푸딩 2주차 스터디

1	조건문
2	반복문
3	리스트, 튜플, 딕셔너리

if/else 문의 기본구조

```
if 조건문:
    수행할 문장1
    수행할 문장2
    ...
else :
    수행할 문장A
    수행할 문장B
    ...
```

- 조건문이란 참과 거짓을 판단하는 문장
- 조건문을 테스트해서 참이면 if문 바로 다음 문장을 수행, 거짓이면 else문 다음 문장 수행
- else문은 if문 없이 독립적 사용 불가
- if/else 문에 속하는 모든 문장은 들여쓰기 필수
- if/else 문 뒤에는 반드시 콜론(:)을 붙여준다

비교 연산자

```
### 비교연산자
a=100
b=200
a!=b
a>b
a<b
a>=b
a<=b
```

- 비교 연산자를 넣고 조건문을 수행하면 참과 거짓을 판단해 각각 True, False를 반환

[표 2-7] 파이썬의 비교 연산자와 설명

비교연산자	설명	a = 100, b = 200일 때
==	두 피연산자의 값이 같으면 True를 반환한다.	a == b는 False
!=	두 피연산자의 값이 다르면 True를 반환한다.	a != b는 True
>	왼쪽 피연산자가 오른쪽 피연산자보다 클 때 True를 반환한다.	a > b는 False
<	왼쪽 피연산자가 오른쪽 피연산자보다 작을 때 True를 반환한다.	a < b는 True
>=	왼쪽 피연산자가 오른쪽 피연산자보다 크거나 같을 때 True를 반환한다.	a >= b는 False
<=	왼쪽 피연산자가 오른쪽 피연산자보다 작거나 같을 때 True를 반환한다.	a <= b는 True

and, or, not

```
###or 연산자
a = 100
b = 200
if a>100 or b>100 :
    print('참')
else :
    print('거짓’)

###and 연산자
a = 100
b = 200
if a>100 and b>100 :
    print('참')
else :
    print('거짓')
```

▪ or 연산자
: x 와 y 중 하나만 참이어도 참이다

▪ and 연산자
: x 와 y 모두 참이어야 참이다

▪ not 연산자
: x 가 거짓이면 참이다

elif

```
if 조건문:
    수행할 문장1-1
    수행할 문장1-2
    ...
elif 조건문2 :
    수행할 문장2-1
    수행할 문장2-2
    ...
elif 조건문 N :
    수행할 문장N-1
    수행할 문장N-2
    ...
...
else :
    수행할 문장A
    수행할 문장B
    ...
```

- elif문은 다양한 조건을 판단하기 용이, 다중조건판단
- 개수에 재한 없이 사용가능
- elif 문에 속하는 모든 문장은 들여쓰기 필수
- elif 문 뒤에는 반드시 콜론(:)을 붙여준다

조건부 표현식

```
###조건부표현식

message = 'success' if score >= 60 else
'failure’
```

▪ 조건부 표현식의 정의
: 조건문이 참인 경우 if 조건문 else 조건문이 거짓인 경우

▪ 가독성 유리, 한 줄로 작성할 수 있어 활용성이 좋음

while 문의 기본구조

```
### while문의 기본구조

while 조건문:
    수행할 문장1
    수행할 문장2
    수행할 문장3
    ...
    break
```

- 반복해서 문장을 수행해야 할 경우 while문
- while 문에 속하는 모든 문장은 들여쓰기 필수
- while 문 강제로 빠져나가기
: break

continue

```
###continue문

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

- continue
: while문을 빠져 나가지 않고 while문의 맨 처음으로 돌아감

- else문은 if문 없이 독립적 사용 불가

- if/else 문에 속하는 모든 문장은 들여쓰기 필수

무한루프

```
###무한루프

while True :
    print('Ctrl+C를 눌러야 while문을 빠져나갈 수 있습니다.')
```

▪while 문의 조건문이 True인 경우 항상 참
: 무한히 수행

▪ Ctrl+C 를 눌러서 while 문을 빠져나갈 수 있음

for문의 기본구조

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

one
two
three
```

- 리스트나 튜플, 문자열의 첫 번째 요소부터 마지막요소까지 차례로 변수에 대입

- continue 문을 만나면 for문의 처음으로 돌아가게 된다

for문 에서의 range 함수

```
###range함수

a = range(1,11)
print(a)
#1,2,3,4,5,6,7,8,9,10
```

- 시작과 끝 숫자 지정

: range(시작 숫자, 끝 숫자)
이때 끝 숫자는 포함 되지 않는다

for문 에서의 range 함수

```
###예제
###구구단을 2단부터 9단까지 차례대로 출력해
보자

#A.
for i in range(2,10):
    for j in range(1,10):
        print(i*j, end= ' ')
    print('')
```

- for과 range 함수를 사용하여 구구단을 출력해보자

리스트 내포

```
###리스트 내포

a= [1,2,3,4]
result =[num * 3 for num in a]
print(result)
```

- 리스트 내포의 일반 문법
: [표현식 for 항목 in 반복 가능 객체 if 조건]

- for문 2개 이상 사용 가능

리스트

```
###리스트

#리스트명 = [요소1, 요소2, 요소1, ...]

a=[]
b=[1,2,3]
c=['life','is','too','short']
d=[1,2,'life','is']
e=[1,2,['life','is']]
```

- 리스트를 만들 때는 대괄호([])로 감싸주고, 각 요솟값은 쉼표(,)로 나타낸다

- 리스트 안에는 어떤 자료형도 포함 가능

리스트 인덱싱

```
###리스트 인덱싱

a=[1,2,3,['a','b','c']]
print(a[0])
print(a[-1])
print(a[3])
print(a[3][0])
```

- 문자열처럼 인덱싱 적용가능
- 리스트에 내포된 리스트에서 값 도출
:리스트[i][j]...
- 리스트[-1] : 리스트의 마지막 요소

리스트 슬라이싱

```
###리스트 슬라이싱

a = [1,2,3,4,5]
a[0:2] #[1,2]
a[:2] #[1,2]
a[2:] #[3,4,5]
```

- 문자열 슬라이싱과 동일함
- 중첩된 리스트에서도 동일하게 적용

리스트 연산

```
###리스트연산

a=[1,2,3]
b=[4,5,6]
print(a+b) #[1,2,3,4,5,6]
print(a*3) #[1,2,3,1,2,3,1,2,3]
print(len(a)) #3
```

▪ 리스트 더하기(+)
: 2개의 리스트를 합치는 기능

▪ 리스트 반복하기(*)
: 리스트가 n번 반복

▪ 리스트 길이 구하기
: len 함수 사용

리스트 수정과 삭제

```
###리스트 수정과 삭제

a=[1,2,3]
a[2] = 4
del a[1]
del a[2:]
```

- 리스트 수정
: 리스트[x] = 새로운 값

- 리스트 삭제
: del 함수 사용, del 객체로 사용

리스트 append()

```
###append

a=[1,2,3]
a.append(4)
print(a) #[1,2,3,4]
```

- append의 사전적 의미
: 첨부하다, 덧붙이다

- append(x)는 리스트 맨 마지막에 x를 추가하는 함수

리스트 sort()

```
###sort

a = [1,4,3,2]
a.sort()
print(a) #[1,2,3,4]
```

- sort
: 리스트 요소를 순서대로 정렬해준다

- 숫자는 오름차순, 문자는 알파벳 순서로 정렬

리스트 reverse()

```
###reverse

a = [1,4,3,2]
a.reverse()
print(a) #[2,3,4,1]
```

- reverse
: 리스트를 역순으로 뒤집어준다

리스트 index()

```
###index
a=[1,2,3]
a.index(3) #2
a.index(1) #0
```

- index
: 리스트에 x 값이 있으면 위치 값을 돌려준다

리스트 insert()

```
###insert

a=[1,2,3]
a.insert(0,4)
print(a) #[4,1,2,3]
```

- insert(a,b)는 리스트의 a번째 자리에 b를 삽입하는 함수
- 파이썬에서는 숫자를 0부터 센다는 것을 기억

리스트 remove()

```
###remove

a=[1,2,3,1,2,3]
a.remove(3)
print(a) #[1,2,1,2,3]
```

- 리스트에서 첫 번째로 나오는 x를 삭제하는 함수
- x값이 2개 이상 있을 경우 첫번째 하나만 제거

리스트 pop()

```
###pop

a=[1,2,3]
a.pop() #3
print(a) #[1,2]

a=[1,2,3]
a.pop(1) #2
print(a) #[1,3]
```

- pop()
: 리스트의 맨 마지막 요소를 돌려주고 그 요소는 삭제

- pop(x)
:리스트의 x번째 요소를 돌려주고 그 요소는 삭제

리스트 count()

```
###count

a =[1,2,3,1]
a.count(1) #2
```

- count
: 리스트 안에 x가 몇 개 있는지 그 개수를 돌려주는 함수

리스트 extend()

```
###extend

a=[1,2,3]
a.extend([4,5])
print(a) #[1,2,3,4,5]
```

- extend(x)에서 x에는 리스트만 올 수 있으며, 원래의 a 리스트에 x 리스트를 더함
- a.extend([4,5])는 a+=[4,5]와 동일

튜플

```
###튜플

t1 = ()
t2 = (1,)
t3 = (1,2,3)
t4 = 1,2,3
t5 = (1,2,(1,2,3))
```

- 튜플은 소괄호()로 둘러싼다
- 튜플은 그 값을 바꿀 수 없다
- 한 개의 요소만 가질 때 그 요소 뒤에 콤마(,)를 반드시 붙여야한다
- 괄호()를 생략해도 무방하다

튜플 다루기

```
###튜플다루기

t1 = (1,2,'a','b')
print(t1[0])
print(t1[1:])
t2 = (3,4)
print(t1 + t2)
print(t2 *3)
print(len(t1))
```

- 인덱싱하기
- 슬라이싱하기
- 튜플 더하기
- 튜플 곱하기
- 튜플 길이구하기

딕셔너리

```
###딕셔너리

{key1:value1, key2:value2, key3:value3…}
```

- 딕셔너리는 key와 value를 한 쌍으로 갖는 자료형

- 순차적으로 해당 요솟값을 구하지 않고 key를 통해 value 값을 구한다

딕셔너리 요소 추가,삭제

```
###딕셔너리 요소 추가, 삭제

a ={1,'a'}
a[2]= 'b'
print(a) #{1:'a', 2:'b’}
del a[1]
print(a) #{1:'a'}
```

- 딕셔너리 요소 추가

: a[key] = 새 요소

- 딕셔너리 요소 삭제

: del a[key]
지정한 key 값에 해당하는 {key:value} 쌍 삭제

딕셔너리

```
###딕셔너리 호출
a = {1:'a', 2:'b'}
print(a[1]) #'a'
print(a[2]) #'b'
```

- key를 이용하여 value 얻기
: '딕셔너리 변수 이름[key]' 사용

keys(), values()

```
###keys, values

a = {1:'a', 2:'b', 3:'c'}
a.keys()
a.values()
```

- keys()
: 딕셔너리의 key만을 모아서 dict_keys 객체를 돌려준다

- values()
: 딕셔너리의 value 만을 모아서 dict_values 객체를 돌려준다

value쌍 얻기, 지우기

```
###딕셔너리 쌍 얻기, 지우기

a = {1:'a', 2:'b', 3:'c'}
a.items()
a.clear()
print(a) #{}
```

▪ item 함수
: key 와 value값의 쌍을 튜플로 묶은 값을 dict_items 객체로 돌려준다

▪ clear 함수
: 딕셔너리 안의 모든 요소를 삭제

딕셔너리 get()

```
###get
a = {1:'a', 2:'b', 3:'c'}
a.get(1) #'a'
a.get(4) #None
```

- get(x) 함수
: x라는 key에 대응되는 value를 돌려준다

- 존재하지 않는 키로 값을 가져오려 할 경우 'None'을 돌려준다

- 딕셔너리 안에 찾으려는 키 값이 없을 경우 get(x) '디폴트 값'을 사용하면 편리

딕셔너리 in

```
###in
a = {1:'a', 2:'b', 3:'c'}
print('2' in a) #True
print('5' in a) #False
```

- 'key' in a 를 호출하면 참 또는 거짓으로 돌려준다

- 딕셔너리 안에 존재 : True
딕서너리 안에 존재하지 않음 :False