제어문

조건문

조건에 따라 분기하는 if 문

- 조건에 따라 코드를 다르게 실행하려면 if 문을 이용
- if 문에서는 지정한 조건에 따라 다르게 분기해 명령을 수행
- if 문에서는 조건의 만족 여부에 따라서 코드 수행 결과가 달라짐

조건에 따라 분기하는 if 문

단일 조건에 따른 분기(if)

- 단일 조건에 따라서 분기하는 가장 기본적인 if 문의 구조

```
If <조건문> :
    <코드 블록>
```

- if 문에서〈조건문>을 만족하면(즉, 참이면) 그 아래의〈코드 블록>을 수행하고, 만족하지 않으면 (즉, 거짓이면) 수행하지 않음
- 〈조건문> 다음에는 콜론(:)을 입력
- 다음 줄에서〈코드 블록>을 입력할 때 키보드의 탭(Tab)이나 공백을 이용해 들여쓰기
- 〈코드 블록>의 코드는 한 줄일 수도 있고 여러 줄일 수도 있음

조건에 따라 분기하는 if 문

단일 조건에 따른 분기(if)

- if 문의〈조건문〉에는 조건을 판단하기 위해 비교 연산 및 논리 연산을 이용
- 비교 연산과 논리 연산 여러 개를 조합해 사용

비교 연산자	의미	활용 예	설명
==	같다	a==b	a는 b와 같다
!=	같지 않다	a!=b	a는 b와 같지 않다
<	작다	a<b	a는 b보다 작다
>	크다	a>b	a는 b보다 크다
<=	작거나 같다	a<=b	a는 b보다 작거나 같다
>=	크거나 같다	a>=b	a는 b보다 크거나 같다

논리 연산자	의미	활용 예	설명
and	논리곱	A and B	A와 B 모두 참일 때만 참이고 나머지는 거짓
or	논리합	A or B	A와 B 중 하나라도 참이며 둘 다 거짓일 때 거짓
not	논리부정	Not A	A가 참이며 거짓이고, 거짓이면 참

조건에 따라 분기하는 if 문

단일 조건에 따른 분기(if)

- if 문을 이용해 변수 때 값이 90보다 크거나 같으면 'Pass'를 출력

```
[ ] x = 95
    if x >= 90:
        print("Pass")
```

- 변수 x에는 95가 할당돼 90보다 크거나 같으므로 'Pass'를 출력

조건에 따라 분기하는 if 문

단일 조건 및 그 외 조건에 따른 분기(if ~ else)

- if 문 구조에서는〈조건문>을 만족하면 <코드 블록>을 수행하지만 만족하지 않는 경우는 아무것도 수행하지 않았음
- if 문에서〈조건문〉의 만족 여부에 따라 코드를 다르게 수행하려면 'if ~ else' 구조의 조건문을 이용

조건에 따라 분기하는 if 문

단일 조건 및 그 외 조건에 따른 분기(if ~ else)

If 〈조건문〉 :

<코드 블록 1>

Else :

<코드 블록 2>

- 'if ~ else' 구조의 조건문에서 <조건문>을 만족하면 <코드 블록1>을 수행하고, 만족하지 않으면 〈코드 블록2>를 수행
- <조건문〉다음에 콜론(☺을 입력했듯이 else 다음에도 콜론(☺을 입력
- 'else:' 다음의〈코드 블록 2〉도 들여쓰기를 함
- Else는 단독으로 쓸 수 없고 반드시 if와 함께 써야 함

조건에 따라 분기하는 if 문

단일 조건 및 그 외 조건에 따른 분기(if ~ else)

- 변수 x의 데이터가 90보다 크거나 같으면 'Pass'를 출력하고 90보다 작으면 'Fail'을 출력

```
[ ] x = 75
    if x >= 90:
        print("Pass")
    else:
        print("Fail")
```

  - 변수 x의 값이 75로 90보다 작으므로 'Fail'을 출력

조건에 따라 분기하는 if 문

여러 조건에 따른 분기(if ~ elif ~ else)

- 여러 조건에 따라 코드를 각각 다르게 수행하려면, if ~ elif ~ else‘ 조건문을 이용
- elif는 필요에 따라 여러 개를 사용할 수도 있음

조건에 따라 분기하는 if 문

여러 조건에 따른 분기(if ~ elif ~ else)

```
If <조건문 1> :
        <코드 블록 1>
elif <조건문 2> :
        〈코드 블록 2〉
        .
elif <조건문 n>:
        <코드 블록 n>
else :
        〈코드 블록 m〉
```

조건에 따라 분기하는 if 문

여러 조건에 따른 분기(if ~ elif ~ else)

- 'if ~ elif - else' 구조의 조건문에서 우선<조건문1>이 만족하는지 검사하고 만족하면 <코드 블록 1>을 수행
- 만약 <조건문 1>을 만족하지 않는다면 그 다음 조건문인 <조건문 2>를 검사하고 만족하면 <코드 블록 2>를 수행
- 만약 <조건문 2>도 만족하지 않는다면 그 다음 조건문을 검사함
- 마지막 조건문인〈조건문 n>을 검사하고 만족하면 <코드 블록 n>을 수행
- 마지막 조건문까지 만족하지 않으면 'else:' 아래의 <코드 블록 m>을 실행
- 여기서 'else:' 이후는 생략하고 'if ~ elif'만 이용할 수도 있음

조건에 따라 분기하는 if 문

여러 조건에 따른 분기(if ~ elif ~ else)

- 변수 x 값이 90보다 크거나 같으면 'Very good'을 출력하고 80보다 크거나 같고 90보다 작으면 'Good'을 출력하고 80보다 작으면 'Bad'를 출력

```
[ ] x = 85
    if x >= 90:
        print("Very good")
    elif  (x >= 80) and (x < 90):
        print("Good")
    else:
        print("Bad")
```

조건에 따라 분기하는 if 문

여러 조건에 따른 분기(if ~ elif ~ else)

- 비교 연산과 논리 연산을 조합해서 만든 조건
- (x >= 80) and (x < 90)'은 '(80 <= x) and (x < 90)'과 같음
- 파이썬에서는 '(80 <= x < 90)'처럼 더 직관적으로 표현

```
[ ] x = 85
    if x >= 90:
        print("Very Good")
    elif  80 <= x < 90:
        print("Good")
    else:
        print("Bad")
```

조건에 따라 분기하는 if 문

중첩 조건에 따른 분기

```
If <조건문 1> :
        If <조건문 1-1>:
                <코드 블록 1-1>
        else :
                <코드 블록 1-2>
elif <조건문 2> :
        <코드 블록 2>
else :
        <코드 블록 3>
```

조건에 따라 분기하는 if 문

중첩 조건에 따른 분기

- 중첩 조건문에서 <조건문 1>을 만족하면 다시 <조건문 1-1>을 만족하는지 검사한 후에 만족한다면 <코드 불록 1-1>을 수행하고, 만족하지 않으면 <코드 블록 1-2>를 수행
- 만약 <조건문 1>을 만족하지 않는다면 <조건문 2>를 만족하는지 검사한 후 만족한다면 <코드 블록 2>를 수행하고, 만족하지 않으면 마지막으로 <코드 블록 3>을 수행
- 조건문 if 안에 다시 'if ~ else' 조건문 들여 쓰기를 함
- if 문 안에 또 다른 if 문을 사용해 두 개의 중첩 조건문을 만들었지만 더 많은 중첩 조건문을 만들 수도 있음

조건에 따라 분기하는 if 문

중첩 조건에 따른 분기

- 변수 x가 'x >= 90'을 만족하는 가운데 x가 100이면 'Perfect'를 출력하고, 그렇지 않으면 'Very good'을 출력하는 조건문을 추가

```
[ ] x = 100
    if x >= 90:
        if x==100 :
            print("Perfect")
        else:
            print("Very Good")
    elif (x >= 80) and (x < 90):
        print("Good")
    else:
        print("Bad")
```

  - 변수 x에는 100이 할당돼 첫 번째 조건(x >= 90)을 만족하고 다시 그 안의 조건문에서 'x==100'을 만족해 'Perfect'가 출력

반복문

반복문

- 코드를 작성할 때 지정된 횟수만큼 작업을 반복하거나 지정된 조건을 만족하면 작업을 반복해야 할 때
- 작업을 반복적으로 수행하는 구문을 반복문
- 파이썬에서는 반복문을 for문과 while문을 이용해 구현

반복문

반복문의 필요성

- 입력한 순서대로 코드가 실행되는 구조로 0부터 5까지 순차적으로 출력하는 코드를 작성

```
[ ] a = 0 # 변수 a를 0으로 초기화
    print(a) # 변수 a 출력

    a = a + 1 # 변수 a에 1을 더한 후, 다시 a에 대입
    print(a)  # 변수 a 출력

    a = a + 1 # 같은 코드 반복
    print(a)

    a = a + 1 # 같은 코드 반복
    print(a)

    a = a + 1 # 같은 코드 반복
    print(a)

    a = a + 1 # 같은 코드 반복
    print(a)
```

반복문

반복문의 필요성

- 변수 a에 초기값 0을 넣고 출력한 후 a에 1을 더한 결과를 다시 a에 대입한 후 결과를 출력하는 코드를 5회 반복
- 0부터 100까지 출력하는 경우를 생각한다면 똑같은 코드를 반복하는 것은 비효율적
- 반복문을 이용하면 같은 코드를 반복해서 입력하지 않고 효율적으로 코드를 작성할 수 있음

지정된 범위 만큼 반복하는 for 문

for 문의 구조

- 구조적인 특징으로 인해 'for ~ in' 문이라고도 함

```
for <반복 변수> in <반복 범위> :
    <코드 블록>
```

- for 문에서 <반복 변수>는 <반복 범위>에 따라 변하면서 <코드 블록>을 실행
- for 문을 시작하면〈반복 범위>의 첫 번째 데이터가〈반복 변수〉에 들어가고 <코드 블록>을 실행
- 다음은〈반복 범위〉의 두 번째 데이터가〈반복 변수〉에 들어가고 <코드 블록>을 또 실행
- 이런 과정을<반복 범위>의 마지막까지 반복
- <코드 블록>에서는 <반복 변수>를 이용할 수 있음
- for 문에서도〈반복 범위〉 다음에 콜론(:)을 입력하고 그 다음 줄의 <코드 블록>을 입력할 때는 들여쓰기

지정된 범위 만큼 반복하는 for 문

반복 범위 지정

- for 문에서 <반복 범위>는 리스트와 range() 함수를 이용해 지정
- 리스트 이용
- 리스트는 for 문에서〈반복 범위〉에 이용할 수 있음
- for 문에서〈반복 범위>는 출력될 숫자 리스트([0, 1, 2, 3, 4, 5])로 지정
- <반복 변수>는 a로 지정하고 <코드 블록>에서는 print() 함수로 a를 출력

지정된 범위 만큼 반복하는 for 문

반복 범위 지정

- 리스트 이용

```
[ ] for a in [0, 1, 2, 3, 4, 5]:
        print(a)
```

  - for 문이 처음 실행될 때 a에는 리스트의 첫 번째 항목인 0이 대입되고 print(a)가 실행
  - 두 번째로 for 문이 실행될 때 a에 리스트의 두 번째 항목인 1이 대입되고 print(a) 실행
  - 계속 반복해 실행되다가 a에 리스트의 마지막 항목인 5가 대입되고 print(a)가 실행된 후 반복문은 끝남
  - for 문을 이용하면 코드를 반복하지 않고 짧고 간단하게 작성

지정된 범위 만큼 반복하는 for 문

반복 범위 지정

- 문자열을 이용해 리스트를 만들고 이를 for문의 <반복 범위>로 지정
- for 문의〈반복 범위〉로 이용하 고 변수 myFriend를 <반복 변수〉로 지정한 후 <코드 블록> 에서는 print() 함수로 변수 myFriend를 출력

```
[ ] myFriends = ['James', 'Robert', 'Lisa', 'Mary']  # 리스트를 변수에 할당
    for myFriend in myFriends:
        print(myFriend)
```

  - for 문이 처음 실행될 때 myFriend에는 리스트 myFriends의 첫 번째 항목('James')이 대입 되고 print(myFriend)가 실행
  - 리스트 myFriends의 마지막 항목{'Mary')까지 순차적으로 반복하고 반복문은 끝남

지정된 범위 만큼 반복하는 for 문

반복 범위 지정

- range() 함수 이용
- for 문의 <반복 범위>를 지정하는 또 다른 방법은 범위를 반환하는 range() 함수를 이용하는 것
- 파이씬 내장 함수 range()는 for 문에서 숫자로 <반복 범위>를 지정할 때 많이 이용

지정된 범위 만큼 반복하는 for 문

반복 범위 지정

- range(start, stop, step)
- start는 범위의 시작 지점, stop은 범위의 끝 지점, step은 증감의 크기
- start부터 시작해서 stop 전까지(stop은 포함되지 않음) step만큼 계속 더해 <반복 범위>를 만듬
- start와 stop은 양의 정수, 음의 정수, 0 모두 사용할 수 있으며 step은 양의 정수와 음의 정수만 사용

지정된 범위 만큼 반복하는 for 문

반복 범위 지정

- range() 함수 이용

```
[ ] print(range(0, 10, 1))
```

- print() 함수를 이용해 range() 함수의 결과를 출력하려고 했으나 실제로 출력이 되지 않음

- range()함수로 만들어진 숫자의 리스트를 출력하려면 list() 함수를 이용하여 리스트 타입으로 변환한 후 출력

```
[ ] print(list(range(0, 10, 1)))
```

  - 출력 결과에서 range() 함수로 생성된 범위는 start부터 stop 전까지(stop을 포함하지 않음)

지정된 범위 만큼 반복하는 for 문

반복 범위 지정

- range() 함수를 이용해 for 문에서 <반복 범위>를 지정

```
[ ] for a in range(0, 6, 1):
        print(a)
```

- range(0, 6, 1)로 생성하는 범위는 0부터 6전까지 for 1씩 증가하는 값으로 {0, 1, 2, 3, 4, 5}가 됨
- for 문의 실행 결과를 보면 0부터 5까지 1씩 증가해서 출력

지정된 범위 만큼 반복하는 for 문

반복 범위 지정

- range() 함수를 이용해 for 문에서 <반복 범위>를 지정
- step을 1이 아니라 2로 조정해서 <반복 범위>를 바꾸고 싶다면

```
[ ] for a in range(0, 6, 2):
        print(a)
```

지정된 범위 만큼 반복하는 for 문

반복 범위 지정

- range() 함수를 이용할 때 start, start, step을 모두 지정했음
- step이 1인 경우에는 step을 생략해 사용할 수 있음
- range(start, stop)
- step이 1 이고 start가 0인 경우는 start 역시 생략하고 stop만 지정할 수도 있음
- range(stop)

지정된 범위 만큼 반복하는 for 문

반복 범위 지정

- range(0, 10, 1)에서 step이 1이므로 생략할 수 있어서 range(0, 10)으로 지정했고,
- 다시 start가 0이므로 생략할 수 있어서 stop만 지정해서 range(10)으로 입력

```
[ ] print(list(range(0, 10, 1)))
    print(list(range(0, 10)))
    print(list(range(10)))
```

- range(0, 10, 1), range(0, 10), range(10)이 모두 같은 것

지정된 범위 만큼 반복하는 for 문

반복 범위 지정

- range() 함수를 이용

```
[ ] print(list(range(0, 20, 5)))     # Line 1
    print(list(range(-10, 0, 2)))    # Line 2
    print(list(range(3, -10, -3)))   # Line 3
    print(list(range(0, -5, 1)))     # Line 4
```

- 첫 번째(Line 1)는 0부터 5씩 증가해 20보다 작은 숫자까지를 출력
- 두 번째(Line 2)는 -10에서 2씩 더해 0보다 작은 숫자까지 출력
- 세 번째(Line 3)는 3부터 -3 씩 증가해서 (혹은 3씩 감소해서) -10보다 큰 숫자까지 출력
- 네 번째(Line 4)는 0부터 1 씩 증가해서 -5보다 작은 숫자까지 출력한 예지만, 이런 조건을 만족하는 숫자는 없음(빈 리스트를 반환)

지정된 범위 만큼 반복하는 for 문

중첩 for 문

- for 문도 중첩 for 문 구조를 이용해 중첩 반복문을 만들 수 있음

```
for <반복 변수 1> in <반복 범위 1> :
    for <반복 변수 2〉 in 〈반복 범위 2〉 :
        〈코드 블록〉
```

  - 중첩 for 문에서〈반복 변수1〉의 첫 번째 데이터가 실행될 때 그 안에 있는 for 문을 만나게 되어 내부 for 문을 실행하고 다시〈반복 변수1〉의 두 번째 데이터가 실행될 때 내부 for 문을 수행
  - 이런 과정을 모두 거쳐서 중첩 for 문을 모두 마치면 코드가 끝남
  - 중첩 for 문에서 내부 for 문과 <코드 블록>을 입력할 때도 각각 들여쓰기

지정된 범위 만큼 반복하는 for 문

중첩 for 문

- 리스트변수 x와 y에 각각 ['x1', 'x2']와 ['y1','y2']를 할당하고
- for 문 두개를 이용해 각 요소 로 이뤄진 (x, y)의 쌍을 출력

```
[ ] x_list = ['x1', 'x2']
    y_list = ['y1', 'y2']

    print("x y")
    for x in x_list:
        for y in y_list:
            print(x,y)
```

지정된 범위 만큼 반복하는 for 문

중첩 for 문

- 중첩 for 문을 이용해 <반복 변수 1> x의 첫 번째 요소(x1)에 대해 <반복 변수 2> y의 첫 번째 요소(y1)와 두 번째 요소(y2)를 출력
- x의 두 번째 요소(x2)에 대해 〈반복 변수 2〉 y의 첫 번째 요소(y1)와 두 번째 요소(y2)를 출력
- print() 함수의 괄호에 출력하고자 하는 것을 콤마(,)로 구분해서 모두 쓰면 각각을 공백으로 구분해서 전체를 한 줄에 출력 할 수 있음

지정된 범위 만큼 반복하는 for 문

여러 개의 리스트 다루기

- 여러 개의 리스트가 있을 때 for 문을 이용해 리스트의 데이터를 다루는 방법

```
[ ]  names = ['James', 'Robert', 'Lisa', 'Mary']
     scores = [95, 96, 97, 94]
```

- 리스트가 하나라면 <반복 범위>를 리스트로 지정해 for 문을 이용할 수 있지만 리스트가 두 개이므로 리스트를<반복 범위>에 이용할 수 없음
- 리스트, 튜플, 세트, 딕셔너리의 항목 개수(데이터의 '길이'라고도 함)를 반환하는 len() 함수와 범위를 반환하는 range() 함수를 이용

지정된 범위 만큼 반복하는 for 문

여러 개의 리스트 다루기

- for 문의 <반복 범위>를 지정하고 <반복 변수>를 이용해 리스트의 요소를 하나씩 부르면 됨

```
[ ] for k in range(len(names)):
        print(names[k], scores[k])
```

- for 문에서 <반복 범위>를 알기 위해 len() 함수로 리스트의 길이(항목 개수)
- len(names)의 항목 개수는 4이고 range() 함수의 인자로 넣어서 <반복 범위>를 설정
- k는 0부터 3까지 1씩 증가하면서 두 리스트(names와 scores)의 항목을 출력

지정된 범위 만큼 반복하는 for 문

zip( )함수 이용

- 길이가 같은 리스트가 여러 개 있는 경우 for 문을 이용해도 되지만
- 같은 길이의 데이터를 하나로 묶어주는 zip() 함수를 이용해
- <반복 범위>를 지정하고
- 데이터별로 <반복 변수>를 지정해 이용할 있음

지정된 범위 만큼 반복하는 for 문

zip( )함수 이용

```
for var1, var2 in zip(list1, list2) :
    <코드 블록>
```

- zip() 함수를 이용해 for 문을 구성하면 〈반복 범위〉인 zip() 안에 있는 list1과 list2 의 항목이 각각 순서대로 동시에
- <반복 변수>인 var1, var2에 대입되고
- <코드 블록>을 수행

지정된 범위 만큼 반복하는 for 문

zip( )함수 이용

- for 문에서 zip() 함수를 사용

```
[ ] for name, score in zip(names, scores):
        print(name, score)
```

- for문에서 길이가 같은 여러 개의 리스트를 처리해야 할 때
- zip() 함수를 이용하면 좀 더 알아 보기 쉽게 코드를 작성

조건에 따라 반복하는 while 문

조건에 따라 반복하는 while 문

- 반복 수행을 할 수 있는 또 다른 방법은 while 문을 이용하는 것
- while 문은 조건에 따라 반복 여부를 결정
- 반복 범위가 정해진 반복을 수행할 경우에는 for 문을 주로 이용
- 반복 범위 없이 조건에 따라서 반복 수행 여부를 결정하는 경우에는 while 문을 주로 이용

조건에 따라 반복하는 while 문

while 문의 구조

```
while <조건문>:

    <코드 블록>
```

- while 문에서 <조건문>을 만족하면 <코드 블록>을 계속 수행하고
- <조건문>을 만족하지 않으면 <코드 블록>을 실행하지 않고 while 문을 빠져나오게 됨
- <조건문>다음에 는 콜론(:)을 쓰고 <코드 블록>은 들여쓰기

조건에 따라 반복하는 while 문

while 문의 구조

- 변수 i와 sum을 0으로 초기화
- 그 후에 while 문에서 sun이 20보다 작을 경우만 <코드 블록>을 반복해서 수행
- <코드 블록 >에서는 변수 i를 증가하고 이전의 sum과 현재의 i를 더하고 변수 i와 sum을 출력
- sum이 20 이상이 되면 while 문을 빠져나오게 됨

조건에 따라 반복하는 while 문

while 문의 구조

```
[ ]  i = 0      # 초기화
     sum = 0    # 초기화

     print("i sum")
     while (sum < 20):  # 조건 검사
         i = i + 1      # i를 1씩 증가
         sum = sum + i  # 이전의 sum과 현재 i를 더해서 sum을 갱신
         print(i, sum)  # i와 sum을 출력
```

- sum이 21로 20 이상이 되자 while 문의 <코드 블록>을 실행하지 않고 while 문을 빠져나왔음

조건에 따라 반복하는 while 문

무한 반복 while 문

- while 문을 조건이 만족하는 경우에만 수행
- 어떤 경우에는 코드 블록을 무조건 계속 반복하라고 명령을 내려야 할 때가 있음

```
while True :
```

  - while 문에서 〈조건문〉이 항상 참이므로 코드 블록을 무조건 수행

```
while True :
    print("while test")
```

  - while 문에서 <조건문> 이 항상 참일 경우 <코드 블록>에 있는 코드가 무한 반복하므로 주의가 필요

반복문을 제어하는 break와 continue

반복문을 제어하는 break와 continue

- for 문에서는 <반복 범위> 동안, while 문에서는<조건문>을 만족할 때까지 계속해서 <코드 블록>의 코드를 실행
- 반복문이 수행되고 있는 동안 에 특정 조건을 만족하는 경우 반복을 멈추고 <코드 블록>을 빠져나오거나 다음 반복을 수행하게 하려면?
- break와 continue를 이용

반복문을 제어하는 break와 continue

반복문을 빠져나오는 break

- 반복문(for 문 혹은 while 문) 안에서 break를 만나게 되면 반복문을 빠져나옴
- <코드 블록>의 구조에 따라 <코드 블록 1>이나 <코드 블록 2>는 없을 수도 있음

반복문을 제어하는 break와 continue

반복문을 빠져나오는 break

- break를 이용해 while 문을 빠져나옴

```
[ ]  k=0
     while True:
         k = k + 1  # k는 1씩 증가

         if(k > 3): # k가 3보다 크면
             break  # break로 while 문을 빠져나옴

         print(k)   # k 출력
```

- while 다음에 True 조건이 있으므로 <코드 블록>을 계속 반복
- 〈코드 블록>에서는 k가 4가 되면 지정한 조건('if(k > 3)')을 만족해서 break 명령을 수행해 while 문을 빠져나옴
- 출력 결과에 3까지만 출력되고 그 이후는 출력되지 않음

반복문을 제어하는 break와 continue

반복문을 빠져나오는 break

- for문에서 break를 이용한 코드

```
[ ] for k in range(10):
        if(k > 2):    # k 가 2보다 크면
            break     # break로 for 문을 빠져나옴

        print(k)      # k 출력
```

- k가 3이 되면 지정한 조건('if(k > 2)')을 만족해서 break 명령을 수행해 반복문을 빠져나옴
- break 명령어를 이용하면 특정 조건을 만족할 때 반복문을 멈추게 할 수 있음

반복문을 제어하는 break와 continue

다음 반복을 실행하는 continue

- 반복문 안에서 continue를 만나면 반복문의 처음으로 돌아가서 다음 반복을 진행

반복문을 제어하는 break와 continue

다음 반복을 실행하는 continue

- 반복문 안에서 continue를 만나면 반복문의 처음으로 돌아가서 다음 반복을 진행

```
[ ] for k in range(5):
        if(k == 2):
            continue

        print(k)
```

- for 문을 이용해 <반복 범위>({0,1,2,3,4}) 만큼 <코드 블록>을 실행
- 그러다 k가 2일 때는 지정 조건('if(k==2)')을 만족해 continue가 실행돼 반복문의 처음으로 돌아가서 다음 반복을 진행
- continue 다음에 있는 print (k)를 실행하지 않아 출력 결과에 2는 빠짐

반복문을 제어하는 break와 continue

while 문에서 break와 continue를 모두 사용

- while 문에서 break와 continue를 모두 사용

```
[ ]  k = 0
     while True:
         k = k + 1

         if(k == 2):
             print("continue next")
             continue
         if(k > 4):
             break

         print(k)
```

반복문을 제어하는 break와 continue

while 문에서 break와 continue를 모두 사용

- while 문에서 break와 continue를 모두 사용
  - 변수 k가 2일 경우에 continue를 실행
  - 반복문에서 continue를 만나면 그 이후의 코드는 실행하지 않고 반복문의 처음으로 가서 바로 다음 반복을 수행
  - continue 문 다음에 있는 print(k)가 수행되지 않아서 2가 출력되지 않음
  - k가 3과 4인 경우에는 print(k)가 수행하여 출력됐고
  - k가 5가 되면 지정 조건('if((k > 4)')을 만족해 break가 수행돼 while 문을 빠져나옴
  - 5 이상은 출력되지 않음

간단하게 반복하는 한 줄 for 문

간단하게 반복하는 한 줄 for 문

- 파이썬에서는 리스트, 세트, 딕셔너에서 실행할 수 있는 한 줄 for 문도 지원
- 각각 '리스트 컴프리헨션(List comprehension)', '세트 컴프리헨션(Set comprehension)', '딕셔너리 컴프리 헨션(Dictionary comprehension)'
- 컴프리헨션(Comprehension)은 우리말로 내포 (혹은 내장)라는 뜻으로서 리스트, 세트, 딕셔너리 컴프리헨션은 각각 리스트, 세트, 딕셔너리내에 코드가 내포돼 실행되는 것을 의미
- 컴프리헨션을 잘 이용하면 리스트, 세트, 딕셔너리 데이터를 반복해서 처리해야 할 때 코드를 한 줄로 작성할 수 있어서 편리

간단하게 반복하는 한 줄 for 문

리스트 컴프리헨션의 기본 구조

- 리스트 컴프리헨션의 기본적인 구조
- [〈반복 실행문〉 for <반복 변수> in 〈반복 범위〉]
- for 문에서는 〈반복 실행문〉이 'for 〈반복 변수〉 in <반복 범위>:' 다음 줄에 왔는데 한 줄 for 문에서는 <반복 실행문>이 먼저 나옴
- 콜론(:)도 이용하지 않음

간단하게 반복하는 한 줄 for 문

리스트 컴프리헨션의 기본 구조

- 1~5까지 숫자가 들어있는 리스트에서 각 항목의 숫자를 제곱하려면?

```
[ ] numbers = [1,2,3,4,5]
    square = []

    for i in numbers:
        square.append(i**2)

    print(square)
```

- 리스트 컴프리헨션 방법을 이용하면 위의 코드보다 좀 더 간단하게 코드를 작성할 수 있음

```
[ ] numbers = [1,2,3,4,5]
    square = [i**2 for i in numbers]
    print(square)
```

간단하게 반복하는 한 줄 for 문

조건문을 포함한 리스트 컴프리헨션

- 리스트 컴프리헨션은 for 문 다음에 if <조건문>을 추가할 수 있음
- [〈반복 실행문〉 for <반복 변수〉 in <반복 범위〉 if <조건문>】
- 반복문을 수행하다가 if 〈조건문>을 만족하는 경우에만 <반복 실행문>을 실행

간단하게 반복하는 한 줄 for 문

조건문을 포함한 리스트 컴프리헨션

- 리스트 컴프리헨션을 이용하지 않고 리스트의 각 항목에서 3 이상의 숫자만 제곱하도록 하려면 if 문을 포함한 for문을 작성

```
[ ]  numbers = [1,2,3,4,5]
     square = []

     for i in numbers:
         if i >= 3:
             square.append(i**2)

     print(square)
```

간단하게 반복하는 한 줄 for 문

조건문을 포함한 리스트 컴프리헨션

- 조건문을 포함한 리스트 컴프리헨션을 이용해 다시 작성

```
[ ]  numbers = [1,2,3,4,5]
     square = [i**2 for i in numbers if i>=3]

     print(square)
```

- 조건문을 포함한 리스트 컴프리헨션 방법을 이용해 한 줄로 조건에 맞는 리스트의 항목만 제곱하는 코드를 작성

정리

정리

- 조건문인 if 문
- 지정된 범위만큼 반복하는 for 문
- 조건이 맞을 때만 반복하는 while 문
- 반복문을 빠져나오는 break와 반복문의 처음으로 돌아가서 다음 반복을 수행하는 continue
- 리스트, 세트, 딕셔너리에서 이용할 수 있는 한 줄 for 문