Study Title _____ Date _____

Chapter4. 반복문

4-1 리스트와 반복문

-리스트란 여러가지 자료를 저장할 수 있는 자료입니다

ex)

```
array = [273, 32, 103, “문자열”, True]
print(array)
>> [273, 32, 103, “문자열”, True]
```

리스트 선언하고 요소에 접근하기

-대괄호[]안에 넣는 자료를 요소 혹은 element라고 부릅니다. 그리고 []안의 인덱스는 다음과 같습니다

```
list_a = [ 0, 1, 2, 3, 4, ……]
```

ex)

```
list_a = [273, 32, 103, “문자열”, True, False]
list_a[0]
>>273
list_a[1:3]
>>[32, 103]
list_a[3][0]
>>’문’
list_a[0] = “변경”
list_a
>> [‘변경’, 32, 103, “문자열”, True, False]
```

리스트 연산자 : 연결+, 반복*, len()

ex)

```
list_a = [1, 2, 3]
list_b = [4, 5, 6]
print(list_a + list_b)
>>[1, 2, 3, 4, 5, 6]
print(list_a * 3)
>>[1, 2, 3, 1, 2, 3, 1, 2, 3]
print(len(list_a)) #요소의 개수를 세어줍니다
>>3
```

리스트에 요소 추가하기 : append, insert

-리스트에 요소를 추가할 때는 두가지 방법이 있습니다

ex)

```
list_a = [1, 2, 3]
list_a.append(4)   #리스트 뒤에 요소 추가
print(list_a)
>>[1, 2, 3, 4]
list_a.insert(0, 10)   #리스트 중간에 요소 추가
print(list_a)
>>[10, 1, 2, 3, 4]
list_a.extend([5, 6, 7])   #한번에 여러요소 추가
print(list_a)
>>[10, 1, 2, 3, 4, 5, 6 7]
```

리스트에 요소 제거하기

-리스트의 요소를 제거하는 방법에는 크게 두가지로 나뉩니다

1.인덱스로 제거하기 : del, pop

2.값으로 제거하기 : remove

ex)

```
list_a = [0, 1, 2, 3, 4, 5]
del list_a[1]   #인덱스로 제거, 범위설정가능
print(list_a)
>>[0, 2, 3, 4, 5]
list_a.pop(2)   #인덱스로 제거
print(list_a)
>>[0, 2, 4, 5]

list_c = [1, 2, 1, 2]
list_c.remove(2)   #값으로 제거,먼저발견되는거부터 제거
print(list_c)
>>[1, 1, 2]
list_c.clear()   #모든 요소 제거
print(list_c)
>>[]
```

리스트 내부에 있는지 확인하기 : in/not in 연산자

-특정 값이 리스트 내부에 있는지 확인하는 방법입니다 결과는 불값으로 나옵니다

ex)

```
list_a = [273, 32, 103, 57, 52]
273 in list_a
>>True
99 in list_a
>>False
273 not in list_a
>>False
99 not in list_a
>>True
```

for 반복문 : 리스트와 함께 사용하기

ex)

```
array = [273, 32, 103]
for element in array :
    print(element)
>>273
  32
  103
```

Study Title _____ Date _____

4-2 딕셔너리와 반복문

-리스트가 인덱스를 기반으로 값을 저장하는것이라면 딕셔너리는 키를 기반으로 값을 저장합니다

딕셔너리 선언하기

-딕셔너리는 중괄호{}로 선언하며 키는 문자열, 숫자, 불등으로 선언할수 있지만, 일반적으로는 문자열을 사용합니다

ex)

```
dict_a = {
      "name" : "어밴저스 엔드게임",
      "type" : "히어로 무비"
      }
print(dict_a)
>>{'name' : '어밴저스 엔드게임', 'type' : '히어로 무비'}
# 키값을 선언할때 ""을 빼먹으면 NameError 발생
print(dict_a["name"])
>>'어밴저스 엔드게임'
print(dict_a["type"])
>>'히어로 무비'
dict_a["name"] = "스파이더맨"
print(dict_a["name"])
>>'스파이더맨'
```

딕셔너리에 값 추가하기/제거하기

-딕셔너리에 값을 추가할때는 다음과 같이 입력합니다

딕셔너리[새로운키] = 새로운값

ex)

```
dictionary = {
      "name" : "망고절임"
      "type" : "당절임"
      "ingredient" : ["망고", "설탕", "치자황색소"]
}
dictionary["price"] = 5000
print(dictionary)
>>{'name' : '망고절임', 'type' : '당절임', 'ingredient' : ['망고', '설탕', '치자황색소'], 'price' : 5000}
#기존에 존재하고있는 키 값을 넣으면 기존의 값을 대채합니다
del dictionary["ingredient"]
print(dictionary)
>>{'name' : '망고절임', 'tpye' : '당절임', 'pirce' : 5000}
#딕셔너리에 존재하지 않는 키에 접근하면 KeyError가 발생합니다
```

딕셔너리 내부에 있는 키가 있는지 확인하기

in 키워드

-딕셔너리 내부에 키가 있는지 확인할때 사용합니다

ex)

```
dictionary = {
      "name" : "망고",
      "type" : "당절임"
}
print("name" in dictionary)
>>True
print("밥" in dictionary)
>>False
```

get()함수

-키에대한 값을 불러올수있지만, 만약 없다면 KeyError가 아니라 None을 출력합니다

ex)

```
print(dictionary.get("name"))
>>'망고'
print(dictionary.get("price"))
>>None
```

for 반복문 : 딕셔너리와 함께 사용하기

-for 키 변수 in 딕셔너리 :
    코드

ex)

```
dictionary = {
      "name" : "망고",
      "type" : "당절임"
}
for key in dictionary :
      print(key, ":" , dictionary[key])
>>name : 망고
   type : 당절임
```

Study Title _____ Date _____

4-3 반복문과 while 반복문

범위

-리스트,딕셔너리 외에 for반복문과 함께 많이 사용되는 범위자료형의 사용법에 대해 알아보겠습니다.

ex)

```
print(list(range(5)))
>>[0, 1, 2, 3, 4]
print(list(range(0, 5)))
>>[0, 1, 2, 3, 4]
print(list(range(0, 10, 2)))
>>[0, 2, 4, 6, 8]
#range()함수의 매개변수로는 반드시 정수를
입력해야합니다
```

for반복문 : 범위와 함께 사용하기

-for반복문과 범위를 조합하는 방법을 살펴보겠습니다.

ex)

```
for i in range(5):
      print(str(i) + "=반복 변수")
>>0 = 반복 변수
    1 = 반복 변수
    2 = 반복변수
    3 = 반복변수
    4 = 반복변수
```

for반복문 : 리스트와 범위 조합하기

ex)

```
array = [273, 32, 103, 57, 52]

for i in range(len(array)):
      print("{}번째 반복: {}".format(i, array[i]))
>>0번째 반복 : 273
    1번째 반복 : 32
    2번째 반복 : 103
    3번째 반복 : 57
    4번째 반복 : 52
```

for 반복문 : 반대로 반복하기

ex)

```
for i in range(4, -1, -1)
      print("현재 반복 변수 : {}".format(i))
>>현재 반복 변수 : 4
   현재 반복 변수 : 3
   현재 반복 변수 : 2
   현재 반복 변수 : 1
   현재 반복 변수 : 0
#reversed()함수 사용방법
```

ex)

```
for i in reversed(range(5)):
      print("현재 반복 변수: {}".format(i))
>>현재 반복 변수 : 4
   현재 반복 변수 : 3
   현재 반복 변수 : 2
   현재 반복 변수 : 1
   현재 반복 변수 : 0
#이 함수를 사용 하면 범위가 반대로 뒤집어짐
```

while 반복문

-기본적인 형태는 다음과 같습니다

```
""""
while 불 표현식 :
      문장
""""
```

불 표현식이 참인 동안 문장을 계속 반복하고 거짓이나와야지만 반복을 멈춥니다

ex)

```
while True :
      print(".", end="")
#end가 "\n"이라 줄바꿈이 일어나지만 빈문자열로
바꿔서 줄바꿈이 일어나지 않게 합니다
>>.....
.....
```

while 반복문 : for 반복문처럼 사용하기

ex)

```
i=0
while i < 10:
      print("{}번째 반복입니다.".format(i))
      i += 1
>>0번째 반복입니다
   1번째 반복입니다
   2번째 반복입니다
   3 번째 반복입니다
   4 번째 반복입니다
   5 번째 반복입니다
   6 번째 반복입니다
   7 번째 반복입니다
   8 번째 반복입니다
   9 번째 반복입니다
```

Study Title _____ Date _____

while반복문 : 상태를 기반으로 반복하기

ex)

```
list_test = [1, 2, 1, 2]
value = 2

while value in list_test:
    list_test.remove(value)

print(list_test)
>>[1, 1]
```

while반복문 : 시간을 기반으로 반복하기

-시간을 기반으로 반복하려면 유닉스 타임이라는 개념을 알아야 합니다. 유닉스 타임이란 세계표준시로 1970년 1월 1일 0시를 기준으로 몇초가 지났는지 정수로 나타낸것을 말합니다. 파이썬에서 유닉스 타임을 구할때 다음과 같은 코드를 사용합니다.

```
import time
time.time()
>>1557241486
```

ex)

```
import time
number = 0
target_tick = time.time() + 5
while time.time() < target_tick:
    number += 1
print("5초 동안 {}번 반복했습니다.".format(number))
>>5초 동안 14223967번 반복했습니다.
```

while반복문 : break키워드/continue키워드

-break 키워드는 반복문을 벗어날때 사용하는 키워드입니다.

ex)

```
i=0
while True:
    print("{}번째 반복문입니다.".format(i))
    i = i + 1
    input_text = input(">종료하시겠습니까?(y/n):")
    if input_text in ["y", "Y"]
        print("반복을 종료합니다.")
        break
>>0번째 반복문입니다.
  > 종료하시겠습니까?(y/n): y
  반복을 종료합니다.
```

continue키워드는 현재 반복을 생략하고, 다음 반복으로 넘어갈때 사용하는 키워드입니다.

ex)

```
numbers = [5, 15, 6, 20, 7, 25]
for number in numbers:
    if number < 10:
        continue
    print(number)
>>15
  20
  25
```

4-4 문자열, 리스트, 딕셔너리와 관련된 기본 함수

리스트에 적용할 수 있는 기본 함수 : min(), max(), sum()

-min,max,sum함수들은 리스트를 매개변수로 넣어 사용하는 함수입니다

ex)

```
number = [103, 52, 273, 32, 77]
min(numbers)
>>32
max(numbers)
>>273
sum(numbers)
>>537
```

reversed()함수로 리스트 뒤집기

ex)

```
list_a = [1, 2, 3, 4, 5]
list_reversed = reversed(list_a)
print(list(list_reversed))
>>[5, 4, 3, 2, 1]
for i in reversed(list_a) :
    print("-", i)
>>-5
  -4
  -3
  -2
  -1
#확장슬라이싱을 이용한 뒤집기
number = [1, 2, 3, 4, 5]
print(number[::-1])
>>[5, 4, 3, 2, 1]
```

Study Title _____ Date _____

enumerate()함수와 반복문 조합하기

-리스트의 요소를 반복할때 현재 인덱스가 몇번째인지 확인해야 하는 경우가 많은데,이런 코드를 쉽게작성할 수 있도록 합니다

ex)

```
example_list = ["요소A", "요소B", "요소C"]

print(list(enumerate(example_list)))
>>[(0, '요소A'), (1, '요소B'), (2, '요소C')]

#enumerate()함수를 사용하면 반복 변수를 이런 형태로 넣을 수 있습니다.
for i, value in enumerate(example_list) :
    print("{}번째 요소는 {}입니다".format(i, value))
>>0번째 요소는 요소A입니다.
  1번째 요소는 요소B입니다.
  2번째 요소는 요소C입니다.
```

딕셔너리의 items() 함수와 반복문 조합하기

-enumerate함수와 비슷하게 딕셔너리는 items() 함수와 함께 사용하면 키와 값을 조합해서 쉽게 반복문을 작성할 수 있습니다.

ex)

```
example_dictionary = {
    "키A" : "값A",
    "키B" : "값B",
    "키C" : "값C",
}
print(example_dictionary.items())
>>dict_items([('키A', '값A'), ('키B', '값B'), ('키C', '값C')])

for key, element in example_dictionary.items() :
    print("dictionary[{}] = {}".format(key, element))
>>dictionary[키A] = 값A
  dictionary[키B] = 값B
  dictionary[키C] = 값C
```

리스트 내포

-프로그램을 만들때는 반복문을 이용해 리스트를 재조합하는 경우가 많습니다.

ex)

```
array = [i*i for i in range(0, 20 ,2)]
print(array)
>>[0, 4, 16, 36, 64, 100, 144, 196, 256, 324]
#이런 구문을 리스트 내포라고 부릅니다.
리스트 이름 = [표현식 for 반복자 in 반복할 수 있는것]
```

-뒤에 if구문을 넣어 조건을 조합할 수도 있습니다.

ex)

```
array = ["사과", "자두", "초콜릿", "바나나", "체리"]
output = [fruit for fruit in array if fruit != "초콜릿"]
print(output)
>>['사과', '자두', '바나나', '체리']
리스트이름 = [표현식 for 반복자 in 반복할 수 있는것 if 조건문]
```

문자열 연결하기

```
#괄호로 문자열 연결하기
ex)
test = (
    "이렇게 입력해도"
    "하나의 문자열로 연결되어"
    "생성됩니다."
print(test)
>> 이렇게 입력해도 하나의 문자열로 연결되어 생성됩니다.

#문자열의 join()함수
-문자열.join(문자열로 구성된 리스트)
ex)
print("::".join(["1", "2", "3", "4", "5"]))
>>1::2::3::4::5
```

이터레이터

반복문의 구문은 다음과 같습니다.
[for 반복자 in 반복할 수 있는것]

여기서 반복할 수 있는 것을 이터러블이라고 하고, 이터러블 중에서 next()함수를 적용해 하나하나 꺼낼 수 있는 요소를 이터레이터라고 합니다.

예로써,reversed()함수와 enumerate()함수의 리턴값은 이터레이터 입니다.