 순서대로 가지고 있는 리스트를 만드세요.(한줄로)

```
[ ] lc_1 = [x for x in range(10)]  
    print(lc_1)
```

 [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

[ 저장할 값 **for** 원소 **in** 반복 가능한 객체 ]

# List comprehension

## 1) for 문

Quiz 1) 구구단 2단을 list comprehension을 이용하여 구현하고 리스트를 화면에 출력해보자.

Quiz 2) 다음의 문장을 분석해보자.

"코로나 바이러스를 예방하기 위해 사회적 거리두기를 실천합시다. 마스크를 끼고 손씻기를 생활화합시다." 라는 문장을 띄어쓰기별로 분석하려고 한다. 띄어쓰기별로 문장을 나눈 후 각 요소의 길이를 리스트에 저장하라.

(Hint : 띄어쓰기는 split 함수를 써라.)



# List comprehension

## 2) for 문 + if 문

10부터 20 사이의 숫자들 중에서 짝수만을 담은 리스트를 만들어보자.  
(List Comprehension을 사용하시오.)

```
[33] list_3 = []  
    for x in range(10, 21) :  
        if x % 2 == 0 :  
            list_3.append(x)  
    print(list_3)
```

```
☞ [10, 12, 14, 16, 18, 20]
```

# List comprehension

## 2) for 문 + if 문

10부터 20 사이의 숫자들 중에서 짝수만을 담은 리스트를 만들어보자

```
[34] lc_2 = [x for x in range(10, 21) if x % 2 == 0]
```

```
[35] lc_2
```

```
↳ [10, 12, 14, 16, 18, 20]
```

[ 저장할 값 **for** 원소 **in** 반복 가능한 객체 **if** 조건 ]

# List comprehension

## 2) for 문 + if 문

Quiz 1) 1부터 10의 제곱수 중, 50 이하인 수만 리스트에 저장하라.

Quiz 2) 다음의 문장을 분석해보자.

"코로나 바이러스를 예방하기 위해 사회적 거리두기를 실천합시다. 마스크를 끼고 손씻기를 생활화합시다." 라는 문장을 띄어쓰기별로 분석하려고 한다. 띄어쓰기별로 문장을 나눈 후 각 요소의 길이가 5미만인 것 들만 리스트에 저장하라.

# List comprehension

- 2) for 문 + if 문
- 1부터 10 까지의 숫자들 중 홀수 이면 제곱수를, 짝수 이면 세제곱수 를 담은 리스트를 만들어보자.

```
[40] list_4 = []  
    for x in range(1, 11) :  
        if x % 2 == 1 :  
            list_4.append(x ** 2)  
        else :  
            list_4.append(x ** 3)
```

```
[41] list_4
```

```
↳ [1, 8, 9, 64, 25, 216, 49, 512, 81, 1000]
```

[ 저장할 값 if 조건 else 저장할 값 for 원소 in 반복 가능한 객체 ]

# List comprehension

## 2) for 문 + if 문

Quiz 1) 40 이하의 숫자는 5를 더하고, 40 초과인 숫자는 41로 바꾸어 리스트로 저장하고, 리스트를 출력하라.

Quiz 2) 컷라인이 60점일때, 사람이름과 통과여부를 리스트로 담아서 출력하라. 이름과 통과여부는 튜플로 묶어있는 자료이다.

```
[ ] students = {"보라돌이" : 61,  
                "뚜비" : 35,  
                "나나" : 78,  
                "뽀" : 88}
```

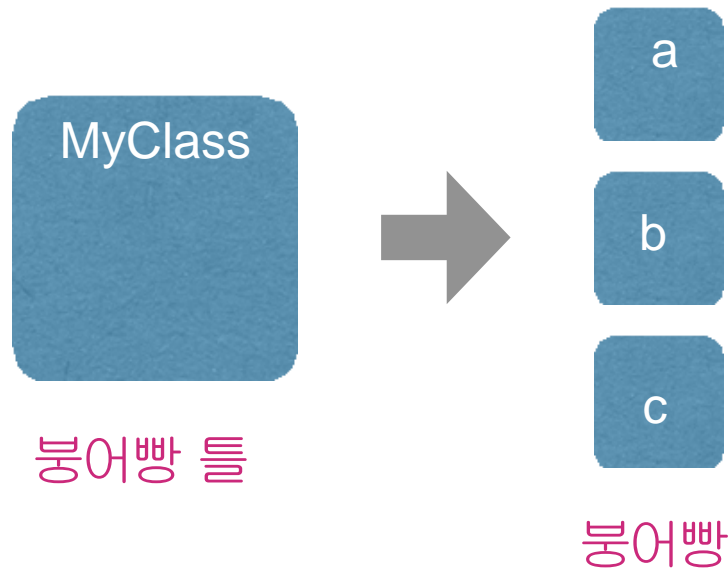
# Class

# Class

- Class 만들기
- instance 이해하기
- 상속
- 오버라이딩

# 클래스(class)

- Class
  - 객체(Object)
  - 클래스(Class) : 객체를 만드는 구조 / 틀
  - 인스턴스(Instance) : 클래스가 실질적으로 객체를 만들었을 때 그 객체를 부르는 용어



a, b, c는 각각 객체

a, b, c는 MyClass의 인스턴스



# 클래스(class)

**class** 클래스 이름():

클래스 변수,

메소드 인스턴스 변수,

메소드 등 클래스의 구조와 내용

```
class 클래스 이름:
    변수1
    변수2
    .....필요한 만큼 변수를 제공.....
    def 메소드1(인수):
        .....메소드의 처리.....
    def 메소드2(인수):
        .....메소드의 처리.....
    필요한만큼 메소드를 제공.....
```

```
class MyClass() :
    class_var = '클래스 변수'

    @classmethod
    def class_method(cls):
        print('클래스의 메소드')

    def instance_method(self):
        self.instance_var = '인스턴스 변수'
        print("인스턴스의 메소드")
```

# 클래스(class)

- 인스턴스(instance)

클래스가 실질적으로 객체를 만들었을 때 그 객체를 부르는 용어

```
[5] class MyClass():  
    pass
```

→ 클래스를 선언

```
▶ a = MyClass()  
  b = MyClass()  
  c = MyClass()
```

→ 함수처럼 콜(call)하여 사용

```
▶ print(a,b,c)
```

```
☞ <__main__.MyClass object at 0x7ff058193eb8>  
   <__main__.MyClass object at 0x7ff058193e80>  
   <__main__.MyClass object at 0x7ff058193ef0>
```

객체(object) 생성 →

# 클래스(class)

- 인스턴스(instance)

클래스가 실질적으로 객체를 만들었을 때 그 객체를 부르는 용어

1. instance variable(인스턴스 변수)

인스턴스 고유의 변수

2. instance method(인스턴스 메소드)

인스턴스에 적용되는 함수

# 클래스(class)

- 인스턴스(instance)

```
[1] class Account():  
    def make_account(self):  
        self.balance = 0  
  
    def deposit(self, money):  
        self.balance += money  
  
    def draw(self, money):  
        self.balance -= money
```

인스턴스

```
[2] a1 = Account()
```

self : 인스턴스를 위한 placeholder

→ 클래스를 선언

# 클래스(class)

- 인스턴스(instance)

```
[1] class Account():  
    def make_account(self) :  
        self.balance = 0  
  
    def deposit(self, money) :  
        self.balance += money  
  
    def draw(self, money) :  
        self.balance -= money
```

```
[2] a1 = Account()  
    a1.make_account()  
    a1.deposit(1000)  
    print(a1.balance)
```

☞ 1000

인스턴스.변수


인스턴스.메소드

→ 위와 같은 방식으로 클래스 선언 시에  
만들었던 변수들, 메소드 들을 사용 할 수 있음

# 클래스(class)

- 인스턴스 메소드(Instance Method)

**\*주의\*** 인스턴스 메소드의 매개변수 중 가장 첫번째는 인스턴스의 자리

C++	Python
<pre>class Person{     int age; public:     Person(){         this-&gt;age = 17;     } }</pre>	<pre> class person():     def __init__(self):         self.age = 17</pre>
	<pre>[51] p1 = person()</pre>
	<pre>[52] p1.age 17</pre>

# 클래스(class)

- 인스턴스 메소드(Instance Method)

**\*주의\*** 인스턴스 메소드의 매개변수 중 가장 첫번째는 인스턴스 의 자리

```
[1] class TestClass1():  
    def __init__(self, in1, in2):  
        self.v1 = in1  
        self.v2 = in2
```

인스턴스

인스턴스 메소드의 매개변수

```
[2] t_cls1 = TestClass1(10, 20)
```

```
[3] t_cls1.v1
```

```
10
```

→ 파이썬은 인스턴스를 자동으로 넘겨줌

# 클래스(class)

- 인스턴스 메소드(Instance Method)

**\*주의\*** 인스턴스 메소드의 매개변수 중 가장 첫번째는 인스턴스의 자리

```
[4] class TestClass2():  
    def __init__(in1, in2):  
        self.v1 = in1  
        self.v2 = in2
```

```
[5] t_cls2 = TestClass2(10, 20)
```

```
-----  
TypeError                                Traceback (most recent call last)  
<ipython-input-5-32e19ab7347c> in <module>()  
----> 1 t_cls2 = TestClass2(10, 20)
```

**TypeError:** \_\_init\_\_() takes 2 positional arguments but 3 were given

SEARCH STACK OVERFLOW



# 클래스(class)

- Instance Method(인스턴스 메소드)

- 1) `__init__`(초기화자): 인스턴스가 생성될 때, 객체의 초기값을 설정하는 메소드로, 자동으로 호출됨
- 2) `__del__`(소멸자): 인스턴스가 소멸될 때, 자동으로 호출되는 메소드

```
[1] class TestClass1():  
    def __init__(self, in1, in2):  
        self.v1 = in1  
        self.v2 = in2
```

```
[2] t_cls1 = TestClass1(10, 20)
```

```
[3] t_cls1.v1
```

```
↳ 10
```

→ 초기화자를 통해서 객체의 초기값을 자동으로 설정

# 클래스(class)

- 인스턴스(instance)

```
[1] class Account():  
    def make_account(self) :  
인스턴스 변수 self.balance = 0  
  
    def deposit(self, money) :  
        self.balance += money  
  
    def draw(self, money) :  
        self.balance -= money
```

인스턴스 메소드

## 인스턴스

```
[2] a1 = Account()  
    a1.make_account()  
    a1.deposit(1000)  
    print(a1.balance)
```



☞ 1000

통장1에 1000원을 넣음

```
[3] a2 = Account()  
    a2.make_account()  
    a2.deposit(5000)  
    print(a2.balance)
```



☞ 5000

통장2에 5000원을 넣음

# 클래스(class)

- 인스턴스(instance)

```
[1] class Account():  
    def make_account(self) :  
인스턴스 변수 self.balance = 0  
  
    def deposit(self, money) :  
        self.balance += money  
  
    def draw(self, money) :  
        self.balance -= money
```

## 클래스 이름



Account.balance

```
-----  
AttributeError                                Traceback (most recent  
<ipython-input-4-35da668b8893> in <module>()  
----> 1 Account.balance  
  
AttributeError: type object 'Account' has no attribute 'balance'
```



→ 통장끼리는 잔고를 공유하지 않음

# 클래스(class)

- 인스턴스(instance)

```
class test():
    name = '아무나'
    age = 0
    def __init__(self, name, age):
        print('생성자 호출!')
        self.name = name
        self.age = age
    def __del__(self):
        print('소멸자 호출!')
    def info(self):
        print('나의 이름은', self.name, '입니다!')
        print('나이는', self.age, '입니다!')
```

```
r = test('류영표', 7)
```

생성자 호출!

```
r.info()
```

나의 이름은 류영표 입니다!  
나이는 7 입니다!

```
[14] test('류영표', 7)
```

생성자 호출!

<\_\_main\_\_.test at 0x7ff089935e80>

```
[15] test.info()
```

-----  
TypeError

Traceback (most recent call last)

<ipython-input-15-a72a0779be47> in <module>()

----> 1 test.info()

TypeError: info() missing 1 required positional argument: 'self'

# 클래스(class)

- 인스턴스(instance)

```
▶ class ClassExample:
    def __init__(self, name, age): # 생성자(Constructor) 생성 시 자동 수행
        self.name = name
        self.age = age + 5 # 변수 생성 후 파라미터로 보낸 값 할당
        print("[생성자] self와 변수 비교 " + str(self.age) + " age : " + str(age))
        print("[생성자] 이름 : " + self.name + " 나이 : " + str(self.age))

    def ten_year_call(self, val): # 함수
        return self.age + 10 # 반환
```

```
[2] a = ClassExample("철수", 20) # 객체화 (ClassExample의 정보를 a에 담음)
print(a.age) # a의 age 출력
print(a.ten_year_call(50)) # a의 ten_year_call 메소드 수행
```

[생성자] self와 변수 비교 25 age : 20

[생성자] 이름 : 철수 나이 : 25

25

35

# 메소드 오버로딩(method overriding)

```
class Adder{
    static int add(int a,int b)
    {
        return a+b;
    }
    static int add(int a,int b,int c)
    {
        return a+b+c;
    }
}
```

자바는 함수이름이 같아도 허용.

```
class Korea:

    def __init__(self, name,population, captial):
        self.name = name
        self.population = population
        self.capital = captial

    def show(self):
        print(
            """
            국가의 이름은 {} 입니다.
            국가의 인구는 {} 입니다.
            국가의 수도는 {} 입니다.
            """.format(self.name, self.population, self.capital)
        )

    def show(self, abc):
        print('abc :', abc)
```

```
>>> from inheritance import *
>>> a = Korea('대한민국',50000000, '서울')
>>> a.show()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: show() missing 1 required positional argument: 'abc'
```

# 클래스(class)

- Class Variable과 Class Method
  1. class variable(클래스 변수)  
클래스와 인스턴스 전체가 공유하는 변수
  2. class method(클래스 메소드)  
클래스와 인스턴스 전체가 공유하는 함수

인스턴스와 상관없이 사용(인스턴스가 없어도 사용 가능)  
클래스와 인스턴스 전체에 공유됨

# 클래스(class)

- Class Variable과 Class Method

1. class variable(클래스 변수)

클래스와 인스턴스 전체가 공유하는 변수

2. class method(클래스 메소드)

클래스와 인스턴스 전체가 공유하는 함수

```
[8] class MyClass2():  
    class_var = '클래스 변수'  
  
    @classmethod  
    def class_method(cls):  
        print('클래스의 메소드')
```

데코레이터(decorator) : wrapping을  
통해 특정 코드를 재사용할 수 있게  
해주는 역할

cls : 클래스를 위한 placeholder  
(self와 유사한 역할)

→ 파이썬은 클래스 또한 자동으로 넘겨줌



# 클래스(class)

데코레이터(decorator)

함수를 수정하지 않은 상태에서 추가 기능을 구현할 때 사용

# 클래스(class)

- Class Variable과 Class Method

```
[8] class MyClass2() :  
    class_var = '클래스 변수'  
  
    @classmethod  
    def class_method(cls):  
        print('클래스의 메소드')
```

인스턴스.변수

클래스이름.변수

인스턴스.메소드

클래스이름.메소드

→ 위와 같은 방식으로 클래스 변수와  
클래스 메소드를 사용할 수 있음

```
[10] e = MyClass2()  
      print(e.class_var)  
      print(MyClass2.class_var)
```

☞ 클래스 변수  
클래스 변수

```
▶ e.class_method()  
   MyClass2.class_method()
```

☞ 클래스의 메소드  
클래스의 메소드

# 클래스(class)

## • Class Variable과 Class Method

[6] `class Account2() :`

```
    bank = '모두은행'  
    total = 0
```

클래스 변수

```
    @classmethod  
    def merge(cls, acc1, acc2) :  
        cls.total = acc1.balance + acc2.balance  
        print("당신의 재산은 %d" %cls.total)
```

```
    def __init__(self) :  
        self.balance = 0
```

클래스 메소드

```
    def deposit(self, money) :  
        self.balance += money
```

```
    def draw(self, money) :  
        self.balance -= money
```



은행  
내 자산

[7] `Account2.bank`

↳ '모두은행'

[8] `b1 = Account2()  
b1.deposit(4000)  
print(b1.balance)  
print(b1.bank)`



↳ 4000  
모두은행

[9] `b2 = Account2()  
b2.deposit(7000)  
print(b2.balance)  
print(b2.bank)`



↳ 7000  
모두은행

[10] `Account2.merge(b1, b2)`

↳ 당신의 재산은 11000

[11] `Account2.total`

↳ 11000

# 클래스(class)

```
[22] class Hotel() :  
      def __init__(self) :  
          self.room = []  
  
      def add_person(self, name):  
          self.room.append(name)
```

```
[23] r1 = Hotel()  
      r2 = Hotel()  
  
      r1.add_person('뽀로로')  
      r2.add_person('펍수')
```

```
[26] Hotel.room
```

```
↳ -----  
AttributeError                                Traceback (most recent call last)  
<ipython-input-26-b87892c63c8a> in <module>()  
----> 1 Hotel.room  
  
AttributeError: type object 'Hotel' has no attribute 'room'
```

# 클래스(class)

```
[27] class GuestHouse() :  
  
    guest = []  
  
    def __init__(self) :  
        self.room = []  
  
    @classmethod  
    def check_in(cls, name):  
        cls.guest.append(name)  
  
    def add_person(self, name):  
        self.check_in(name)  
        self.room.append(name)
```

```
[28] r3 = GuestHouse()  
      r4 = GuestHouse()  
  
      r3.check_in('뚝딱이')  
      r4.check_in('뽕뽕이')
```

```
[29] GuestHouse.guest
```

```
↳ ['뚝딱이', '뽕뽕이']
```

```
[30] r3.room
```

```
↳ []
```

```
[31] r4.room
```

```
↳ []
```

```
[32] r3.add_person('홍길동')  
      r4.add_person('텔레토비')
```

```
[33] r3.room
```

```
↳ ['홍길동']
```

```
[34] r4.room
```

```
↳ ['텔레토비']
```

```
[35] GuestHouse.guest
```

```
↳ ['뚝딱이', '뽕뽕이', '홍길동', '텔레토비']
```

# 클래스(class)

- 클래스 상속(inheritance)
  - 상속: 부모의 클래스를 물려받음
    - 부모 클래스가 가지고 있는 함수/변수를 그대로 사용할 수 있음
  - 상속의 장점: 기존 클래스를 변형하지 않고 추가/변경이 가능함

상속받고자 하는 클래스 (부모 클래스(super class))

```
[ ] class MyPhone2(MyPhone):
```

```
    def has_case(self, val=False):  
        self.case = val
```

: 자식 클래스(sub class)의 새로운 메소드

# 클래스(class)

- 클래스 상속(inheritance)

```
[ ] class MyPhone():  
    def __init__(self, model, color) :  
        self.model = model  
        self.color = color  
  
    def set_name(self, name):  
        self.user = name  
        print("사용자의 이름은 : %s" %self.user)  
  
    def set_number(self, number):  
        self.number = number
```

```
[ ] class MyPhone2(MyPhone):  
    def has_case(self, val=False) :  
        self.case = val
```

```
[48] p2 = MyPhone2('iphone', 'red')
```

```
[49] p2.set_name("MJ")
```

☞ 사용자의 이름은 : MJ

부모 클래스의 메소드 사용 가능

```
[50] p2.has_case(True)
```

```
[51] p2.case
```

☞ True

자식 클래스의 새로운 메소드도 사용 가능

# 클래스(class)

- 메소드 오버라이딩(overriding)
  - 상속을 받은 자식 클래스가 상속해 준 부모 클래스의 메소드를 변형하는 방법
  - 원본(부모 클래스)의 소스코드는 그대로 유지한채 소스코드를 확장, 개인화 할 수 있다는 장점이 있음



# 클래스(Class)

```
class Calculate:
    type = 'low'

    def __init__(self, n1, n2):
        self.n1 = n1
        self.n2 = n2

    def sum(self):
        print(self.n1 + self.n2)
```

```
c = Calculate(4, 2)
print(c.type)
c.sum()
```

low  
6

```
class Calculate_1(Calculate):
    type = 'high'

    def sub(self):
        print(self.n1 - self.n2)

    def mul(self):
        print(self.n1 * self.n2)
```

```
c1 = Calculate_1(4, 2)
print(c1.type)
c1.sum()
c1.sub()
c1.mul()
```

high  
6  
2  
8

# 클래스(class)

- 메소드 오버라이딩(overriding)

```
[46] class MyPhone():
    def __init__(self, model, color) :
        self.model = model
        self.color = color

    def set_name(self, name):
        self.user = name
        print("사용자의 이름은 : %s" %self.user)

    def set_number(self, number):
        self.number = number
```

```
[52] p1 = MyPhone('iphone', 'red')
      p1.set_number("010-xxxx-xxxx")
```

```
[55] class MyPhone3(MyPhone) :
    def set_number(self, num) :
        self.number = num
        print("이 핸드폰의 번호는 : %s" %self.number)
```

```
[56] p3 = MyPhone3('iphone', 'red')
      p3.set_number("010-xxxx-xxxx")
```

☞ 이 핸드폰의 번호는 : 010-xxxx-xxxx

# 연습문제

## 1번

Human 이라는 클래스를 만들어보자.

- Human 클래스는 아래와 같은 특징이 있다.
- Human 클래스는 인스턴스 생성 시 `birth_date`, `sex`, `nation`을 변수로 가지고 있다.
- `give_name`은 인스턴스 메소드로 이름을 input으로 받아서 `name`이라는 인스턴스 변수로 저장하고 화면에 이름을 출력하는 역할을 하는 함수이다.
- `can_sing`이라는 함수는 `True/False`값을 input으로 받으며, 참이면 "Sing a song"을 화면에 출력하는 함수이다.

# 연습문제

## 2번

Human 이라는 클래스를 상속하는 Child라는 클래스를 만들어보자.

- Child의 클래스에는 아래와 같은 변수와 함수들이 추가된다.
- 눈동자 색깔을 나타내는 변수 `eye`를 인스턴스 선언시 사용할 수 있게 추가해보자.
- Child의 클래스는 노래하는 능력이 없다. 따라서 `can_sing`이라는 메소드가 호출되면 무조건 "Can't Sing"이고 출력하도록 바꿔보자.
- Child는 노래 대신 춤을 출 수 있다. `can_dance`라는 메소드가 호출되면 "Dance Time!"을 출력하도록 메소드를 작성해보자.

# 모듈과 패키지

# 모듈과 패키지

- 모듈
- 패키지

# 모듈과 패키지

- 모듈(module)

## 1. 모듈

- 함수나 변수, 클래스 등을 가진 파일 (.py) / jupyter notebook(.ipynb)
- 모듈 안에는 함수, 클래스 또는 변수들이 정의되어 있음
- 파이썬은 많은 표준 라이브러리 모듈을 제공함

```
[ ] import math
```

```
[ ] math.factorial(4)
```

 24

```
[ ] from math import factorial
```

```
[ ] factorial(4)
```

 24

**import** 모듈이름

**from** 모듈이름

**import** 변수/함수/클래스 이름

# 모듈과 패키지

- 모듈(module)

## 2. 패키지(package)

- 모듈을 효율적으로 관리하기 위한 모듈의 상위 개념
- 공동 작업이나 코드의 유지 보수 등에 유리

```
[ ] import 패키지.모듈  
import 패키지.모듈.변수  
import 패키지.모듈.함수  
import 패키지.모듈.클래스
```

```
[ ] from 패키지.모듈 import 변수/함수/클래스
```

