Python Basic

자료구조(Data Structure)

리스트(List)

- 리스트의 특징
  - 순서가 있는 요소의 집합.
  - 수정 가능(mutable).
  - 문자열 또한 리스트의 일종.
  - 서로 다른 자료형의 요소를 입력 가능.

    ex) [1, 3.14, "a", True, None]
  - 리스트의 요소로 리스트를 입력 가능(중첩 리스트).

    ex) [1, [1, 2], [1, [1, 2, 3]]]

- 리스트 생성
  - [] 또는 list() 함수로 생성.

    ex) list("python") # ['p', 'y', 't', 'h', 'o', 'n']

리스트(List)

- 리스트 인덱싱과 슬라이싱
  - 문자열과 마찬가지로 인덱싱, 슬라이싱 가능.

표현	의미
[index]	해당 index 위치의 요소를 선택
[:]	처음부터 끝까지
[start:]	start offset부터 끝까지
[:end]	처음부터 end-1까지
[start:end]	start offset부터 end-1까지
[start:end:step]	step만큼 요소를 건너뛰면서, start 오프셋부터 (end-1) 오프셋까지 시퀀스를 추출

리스트(List)

- 중첩 리스트의 인덱싱과 슬라이싱

```
l = [1, [1, 2], [1, [1, 2, 3]]]

l[0] == l[1][0] == l[2][0] == l[2][1][0] # True
l[2][1] # [1, 2, 3]
l[2][1:] # [[1, 2, 3]]
l[2][1][0:2] # [1, 2]
l[2][1] == l[2][1][:] # True
```

리스트(List)

► 요소 입력

표현	의미
[list].append(e)	전달받은 e를 리스트의 끝에 추가한다.
[list].insert(index, e)	전달받은 e를 리스트의 index에 추가한다.

```
l = [0, 1, 2]
l[1] = [3, 4, 5]
print(l)

l = [0, 1, 2]
l[1:1] = [3, 4, 5]
print(l)
```

리스트(List)

► 요소 삭제

표현	의미
[list].remove(value)	전달받은 value를 찾아 삭제한다.
del list[i]	해당 요소를 삭제한다.
[list].pop(index)	해당 index의 요소를 꺼낸다.
[list].clear()	모든 요소를 삭제한다.

리스트(List)

- 요소 삭제

```
num_list = list(range(1, 11))
print(num_list)

num_list.remove(3)
print(num_list)

del num_list[3]
print(num_list)

del num_list[3:6]
print(num_list)

num_list.pop(3)
print(num_list)

num_list.clear()
print(num_list)
```

리스트(List)

► 요소 검색

표현	의미
[list].index(value)	전달받은 value에 해당하는 index를 찾는다.
[list].count(value)	전달받은 value가 몇 개 있는지 찾는다.
len([list])	리스트의 길이를 반환한다.
min([list]), max([list])	리스트의 요소 중 최소/최대값을 찾는다.
e in [list] e not in [list]	리스트에 e가 있는지 유무를 검사한다.

리스트(List)

- 요소 검색

```
str_list = list("python")
print(str_list)
print(len(str_list))
print(min(str_list))
print(max(str_list))
print(str_list.index("o"))
print("z" in str_list)

ans_list = ["yes", "y", "예", "네"]
ans = input("결제하시겠습니까?")
if ans.lower() in ans_list:
    print("결제가 완료되었습니다.")
else:
    print("결제가 취소되었습니다.")
```

리스트(List)

- 리스트 정렬

표현	의미
[list].sort() [list].sort(reverse=True)	요소의 순서를 정렬한다.
[list].reverse()	요소의 순서를 뒤집는다.
sorted([list])	요소의 순서를 정렬한 새 리스트를 반환한다.

```
num_list = [2, 11, 3, 5, 7]
num_list.sort()
print(num_list)
num_list.reverse()
print(num_list)
num_list.sort(reverse=True)
print(num_list)
```

리스트(List)

► 리스트 병합

표현	의미
[list].extend([list])	기존의 리스트에 전달받은 리스트를 병합한다.
[list] + [list]	두 리스트를 병합한 새로운 리스트를 반환한다.
[list] * 숫자	리스트의 요소를 반복한 새로운 리스트를 반환한다.

```
list1 = ["A", "B", "C"]
list2 = ["D", "E", "F"]
print(list1 + list2)
list1.extend(list2)
print(list1)
print(list2)
```

튜플(Tuple)

- 튜플의 특징
  - 순서가 있는 요소의 집합.
  - 수정 불가능(immutable).
  - 리스트와 비슷함.
  - 서로 다른 자료형의 요소를 입력 가능.

    ex) (1, 3.14, "a", True, None)
  - 튜플의 요소로 튜플을 입력 가능(중첩 튜플).

    ex) (1, (1, 2), (1, (1, 2, 3)))

- 튜플 생성
  - () 또는 tuple() 함수로 생성.
  - () 생략 가능.

    ex) tuple("python") # ('p', 'y', 't', 'h', 'o', 'n')

    ex2) new_tuple = 1, 2, 3 # (1, 2, 3)

튜플(Tuple)

- 튜플 인덱싱과 슬라이싱
  - 문자열, 리스트와 마찬가지로 인덱싱, 슬라이싱 가능.

```
t1 = (1, 2, 'a', 'b')
print(t1[0])         # 1
print(t1[3])         # b
print(t1[1:])        # (2, 'a', 'b')
print(t1[:2])        # (1, 2)
print(t1[::2])       # (1, 'a')
```

튜플(Tuple)

- 요소 검색
  - 리스트와 마찬가지로 요소 검색 가능.

표현	의미
(tuple).index(value)	전달받은 value에 해당하는 index를 찾는다.
(tuple).count(value)	전달받은 value가 몇 개 있는지 찾는다.
len((tuple))	튜플의 길이를 반환한다.
min((tuple)), max((tuple))	튜플의 요소 중 최소/최대값을 찾는다.
e in (tuple) e not in (tuple)	튜플에 e가 있는지 유무를 검사한다.

튜플(Tuple)

► 튜플 연산

표현	의미
(tuple) + (tuple)	두 튜플을 병합한 새로운 튜플을 반환한다.
(tuple) * 숫자	튜플의 요소를 반복한 새로운 튜플을 반환한다.

```
t1 = ("A", "B", "C")
t2 = ("D", "E", "F")
print(t1 + t2)
print(t1 * 2)
```

튜플(Tuple)

- Tuple Unpacking
  - 여러 개의 변수에 값을 한꺼번에 대입하는 기능

```
ex) breakfast, lunch, dinner = \
        (“Sandwich”, None, “Chicken”)
```

  - 변수의 값 치환을 쉽게 구현할 수 있다.

```
ex) a, b = (12, 34)
    a, b = b, a
    print(a, b)
```

사전(Dictionary)

- 사전의 특징
  - 키(Key)와 값(Value)을 쌍의 형태로 저장하는 자료구조.
  - {Key1: Value1, Key2: Value2, Key3: Value3, ...}
  - 순서가 존재하지 않음.
  - 키를 이용하여 값을 호출할 수 있음.
  - 사전 자체는 mutable 객체.
  - 키는 immutable 객체(숫자, 문자열, 불, 튜플)만 사용할 수 있음.
  - 값은 중복될 수 있지만 키는 중복될 수 없음.

사전(Dictionary)

- 사전 생성
  - {} 또는 dict() 함수로 생성

```
ex) scores = {"Java": 100, "Oracle": 90, "Python": 80}
ex2) scores = dict(Java = 100, Oracle = 90, Python = 80)
ex3) list_in_list = [["Java", 100], ["Python", 80]]
     print(dict(list_in_list))
     tuple_in_list = [("Java", 100), ("Python", 80)]
     print(dict(tuple_in_list))
     list_in_tuple = (["Java", 100], ["Python", 80])
     print(dict(list_in_tuple))
     tuple_in_tuple = (("Java", 100), ("Python", 80))
     print(dict(tuple_in_tuple))
```

사전(Dictionary)

► 요소 입력

명령어	설명
dict[key] = value	사전에 key와 value의 쌍을 입력한다. key가 이미 존재하면 기존의 value를 덮어쓴다.

```
new_dict = {0: "Hello"}
print(new_dict)

new_dict[1] = "World"
print(new_dict)

new_dict[0] = "Hi"
print(new_dict)
```

사전(Dictionary)

► 요소 삭제

명령어	설명
del dict[key]	value를 삭제한다. value를 찾을 수 없으면 예외를 발생시킨다.
{dict}.pop(key)	value를 꺼낸다. value를 찾을 수 없으면 예외를 발생시킨다.
{dict}.clear()	모든 요소를 삭제한다.

사전(Dictionary)

- 요소 삭제

```
time_dict = {0: "morning", 1: "noon", 2: "evening", 3: "night"}
print(time_dict)

del time_dict[0]
# del time_dict[0]
print(time_dict)

print(time_dict.pop(1))
# print(time_dict.pop(1))
print(time_dict)

time_dict.clear()
print(time_dict)
```

사전(Dictionary)

► 요소 검색

명령어	설명
dict[key]	value를 찾아서 반환한다. value가 없으면 에러를 발생시킨다.
{dict}.get(key) {dict}.get(key, 반환 값)	value를 찾아서 반환한다. value가 없으면 설정한 반환 값 또는 None을 반환한다.
key in {dict} key not in {dict}	전달받은 key가 사전에 존재하는지 유무를 판별한다.

```
eng_to_kor = {"boy": "소년", "school": "학교", "book": "책"}

print(eng_to_kor["boy"])
print(eng_to_kor.get("school"))
print(eng_to_kor.get(0))
print(eng_to_kor.get("student", "등록되지 않은 단어입니다."))
print("book" in eng_to_kor)
print("book" not in eng_to_kor)
```

사전(Dictionary)

► 사전 수정

명령어	설명
{dict}.update({dict})	두 사전을 병합한다. 중복된 key가 존재하면 전달받은 사전 내 해당 value로 덮어쓴다.

```
dict1 = {0: "zero", 1: "one"}
dict2 = {1: "일", 2: "이"}
print(dict1)
print(dict2)

dict1.update(dict2)
print(dict1)
print(dict2)
```

사전(Dictionary)

- 기타 연산

명령어	설명
{dict}.keys()	key의 목록을 반환한다.
{dict}.values()	value의 목록을 반환한다.
{dict}.items()	key와 value의 쌍을 튜플로 묶어 반환한다.
min({dict}) max({dict})	key 중 가장 작은/큰 값을 반환한다.
sorted({dict})	key만 추려서 정렬한 리스트를 반환한다.

사전(Dictionary)

- 기타 연산

```
stocks = {'eggs': 200, 'sausage': 100, 'bacon': 100, 'spam': 500}
keys = stocks.keys()
values = stocks.values()
items = stocks.items()

for key in keys:
    print("품목명:", key, "/ 수량:", stocks[key])

for item in items:
    print("품목명:", item[0], "/ 수량:", item[1])

n = 0
for val in values:
    n += val
print("총 재고량:", n)
```

집합(Set)

- 집합의 특징
  - 중복을 허용하지 않는 비순서적 자료구조.
  - {Element1, Element2, Element3, ...}
  - mutable 객체.

집합(Set)

- 집합 생성
  - {} 또는 set() 함수로 생성
  - 공집합을 만들 때 {} 기호로 만들 수 없음.

```
ex) basket = \
        {"apple", "orange", "pear", "apple", "orange"}
        print(basket)
ex2) spell = set("abracadabra")
         print(spell)
```

집합(Set)

► 집합 함수

명령어	설명
{set}.add(e)	요소를 추가한다.
{set}.remove(value)	요소를 제거한다.
{set}.update({set})	집합을 전달받아 병합한다.

```
prime1 = {2, 3, 5, 7}
prime2 = {11, 13, 17, 19}
print(prime1)
print(prime2)

prime1.add(9)
print(prime1)
prime1.remove(9)
print(prime1)
prime1.update(prime2)
print(prime1)
```

집합(Set)

► 집합 연산

명령어	설명
| union()	합집합
& intersection()	교집합
- difference()	차집합
^ symmetric_difference()	배타적 차집합

집합(Set)

- 집합 연산

```
two = {2, 4, 6, 8, 10, 12}
three = {3, 6, 9, 12}

print(two | three)
print(two & three)
print(two - three)
print(two ^ three)
print(two & three | two ^ three)
print(two - three & three - two)

print(three.union(two))
print(three.intersection(two))
print(three.difference(two))
print(three.symmetric_difference(two))
print(three.intersection(two).union(three.symmetric_difference(two)))
print(three.difference(two).intersection(two.difference(three)))
```

집합(Set)

► 집합 연산

명령어	설명
<= issubset()	부분집합
<	진성 부분집합
>= issuperset	포함집합
>	진성 포함집합

집합(Set)

- 집합 연산

```
two = {2, 4, 6, 8, 10, 12}
three = {3, 6, 9, 12}

print(two & three <= two | three)
print((two & three).issubset(two | three))
print(two ^ three >= two - three)
print((two ^ three).issuperset(two - three))

print(two < two)
print(two <= two)
print(three > three)
print(three >= three)
```

컬렉션 고급 기법

- 컴프리헨션(Comprehension)
  - 하나 이상의 이터러블로부터 파이썬의 자료구조를 만드는 방법

1) List Comprehension

```
[ 표현식 for 항목 in 순회 가능한 객체 ]

[ 표현식 for 항목 in 순회 가능한 객체 if 조건]
```

2) Tuple Comprehension

```
tuple(표현식 for 항목 in 순회 가능한 객체 )
```

컬렉션 고급 기법

- List Comprehension

```
squares = []
for x in range(10):
   squares.append(x**2)
print(squares)

squares = [x**2 for x in range(10)]
print(squares)

combs = []
for x in [1,2,3]:
   for y in [3,1,4]:
      if x != y:
         combs.append((x, y))
print(combs)

combs = [(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]
print(combs)
```

컬렉션 고급 기법

- List Comprehension

```
from math import pi
[str(round(pi, i)) for i in range(1, 6)]

vec = [[1,2,3], [4,5,6], [7,8,9]]
num_list = [num for elem in vec for num in elem]
print(num_list)

vec = [[[1,2,3],[4,5,6],[7,8,9]], [[1,2,3],[4,5,6],[7,8,9]], [[1,2,3],[4,5,6],[7,8,9]]]
num_list = [num for a in vec for b in a for num in b]
print(num_list)

gugudan = ["%d * %d = %d" %(i, j, i*j) for i in range(2, 10) for j in range(1, 10)]
print(gugudan)
```

3) Dictionary Comprehension

- List Comprehension처럼 if, for … 다중 절을 가질 수 있다.

```
{키 표현식 : 값 표현식 for 표현식 in 순회가능한 객체}
```

4) Set Comprehension

```
{ 표현식 for 표현식 in 순회가능한 객체 }
```

컬렉션 고급 기법

- Dictionary Comprehension

```
gugudan = {"%d * %d" %(i, j): i*j for i in range(2, 10) for j in range(1, 10)}
print(gugudan)
```

- Set Comprehension

```
prime_set = set(range(2, 21)) - {x for x in range(2, 21) for y in range(2, x) if x % y == 0}
print(prime_set)
```

컬렉션 고급 기법

- enumerate()
  - 컬렉션의 각 요소와 해당 요소의 순서값을 추출하여 리턴하는 함수.
  - 추출할 컬렉션을 매개변수로 전달받아 enumerate 객체로 리턴함.
  - enumerate 객체를 다른 컬렉션으로 변환하여 사용함.

```
menu = ["입력", "삭제", "수정", "조회"]
print(menu)

menu_list = list(enumerate(menu))
print(menu_list)

menu_dict = dict(enumerate(menu))
print(menu_dict)

dict_to_enum = enumerate(menu_dict)
print(type(dict_to_enum))
print(list(dict_to_enum))
```

컬렉션 고급 기법

- zip()
  - 두 개 이상의 컬렉션을 병렬적으로 묶는 함수.
  - 묶을 컬렉션들을 매개변수로 전달받아 zip 객체로 반환함.
  - zip 객체를 다른 컬렉션으로 변환하여 사용함.

```
time = ["아침", "점심", "저녁"]
menu = ["토스트", "샐러드", "김밥", "라면", "샌드위치"]
print(list(zip(time, menu)))
print(dict(zip(time, menu)))

import random
random.shuffle(menu)
print(list(zip(time, menu)))
random.shuffle(menu)
print(dict(zip(time, menu)))
```

컬렉션 고급 기법

- any(), all()
  - any(): 컬렉션의 요소 중 참이 하나라도 있는지 조사하는 함수.
  - all(): 컬렉션의 요소가 모두 참인지 조사하는 함수.

```
prime_list = [i for i in range(2, 21) if all(i % j != 0 for j in range(2, i))]
print("소수:", prime_list)

composite_list = [i for i in range(2, 21) if any(i % j == 0 for j in range(2, i))]
print("합성수:", composite_list)
```

컬렉션 고급 기법

- 컬렉션 복사
  - 다른 변수에 바로 대입. 사실상 같은 객체.

    ex) a = [1, 2, 3]
        b = a
  - Slicing을 이용해 대입. 요소를 복사하여 새로운 객체를 만듦.

    ex) a = [1, 2, 3]
        b = a[:]
  - copy() 함수를 이용해 대입(얕은 복사, shallow copy)

    ex) import copy; a = [1, 2, 3]
        b = copy.copy(a)
  - deepcopy() 함수를 이용해 대입(깊은 복사, deep copy).

    ex) import copy; a = [1, 2, 3]
        b = copy.deepcopy(a)