0128

〈 Operator Overloading 〉

- 연산자의 기능을 수정해서 사용하는 것

: 자바에서 + 는 숫자를 더하는 기능
문자열 + 문자열 ——〉 문자열을 결합

(이 경우가 Overloading 의 대표적인 예)

* python 이나 C++ 에서 사용자가 직접 연산자의 기능을 정의할 수 있다.

특정한 이름을 재정의 시
method 이름 = __연산자이름__

parameter의 개수를 변경할 수는 없다

Ex) + 는 숫자 2개를 더하는 기능
첫번째 데이터는 호출하는 Instance
Parameter 는 인스턴스 자신과 연산을 수행해야하는 Instance 2개

이 경우 Unbound 호출을 많이 사용

```
class Student:
    def __init__(self, name="What is your name"):
        self.__name = name

    def getName(self):
        return self.__name

    def setName(self, name):
        self.__name = name
    name = property(getName, setName)


# Instance 생성

st = Student()
print(st.name)
st.name = "심청이"
print(st.name)
```

```
What is your name
심청이
```

```
st1 = Student()
st1.name = "콩쥐"
print(st1.name)
print(st+st1)
```

```
콩쥐
Traceback (most recent call last):
  File "/Users/mac/PycharmProjects/pythonProject/0128_4th.py", line 24, in <module>
    print(st+st1)
TypeError: unsupported operand type(s) for +: 'Student' and 'Student'
```

```
def __add__(self, other):
    return self.name + other.name
```

```
What is your name
심청이
콩쥐
심청이콩쥐
```

```python
class Student:
    def __init__(self, name="What is your name"):
        self.__name = name

    def getName(self):
        return self.__name

    def setName(self, name):
        self.__name = name
    name = property(getName, setName)

    def __add__(self, other):
        return self.name + other.name


# Instance 생성

st = Student()
print(st.name)
st.name = "심청이"
print(st.name)


st1 = Student()
st1.name = "콩쥐"
print(st1.name)
print(st+st1)
```

〈 Inheritance 〉

하위 클래스가 상위 클래스의 모든 것을 물려 받는 것

(상속을 이용하는 경우)
: 클래스 들에 공통된 내용에 있어 상위 클래스를 지정
다른 클래스가 물려받을 경우
(JPA 의 BaseEntity)

(python에서의 상속)

class 하위클래스이름 (상위 클래스 이름)

```
class BaseClase:
    def greeting(self):
        print("Super class ")

class DerivedClass(BaseClase):
    def study(self):
        print("Sub Class")

# Instance 생성
ins0 = DerivedClass()

# method 호출
ins0.study()
ins0.greeting()
```

Sub Class
Super class

〈 상속을 받았는지 확인 하는 방법 〉

Issubclass(하위클래스이름 , 상위클래스이름)
—〉 method 이름을 보고 return type 을 예측가능해야한다

* method 이름이 is 로 시작하면 Return type 은 무조건 boolean 타입이다

〈 Overriding 〉

상위 클래스에 존재하는 method를 하위 클래스에서 다시 정의
—〉기능을 추가하기 위한 목적으로 사용

상위 클래스의 기능은 수행하고 추가

상위 클래스 기능 호출 시 **Super** 사용

```
class BaseClase:
    def greeting(self):
        print("Super class ")

class DerivedClass(BaseClase):
    def study(self):
        super().greeting()
        print("Sub Class")

# Instance 생성
ins0 = DerivedClass()

# method 호출
ins0.study()
print("------------")
ins0.greeting()
```

```
Super class
Sub Class
------------
Super class
```

〈 Abstract class 〉

자신의 Instance 를 생성할 수 없는 클래스
——〉 상속을 통해서만 사용하는 클래스

만드는 이유 ?

1. 유사한 역할을 하는 class 들의 공통된 method 를 소유
   (이름을 적게 기억하기 위해서)

2. Template method pattern 구현
   (Template method pattern : 이름만 먼저 보여줌)

〈 생성 방법 〉
abc 모듈을 import
——〉 클래스에 metaclass = ABCMeta 지정
——〉 추상 method 위에 @abstractmethod 를 설정

```
class Super(metaclass=ABCMeta):
    @abstractmethod
    def abstractmethod(self):
        pass


class Sub(Super):
    def abstractmethod(self):
        print("Overriding complete")

su = Sub()
su.abstractmethod()
```

————〉 method 재정의 반드시 해야함

Overriding complete

〈 Delegation 〉 - 위임

delegate : 어떤 객체에
이벤트가 발생하면 이벤트 를 처리하는 것은 자신이 해야함
이벤트 처리를 다른 객체에게 넘기는 경우가 있다

—〉 이 경우를 delegate pattern 이라고 한다

Instance 가 클래스에 정의하지 않은 method 를 호출하면 Error
__getter__(self, name) 을 생성하면

Instance 가 클래스에 정의되지 않은 method 를 호출했을 때

처리를 수행 name 은 method 이름

```
class Student:
    def __getattr__(self, name):
        print(name, "this method is not exists")
        return getattr(1, name)

    pass


st = Student()
st.getName()
```

```
getName this method is not exists
Traceback (most recent call last):
  File "/Users/mac/PycharmProjects/pythonProject/0128_4th.py", line 83, in <module>
    st.getName()
  File "/Users/mac/PycharmProjects/pythonProject/0128_4th.py", line 77, in __getattr__
    return getattr(1, name)
AttributeError: 'int' object has no attribute 'getName'
```

〈 Iterator(반복자) 〉

데이터를 순차적으로 접근할 수 있도록 해주는 것

__iter__ 와 __next__ 를 구현해서 생성

for 에 사용할 수 있는 data 는
    iterator 가 구현되어 있어야 한다

__iter__ 가 존재해야한다

for 에
    enumerator(iterator 구현된 객체) 를 이용하면 index 와 값이 tuple로 넘어옴

〈 Generator 〉

iterator 와 특수한 형태
yield를 이용하여 다음 데이터를 리턴

〈 coroutline 〉

서브 루틴을 호출하는 것
    메인 루틴과 Sub 루틴을 순회하면서 수행

일반 함수 호출 —〉 Sub Routine 을 호출하고 return 되며 종료

coroutlin ——〉 양쪽을 계속해서 순회하면수 수행가능

〈 dcforator 〉

@로 시작하는 것
    예약어를 이용해서 파이썬의 코드를 삽입하는 경우

** Module

〈 module 〉
독자적으로 memory를 할당받아서 수행하는 코드 블럭
— 단독으로 존재할 수 없음

Program : 독자적으로 memory를 받아서 일을 수행하고 혼자서 존재

module : 독자적으로 memory를 할당 받지만 단독으로 존재할 수 없음

python 에서는 하나의 file 을 module 로 간주

〈 종류 〉
1. builtin module
: python 에서 제공하는 모듈

2. 사용자 정의 모듈
—〉3rd Party 모듈 : 다른 개발자들이 만든 module

〈 module 가져오기 〉
import module 이름
: 모듈이름을 이용해서 모듈을 사용

from package Name import module 이름
: package 에 나열된 모듈만 가져오는 것
이 경우도 모듈 이름을 이용해서 모듈 내부의 요소를 사용

from package Name import *
: package 의 모든 모듈을 가져오는 것
이 경우에는 모든 모듈을 현재 모듈에 포함
때문에 모듈 내부의 요소를 사용할 때 모듈이름을 기재하지 않아도 됨

import module Name as NickName
: module 의 내용을 별명으로 가져와서 별명으로 이용

Ex)
import numpy as np : numpy 모듈을 np 라는 이름으로 가져와서 사용
import pandas as pd : pandas 모듈을 pd 라는 이름으로 가져와서 사용
——〉 from package Name import module as 별명

```python
# sys module 에 있는 path 라는 data 모임 속성을 접근해서 모든 요소를 출력
import sys
for attr in sys.path:
    print(attr)
```

```
/Users/mac/PycharmProjects/pythonProject
/Users/mac/PycharmProjects/pythonProject
/Library/Frameworks/Python.framework/Versions/3.9/lib/python39.zip
/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9
/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/lib-dynload
/Users/mac/PycharmProjects/pythonProject/venv/lib/python3.9/site-packages
```

〈 package 〉

유사한 역할을 수행하는 module 을 모아놓은 것

directory 와 유사한 개념
directory에 __init__. py 라는 파일이 있으면 package 로 간주

〈 python 실행 파일 〉

directory에 __main__.py 파일을 만들면
    python directory이름을 이용하면 __main__.py 파일이 실행

Zip 파일로 압축이 되어있다면
    directory 이름 대신 zip 파일의 이름을 사용하면 된다

〈 pip 〉

다른 모듈을 설치하는 명령어

Command 혹은 Terminal 창에서 실행

Ex)
Pip install 모듈이름

Pip install 모듈이름 - upgrade

Pip install 모듈이름 = version

Pip install 모듈이름 >= version

—— 〉 pip 자체 upgrade : python -m pip install --upgrade pip

〈현재 설치된 package 목록을 출력 〉
pip list --format= columns

〈 package 이름 검색 〉
pip search package Name

〈 Package 삭제 〉
pip uninstall packageName -y

〈 현재 환경의 package 설정을 저장 〉
pip freeze 〉 파일 경로

〈 복원 〉
pip install -r 파일 경로

〈 package download 〉
pip download package Name

—〉 존재 여부 확인

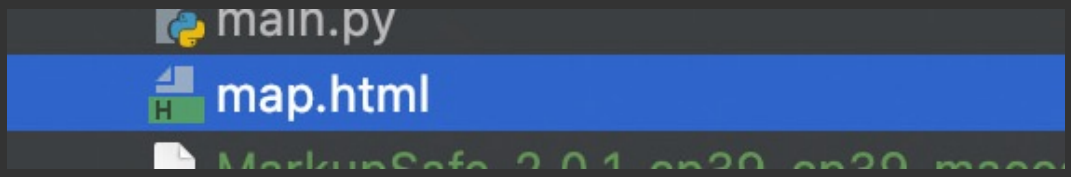
```
(venv) (base) mac@SungMak pythonProject % pip list --format=columns
Package    Version
---------- -------
numpy      1.22.1
pip        21.1.2
setuptools 57.0.0
wheel      0.36.2
```

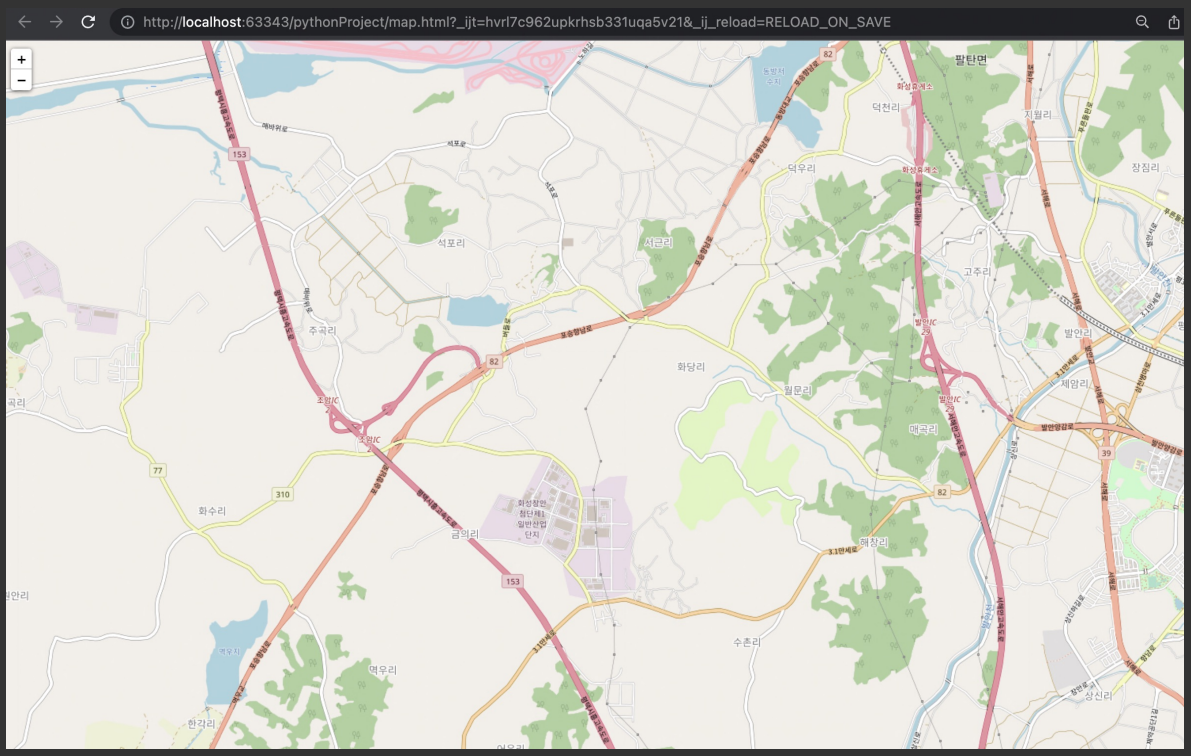
```python
# folium package 사용
import folium
m = folium.Map(location=[37.1233, 126.8732], zoom_start=15)
m.save("map.html")
```

```
Traceback (most recent call last):
  File "/Users/mac/PycharmProjects/pythonProject/0128_4th.py", line 101, in <module>
    import folium
ModuleNotFoundError: No module named 'folium'
```

```
Saved ./folium-0.12.1.post1-py2.py3-none-any.whl
Saved ./branca-0.4.2-py3-none-any.whl
Saved ./Jinja2-3.0.3-py3-none-any.whl
Saved ./MarkupSafe-2.0.1-cp39-cp39-macosx_10_9_x86_64.whl
Saved ./requests-2.27.1-py2.py3-none-any.whl
Saved ./certifi-2021.10.8-py2.py3-none-any.whl
Saved ./charset_normalizer-2.0.10-py3-none-any.whl
Saved ./idna-3.3-py3-none-any.whl
Saved ./urllib3-1.26.8-py2.py3-none-any.whl
Successfully downloaded folium branca jinja2 MarkupSafe numpy requests certifi charset-normalizer idna urllib3
WARNING: You are using pip version 21.1.2; however, version 21.3.1 is available.
You should consider upgrading via the '/Users/mac/PycharmProjects/pythonProject/venv/bin/python -m pip install --u
(venv) (base) mac@SungMak pythonProject %
```

```python
import folium
m = folium.Map(location=[37.1233, 126.8732], zoom_start=15)
m.save("map.html")
```





〈 Data Type 〉

(데이터 분류)

1. Scala Data & Vector Data

2. 데이터 형태에 따른 분류

정형 데이터
: 테이블 형태로 존재하는 data

비정형 데이터
; 테이블이 아닌 일반 파일 형태로 존재하는 데이터 (음성, 영상)

반정형 데이터
: 비정형 데이터인데 meta data 를 가지고 있어 정형처럼 해석이 가능 (JSON)
—〉 ex) Sesor

〈 python 기본 자료형 〉

- bool
- int, float , complex
- str
- bytes
- list
- tuple
- set
- dict

〈 bool 〉

: True 또는 false를 저장하는 자료형
— 파이선에는 0 이 아닌 숫자나 데이터가 존재하는 Vector 타입도 True

0 이나 데이터가 없는 Vector 타입은 false
—〉 다른 자료형을 bool 로 변환할 때는 bool(다른 종류의 데이터) 를 이용

〈 수치 데이터 〉

- int -
: 정수를 저장하기 위한 자료형
— python 의 경우 메모리가 허용하는 한 무한대의 정수를 다룰수 있음

다른 자료형의 데이터를 정수로 변환하고자 할 때 int(사용)

- float -
: 실수를 저장하기 위한 자료형
8 byte를 이용해서 표현
소수점을 움직여서 소수를 표현 (부동 소수점)

1.2 —〉 고정 소수점 형태로 표현 가능

3 e4 -〉 3 의 10의 4 승 , 부동 소수점 형태로도 표현 가능

다른 자료형의 데이터를 실수로 변환할 때는 float(데이터)를 이용

실수 ——〉 정수
사용할 수 있는 함수 : round , math . ceil, math floor

```
sum = 0.0
for i in range(0, 1000, 1):
    sum = sum + 0.1
print(sum)

print((1.0-0.8) == 0.2)

t = 1.0 - 0.8
print(t)
print(t == 0.2)
```

```
99.99999999999986
False
0.19999999999999996
False
```

- complex -
복소수로 저장하기 위한 자료형
—) 실수부와 허수부로 구성
(실수 나 허수부로 구성)

〈 sequential Type 〉
여러 개의 객체를 순서대로 저장하는 자료형

종류
str , bytes, list, tuple, range 함수로 만든 데이터

- 공통된 연산 -

[index]
: index 번째 데이터를 리턴
음수를 이용하면 뒤에서부터 indexing
-1 은 가장 마지막 데이터
-2 는 뒤에서 두번째 데이터
없는 index를 사용하면 index out of range 예외 발생

[시작 위치 : 종료위치 : 간격]
- 데이터를 슬라이싱 , 간격을 생략하면 1, 종료위치를 생략하면 마지막 까지

+ 연산자 : 결합

* 정수 : 반복

in : 데이터가 존재하는지 확인

len(데이터) : 데이터의 개수

```
msg = "BigInteger"
print(msg[3])
print(msg[2])
print(msg[0:3])
print(msg[1:])
```

```
I
g
Big
igInteger
```

〈 STR 〉

문자의 집합

(한 줄)

" "

" "

를 이용해서 생성

```
msg1 = "Hello"
msg2 = "s"

print(msg1<msg2)

print(msg1[0] == 'H')
```

True
True

(여러줄)

''' '''

""" """

를 이용해서 생성

```
print(" I wanna go a john")
print(" I wanna go \n a john")
```

I wanna go a john
I wanna go
a john

내부 데이터를 읽기는 가능하지만 수정은 안됨

파이썬에서는 각 문자의 코드값을 이용해서 크기 비교 가능

〉 , 〉= , 〈, 〈=
연산자 사용 가능

제어문 문자 사용 가능
: 역슬래시 다음에 영문자 1글자를 추가해서
특별한 기능을 갖도록 한 것 , ₩n , ₩t ₩₩

생성할 때
"{index : 서식}".format(데이터 나열) —〉 생성 가능
f"{데이터}" 로도 생성 가능

〈 대소문자 변환 〉

upper()

lower()

swapcase()

capitalize()

title()

이름은 singsiuk 이고 나이는 27 이다

```
print("이름은 {0:10s}이고 나이는 {1:5d} 이다".format("singsiuk", 27))
```

〈 검색 관련 method 〉

count()

find() : 데이터가 없으면 음수를 return

rfind()

index()

startwith()

endswith()

〈 치환 관련 method 〉

strip()

rstrip()

lstrip()

replace()

〈 확인하는 method - is 로 시작 〉

isdigit()

〈 인코딩 관련 된 method 〉

encode
decode method

* 한글자씩 가져오는 것과 len 만으로 구현하는 것이 좋다

〈 Bytes 〉
바이트의 모임

파일의 내용을 읽을 때 주로 이용한다

: 문자열과 변환할 때는 encode 와 decode method 를 사용

b 문자 literal 작성 ——〉 bytes 로 생성

〈list〉
변경 가능한 데이터의 연속적인 모임

1. 생성
[ ] : 비어있는 list
[데이터를 나열]

list(__iter__ 가 구현된 객체)

2 . 내부 데이터 변경

list [index] = 하나의 데이터

list [시작 위치 : 종료 위치] = 여러개의 데이터

——— 삭제
del list [index] :

list [시작위치 : 종료 위치] = []

```
# 전체데이터 순회하기

# 방법 1 빠른 열거
for element in li:
    print(element)

print()
print("hoo")
print()

# 방법 2 len 사용
for i in range(0, len(li)):
    print(li[i])
```

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

hoo

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

3. 데이터 추가

append (데이터)
insert (index, data)

4. 데이터 정렬

sort (key = None , reverse= False)

```
# ------ - sort
li = [100, 200, 1, 0, 5, 27]

# sort
li.sort(reverse=True)
# reverse 가 TRUE 이면 Descending
print(li)

# sort 함수를 적용
li1 = ["1", "2", "3", "a", "b", "c", "J", "R", "C"]
li1.sort(reverse=True)
print(li1)

li1.sort(key=str.lower, reverse=True)
print(li1)
```

```
['c', 'b', 'a', 'R', 'J', 'C', '3', '2', '1']
['R', 'J', 'c', 'C', 'b', 'a', '3', '2', '1']
```

〈 Tuple 〉

내부 데이터를 변경할 수 없는 데이터의 집합

- 생성

(데이터 나열)

데이터를 , 로 구분해서 나열

tuple(__iter__ 가 구현된 객체)

```
row = (1, "ring")
print(row)

row = 1, "song"
print(row)

print(row[0])
# key 값을 return
row[0] = 2
# 값을 변경할 수 없다
# 분석하고 결과를 return 할 때 Tuple 을 자주 사용
```

```
(1, 'ring')
(1, 'song')
1
Traceback (most recent call last):
  File "/Users/mac/PycharmProjects/pythonProject/0128_4th.py", line 170, in <module>
    row[0] = 2
TypeError: 'tuple' object does not support item assignment
```

〈 set 〉

데이터를 순서와 상관없이 중복 되지 않게 저장하는 데이터의 집합

- 생성

{ 데이터의 나열 }

set()

set(__iter__ 를 구현한 객체)

```
hashset = {100, 200, 300, 100, 100, 100, 100}
print(hashset)
```

{200, 100, 300}

〈 dict 〉

key 와 value 를 쌍으로 저장하는 자료형

(생성)
{ key: value , key : value ···}

: key 는 일반적으로 문자열로 만들고 , set 으로 구성되기 때문에 중복될 수 없음

dict(zip(key 의 모임, value의 모임))

(value 찾기)
dict 이름 [key]

* 존재하지 않는 key를 사용하면
예외를 발생

```
print("horizon----------------------------------------")

dictionary = {"name": "singsiuk", "age": 27, "hobby": "Programming"}
print(dictionary["name"])

# 없는 key를 사용하면 Error 가 뜬다
print(dictionary["Nothing in Dictionary"])
```

```
---------
Overriding complete
Traceback (most recent call last):
  File "/Users/mac/PycharmProjects/pythonProject/0128_4th.py", line 189, in <module>
    print(dictionary["Nothing in Dictionary"])
KeyError: 'Nothing in Dictionary'
```

```
print("horizon----------------------------------------")

dictionary = {"name": "singsiuk", "age": 27, "hobby": "Programming"}
print(dictionary["name"])

# 없는 key를 사용하면 Error 가 뜬다
# print(dictionary["Nothing in Dictionary"])

# dict 는 빠른 열거를 이용하면 key를 순서대로 접근한다
for key in dictionary:
    print(key, ":", dictionary[key])
```

```
horizon----------------------------------------
singsiuk
name : singsiuk
age : 27
hobby : Programming
```

(DATA Delete)
del dict 이름 [key]

(Method)

keys() : 모든 key 를 return

values() : 모든 value 를 리턴

items() : 모든 key 와 value 를 tuple 로 묶어서 리턴

clear() : 모든 데이터를 삭제

〈 테이블 형태의 데이터 구조 〉

DTO 의 List
dict 의 List
—— 저 데이터의 행 단위 접근과
열 단위 접근

```
print("horizon-----------------------------")

dic1 = {"name": "sing", "age": 2}
dic2 = {"name": "si", "age": 7}
dic3 = {"name": "uk", "age": 27}

table =[dic1, dic2, dic3]
print(table)

# 행 단위 접근
for idx in range(0, len(table)):
    print(table[idx])

# 열 단위 접근
for row in table:
    print(row["name"])
```

```
horizon-----------------------------
[{'name': 'sing', 'age': 2}, {'name': 'si', 'age': 7},
{'name': 'uk', 'age': 27}]
{'name': 'sing', 'age': 2}
{'name': 'si', 'age': 7}
{'name': 'uk', 'age': 27}
sing
si
uk
```

〈 sequence 객체를 이용한 list 생성 〉

연산을 수행해서 생성

——〉 map 함수 이용 가능

[연산식 for 임시변수 in sequence 객체]

조건을 설정해서 생성하는 것도 가능

시퀀스 객체 뒤에 if 조건을 추가하면 된다

```
print("horizon-----------------------------")

ar = range(1, 11, 1)

# 원래
map("함수", "데이터의 모임")

# 제곱을 이용해서 생성 가능
result = [i*i for i in ar]
print(result)

# 걸러 내는 것 ---> filter 사용
ar = {"하나", "둘", "셋", "넷", "다섯"}
result = [i for i in ar if len(i)==2]
print(result)
```

```
horizon-----------------------------
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
['다섯', '하나']
```

```
# 이중 데이터
li1 = [1, 2, 3]
li2 = [4, 5, 6]
result = [x+y for x in li1 for y in li2]
print(result)
```

```
[5, 6, 7, 6, 7, 8, 7, 8, 9]
```