LIKE LION
멋쟁이 사자처럼

파이썬

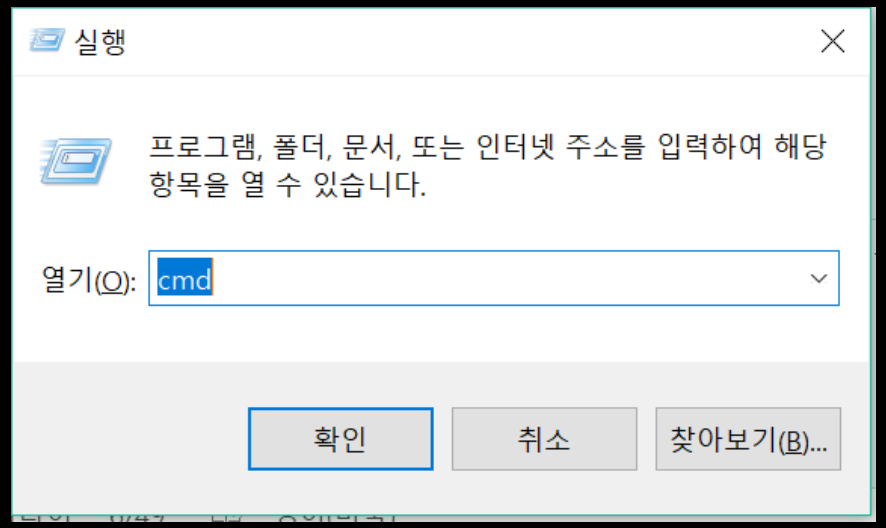
LIKE LION
멋쟁이 사자처럼

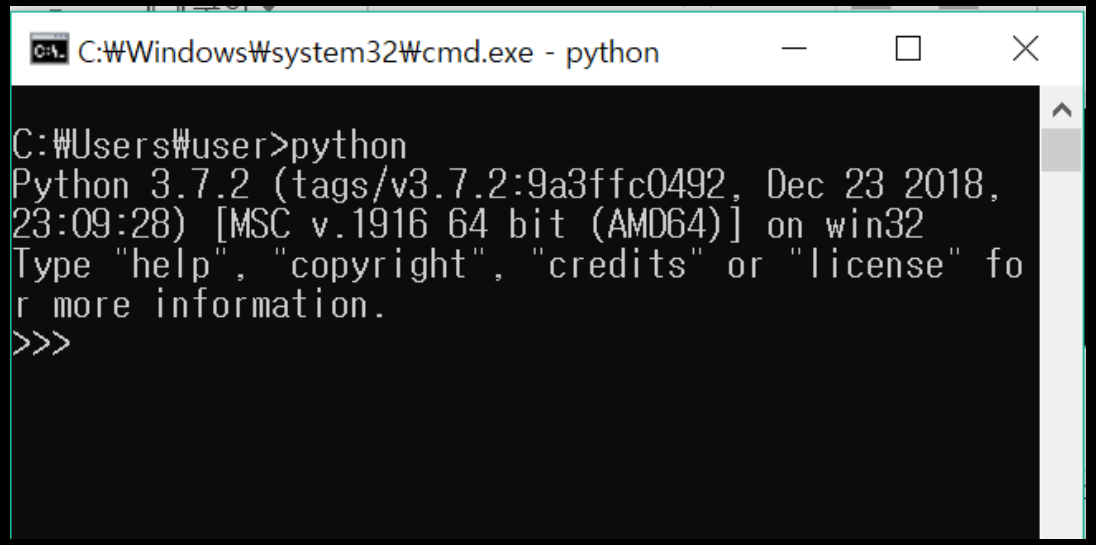
실습환경

파이참같은 걸 쓰셔도 되는데

window+R ->cmd->python

LIKE LION
멋쟁이 사자처럼





LIKE LION
멋쟁이 사자처럼

아니면
visual code

LIKE LION
멋쟁이 사자처럼

차례

LIKE LION
멋쟁이 사자처럼

파이썬은?

프로그래밍언어

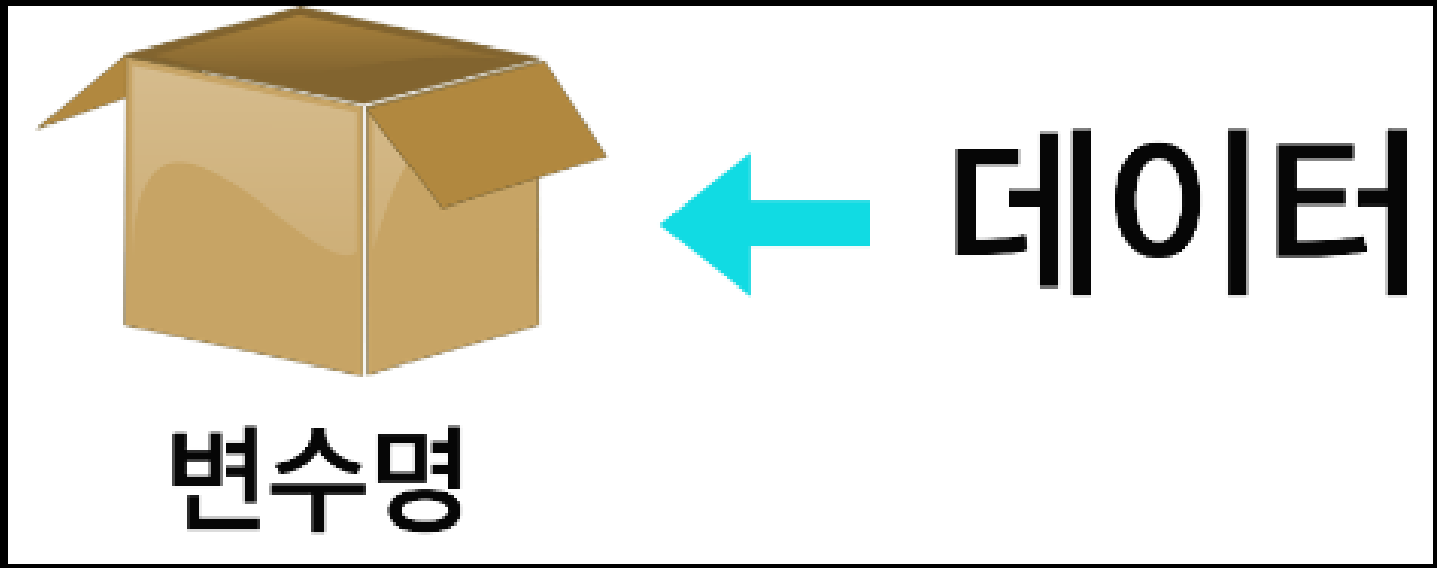
읽고 사용하기 편함

hot한 언어

쉽습니다 (아마도)

LIKE LION
멋쟁이 사자처럼

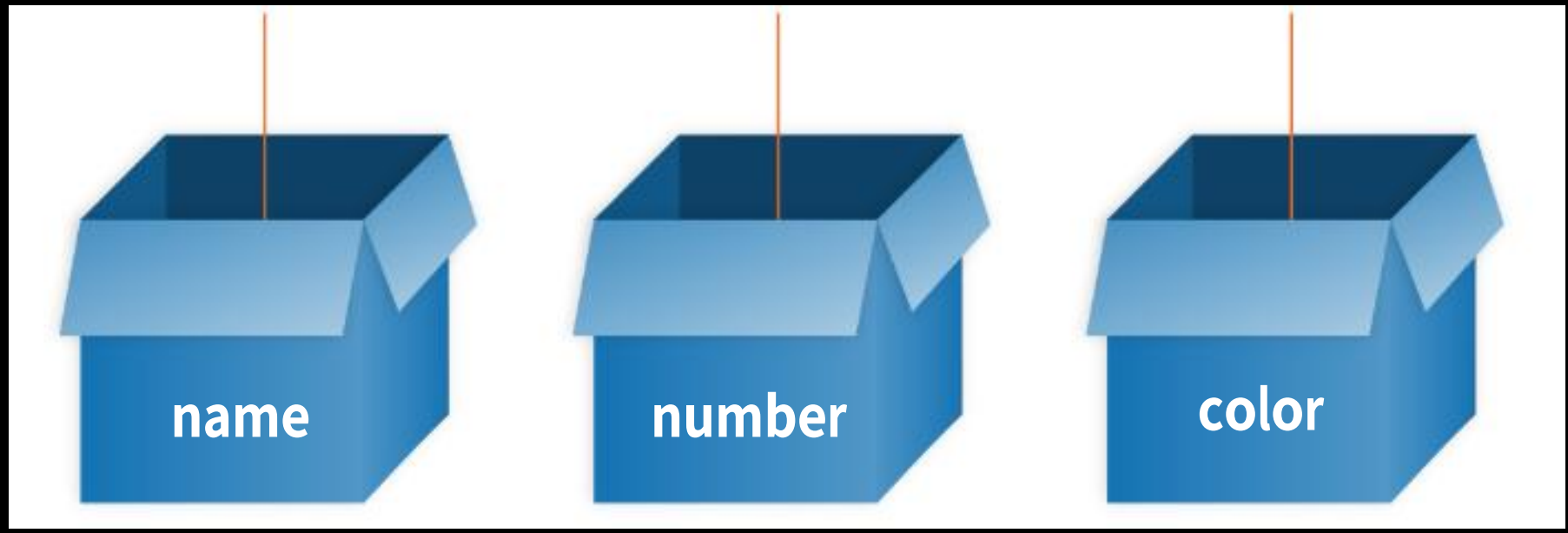
변수란..?



LIKE LION
멋쟁이 사자처럼

데이터 값 “파이썬” 123456 “red”

변수명



LIKE LION
멋쟁이 사자처럼

변수란..?

값을 저장하는 공간의 이름

이름을 정하고 등호(=)
인 할당문을 사용하여
값을 대입하면 자동생성

Name이라는 '문자열' 변수
Number이라는 '정수형' 변수

LIKE LION
멋쟁이 사자처럼

변수는 응용 가능

```
>>> a = 100
>>> b = 11
>>> c = a+b
>>> c
111
```

정수형a와 b의 덧셈

LIKE LION
멋쟁이 사자처럼

```
>>> A = 'like'
>>> B = 'lion'
>>> A+B
'likelion'
```

LIKE LION
멋쟁이 사자처럼

사용자 입출력

```
>>> a=input()
hello world
>>> a
'hello world'
```

Input() 을 이용하여 변수에 입력할 수 있습니다.

```
>>> b= input("어떤 색깔을 좋아하세요?")
어떤 색깔을 좋아하세요?아이보리
>>> b
'아이보리'
```

이렇게 질문의 내용도
추가해서 대화형식으로
사용 가능합니다.

LIKE LION

멋쟁이 사자처럼

```
>>> a='멋쟁이사자처럼'
>>> print(a)
멋쟁이사자처럼
```

출력을 할 땐 print()를
사용해서 출력하시면 됩니다.

LIKE LION
멋쟁이 사자처럼

```
>>> print("hello world")
hello world
```

변수가 아니더라도 바로 출력을
할 수 있어요.

LIKE LION
멋쟁이 사자처럼

```
>>> a= '123'
>>> b='456'
>>> a+b
```

```
>>> a = 123
>>> b = 456
>>> a+b
```

왜 다를까요?

LIKE LION
멋쟁이 사자처럼

자료형

숫자형 문자열 부울형 등등

```
num = 10
num2 = 110.1
str_ing  =  'string'
my_bool = True
```

LIKE LION
멋쟁이 사자처럼

파이썬은 자료형을 미리 선언해줄 필요 X

C언어

```
int num = 10;
```

python

```
num = 10;
```

프로그램 실행시에 자료형을 판단해서 실행해줍니다.

LIKE LION
멋쟁이 사자처럼

숫자형

실수형, 정수형, 복소수형 등등

LIKE LION
멋쟁이 사자처럼

사칙연산
더하기/빼기/곱하기/나누기
+ - * /

```
>>> a=3
>>> b=4
>>> c=a+b
>>> print(c)
7
```

LIKE LION
멋쟁이 사자처럼

연산 우선순위

괄호 > 거듭제곱 > 곱셈, 나눗셈 > 덧셈, 뺄셈

일반적으로 계산하는 순서대로

```
>>> (1+1)*3
6
```

```
>>> 3**2
9
```

```
>>> 1+1*3
4
```

자료형 type 확인

```
>>> type(a)
<class 'int'>
```

문자형

```
>>> a='likelion'
>>> a
'likelion'
```

LIKE LION
멋쟁이 사자처럼

입력은 편한 방법으로

str=‘qwerty’

str=“qwerty”

str=‘”qwerty”’

str=“””qwerty”””

```
>>> str='likelion'
>>> str="likelion"
>>> str='likelion'
>>> str
'likelion'
>>> str2="likelion"
>>> str2
'likelion'
>>> str3='''likelion'''
>>> str3
'likelion'
>>> str4 = """likelion"""
>>> str4
'likelion'
```

LIKE LION
멋쟁이 사자처럼

여러 줄을 입력할 수 있어요.

```
>>> str='''
... 파이썬 너무너무
... 재밌다!'''
```

```
>>> print(str)

파이썬 너무너무
재밌다!
```

LIKE LION
멋쟁이 사자처럼

\n을 쓰면 다음줄로

```
>>> str1='hello₩nworld'
>>> print(str1)
hello
world
```

LIKE LION
멋쟁이 사자처럼

```
>>> a= '123'
>>> b='456'
>>> a+b
```

```
>>> a = 123
>>> b = 456
>>> a+b
```

이제 왜 다른 지 이해하셨나요?

```
>>> a='hello'
>>> b='world'
>>> c=a+b
>>> c
'helloworld'
```

문자열끼리 더하면 연결됩니다.

LIKE LION
멋쟁이 사자처럼

문자열 slice

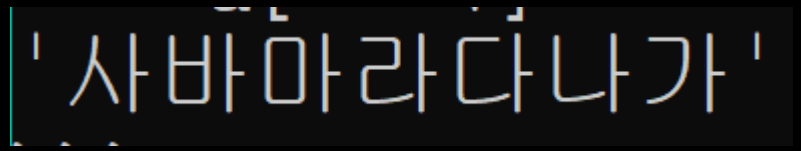
3
a= “라이크 라이온 "
0 1 2 4 5 6

LIKE LION
멋쟁이 사자처럼

```
>>> a='라이크 라이온'
>>> a[0:7]
'라이크 라이온'
>>> a[0:3]
'라이크'
>>> a[2:6]
'크 라이'
```

LIKE LION
멋쟁이 사자처럼

한번 반대로 출력해보세요



LIKE LION
멋쟁이 사자처럼

자료구조

리스트 튜플 딕셔너리

LIKE LION
멋쟁이 사자처럼

list	Tuple	Dictionary
리스트	튜플	딕셔너리
변함	불변	변함
순서있음	순서있음	순서없음
슬라이스[:]	슬라이스[:]	키 : 벨류
[]	()	{}

LIKE LION
멋쟁이 사자처럼

리스트

```
likelion=['temi','sewha','jin','enji']
```

0 1 2 3

```
>>> likelion[0:1]
['temi']
>>> likelion[0:2]
['temi', 'sewha']
>>> likelion[0:]
['temi', 'sewha', 'jin', 'enji']
>>> likelion[3]
'enji'
```

• append()	- 리스트의 끝에 새 요소 추가
• remove()	- 매개변수로 입력한 데이터를 리스트에서 찾아 발견한 첫 번째 요소 제거
• pop()	- 리스트의 마지막 요소 제거 및 리턴
• insert()	- 첨자로 적은 리스트 내의 위치에 새 요소 삽입
• extend()	- 기존 리스트에 다른 리스트를 이어 붙임
• index()	- 리스트 내에서 매개변수로 입력한 데이터와 일치하는 첫 번째 요소 첨자 리턴
• count()	- 매개변수로 입력한 데이터와 일치하는 요소 카운트
• sort()	- 요소 정렬(기본값 : 오름차순)
• reverse()	- 리스트 내 요소의 순서를 반대로 뒤집기
• len()	- 리스트의 길이 리턴

LIKE LION
멋쟁이 사자처럼

리스트 추가

append()

```
>>> likelion.append('dongjin')
>>> likelion
['temi', 'sewha', 'jin', 'enji', 'dongjin']
```

```
>>> likelion.insert(2,'gijin')
>>> likelion
['temi', 'sewha', 'gijin', 'jin', 'enji', 'dongjin']
```

삭제

remove()

```
>>> likelion.remove('dongjin')
>>> likelion
['temi', 'sewha', 'gijin', 'jin', 'enji']
```

LIKE LION

멋쟁이 사자처럼

```
>>> likelion.pop()
'enji'
>>> likelion
['temi', 'sewha', 'gijin', 'jin']
```

리스트의 마지막 요소 제거 및 리턴

```
>>> likelion.extend('changhyun')
>>> likelion
['temi', 'sewha', 'gijin', 'jin', 'c', 'h', 'a', 'n
', 'g', 'h', 'y', 'u', 'n']
```

기존의 리스트에 'changhyun' 리스트 추가

```
>>> likelion.index('jin')
3
```

문자열 jin이 있는 index리턴

```
>>> likelion
['temi', 'sewha', 'gijin', 'jin', 'c', 'h', 'a', 'n
', 'g', 'h', 'y', 'u', 'n']
>>> likelion.sort()
>>> likelion
['a', 'c', 'g', 'gijin', 'h', 'h', 'jin', 'n', 'n',
 'sewha', 'temi', 'u', 'y']
```

오름차순 정렬

LIKE LION
멋쟁이 사자처럼

튜플
tuple

lion = (‘a’,’b’,’c’,’d’)

Index()	튜플내에서 매개변수로 입력한 데이터와 일치하는 첫번째 요소 index 리턴
count()	매개변수로 입력한 데이터와 일치하는 요소 카운트

```
>>> lion.index('a')
0
```

```
>>> lion.count('a')
1
```

LIKE LION
멋쟁이 사자처럼

packing / unpacking

여러 가지 데이터를 튜플로 묶음 -> 패킹

하나의 튜플을 여러가지 데이터로 품 -> 언패킹

```
>>> a= 1,2,3
>>> a
(1, 2, 3)
>>> one, two, three = a
```

```
>>> print(one,two,three)
1 2 3
>>> one
1
>>> two
2
>>> three
3
```

LIKE LION
멋쟁이 사자처럼

딕셔너리
Dictionary

dictionary={‘일’:1,’이’:2}

key : value의 형식

```
>>> dictionary={'일':1,'이':2}
>>> dictionary['일']
1
>>> dictionary['이']
2
```

LIKE LION
멋쟁이 사자처럼

새로운 데이터를 삽입방법
key와 value를 입력

```
>>> dictionary['사']=4
>>> print(dictionary)
{'일': 1, '이': 2, '사': 4}
```

만약 key가 있었다면
새로운 value로 대체

```
>>> dictionary['사']=6
>>> print(dictionary)
{'일': 1, '이': 2, '사': 6}
```

LIKE LION
멋쟁이 사자처럼

```
>>> dictionary={'일':1,'이':2,'사':6}
>>> dictionary.items()
dict_items([('일', 1), ('이', 2), ('사', 6)])
>>> dict(dictionary.items())
{'일': 1, '이': 2, '사': 6}
```

items() – 모든 key와 value를 리턴

```
>>> dictionary.keys()
dict_keys(['일', '이', '사'])
```

keys() - 모든 key를 리턴

```
>>> dictionary.values()
dict_values([1, 2, 6])
```

values() – 모든 value를 리턴

```
>>> dictionary
{'일': 1, '이': 2, '사': 6}
>>> del dictionary['이']
>>> dictionary
{'일': 1, '사': 6}
```

del – dictionary요소 삭제

```
>>> dictionary.pop('일')
1
```

pop() – dictionary 요소 삭제

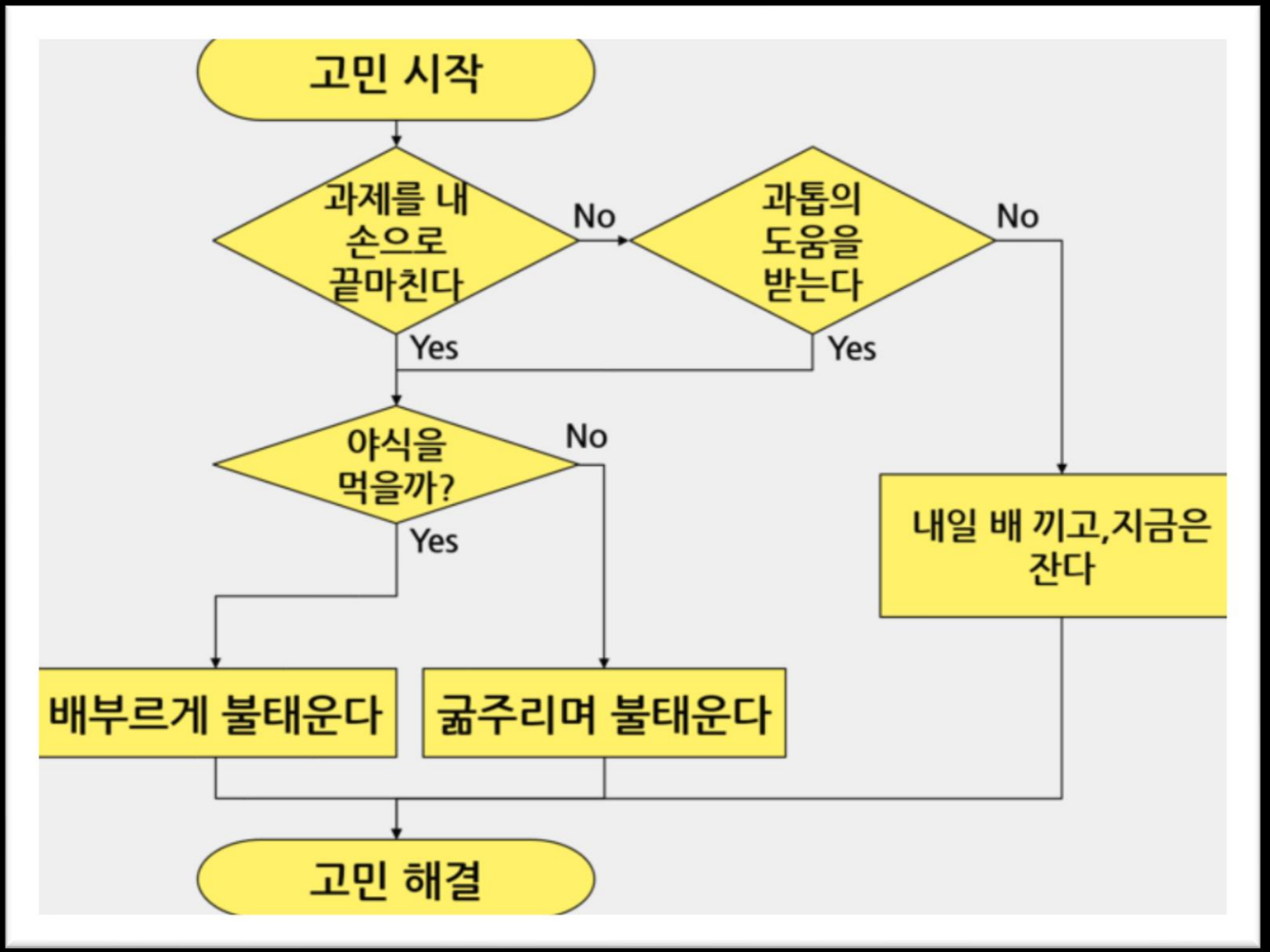
```
>>> dictionary
{'사': 6}
>>> dictionary.clear()
>>> dictionary
{}
```

clear() - dictionary 모든 내용 삭제

LIKE LION
멋쟁이 사자처럼

조건문

LIKE LION
멋쟁이 사자처럼



LIKE LION
멋쟁이 사자처럼

if (배가 고프다) → 먹을 걸 사 먹는다.
else (배가 고프지 않다) → 그냥 잔다.

LIKE LION
멋쟁이 사자처럼

비교연산자

a>b, a<b : 크다, 작다
a>=b, a<=b : 크거나 같다, 작거나 같다
a==b, a!=b : (value) 같다, 같지않다
a is b : (reference) 같다

```
if a>b:
    print('a가 더 크다')

if a>=b:
    print('a가 크거나 같다')

if a==b:
    print('a와 b가 같다')

if a!=b:
    print('a와 b가 다르다')

if a is b:
    print('a와 b가 같다')
```

맞으면 true 아니면 false

LIKE LION
멋쟁이 사자처럼

논리연산자

x and y : 둘 다 참?
x or y : 둘 중 하나 참?
not a : a의 부정 (참->거짓, 거짓->참,
0은 부정 나머지는 참)

```
if a>b and b>c:
    print('둘 다 참이면 출력')

if a>b or b>c:
    print('둘중 하나만 참이여도 출력')

if not a:
    print('a의 부정')
```

LIKE LION
멋쟁이 사자처럼

if , elif, else

```
if (조건문):
    (실행문)

elif (조건문):
    실행문

else:
    실행문
```

첫번째 if

elif

else

순으로 확인

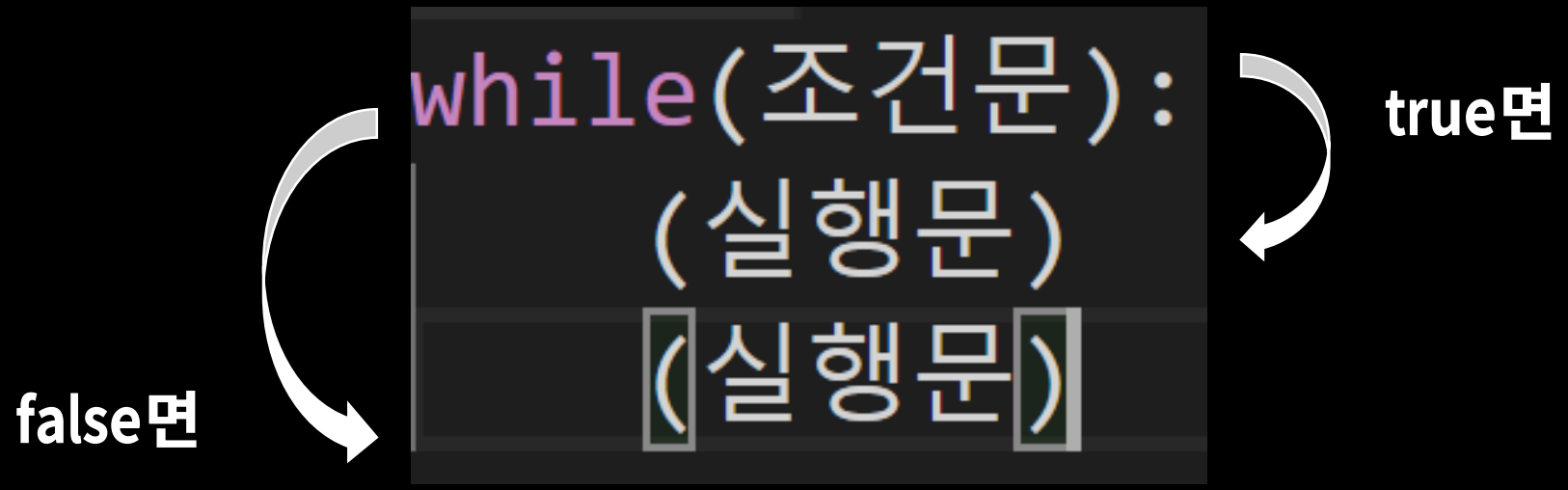
LIKE LION
멋쟁이 사자처럼

반복문
(while, for)

LIKE LION
멋쟁이 사자처럼

While

LIKE LION
멋쟁이 사자처럼



조건에 만족하면 실행문 실행 후 다시 조건문으로
만족하지 않으면 탈출

LIKE LION
멋쟁이 사자처럼

```
a=0
while(a<3):
    print(a)
    a=a+1
```

그러면 이건 출력이
0 1 2 이렇게 되겠죠?

LIKE LION
멋쟁이 사자처럼

```
a=1
while(a<3):
    print(a)
```

이렇게 하면 어떻게 될까요?

LIKE LION
멋쟁이 사자처럼

```
1
1
1
1
1
1
1
1
1
1
1
1
1
```

무한 루프 라고 계속~
그 안에서 무한히 안 끝나요.
이럴땐 ctrl+c누르면 stop

LIKE LION
멋쟁이 사자처럼

```
a=1
while a<5:
    print('print %d번 출력'%a)
    a=a+1
```

```
print('print 1번 출력')
print('print 2번 출력')
print('print 3번 출력')
print('print 4번 출력')
```

어떤 게 더 편할까요?

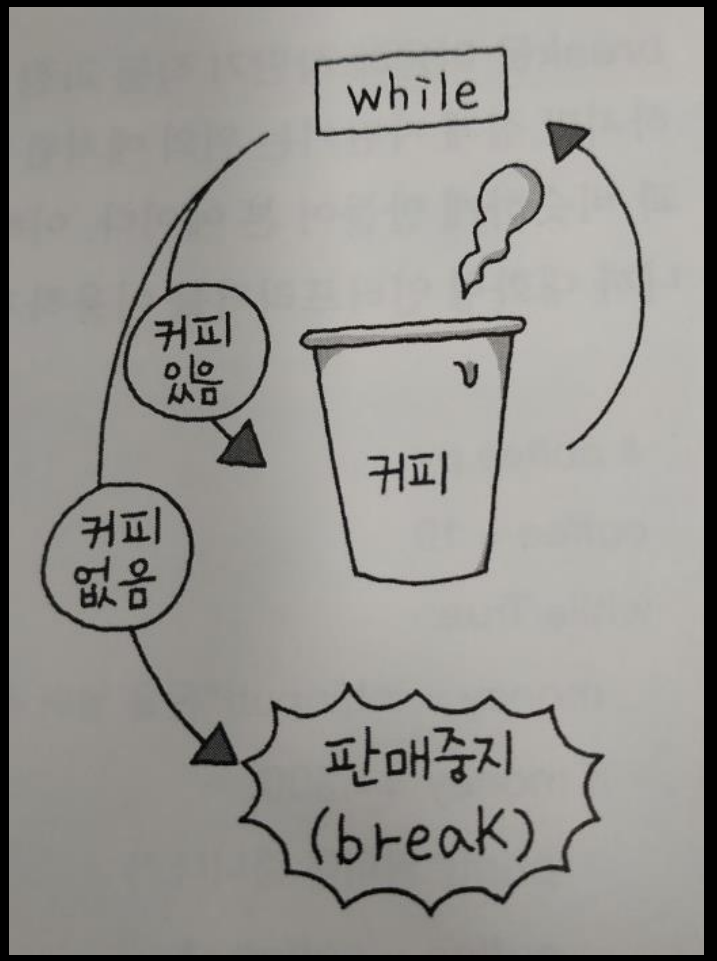
LIKE LION
멋쟁이 사자처럼

break

while문 강제로 빠져나가기

```
coffee = 10
while True:
    order = int(input("커피를 시키시겠습니까(시킨다:1 안 시킨다:2): "))
    if order==1:
        print('커피 나왔습니다')
        coffee=coffee-1
    elif order==0:
        print('커피 안시키셨네요')
        coffee=coffee
    else:
        print('다시')

    if coffee==0:
        print('커피가 다 떨어졌습니다. 판매중지')
        break
```



LIKE LION
멋쟁이 사자처럼

continue

while문의 맨 처음(조건문)으로 다시 돌아가게 만들고 싶은 경우

```
a = 0
while a < 10:
    a = a + 1
    if a % 2 == 0:
        continue
    print(a)
```

```
1
3
5
7
9
```

LIKE LION
멋쟁이 사자처럼

pass

다음줄의 코드 실행

```
a=0
while a<10:
    a=a+1
    if a%2==0:
        pass
    print(a)
```

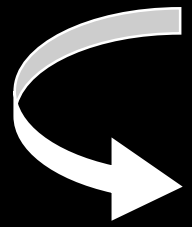
```
1
2
3
4
5
6
7
8
9
10
```

LIKE LION
멋쟁이 사자처럼

for

LIKE LION
멋쟁이 사자처럼

기본 구조



```
for 변수 in 리스트(또는 튜플, 문자열):
    수행할 문장1
    수행할 문장2
```

LIKE LION
멋쟁이 사자처럼

```
test_list = ['one', 'two', 'three']
for i in test_list:
    print(i)
```

```
one
two
three
```

리스트

```
a = (1,2), (3,4), (5,6)
for (first, last) in a:
    print(first + last)
```

```
3
7
11
```

튜플

LIKE LION
멋쟁이 사자처럼

range(a,b) 함수

```
for i in range(1, 11):
    print(i)
```

(a)~(b-1)까지를 범위 지어주는 함수

```
1
2
3
4
5
6
7
8
9
10
```

LIKE LION
멋쟁이 사자처럼

```python
sum = 0
for i in range(1, 11):
    sum = sum + i

print(sum)
```

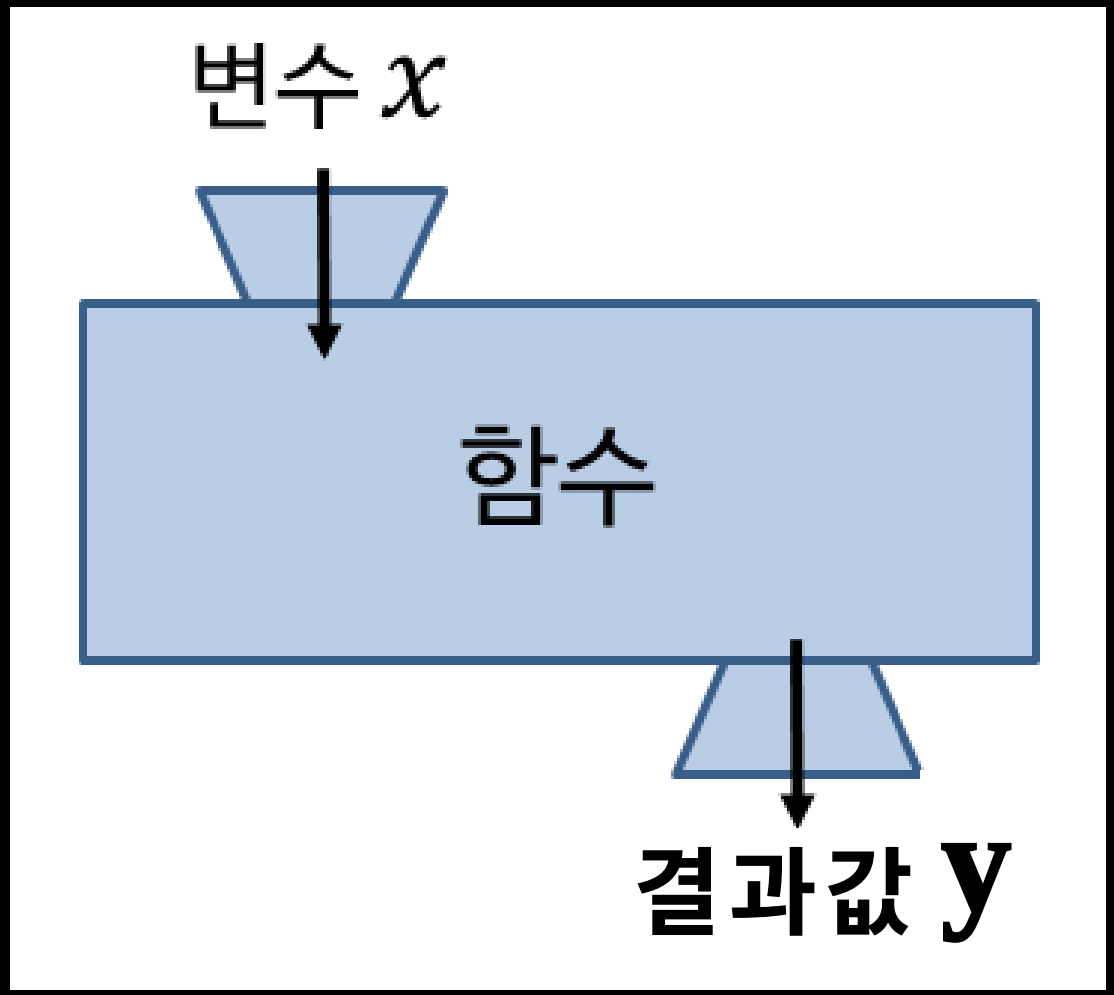
출력

```
55
```

LIKE LION
멋쟁이 사자처럼

함수

특정 기능을 하나로 묶어서 쓰기 위해 사용됩니다


변수 x
함수
결과값 y

일반적인 구조

```
def 함수명(매개변수):
    수행할 문장
    
    return 결과값
```

LIKE LION
멋쟁이 사자처럼

매개변수 - 함수에 입력으로 전달된 값을 받는 변수
인수 -함수를 호출할 때 전달하는 입력값

매개변수

```
def add(a,b):
    return a+b


print(add(1,2))
```

인수

```
def add(a,b):
    print(a+b)


add(1,2)
```

```
def 함수명(매개변수):
    수행할 문장
    결과값을 return 안할수도

def 함수명():
    수행할 문장
    매개변수가 없을수도 있어요.
```

```
def print_name():
    print('김창현')

print_name()
```

입력 값이 몇 개가 될지 모를 때는 어떻게 해야 할까?

```
def 함수이름(*매개변수):
    수행할 문장
```

```python
def add_many(*args):
    result = 0
    for i in args:
        result = result + i
    return result

add_many(1,2,3,4,5,6)
```

*매개변수 쓰면 여러 개 받을 수 있습니다.

LIKE LION
멋쟁이 사자처럼

딕셔너리 형태도 가능?

가능!

```python
def get_kwarg(**kwarg):
    print(kwarg)

get_kwarg(a=1,name='python', sport='baseball')
```

```
{'a': 1, 'name': 'python', 'sport': 'baseball'}
```

이렇게 하면 매개변수를 딕셔너리 형태로 받을 수 있습니다.

LIKE LION
멋쟁이 사자처럼

클래스와 객체

LIKE LION
멋쟁이 사자처럼

클래스

어떤 목적을 위해 필요한 값과 기능들을 묶어 놓은 것

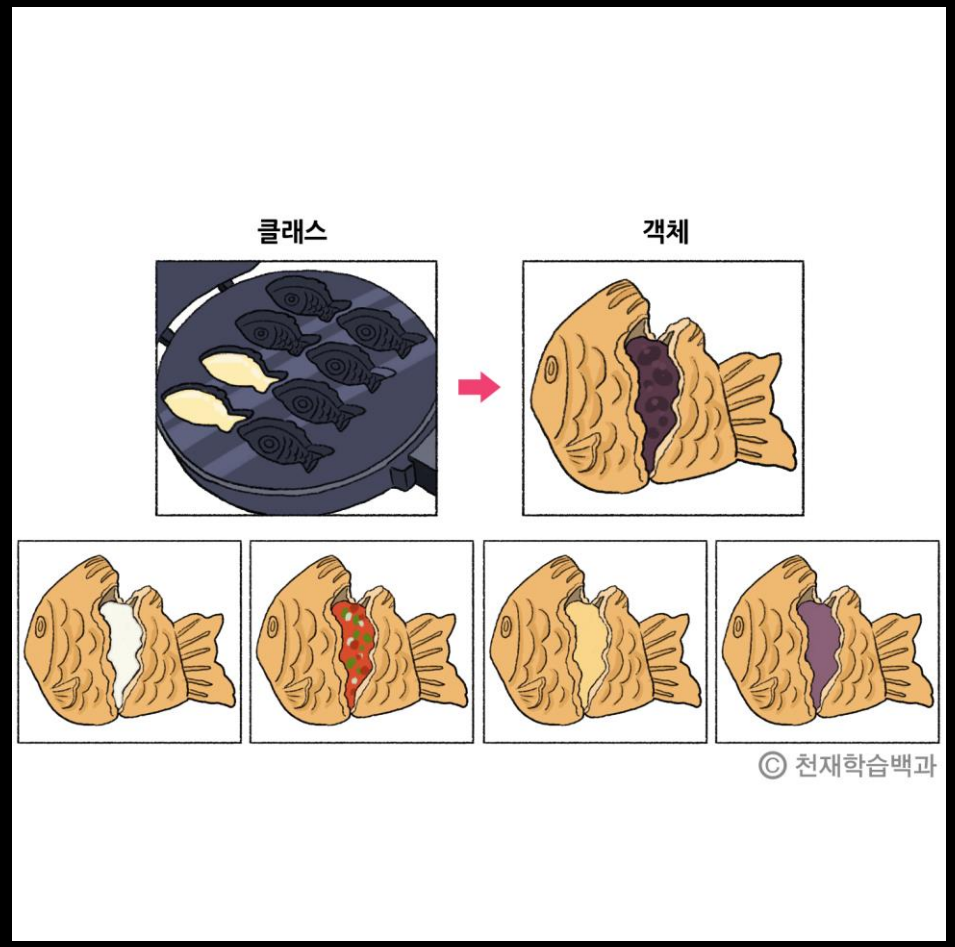
VS

함수

특정 기능을 하나로 묶어서 쓰기 위해 사용됩니다.

클래스 : 똑같은 무엇인가를 계속해서 만들어 낼 수 있는 틀
객체 : 클래스에 의해서 만들어진 결과물

LIKE LION
멋쟁이 사자처럼



붕어빵 틀 = 클래스

붕어빵 = 객체

LIKE LION
멋쟁이 사자처럼

클래스

객체

```python
class 붕어빵틀:
    def __init__(self,내용물):
        self.내용물=내용물

팥빵=붕어빵틀("팥")
슈크림빵=붕어빵틀("슈크림")

print(팥빵.내용물)
print(슈크림빵.내용물)
```

```
팥
슈크림
```

메소드 - 클래스 내에 있는 함수

객체의 중요한 특징

- 객체별로 독립적인 성격을 갖는다.
- 동일한 클래스에 의해 생성된 객체들은 서로에게 영향을 주지 않는다.

LIKE LION
멋쟁이 사자처럼

참고로 클래스 이름은 대문자!

```
class Person:
    def name(self,이름):
        self.이름=이름
        print(이름)

like=Person()
lion=Person()

like.name('창현')
lion.name('likelion')
```

출력

```
창현
likelion
```

객체가 생성된 이상 서로 독립된 객체

LIKE LION
멋쟁이 사자처럼

계산기 클래스

클래스

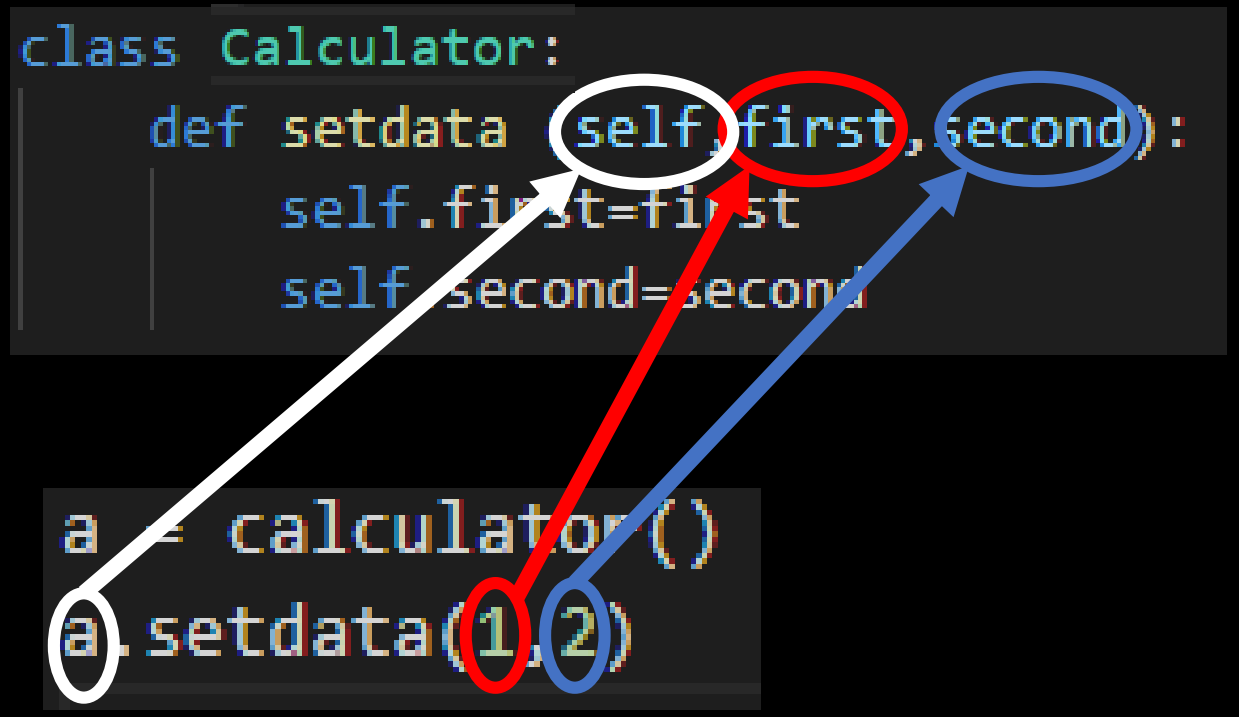
객체

```
class Calculator:
    def setdata (self,first,second):
        self.first=first
        self.second=second

a=calculator()
```

객체 내부에 수를 저장하는 setdata함수 추가.

LIKE LION
멋쟁이 사자처럼



self는 자기 자신을 가리키는 기본 형식

```
class Calculator:
    def setdata (self,first,second):
        self.first=first
        self.second=second

a = calculator()
a.setdata(1)
```

만약 두개를 넣어줘야 하는데 하나만 넣어주면?

LIKE LION
멋쟁이 사자처럼

```
class Calculator:
    def setdata (self,first,second):
        self.first=first
        self.second=second

a = calculator()
a.setdata(1,2)
print(a.first)
print(a.second)
```

```
1
2
```

객체 내의 변수에 접근하는 방법
객체.변수

LIKE LION
멋쟁이 사자처럼

calculator에 덧셈 함수 추가

```
class Calculator:
    def setdata (self,first,second):
        self.first=first
        self.second=second
    def sum (self):
        result=self.first+self.second
        return result

a = calculator()
a.setdata(1,2)

print(a.sum())
```

```
3
```

LIKE LION
멋쟁이 사자처럼

```python
class Calculator:
    def setdata (self,first,second):
        self.first=first
        self.second=second
    def sum (self):
        result=self.first+self.second
        return result
    def mul (self):
        result=self.first*self.second
        return result
    def sub (self):
        result=self.first-self.second
        return result
    def div (self):
        result=self.first/self.second
        return result


a = calculator()
a.setdata(1,2)

print(a.sum())
print(a.mul())
print(a.sub())
print(a.div())
```

계산기 클래스 완성

```
3
2
-1
0.5
```

생성자

LIKE LION
멋쟁이 사자처럼

```python
class Calculator:
    def setdata (self,first,second):
        self.first=first
        self.second=second
    def sum (self):
        result=self.first+self.second
        return result
    def mul (self):
        result=self.first*self.second
        return result
    def sub (self):
        result=self.first-self.second
        return result
    def div (self):
        result=self.first/self.second
        return result

a = calculator()
a.sum()
```

```
Traceback (most recent call last):
  File "c:/Users/user/Desktop/test3.py", line 20, in <module>
    a.sum()
  File "c:/Users/user/Desktop/test3.py", line 6, in sum
    result=self.first+self.second
AttributeError: 'calculator' object has no attribute 'first'
```

객체에 데이터를 입력하지 않고
출력을 하려고 하면

에러가 나겠쥬?

LIKE LION
멋쟁이 사자처럼

그래서

생성자 메소드

```
def __init__(self)
```

LIKE LION
멋쟁이 사자처럼

```python
class Calculator:
    def setdata (self,first,second):
        self.first=first
        self.second=second
    def sum (self):
        result=self.first+self.second
        return result
    def mul (self):
        result=self.first*self.second
        return result
    def sub (self):
        result=self.first-self.second
        return result
    def div (self):
        result=self.first/self.second
        return result

a = calculator()
a.setdata(2,3)
print(a.sum())
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
user@kayden MINGW64 ~
$ C:/Users/user/AppData/Local/Programs/Python/Pyth
5
```

```python
class Calculator::

    def __init__ (self,first,second):
        self.first=first
        self.second=second
    def sum (self):
        result=self.first+self.second
        return result
    def mul (self):
        result=self.first*self.second
        return result
    def sub (self):
        result=self.first-self.second
        return result
    def div (self):
        result=self.first/self.second
        return result

a = calculator(2,3)
print(a.sum())
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
user@kayden MINGW64 ~
$ C:/Users/user/AppData/Local/Programs/Python/Pyth
5
```

```
a = Calculator()
print(a.sum())
```

```
Traceback (most recent call last):
  File "c:/Users/user/Desktop/test3.py", line 18, in <module>
    a = calculator()
TypeError: __init__() missing 2 required positional arguments: 'first' and 'second'
```

생성자를 만들어주고 객체 생성시에
값을 넣어주지 않으면 오류가 납니다.
생성자는 객체 생성시 자동으로 실행되기 때문

객체를 생성할 때 데이터를 넣어주세요.

```
a = Calculator(2,3)
print(a.sum())
```

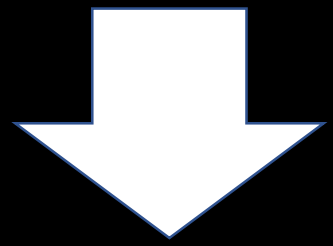
LIKE LION
멋쟁이 사자처럼

상속

자식 클래스가 부모 클래스의 기능을 받아 쓰는 것

LIKE LION
멋쟁이 사자처럼

상속을 통해서 기존 클래스의 기능들을 물려 받을 수 있습니다.



기존 클래스의 기능을 따로 구현하지 않고 바로 사용

LIKE LION
멋쟁이 사자처럼

class 이름(상속하는 클래스명)

```
class A:
    pass

class B(A):
    pass
```

B는 A를 상속받았다.

LIKE LION
멋쟁이 사자처럼

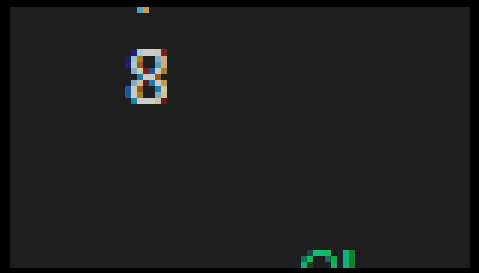
상속을 이용한

calculator에 거듭제곱함수를 추가

LIKE LION
멋쟁이 사자처럼

```python
class Calculator:
    def __init__ (self,first,second):
        self.first=first
        self.second=second
    def sum (self):
        result=self.first+self.second
        return result
    def mul (self):
        result=self.first*self.second
        return result
    def sub (self):
        result=self.first-self.second
        return result
    def div (self):
        result=self.first/self.second
        return result
class Morecalculator(calculator):
    def pow(self):
        result= self.first**self.second
        return result

a =Morecalculator(2,3)
print(a.pow())
```



a는 calculator가 아니고
morecalculator에서 생성

만들어 놓은 객체로 1/0을 해봅시다.

```
a = Morecalculator(2,0)
print(a.div())
```

1을 0으로 나눌 수 없으니 오류가 뜹니다.

```
Traceback (most recent call last):
  File "c:/Users/user/Desktop/test3.py", line 23, in <module>
    print(a.div())
  File "c:/Users/user/Desktop/test3.py", line 15, in div
    result=self.first/self.second
ZeroDivisionError: division by zero
```

LIKE LION
멋쟁이 사자처럼

고치고 싶은데 이미 상속을 받아 놨던 함수

이럴 땐.

LIKE LION
멋쟁이 사자처럼

메소드 오버라이딩

LIKE LION
멋쟁이 사자처럼

부모 클래스로부터 상속받은 메서드의 내용을
변경하는 것을 오버라이딩이라 한다.

이제 0으로 나눌 때는 메시지는 출력해주는 함수로 바꿔봅시다.

LIKE LION
멋쟁이 사자처럼

```python
class Calculator:
    def __init__ (self,first,second):
        self.first=first
        self.second=second
    def div (self):
        result=self.first/self.second
        return result
class Morecalculator(calculator):
    def pow(self):
        result= self.first**self.second
        return result
    def div(self):
        if self.second==0:
            print("0으론 못나눠")
            return 0

        else:
            return self.first/self.second
```

LIKE LION
멋쟁이 사자처럼

다음을 실행하면

```
a = Morecalculator(2,0)
a.div()
```

출력

```
0으론 못나눠
0
```

LIKE LION
멋쟁이 사자처럼

실습

```python
class Person:
    def greeting(self):
        print('안녕하세요.')

class Student(Person):
    def greeting(self):
        print('likelion짱짱.')

james = Student()
james.greeting()
```

출력

```
안녕하세요.
likelion짱짱.
```

LIKE LION
멋쟁이 사자처럼

예외처리

LIKE LION
멋쟁이 사자처럼

try / except

프로그래밍 중 오류 발생시 예외로 별도 처리해주는 것

예외 종류는 엄청 많아요.

https://docs.python.org/3/library/exceptions.html#

나중에 한번 쭉 읽어보시던가.

대표적인 예

NameError
a라는 변수를 찾지 못하여 발생함
ZeroDivisionError
정수를 0으로 나눌 때 발생함
IndexError
리스트의 인덱스범위를 넘어선 참조 시 발생함
TypeError
정수 나누기 문자, type맞지 않는 연산으로 인해 발생함

LIKE LION
멋쟁이 사자처럼

0으로 나누는 예외처리

zerodivisionError

LIKE LION
멋쟁이 사자처럼

```
class Morecalculator(calculator):
    def pow(self):
        result= self.first**self.second
        return result

    def div(self):
        try:
            result=self.first / self.second
            return result
        except ZeroDivisionError:
            print("0으로 나눌 수 없습니다.")
```

```
a = morecalculator(2,0)
a.div()
```

```
0으로 나눌 수 없습니다.
```

try블록을 실행하다가 오류가 발생하면 바로
except 블록으로 이동하여 진행하게 됩니다.
물론 오류가 안 나오면 try만 실행합니다.

LIKE LION
멋쟁이 사자처럼

```python
class Morecalculator(calculator):
    def pow(self):
        result= self.first**self.second
        return result

    def div(self):
        try:
            result=self.first / self.second
            return result
        except ZeroDivisionError as e:
            print(e)


a = morecalculator(2,0)
a.div()
```

```
division by zero
```

오류 변수 e를 사용해서
파이썬 내부에서 어떤 오류로 처리되는지 출력가능

LIKE LION
멋쟁이 사자처럼

끝

LIKE LION
멋쟁이 사자처럼

신나는 과제

키오스크 카페