Python 2주차

<조건문과 반복문>

[1. 제어문]

- 제어문은 프로그래밍 문법 중 하나입니다. 문법이란 우리가 영어배울 때 배웠듯이, 문장을 만드는 규칙입니다. 프로그래밍에서도 마찬가지로 우리가 생각하는 문장을 파이썬 같은 언어로 표현하는 규칙이라고 생각하시면 됩니다. 이 제어문은 프로그램의 가장 뼈대가 되는 구조를 만드는 도구이므로, 정확히 이해하고 넘어가셔야 합니다. 파이썬의 제어문에는 기본적으로 조건문과 반복문이 있습니다.

[2. if(조건문)]

[2-1. 기본 형식]

- if는 조건문이라고도 합니다. 우선 아래 문장을 if 문으로 표현한 모습을 한번 볼게요.

> "돈이 5000원 이상 있으면 제육덮밥을 먹고, 5000원 미만이면 컵라면을 먹는다."

```
if my_money >= 5000:
      print("제육 덮밥 먹어야지 | | ~~")
else:
      print("컵라면 지겹..ㅠ")
```

if 문에 대해서 잘 몰라도 직관적으로 어떻게 표현할 수 있는지 잘 알 수 있습니다~. 여기서 이 if 문 위에 my_money라는 변수에 내 돈이 얼마들었는지를 알려주면 다음과 같은 결과를 얻을 수 있습니다~.

```
my_money = 2000
if my_money >= 5000:
    print("제육 덮밥 먹어야지 | | ~~")
else:
    print("컵라면 지겹..ㅠ")
```

```
C:\Users\Admin\PycharmProjects\Hom
컵라면 지겹..ㅠ

Process finished with exit code 0
```

형식을 자세히 뜯어보면 다음과 같습니다.

```
if <조건부>:
        수행할 문장1
        수행할 문장2
        ...
else:
        수행할 문장1
        수행할 문장2
        ...
```

if라는 글자 뒤에는 띄어쓰기 후 조건부 문장이 옵니다. 그리고 그 조건부가 만족을 한다면(True 라면) 아래에 적혀있는 문장들을 수행합니다. 혹시 그 조건부가 만족을 하지않는다면(False 라면) if 아래의 문장들을 모두 건너띄고, else 부분으로 넘어가 else 아래의 문장들을 수행합니다.
정리하자면,

```
   if 조건부 확인 -> True면? if 아래의 문장들 수행
                  -> False면? else 아래의 문장들 수행
```

여기서 주의할 점이 있습니다.
if와 else 각각 아래에 위치한 '수행할 문장 1', '수행할 문장 2', .. 는 반드시 들여쓰기로 써줘야 합니다. 이 때 들여쓰기는 "스페이스바 4번" 또

는 “탭 1번”을 눌러주면됩니다.

<잠시 딴 얘기 좀..>

여기서 파이썬 개발자들 사이에서 조금 논란이 있는 부분이 있습니다. 스페이스바 4번과 탭 1번 둘 중 뭐가 더 맞는 방법이냐.. 사실 스페이스바 4번이던, 탭 1번이던 같은 간격으로 들여써지기는 합니다. “그럼 이게 왜 논란거리가 되냐..” 라고 하실 수도 있어요. 각자 입장을 들어볼게요.

탭 입장 : 탭은 편하다. 한번만 누르면 된다. 몰라, 그냥 탭이 짱이다..

스페이스바 입장 : 근데 탭은 운영체제, 텍스트 에디터에 따라 2칸, 3칸, 또는 6칸, 8칸을 띄우게 하지 않느냐. 일관성이 없다..

최근에는 스페이스바로 들여쓰기를 하는게 맞다! 라고 많이 기울어져 있어요. PEP8 이라는 파이썬 코딩 스타일 가이드 문서 라는 곳에서는 정확하게 “스페이스바 4번으로 들여쓰기를 해라!” 라고 까지 규칙을 정해놨습니다. (근데 저는 탭 써요 ㅎㅎ.. 몰라요, 그냥 탭이 짱이에요) 하지만 가장 중요한 것은 탭과 스페이스바를 혼용해서 쓰지 않는 것이 제일 중요합니다. 파이썬 2 버전에서는 탭과 스페이스바를 혼용해서 쓸 경우 바로 에러를 내뱉을 정도니까요! 탭을 쓰기 시작했다면 탭으로 통일하고, 스페이스바를 쓰기 시작했다면 스페이스바로 통일하는 버릇이 필요합니다.

[2-2. elif]

위에서 보여드린 예문은 경우의 수가 두가지 밖에 없었습니다. 내 돈이 5000원을 넘는지와 5000원을 안넘는지.. 따라서 이때는 if ~ else로 표현을 할 수가 있었는데, 그럼 3가지 이상의 경우의 수는 어떻게 표현할까요.

3가지 이상의 경우가 있다면 이는 if ~ elif ~ else로 표현하면 됩니다. 예문을 보여드릴게요. 점수에 따라 학점을 출력하는 프로그램입니다.

```
if 90 <= score <=100:
    print("학점 : A")
elif 80<= score <90:
    print("학점 : B")
elif 70<= score <80:
    print("학점 : C")
elif 60<= score <70:
    print("학점 : D")
else:
    print("학점 : F")
```

두가지 경우의 수밖에 없는 경우에는 -> if 조건부를 만족하지 못했을 때, 바로 else문으로 넘어갔었죠? 하지만 지금 예문에서는 if 조건부를 만족하지 못한경우, 바로 아래의 elif로 넘어갑니다. 그 elif 조건부도 만족하지 못하면? 또 바로 아래의 elif 넘어갑니다. elif 조건부를 모두 만족하지 못하면 마지막 else로 넘어가겠죠? (참고로 elif 란 , else if 의 준말입니다.)

만약 중간에 elif를 만족시키는 경우가 있다면, 그 elif 안에 들어있는 문장들을 수행하고, 아래의 elif 또는 else를 모두 스킵해버립니다. 예를 들어 내 점수가 지금 85점이라 했을 때, “학점 : B”를 출력하고는 그 아래의 elif들과 else를 모두 스킵합니다. 아래 예제는 점수를 입력 받고 그 점수에 맞는 학점을 출력하는 프로그램을 완성시킨 모습입니다.

```
score = int(input('점수를 입력하세요 : '))
if 90 <= score <=100:
    print("학점 : A")
elif 80<= score <90:
    print("학점 : B")
elif 70<= score <80:
    print("학점 : C")
elif 60<= score <70:
    print("학점 : D")
else:
    print("학점 : F")
```

```
점수를 입력하세요 : 89
학점 : B

Process finished with exit code 0
```

[2-3. if 조건부에서 활용되는 연산자들]

if 문의 조건부에는 사실 True와 False 밖에 들어가지 못합니다. 하지만 위에 보여드린 예문에는 조건부에 부등식을 사용했죠? 이게 가능한 이유는 부등호가 연산의 결과로 True나 False 반환하기 때문이에요.
즉, my_money 에 2000 이 저장돼있다하면, my_money >= 5000 이 식에서 부등호는 연산의 결과로 False를 내뱉게 되고,

```
if my_money >= 5000:
```

이라고 써있던 조건부는

```
if False:
```

로 바뀌게 되는 겁니다.
이렇게 양쪽의 값을 비교하여 연산의 결과로 True나 False를 내뱉는 연산자를 보고 '비교 연산자'라고 합니다.
이처럼 비교 연산자는 True False를 반환시키기 때문에 if문에서 많이 활용됩니다. 비교 연산자에는 다음과 같은 것들이 있습니다.

```
<   >   <=   >=   ==   !=
```

부등호는 잘 아실텐데, "==" "!=" 이 두 개는 좀 낯설 수도 있어요
== : 연산자의 양 옆에 오는 값들이 같은지를 물어봄.
     같으면 True, 다르면 False를 반환
!= : 연산자의 양 옆에 오는 값들이 다른지를 물어봄.
     다르면 True, 같으면 False를 반환

if 문에서 자주 활용되는 연산자는 비교 연산자 외에 다음과 같은 것들이 더 있습니다.

- and / or / not

x and y : x와 y가 모두 True 일 때만, True를 반환, 나머지는 False 반환
x or y : x와 y 둘 중 하나라도 True이면, True 반환, 둘 다 False 일 때 False 반환
not x : x가 True 면 False 반환, False 면 True 반환
=> 예제 : "돈이 5000원이상 이거나(or) 신용카드가 있다면, 제육 덮밥을 먹는다. 그게 아니면, 컵라면을 먹는다."

```
my_money = 2000
credit_card = True
if my_money >= 5000 or credit_card==True:
    print("제육 덮밥 먹어야지 | | ~~")
else:
    print("컵라면 지겹..ㅠ")
```

```
제육 덮밥 먹어야지 | | ~~

Process finished with exit code 0
```

- in / not in

x in y : y 안에 x가 들어있는지 확인합니다. 들어있으면 True, 없으면 False

x not in y : y 안에 x가 안 들어있는지 확인합니다. 안 들어있으면 True, 있으면 False

이때 y 자리에 올 수 있는 자료형은 “리스트”, “문자열”, “튜플” 입니다.
간단하게 테스트를 해볼게요.

```
>>> "P" in "PYTHON"
True
>>> 1 in [1,2,3]
True
>>> "라" in ("가", "나", "다")
False
```

[2-4. 자료형의 참과 거짓]

그렇다면 조건부에 True나 False가 아닌 그냥 숫자를 넣게 되면 어떻게 될까요?

```
if 100:
    print("OK")
```

```
OK

Process finished with exit code 0
```

우선 에러가 나지 않고 잘 실행이 됐습니다. 그리고 “OK”라는 문자열이 출력된 것을 보니 if 의 조건부에서 숫자 100을 True로 인식했음을 알 수 있겠죠?
이처럼 True나 False가 아닌 자료형 그 자체를 컴퓨터는 다음과 같은 기준으로 True False로 인식합니다.

자료형	참	거짓
숫자	0이 아닌 숫자	0
문자열	"abc"	""
리스트	[1,2,3]	[]
튜플	(1,2,3)	()
딕셔너리	{"a":"b"}	{}

[3. while(반복문)]

[3-1. 기본 형식]

while의 기본 형식은 if 문과 완전히 똑같이 생겼습니다. 기본 형식부터 먼저 알려드릴게요.

```
while <조건부>:
    수행할 문장1
    수행할 문장2
    ...
```

if 문과 마찬가지로 조건부에서 조건을 확인하고, 조건을 만족하면 아래 문장들을 수행합니다. 조건을 만족하지 못하면 그냥 while의 모든 부분을 건너뜁니다.

그러면 차이점은 뭘까요? 우선 이렇게 코딩을 해보고 실행해볼게요.

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

```
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
```

위 코드 실행결과 우측처럼 Hello 라는 게 계속 출력되는 것을 볼 수 있습니다.
while은 그러니까 if문에 반복의 기능을 더한 제어문입니다. 반복문이라고도 합니다.

형태를 좀 더 자세히 얘기해보자면, 우선 if문과 같이 조건부를 먼저 확인하고 조건을 만족하면 while에 속한 문장들을 수행합니다. 문장들을 모두 수행하면 if에서는 그 후에 그냥 쭉 아래로 흘러갔지만, while은 아래로 흘러가지 않고, 다시 조건부로 들어가 조건을 또 검사합니다. 조건이 또 만족되면, 문장들을 또 수행합니다.
정리하자면,
조건부 확인 -> 참? -> 문장 수행 -> 조건부확인 -> 참? -> 문장 수행 -> 조건부확인 -> 참? -> 문장 수행 ... -> 조건부 확인 -> 거짓? -> while문 탈출
이런 과정을 거칩니다.

위 예문과 같이 그러면 while 조건부에 True를 넣게 되면 while에 속한 문장들이 영원히 실행되겠죠?? 이렇게 while 조건부에 True 그 자체를 넣는 것을 보고, “무한 루프” 라고 합니다.
while 문은 다음과 같이 활용할 수 있습니다. 문장을 원하는 횟수만큼 실행시키도록 할 수 있습니다.

```
i=1
while i<=10:
    print("{}번째 줄입니다.".format(i))
    i+=1
```

```
1번째 줄입니다.
2번째 줄입니다.
3번째 줄입니다.
4번째 줄입니다.
5번째 줄입니다.
6번째 줄입니다.
7번째 줄입니다.
8번째 줄입니다.
9번째 줄입니다.
10번째 줄입니다.
```

[3-2. break]

break 라는 명령어는 내가 원하는 때에 while문을 탈출할 수 있게 만들어줍니다. 다음 예제는 “무한 루프”안에서 입력한 돈만큼 돈을 저금하다가 0을 누르면 무한 루프를 빠져나와 프로그램을 종료하는 프로그램입니다. while 문이 어떻게 돌아가는지 그 프로세스를 잘 생각해보면서 코드를 봐주세요.

```python
sum = 0
while True:
    number = int(input('돈을 저금하세요.(0 누르면 종료) : '))
    if number == 0:
        print("현재까지 저금한 돈 : {}원".format(sum))
        print("종료합니다.")
        break
    sum = sum + number
```

```
돈을 저금하세요.(0 누르면 종료) : 1000
돈을 저금하세요.(0 누르면 종료) : 10000
돈을 저금하세요.(0 누르면 종료) : 2000
돈을 저금하세요.(0 누르면 종료) : 5000
돈을 저금하세요.(0 누르면 종료) : 500
돈을 저금하세요.(0 누르면 종료) : 0
현재까지 저금한 돈 : 18500원
종료합니다.
```

이처럼 break 를 만나면 그 break의 위치에서 “가장 가까운” 반복문을 탈출하게 합니다. 여기서 중요한 건 “가장 가까운” 반복문을 탈출한다는 것입니다.

예를들어 위 예제를 다음과 같이 살짝 수정을 해볼게요.

```
sum = 0
while True:
    number = int(input('돈을 저금하세요.(0 누르면 종료) : '))
    if number == 0:
        while True:
            print("현재까지 저금한 돈 : {}원".format(sum))
            print("종료합니다.")
            break
    sum = sum + number
```

while문 안에 while문이 또 들어가 있는 형태입니다.
이 코드는 직접 한 번 실행해보시는 걸 추천드립니다. '백문이 불여일타(打)' 입니다! 제가 백번 설명하는 것보다 한번 코딩해보고 직접 눈으로 보는게 한번에 확 이해가 되실 거라 생각이 듭니다. 코드를 실행해보기 전, 어떤 결과가 나올지 먼저 예상해보세요~ 이를 실행해보시면, break가 가장 가까운 반복문만을 탈출한다는 것을 느끼실 수 있습니다.

[3-3. continue]

continue 라는 명령어는 break와 대조되는 명령어입니다. break는 가장 가까운 반복문으로 "탈출"하는 반면에, continue는 가장 가까운 반복문의 조건부로 다시 "돌아갑니다." 예문을 한번 보실게요.

```
sum = 0
while True:
    number = int(input('돈을 저금하세요.(0 누르면 종료) : '))
    if number == 0:
        print("현재까지 저금한 돈 : {}원".format(sum))
        print("종료합니다.")
        break
    if number < 0:
        print("출금은 못합니다! 저축하세요!!")
        continue
    sum = sum + number
```

```
돈을 저금하세요.(0 누르면 종료) : 10000
돈을 저금하세요.(0 누르면 종료) : 20000
돈을 저금하세요.(0 누르면 종료) : 30000
돈을 저금하세요.(0 누르면 종료) : -5000
출금은 못합니다! 저축하세요!!
돈을 저금하세요.(0 누르면 종료) : 0
현재까지 저금한 돈 : 60000원
종료합니다.

Process finished with exit code 0
```

위 break 예제를 응용해봤습니다. 실행 결과부터 한번 볼게요. 돈을 만원, 2만원, 3만원 저금하다가 5000원을 지출하려 했더니 경고 메시지가 뜨면서 현재까지 저금한 돈을 확인해보니 진짜로 지출이 안된 것을 볼 수가 있죠.
이게 가능하게 됐던 이유가 continue 명령어 때문입니다.
소스코드의 퍼런색으로 드래그한 부분을 보면,
입력한 숫자가 음수일 경우-> 메시지를 출력하고 -> continue를 만납니다.
-> 그 후 “sum = sum + number” 이 문장을 수행하지 않고 (만약 continue가 없었다면 이 문장이 실행되고, 돈이 지출됐겠죠?) -> 다시 바로 while문의 조건부로 향하게 됩니다.

break가 “가장 가까운 반복문”으로 탈출하는 것처럼 continue 또한 “가장 가까운 반복문”으로 돌아갑니다. 주의하세요~

[4. for]

[4-1. 기본 형태]

for문도 while문과 같은 반복문으로 불립니다. 하지만 while문과는 그 사용법이 매우 다릅니다. 이것도 우선 예제를 먼저 보여드릴게요.

```
for word in ["one", "two", "three"]:
    print(word)
```

```
one
two
three

Process finished with exit code 0
```

for문 아래에 들어가 있는 문장이 반복하면서 출력을 하고 있긴한데, while이 맨 처음에 조건부를 확인하는 것과는 다르게 for문은 뭔가 많이 다릅니다.

조금 자세히 뜯어보겠습니다.

저 소스코드에서 “in” 이 보이시죠. in을 기준으로 오른쪽에는 리스트가 위치해있고 왼쪽에는 word라는 변수(제가 아무이름으로 만든 변수입니다.)가 위치해있습니다. 이 word 변수는 리스트의 0번째 값부터 하나씩 가져와서 for문으로 들어가게 됩니다. 그러니까 아래의 과정을 차례대로 거칩니다.

word가 “one”을 들고 for문 아래로 들어감 -> 문장들을 모두 수행 ->

word가 “two”를 들고 for문 아래로 들어감 -> 문장들을 모두 수행 ->

wordrk “three”를 들고 for문 아래로 들어감 -> 문장들을 모두 수행 -> for문을 완전히 빠져나옴

for문의 동작과정이 while문과 조금 많이 달라 처음에는 낯설 수도 있지만, for를 쓰다보면 이렇게 편리한게 또 있나..싶습니다.

****

<주의할 점!>

여기서 주의할 점은 for와 in은 항상 같이 따라다니는 친구입니다. for를 쓰는데 in이 없으면 안되죠. 그런데 우리가 if문에서 배웠던 연산자 중에 in 이라고 있었잖아요. 이 in은 for문의 in과 완전히 다른! 그냥 태생부터 다른 아이입니다. if문에서 쓰이는 in은 True 또는 False를 내뱉는 연산자이고, for 문에서의 in 은 그냥

for <변수> in <리스트> : ==>여기서 변수와 리스트를 구분해주는 그냥 구분 문자? 정도로 알고 있으시면 돼요~

****

for의 정확한 형태를 보여드릴게요.

```
for <변수이름> in <리스트(또는 문자열, 튜플)> :
    수행할 문장 1
    수행할 문장 2
    수행할 문장 3
    ....
```

이런 모습입니다. 여기서 in 뒷부분을 보면 리스트 말고도 문자열, 튜플도 된다고 써놨죠? 튜플과 문자열도 예제를 한번 보여드릴게요.

```
for number in (1,2,3,4,5):
    print(number)
```

```
1
2
3
4
5

Process finished with exit code 0
```

튜플은 리스트와 똑같이 동작합니다.

```
for word in "PYTHON":
    print(word)
```

```
P
Y
T
H
O
N

Process finished with exit code 0
```

문자열을 for문으로 들어갈 때 문자 하나하나씩 첫 글자부터 차례대로 들고 옵니다.

[4-2. break/continue]

for문도 while문과 같은 반복문이기 때문에 break와 continue 명령어를 사용할 수 있습니다. while문에서 모두 설명했으니 건너뛰고, 깔끔하게 예제 하나만 보시고 넘어갈게요.

음.. 예제 생각이 잘 안나서 생각하다가.. 갑자기 떠오른게 있는데.
예전에 군대에서 제가 취사병으로 근무했었거든요.
그 때 격일로 한 번씩 식재료들이 배달이 옵니다. 그러면 쫄래쫄래 나가서 배달 온 식재료들을 하나하나 확인을 해야됐어요. 그러다가 혹시 상한 식재료들이 발견됐다? 그게 뭐 한 두 가지 정도면 그냥 그것들 폐기시키고서, 다른 메뉴로 바꾸면 그만이었는데 상한 음식이 너무 많으면 그 배달차를 돌려보내고 새로 가져오게 했거든요. 갑자기 그게 생각나서..

```
가져온_물건들 = ["생선", "상한 달걀", "우유", "두부", "상한 고기", "소시지", "상한 고등어", "돼지고기"]
상한물건갯수 = 0
for 물건확인 in 가져온_물건들:
    if 물건확인[:2] == "상한":
        상한물건갯수 += 1
        print("{} : 불량!!!!".format(물건확인))
        continue
    if 상한물건갯수 == 3:
        print("돌려보내!!!!!!!")
        break
    print("{} : 양호~~".format(물건확인))
```

```
생선 : 양호~~
상한 달걀 : 불량!!!!
우유 : 양호~~
두부 : 양호~~
상한 고기 : 불량!!!!
소시지 : 양호~~
상한 고등어 : 불량!!!!
돌려보내!!!!!!!

Process finished with exit code 0
```

for문에서도 continue와 break가 잘 작동하는 것을 확인해주세요~

[4-3. range 함수]

for를 사용할 때, 이런 경우가 있습니다. 1부터 ~ 10까지 저장돼있는 리스트가 필요할때!

```
for i in [1,2,3,4,5,6,7,8,9,10]:
    print(i,"번째 줄입니다.")
```

```
1 번째 줄입니다.
2 번째 줄입니다.
3 번째 줄입니다.
4 번째 줄입니다.
5 번째 줄입니다.
6 번째 줄입니다.
7 번째 줄입니다.
8 번째 줄입니다.
9 번째 줄입니다.
10 번째 줄입니다.

Process finished with exit code 0
```

근데 지금은 1부터 10까지라서 충분히 손으로 하나하나하나 타이핑할 수 있지만, 혹시 1부터 100까지라면? 1부터 1000까지라면? 이걸 손으로 하나하나 다 친다고 하면, 손가락 다치겠죠..? (라임..오졌..)

이 때를 대비해서 range 함수라는게 존재합니다.

range 함수는 원하는 숫자들이 나열돼있는 리스트를 손쉽게 만들어주는 기능을 합니다.

range 함수를 쓰는 방법에는 3가지가 있습니다.

1. range(x) : 0부터 x “미만”까지 1씩 증가하는 숫자들을 데이터로 갖는 리스트
2. range(x, y) : x부터 y“미만”까지 1씩 증가하는 숫자들을 데이터로 갖는 리스트
3. range(x, y, z) : x부터 y“미만”까지 “z”씩 증가하는 숫자들을 데이터로 갖는 리스트

예를 들어,

1. range(101) => [0, 1, 2, ... , 98, 99, 100]
2. range(50, 101) => [50, 51, 52, ... , 98, 99, 100]
3. range(0, 101, 2) => [0, 2, 4, 6, ... , 94, 96, 98, 100]

따라서 이 range를 활용하면 내가 원하는 리스트를 순식간에 만들 수 있으므로 for문과 아주 찰떡궁합입니다. range를 활용한 예제입니다.

```
for i in range(1, 101):
    print(i,"번째 줄입니다.")
```

```
HomeWorkActing
test × test ×
88 번째 줄입니다.
89 번째 줄입니다.
90 번째 줄입니다.
91 번째 줄입니다.
92 번째 줄입니다.
93 번째 줄입니다.
94 번째 줄입니다.
95 번째 줄입니다.
96 번째 줄입니다.
97 번째 줄입니다.
98 번째 줄입니다.
99 번째 줄입니다.
100 번째 줄입니다.

Process finished with exit code 0
```

[이중 for문]

for문 안에 또 for문이 들어간 것을 보고 이중 for문이라고 말합니다.

for문의 흐름에 집중해서 봐주세요.

```
for a in ["치즈", "김치", "야채", "고기"]:
    for b in ["버거", "만두", "피자"]:
        print(a+b)
    print() # 한줄 엔터
```

```
치즈버거
치즈만두
치즈피자

김치버거
김치만두
김치피자

야채버거
야채만두
야채피자

고기버거
고기만두
고기피자


Process finished with exit code 0
```

a가 “치즈”를 들고서 안에 있는 for문으로 들어갑니다.
a가 “치즈”인 상태에서, b는 “버거”, “만두”, “피자”를 순서대로 들고오며 a에 저장돼 있는 “치즈”와 합쳐집니다.
다시 바깥의 for문으로 돌아와 a는 “김치”를 들고 들어갑니다.
a가 “김치”인 상태에서, b는 “버거”, “만두”, “피자”를 순수대로 들고오며 a에 저장돼 있는 “김치”와 합쳐집니다.
...

기존의 for문에 대해 기초가 탄탄하다면 for문에 이중으로 오던, 삼중으로 오던 쉽게 해석할 수 있고, 쉽게 활용할 수 있을 거라고 생각합니다!

[리스트 안에 for문 넣기]

아래 코드를 봐주세요.

```
result = []
for num in [1,2,3,4,5,6]:
    result.append(num)
```

for문을 돌리며, result라는 이름의 리스트에 1,2,3,4,5,6을 추가시키는 for문입니다. 파이썬에서는 이를 한줄로 간결하고 직관적이게 표현할 수 있습니다.

```
result = [num for num in [1,2,3,4,5,6]]
```

다시 한번 예를, 들어볼게요. 원래의 리스트 => [1,2,3,4,5,6]의 원소를 각각 2씩 곱해서 result 리스트에 넣어볼게요. 원래는 아래와 같이 코딩을 합니다.

```
result = []
for num in [1,2,3,4,5,6]:
    result.append(num * 2)
```

이를 간단하게 표현하면,

```
result = [num*2 for num in [1,2,3,4,5,6]]
```

자 다시 한번~ 다른 예를 들어볼게요! 이번엔 원래의 리스트 => [1,2,3,4,5,6]의 원소 중 짝수만 골라서 result리스트에 넣어볼게요. 원래는 아래와 같이 코딩합니다.

```
result = []
for num in [1,2,3,4,5,6]:
    if num%2 == 0:
        result.append(num)
```

이를 간단하게 표현하면!

```
result = [num for num in [1,2,3,4,5,6] if num%2 ==0]
```

이해되시나요~? 여러 가지 본인만의 예제를 만들어보면서 이것저것 코딩해보세요! 그러면 정확하게 머릿속에 들어오실 겁니다! 항상 기억하세요! '백문이 불여일타!!'
원래 3~4줄짜리의 for문을 한줄로 표현하니 간결하고, 직관적이게 됐나요? 아닌가요..?ㅠㅠ 이게 불편하다면, 그냥 본인만의 스타일 대로 코딩하시면 됩니다. 코딩은 항상 간결하고 직관적이게 하는 것을 추천하지만.. 간결, 직관의 기준은 주관적이니까요.

이제 여기까지가 1주차 수업내용입니다. 영어도 회화는 재밌지만, 문법은 지루하듯이(하지만 문법을 알아야 회화도 되겠죠?).. 1주차 수업내용이 문법 위주라 조금 지루하셨을 수도 있겠지만 벌써 파이썬의 절반 이상을 배우셨어요!! 고생하셨습니다~