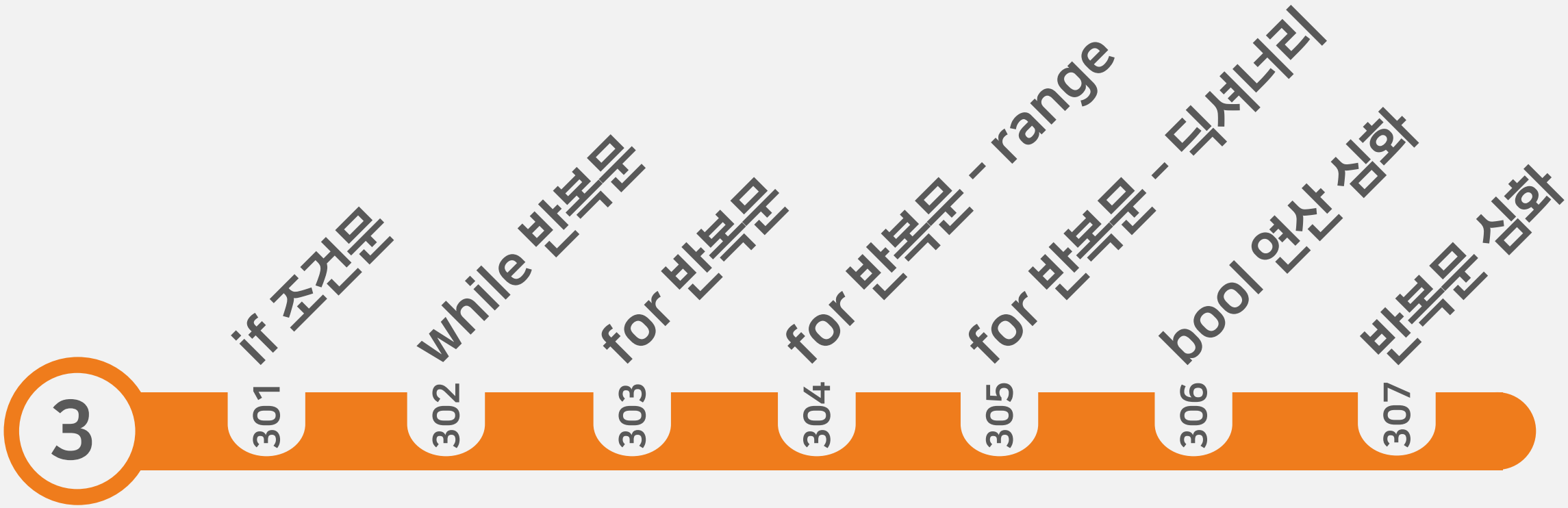
#IF 인터페이스

파이썬 스터디

3 회차

템플릿 김태균
강의 백지오

Study Route Map



if 조건문

if 문은 그 이름 그대로 특정 조건이 만족되면 특정 행동(코드)을 수행하는 조건문이다.

백문이 불여일견 이라고 아래 코드를 한번 보자.

```
>>> a=3
>>> if a > 2:
        print('big!')
출력: big!
```

if 예약어를 사용하여 if문을 사용할 수 있다.

예약어 뒤에는 조건이 오게 되는데, 위에는 변수 a가 2보다 클 경우 참이 되게 하였다.
조건에는 위와 같은 비교연산자 외에도 bool 변수나 함수 등 여러가지를 사용할 수 있다.
(이 강의의 마지막 부분에서 다룰 것이다.)

조건 뒤에는 콜론(:)을 이용해 if문의 시작임을 표현하고, 그 아래에서 들여쓰기(보통 띄어쓰기 4회)로 해당 라인은 if문에 귀속된 코드임을 표기한다.

이러한 형태는 이후 배울 반복문, 함수등에서도 적용된다.

if 조건문

if문과 함께 elif(else if의 줄임말)나 else 함수를 쓸 수 있다.

elif의 경우 if문이 참이 아닐 때, 새로운 조건을 확인한다.

else의 경우 if문이 참이 아닐 때, 무조건 작동한다.

이를 활용하면 아래와 같이 여러 조건을 달 수 있다.

```
if a < 10:
    print('a is small')
elif a < 20:
    print('a is medium')
else:
    print('a is big')
```

if 조건문

```
if a < 10:
    print('a is small')
elif a < 20:
    print('a is medium')
else:
    print('a is big')
```

위 코드에서 else나 elif 대신 그냥 if를 쓰면 되지 않을까 생각할 수도 있다.
그런데, 그렇게 되면 a가 10 이하일 때를 생각해보자. 당연히 a is small은 출력될 것이고, 이후 a가 20보다 작으니 a is mediu도 출력되고.. 문제를 알겠는가?

else와 elif를 적절히 활용하자!

조건/비교 연산자

if문에는 True나 False의 bool 타입을 갖는 모든 변수나 함수를 조건으로 사용할 수 있다.
if문에 사용가능한 몇가지 연산자를 알아보자.

연산자	역할
a > b, a < b	초과/미만 비교
a >= b, a <= b	이상/이하 비교
a == b	변수 a와 b가 같으면 True를 반환
a != b	변수 a와 b가 다르면 True를 반환
a in b	b가 a를 포함하면 True
not (조건), !(조건)	뒤에 오는 조건을 반전시킴
A or B, A || B	조건 A와 B 중 하나 이상이 True이면 True
A and B, A && B	조건 A와 B 모두가 True이면 True

is 연산자는 == 연산자와 같다고 생각할 수도 있지만 완전히 다르다!
이는 심화 과정에 속하므로 여기서 설명하지 않겠다. is를 배우기 전에 절대로 is를 쓰지 말자! (오류의 원인이 된다.)

반복문/조건문/함수의 적용 범위

```
if 조건1:
    print( ' A ' )
    if 조건2:
        print( ' B ' )
print( ' C ' )
```

파이썬에서 조건문이나 반복문, 함수의 범위는 들여쓰기로 결정된다.
백문이 불여일견, 옆의 코드를 보자.

출력들을 간단하게 a,b,c라고 부르겠다.

각 조건에 따라 각 출력의 출력 여부는 아래 표와 같다.

조건1	조건2	출력 A	출력 B	출력 C
True	False	O	X	O
False	True	X	X	O
True	True	O	X	O
False	False	X	X	O

출력 A와 조건2를 포함한 if문은 조건1 if문에 속해있다. 즉 조건2가 True여도 조건1이 False라면 애초에 조건2를 검사하지도 않는다.
출력 C는 조건문 밖에 있기에 조건1/2 어느 쪽의 영향도 받지 않는다.

while 반복문

반복문은 이름 그대로 특정 코드를 반복적으로 수행하는 것이다.

while 반복문은 if문과 같이 특정 조건이 만족된 상태에서 코드를 반복적으로 수행한다.

```
i = 0
while i < 5:
    print(i)
    i = i + 1           # i를 1씩 증가시킨다.
```

위 코드의 결과를 예상해보자.

i가 1씩 증가하다가 5가 되는 순간 i < 5는 False가 되고 반복이 종료될 것이다.

위 코드의 결과는 아래와 같다.

```
0
1
2
3
4
```

while 반복문 - 활용

특정 조건이 만족되는 동안 계속 실행되는 while문을 응용하면 사용자의 입력을 검증하는데 활용할 수 있다.

사용자로부터 10 이하의 정수를 입력받아야 한다고 가정해보자.

```
a = int(input())
while a > 10:
    print('10 이하의 정수만 입력해주세요!')
    int(input())
```

위 방법을 사용하면 사용자가 틀린 입력을 몇 번이고 입력해도 프로그램이 맞는 값을 입력받을 때까지 새 입력을 요구할 것이다!

for 반복문

반복문이 가장 많이 활용되는 부분은 단언컨데 리스트와 같은 자료의 탐색이다.

원소 3개를 갖는 리스트 myList가 있다고 생각해보자.
이 리스트의 모든 원소를 출력하려면 아래와 같이 해야할 것이다.

```
print(myList[0])
print(myList[1])
print(myList[2])
```

그런데 원소가 100개인 리스트의 출력을 저렇게 하라면 할 수 있겠는가?
while 반복문을 배웠다면 우리는 위 코드를 아래처럼 축약할 수 있다.

```
i = 0
while i < len(myList):          # len()함수는 시퀀스 자료형의 길이를 리턴한다. (여기선 3)
    print(myList[i])
```

이 얘기를 for문에서 하는 이유는, 파이썬의 for문이 기본적으로 리스트 등을 탐색하는데 특화된 반복문이기 때문이다.
(다른 언어의 for문과 작동방식이 다르다!!!)

for 반복문

for문은 in 예약어와 짝을 이뤄 아래와 같이 사용한다.

```
for element in myList:
    print(element)
```

for문은 myList에서 순서대로 값을 뽑아 element 변수에 담아준다.

for문은 리스트 뿐만 아니라 여러 값을 담는 대부분의 자료형에 쓸 수 있다!

```
for c in 'hello':
    print(c)
```

결과:

```
h
e
l
l
o
```

for문은 시퀀스 자료형 뿐 아니라
집합 자료형을 비롯한 모든 이터레이블 자료형에 사용 가능하다!
이터레이블 자료형(iterable)에 대해 검색해보는 것을 추천한다.

for 반복문 – range 함수

for 반복문과 range 함수를 함께 활용하면 다른 언어의 for문처럼 점차 증가하는 숫자를 출력할 수도 있다.

range 함수는 한 개에서 3개의 인자를 받고, range라는 클래스(자료형이라고 보면 된다.)를 반환한다.

range(a)는 0~(a-1) 까지의 정수를 원소로 갖는 range 클래스를 생성한다.
range(a,b)는 a~(b-1) 까지의 정수를 원소로 갖는 range 클래스를 생성한다.
range(a,b,c)는 a~(b-1)의 범위를 갖고, c를 공차로 갖는 등차수열을 갖는 range 클래스를 생성한다.

```
for i in range(3):
    print(i)
```

출력:
0
1
2

```
for i in range(7,9):
    print(i)
```

출력:
7
8

```
for i in range(0,7,2):
    print(i)
```

출력:
0
2
4
6

for 반복문 – 딕셔너리

for 반복문은 리스트와 튜플, 문자열 등을 탐색할 때 매우 편리하다.

그런데 딕셔너리에 담아둔 데이터를 모두 출력하고 싶다면 어떻게 할까?

딕셔너리에 사용가능한 함수인 items()와 values(), keys()를 활용하면 된다.

이름에서 알 수 있겠지만 위 함수들은 딕셔너리의 키값과 벨류값, 혹은 둘을 모두 포함한 아이템들을 반환한다.
아래 코드를 수정해서 위 함수들을 모두 실행해보자.

```
>>> myDict = {'k1':'i1','k2':'i2'}
>>> for i in myDict.items():
        print(i)
```

출력:

```
('k1', 'i1')
('k2', 'i2')
```

bool 연산 심화

if문이나 while문에는 bool 자료형 외에도 다른 변수를 조건으로 넣어줄 수 있다.

자료형에 관계없이 0이나 None 이 들어있거나, 값이 아예 지정되어 있지 않은(Null) 변수, 비어있는 리스트 등이 조건문에 들어가면 False로 취급된다.

반면 위 경우를 제외한 경우, 즉, 변수에 무언가 값이 들어있는 경우라면 그 값의 크기에 관계없이 True로 취급된다.

특정 변수에 0 이외의 값이 들어있는지 확인하려면 a != 0 대신 그냥 a만 써도 되는 것이다!

```
a = 32
if a:
    print(a)

myList = []
if myList:
    print('list is not empty')
```

반복문 심화

반복문은 매우 편리하지만, 조건이 True인 동안 계속 반복된다는 것은 불편한 점도 있다.

만약 조건이 한 개가 아니라 여러 개라면 어떨까?

while a is True and b is False and c is True……:

조건을 쭉 나열할 수는 없는 일이다.

이럴 때 활용가능한 것이 break와 continue문이다.

break문은 실행되는 순간 현재 실행중인 반복문을 종료한다.
continue문은 실행되는 순간 현재 실행중인 반복문을 최상단부터 다시 시작한다.

이 또한 백문이불여일견 코드를 보고 배워보자.

반복문 심화

```
while 조건:
    age = int(input('how old are you?')
   # 20살 이상은 아래 코드를 실행할 일이 없다고 생각해보자.
   if age >= 20:
        print('20살 이상은 아래 코드를 실행하지 않습니다!')
        break
    # 20살 이하라면 할 것들…

while 조건:
    # 코드
    a = input('입력하신 정보들이 올바른가요?(y/n)')
    # 입력한 정보가 올바르지 않다면 입력을 다시 받아야 한다.
   if a =='n':
        continue
    # 정보가 올바르면 실행할 코드
```

당장은 감이 오지 않을 수도 있지만, 코딩을 하다보면 break 혹은 continue 문을 적절히 활용해야 할 상황이 많이 생길 것이다!

반복문 심화

```
while 조건:
    age = int(input('how old are you?')
   # 20살 이상은 아래 코드를 실행할 일이 없다고 생각해보자.
   if age >= 20:
        print('20살 이상은 아래 코드를 실행하지 않습니다!')
        break
    # 20살 이하라면 할 것들…

while 조건:
    # 코드
    a = input('입력하신 정보들이 올바른가요?(y/n)')
    # 입력한 정보가 올바르지 않다면 입력을 다시 받아야 한다.
   if a =='n':
        continue
    # 정보가 올바르면 실행할 코드
```

당장은 감이 오지 않을 수도 있지만, 코딩을 하다보면 break 혹은 continue 문을 적절히 활용해야 할 상황이 많이 생길 것이다!

강의자 정보

백지오
컴퓨터공학과 19학번, 인터페이스 32기 학술차장
https://skyil.tistory.com/
https://github.com/skyil7
https://linkedin.com/in/giopaik