Python

<딕셔너리 Key를 이용하여 Value를 뽑아내기.

```
list_a=[1,2,3]
tuple_a=[1,2,3]
str1 = 'apple'
list_a[0]
1
tuple_a[0]
1
str1[0]
'a'
list_a[1:]
[2, 3]
```

인덱싱, 슬라이싱 → 요소를 뽑아내는 것

딕셔너리는 Key를 이용하여 Value를 뽑아냄
동일한 Key가 존재하면 어떤 Key의 Value를 불러낼지 모르니 자기 마음대로 Value를 불러냄.
Key가 중복되었을 때 1개를 제외한 나머지 Key:Value 값이 모두 무시되는 이유는 Key를 통해서 Value를 얻는 딕셔너리의 특징에서 비롯된다. 즉 동일한 Key가 존재하면 어떤 Key에 해당하는 Value를 불러야 할지 알 수 없기 때문이다.

또 한 가지 주의해야 할 사항은 Key에 리스트는 쓸 수 없다는 것이다. 하지만 튜플은 Key로 쓸 수 있다. 딕셔너리의 Key로 쓸 수 있느냐 없느냐는 Key가 변하는 값인지 변하지 않는 값인지에 달려 있다.

```
specialty = {"김연아":"피겨스케이팅", "류현진":"야구", "박지성":"축구", "귀도":"파이썬"}
print(specialty.keys())
```

어떠한 명령을 내렸을 때 결과 값을 되돌려 주는 것을 반환값이라고 함
1+5 > 10 False(거짓)
5+3 : 8
specialty.keys()
⇒dict_keys(["김연아", "류현진", "박지성", "귀도"])
["김연아", "류현진", "박지성", "귀도"] : list

list(a.keys()) : 리스트가 필요한 경우 리스트를 뽑음

dict_keys 객체를 돌려준다. 다음에 소개할 dict_values, dict_items 역시 파이썬 3.0 이후 버전에서 추가된 것들이다. 만약 3.0 이후 버전에서 반환 값으로 리스트가 필요한 경우에는 list(a.keys())를

사용하면 된다.

객체?
a = 3 : 숫자 변수
3 : 숫자 상수
'a' : 문자 상수
a = 'apple' : 문자 변수
이러한 것을 속성값(Attribute)이라고 부름

```
def add(x,y):
return x+y
sum = add(5,3)
8
```

객체는 어떠한 속성값과 행동을 가지고 있는 데이터입니다.
파이썬의 모든것들(숫자, 문자, 함수 등)은 여러 속성과 행동을 가지고 있는 데이터입니다.
object(객체)는 그 본연의 추상적인 개념만 가지고 있는 것이 아니라, 다양한 정보와 행동을 가지고 있습니다.

표현하는 것 뿐만 아니라 자체를 객체로 만들고 다양한 속성과 행동을 넣어뒀습니다.

이러한 객체들이 가진 속성중에 상태들은 value, 또는 attribute라고 부릅니다. 또 객체가 가진 행동들은 method라고 부릅니다.
이러한 프로그래밍 기법을 객체지향프로그래밍이라고 합니다.
객체지향프로그래밍은 컴퓨터 프로그램으로 해결해야 하는 문제를 실제 세상에서 처럼 다양한 정보들을 가진 객체들로 표현하고, 그 객체들간 통신으로 해결하는 기법입니다.

딕셔너리에서 key값을 뽑아내기

```
a = {'name': 'pey', 'phone': '0119993323', 'birth': '1118'}
a.keys()
dict_keys(['name', 'phone', 'birth'])
list(a.keys())
['name', 'phone', 'birth']
a_keys = list(a.keys())
a_keys
['name', 'phone', 'birth']
a_keys[0]
'name'
```

values 뽑아내기

```
a_values
```

```
a_values = list(a.values())
a_values
['pey', '0119993323', '1118']

a
{'name': 'pey', 'phone': '0119993323', 'birth': '1118'}
```
※ 리스트 형태로 뽑아내면 리스트가 됨

```
a = {'name': 'pey', 'phone': '0119993323', 'birth': '1118'}

for k in a.keys():
    print(k)
```
: Key값을 반복하여 가지고 옴

```
for l in a.values():
    print(l)
```
: Value값을 반복하여 가지고 옴

Key와 Value 쌍으로 얻기
```
for i in a.items():
    print(i)
```
: tuple 형태로 (Key , Value)로 값이 출력됨

<Key Value 쌍으로 지우기>
```
a.clear()
a
{}
```
모두 지우고 빈 딕셔너리만 남게됨.

<Key로 Value를 얻기>
```
a.get('name')
'pey'
```
Key값의 Value를 가지고옴

```
a.get('age')
```
a['age'] → 에러

get은 키값이 없으면 아무것도 나타나지 않지만 일반적으로 Key값을 입력하면 오류를 발생.

```
a.get('age', 'phone')
'phone'
```

값이 없을 때 디폴트 값을 주면 디폴트 값을 사용함.

● Key값이 있는지 없는지 확인 할 때(in)

```
'age' in a
False
'name' in a
True

a = {'name': 'pey', 'phone': '0119993323', 'birth': '1118'}
key_value = input("찾고 싶은 키는? ")
if key_value in a:
    print(a.get(key_value))
else:
    print(f"{key_value}는 없는 키입니다.")

찾고 싶은 키는? pey
없는 키입니다.
찾고 싶은 키는? name
pey

b = [1,2,3]
1 in b
True
4 in b
False
```

in은 어떤 형태(튜플, 딕셔너리, 리스트)든지 자료의 속한 것을 파악가능함.

■ 집합

```
a = []
a = list()
a
[]
a = [1,2,3]
a
[1, 2, 3]
a = list(4,5,6) -> 에러
a = list([4,5,6])
```

list는 []안에 넣어줘야 함

tuple

```
b = ()
b
()
b = tuple()
b = (4,5,6)
b
(4,5,6)
b = tuple(8,9,10) -> 에러
b = tuple((8,9,10))
b
(8, 9, 10)
```

dictionary 만들기

```
c = {}
c
{}
c = dict()
c = {1:'a', 2:'b'}
c
{1: 'a', 2: 'b'}
c = dict(a=1, b=2)
c
{'a': 1, 'b': 2}
```

Set

```
s1 = set([1,2,3])
s1
{1, 2, 3}
```

dictionary와 차이점 {key : values} vs {values}

```
s2 = set("Hello")
s2
{'l', 'e', 'H', 'o'}
```

동일한 값 중복 없이 유일한 값만 들어감
비어있는 집합 만들려면

```
s = set()
s
set()
```

Set 명령어는 중복을 허용하지 않는다. 순서가 없습니다.

list와 tuple은 순서가 있는데 set은 순서가 없음
indexing으로는 값을 얻을 수 없음(순서가 없기때문)
⇒ List나 Tuple을 이용해서 인덱싱을 해야함

```
s1 = set([1,2,3,2,4,5,3,4,6,2,4,1,7])
s1
{1, 2, 3, 4, 5, 6, 7}
```
중복된 값을 필터링 함.

■ 교집합, 합집합, 차집합 구하기
교집합
```
s1 = set([1, 2, 3, 4, 5, 6])
s2 = set([4, 5, 6, 7, 8, 9])

s1&s2
{4, 5, 6}
```
s1.intersection(s2) = s2.intersection(s1)로도 사용할 수 있음

합집합
```
s1|s2
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```
s1.union(s2)로도 사용 가능

차집합
```
s1-s2
{1, 2, 3}
s2-s1
{8, 9, 7}
```
s1.difference(s2), s2.difference(s1) 로도 사용 가능

■ Set에서의 자료 추가, 제거
리스트 자료형은 순서가 있기 때문에 append, insert(index번호, 넣을 자료)로 추가 할 수 있음

add는 1개의 값만 추가할 때 사용
```
s1 = set([1, 2, 3])
s1.add(4)
s1
{1, 2, 3, 4}

s1 = set([1,2,3])
```

s1.add(4)
s1
{1, 2, 3, 4}
s1.add(8)
s1.add(7)
s1
{1, 2, 3, 4, 7, 8}
s1.add(5)
s1.add(6)
s1
{1, 2, 3, 4, 5, 6, 7, 8}
집합형은 순서는 없지만 자료 정렬은 해줌

update(여러개 값을 추가)
s1.update([9,10,11,12])
s1
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}

<값을 지우기>
list에서는 del s1[요소]로 값을 제거
set에서는 del로는 불가 remove로 지울 수 있음

remove
s1.remove(4)
s1
{1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12}

■ 불 자료형(논리 값 – 참과 거짓)
bool 자료형은 참과 거짓을 나타내는 자료형

1>5
False
1<5
True
5==5
True

a = True
b = False
첫글자는 대문자로 적어야함.
type(a)

<class 'bool'>
type(b)
<class 'bool'>

자료형의 참과 거짓
자료가 있으면 참 자료가 없으면 거짓

문자열, 리스트, 튜플, 딕셔너리 등의 값이 비어 있으면(" ", [ ], ( ), { }) 거짓이 된다. 당연히 비어 있지 않으면 참이 된다. 숫자에서는 그 값이 0일 때 거짓이 된다.

```
a = [1,2,3,4]
while a:
    print(a.pop())
```

a가 참인동안 반복(4 3 2 1 순으로 출력이 됨)

```
if []:
    print("참")
else:
    print("거짓")
```

빈 리스트라서 거짓이 출력이 됨

```
if [1]:
    print("참")
else:
    print("거짓")
```

값이 하나라도 있기 때문에 참이 출력이 됨

■ 자료형의 값을 저장하는 공간, 변수
변수를 만들 때는 위 예처럼 =(assignment) 기호를 사용한다.
다른 프로그래밍 언어인 C나 JAVA에서는 변수를 만들 때 자료형을 직접 지정해야 한다. 하지만 파이썬은 변수에 저장된 값을 스스로 판단하여 자료형을 지정하기 때문에 더 편리하다.(다른 프로그램에서는 직접적으로 정의를 해주어야함) 즉, python은 값에 따라서 변수의 자료형이 결정

변수란?
객체를 가리키는 것이라고도 말할 수 있다.
객체란 우리가 지금껏 보아 온 자료형과 같은 것을 의미하는 말이다
리스트가 저장된 메모리의 주소를 가리키게 된다.
※ 메모리란 컴퓨터가 프로그램에서 사용하는 데이터를 기억하는 공간이다.

```
a=[1,2,3]
b='apple'
id(a)
```

2160459927744
id(b)
2160489154736
id(b[0])
2160456721200
id(b[1])
2160451949424
변수 자료가 저장된 위치를 보여줌

■ 리스트 복사하기
```
a = [1,2,3]
b = a
id(a)
2646104233472
id(b)
2646104233472
a
[1, 2, 3]
b
[1, 2, 3]
b.append(4)
a
[1, 2, 3, 4]
b
[1, 2, 3, 4]
```
같은 값이기 때문에 b를 바꿨는데 a도 같이 바뀜

```
a = [1,2,3]
b = a[:]
a
[1, 2, 3]
b
[1, 2, 3]
id(a)
2646136638208
id(b)
2646135474752
```
값은 같지만 다르게 나옴

copy 모듈을 이용하여 리스트 복사
```
from copy import copy
```

```
a = [1,2,3]
b = copy(a)

print("a = ", id(a))
print("b = ", id(b))

a =  1414714443840
b =  1414714033984
```

list에서 값을 복사할 때 copy를 쓰거나 a[:] 쓰면 별개의 주소 값이 됨

```
c.f> b = a
```

■ 변수를 만드는 여러 가지 방법

```
a, b = ('python', 'life')
a
'python'
b
'life'

a = 5
b = 3
a, b = b, a
a
3
b
5
```

값이 바뀌어짐.

■우분투 ZSH & NVIM 인스톨

1. ZSH 설치

```
$ sudo apt -y update ; sudo apt -y upgrade
$ echo $SHELL     # 지금 쓰고 있는 쉘 확인
$ sudo apt-get install zsh
$ chsh -s $(which zsh)  # 기본 쉘을 zsh로 변경
$ sudo apt-get install curl     # curl 설치
$ sudo apt-get install git #git 설치
$ sh -c "$(curl -fsSL
https://raw.githubusercontent.com/ohmyzsh/ohmyzsh/master/tools/install.sh)"
```

폰트 설치

```
$ sudo apt-get install fonts-powerline
$ git clone https://github.com/powerline/fonts.git --depth=1
$ cd fonts
$ ./install.sh
$ cd ..
$ rm -rf fonts  # fonts 폴더 삭제./

# 플러그인 설치
$ cd ~/.oh-my-zsh/plugins      # 여기에 플러그인 클론하기
$ git clone https://github.com/zsh-users/zsh-autosuggestions
$ git clone https://github.com/zsh-users/zsh-syntax-highlighting
$ vim ~/.zshrc
ZSH_THEME="agnoster"
plugins=(git zsh-autosuggestions zsh-syntax-highlighting)

$ source ~/.zshrc

2. NVIM 설치
# python을 설치함
$ sudo apt-get install software-properties-common
$ sudo apt-get install python3-dev python3-pip
$ sudo add-apt-repository ppa:neovim-ppa/unstable
$ sudo apt-get update
$ pip3 install --user neovim
$ sudo apt-get install neovim
$ vim ~/.zshrc
alias vi='nvim' # 마지막 줄에 추가
alias vim='nvim'  # 설정시 vim 을 입력해도 neovim 실행
```