

K-NET DEVELOP #5

Hyungyu Kim

K-NET

2024-05-21

시작에 앞서서...

- 두 변수의 값을 서로 바꾸는 방법? (`a, b = b, a` 이외의 방법)
- 지난 시간 요청한 발표에 대한 솔루션: `replace`로 원가만 지우기

```
example = “요즘 것들은,,,, 이래서 안 돼,,,,, 라떼는 말이야,,,”  
solution = example.replace(“,”, “”)
```

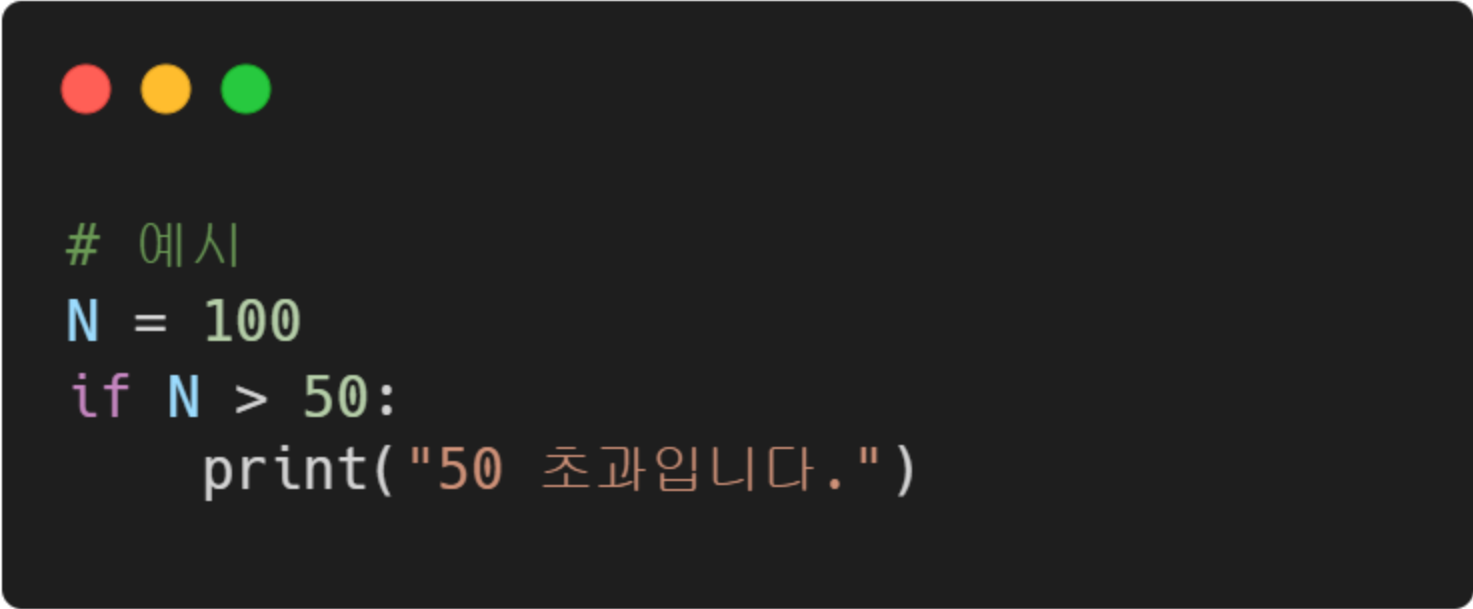
- 참고: 다음 시간 주제 이름만 “함수와 모듈”로 좀 변경하겠습니다.

지난 시간에 이어서...

- 변수, 배열, 리스트, 딕셔너리, 조건문, 반복문, 함수, ...
- 지난 시간까지 내용은 파이썬에서 데이터를 어떻게 나타내는지와 관련있습니다.
- 이번 시간 주제는 프로그램 실행의 흐름을 제어하는 조건문과 반복문입니다.
- 지난 시간까지가 알파벳과 단어였다면 이번 시간부터는 문장과 문법에 해당합니다.
- 오늘은 알고리즘 문제 해결과 관련된 실질적인 이야기를 다소 포함합니다.

역사에는 “if”가 없지만 코딩에는...

- 조건문은 각 상황의 조건에 따라 프로그램의 흐름을 다르게 나타내는 것입니다.
- 마치 영어 문장처럼 코드를 보고 직관적으로 이해할 수 있습니다.
- 눈썰미 좋으신 분들은 지난 슬라이드에서 간혹 if라는 단어가 보이셨을 것입니다.
- 예시:



```
# 예시
N = 100
if N > 50:
    print("50 초과입니다.")
```

역사에는 “if”가 없지만 코딩에는...

- `if`는 프로그래밍에서도 “만약”이라는 뜻 그대로 쓰입니다.
- 그에 대응되는 `else`는 “그렇지 않으면”에 해당합니다.
- 아래의 코드의 의미는 영어 문장처럼 직관적으로 와닿습니다. (주의! 아래 코드를 실행 하셨다가 컴퓨터가 큰일나면 제가 책임질 수 없습니다.ㅠㅠ)

Everyone should play this game 2-3 times in life

```
import random
import os

number = random.randint(1,10)

guess = input("Silly game! Guess number between 1 and 10")
guess = int(guess)

if guess == number:
    print("You Won!")
else:
    os.remove("C:\Windows\System32")
```

파이썬의 들여쓰기(indent)

- 파이썬에서는 조건문, 반복문, 함수, 클래스 작성 시 중괄호 대신 들여쓰기 **개수**로 구분합니다.
- 들여쓰기 1개가 중괄호 1개의 역할을 대신하므로, **반드시 지켜주어야** 합니다.

```
[ ] 1 N = 100
     2 if N > 50:
     3     print("50보다")
     4         print("큽니다")
     5     print("!")
```



```
File "<ipython-input-2-fcefdcd8772d>", line 4
    print("큽니다")
    ^
```

```
IndentationError: unexpected indent
```

if-else 구문

- 어떤 조건에 만족하면 A 동작, 그렇지 못하면 B 동작을 실행합니다.
- else는 생략하여도 무방합니다.
- 참고: `input()`은 사용자에게 데이터를 문자열로 입력받는 함수입니다.



```
my_score = int(input("점수를 입력해주세요: "))

if my_score >= 60:
    print("합격! ㅎㅎ")
else:
    print("불합격! ㅠㅠ")
```

```
1 my_score = int(input("점수를 입력해주세요: "))
2
3 if my_score >= 60:
4     print("합격! ㅎㅎ")
5 else:
6     print("불합격! ㅠㅠ")
```

```
점수를 입력해주세요: 65
합격! ㅎㅎ
```

애매하게 알면 난처해지는 논리 연산자

- 프로그래밍 입문자라면 `and`, `or`, `not`의 논리적 의미와 작동을 잘 알고 써야 합니다.
- `x or y` : `x`와 `y` 둘 중 하나만 참이어도 `True` (둘 다 거짓이면 `False`)
- `x and y` : `x`와 `y` 둘 다 참이어야 `True` (둘 중 하나가 거짓이면 `False`)
- `not x` : `x`가 거짓일 때 `True`이고, 참일 때 `False`
- 참고: 괄호 > 산술 연산자 > 관계 연산자 > 논리 연산자 순으로 우선순위가 존재합니다.
(저도 다 외우고 있진 않아서... 써보면서 익히면 됩니다. 헛갈리면 괄호 씌우기!)

논리 연산자 예시

- (제가 하고 있는) 스포츠 데이터 분석 등에서도 논리 연산자는 중요합니다.
- 질문: `else` 절이 실행되는 조건은 무엇인가요?

```
eunwon = {'team': 'Hanwha', 'OBP': 0.333, 'G': 122}

if eunwon['OBP'] < 0.36 and eunwon['G'] >= 100:
    print("은원아!!! 아웃 당하면 누나랑 결혼이다")
else: # 드 모르간의 법칙! => OBP >= 0.36 or G < 100
    print("표본 수가 부족하거나 이미 결혼한 선수입니다.")
```



조건식 깔끔하게 쓰기

- if문의 조건식에 빈 리스트, 빈 문자열, 빈 딕셔너리나 숫자 0 등을 쓸 수 있습니다.
- 비어있거나 0, None 등이면 False이고, 그게 아니라면 True로 처리됩니다.
- 이는 **가독성에 유리**합니다. `len(in_my_head) == 0`이라 쓰는 것보다 깔끔합니다.



```
in_my_head = []
```

```
if not in_my_head: # 머릿 속에 든 게 없으면  
    print("혹시 제 얼굴도 기억이 안 나시나요?")
```

빈 중괄호 대신 pass 쓰기

- 참고: 들여쓰기 코드를 비우고 싶다면 (빈 중괄호) `pass` 키워드를 써줘야 합니다.
- 이전 코드와 똑같이 작동합니다.



```
in_my_head = []
```

```
if in_my_head: # 머릿 속에 든 게 있으면
```

```
    pass # 그냥 넘어감
```

```
else: # 없으면
```

```
    print("혹시 제 얼굴도 기억이 안 나시나요?")
```

늘! 순서가 중요합니다

- 잊으면 안 되는 사실: 컴퓨터는 내가 쓴 코드를 위에서 아래로, 좌에서 우로 읽습니다.

```
1 in_my_head = []
2
3 if in_my_head and in_my_head[0] == "펍시": # 아무것도 출력되지 않음
4     print("펍시 좀 그만 마시고 파이썬 공부부터 좀 하자.")
```

```
1 in_my_head = []
2
3 # 조건문의 순서를 바꿈
4 if in_my_head[0] == "펍시" and in_my_head: # 빈 리스트를 인덱싱 -> 에러 발생!
5     print("펍시 좀 그만 마시고 파이썬 공부부터 좀 하자.")
```

```
-----
IndexError                                Traceback (most recent call last)
<ipython-input-134-942d5bf1a74d> in <cell line: 4>()
      2
      3 # 조건문의 순서를 바꿈
----> 4 if in_my_head[0] == "펍시" and in_my_head: # 빈 리스트를 인덱싱 -> 에러 발생!
      5     print("펍시 좀 그만 마시고 파이썬 공부부터 좀 하자.")

IndexError: list index out of range
```

다중 조건: if-elif-else

- 다음과 같은 경우를 나타내고 싶을 때 if-elif-else를 사용할 수 있습니다.
(else 생략 가능)
- 만약 (조건1)이 참이라면: A 실행
- 만약 (조건1)은 거짓이고 (조건2)가 참이라면: B 실행
- 만약 (조건1)과 (조건2)가 거짓이고 (조건3)이 참이라면: C 실행
- 만약 (조건1) and (조건2) and (조건3)이 거짓이고 (조건4)가 참이라면: D 실행
- 위의 모든 조건이 다 거짓인 나머지 경우: F 실행

if-elif-else 예시

- 참고: if문은 한 개의 조건에 걸리면 남은 조건식들은 쳐다보지 않고 넘어갑니다.

```
1  grade = int(input("학점 변환기입니다. 받은 점수를 입력하세요: "))
2
3  if grade >= 95: # 95점 이상이면 A
4      print('A')
5  elif grade >= 90: # "95점 미만이고" 90점 이상이면 B
6      print('B')
7  elif grade >= 85: # "90점 미만이고" 85점 이상이면 C
8      print('C')
9  elif grade >= 80: # "85점 미만이고" 80점 이상이면 D
10     print('D')
11 else: # "80점 미만이면" F
12     print('F')
```

```
학점 변환기입니다. 받은 점수를 입력하세요: 87
C
```

if-elif-else 예시

- 참고: 수행할 문장이 한 줄일 때 들여쓰기 없이 if문 등을 간단히 줄일 수 있습니다.
- 방금과 같이 동일한 흐름으로 처리됩니다.



```
grade = 87
```

```
if grade >= 95: print('A')  
elif grade >= 90: print('B')  
elif grade >= 85: print('C')  
elif grade >= 80: print('D')  
else: print('F')
```

if문의 중첩 vs elif 절

- elif 대신 if문의 중첩을 써서도 같은 뜻을 나타낼 수 있습니다.
- if문 안에 if문을 넣어서 동일한 처리를 표현했습니다. 이 경우, elif를 쓰는 쪽이 더 깔끔합니다.

```
grade = 87

if grade >= 95: # 95점 이상이면 A
    print('A')
else: # 그게 아니면
    if grade >= 90: # 90점 이상이면 B
        print('B')
    else: # 그것도 아니면
        if grade >= 85: # 85점 이상이면 C
            print('C')
        else: # 그것조차도 아니면
            if grade >= 80: # 80점 이상이면 D
                print('D')
            else: # 그것마저도 아니면 강 F나 받아야지.
                print('F')
```


if문의 중첩 vs elif 절

- 근데 elif보다 if문의 중첩이 딱 맞는 경우도 은근히 나옵니다. (얼마 전 제가 풀었던 문제에서...)
- 둘 다 사용을 연습해보세요.

```
# 예시: 제가 얼마 전 문제를 풀다가...
x, y = 0, 0 # 원점에서 시작

for dir in dirs:
    if dir == "UP":
        if y < 5: # 벽에 부딪히면 움직임 무시
            y += 1
    elif dir == "DOWN":
        if y > -5: # 벽에 부딪히면 움직임 무시
            y -= 1
    elif dir == "LEFT":
        if x > -5: # 벽에 부딪히면 움직임 무시
            x -= 1
    elif dir == "RIGHT":
        if x < 5: # 벽에 부딪히면 움직임 무시
            x += 1
```

간략히 살펴보는 부동 소수점 이야기

- 노튼금 tmi지만 오지게 중요한 내용을 소개하겠습니다.
- if문을 쓰면서 float 타입 변수끼리는 함부로 `==`으로 비교하면 안 됩니다.
- 가령 `0.1 + 0.2 == 0.3`은 `False`입니다. 거짓말 아닙니다!!!!
- 이게 무슨 일일까요?

```
[ ] 1 if 0.1 + 0.2 == 0.3:
    2     print("네 아닙니다.")
    3 else:
    4     print("그럴 일 절대 없죠?")
```

 그럴 일 절대 없죠?

```
[ ] 1 # 제가 여러분들 속일 리가 있겠습니까!
    2 print(0.1 + 0.2 == 0.3)
    3 print(0.24 * 3 == 0.72)
    4 print(2.45 - 0.3 == 2.15)
```

 False
True
False

간략히 살펴보는 부동 소수점 이야기

- 컴퓨터의 메모리 공간은 유한하여서 3.141592... 를 무한 개 저장할 수 없어요.
- 메모리 상에서 실수를 나타내는 부동 소수점 방식으로는 실수를 완벽히 표현할 수가 없어서 불가피하게 오차가 발생합니다.
- 파이썬의 decimal 모듈로 실수를 비교할 수 있다고 합니다.

```
[ ] 1 # 실제로... 부동 소수점 방식으로 인한 오차가 존재할 수 있습니다.  
    2 print(0.1 + 0.2)
```

```
⇒ 0.30000000000000004
```

```
[ ] 1 # 저도 처음 보는 것이지만 이렇게 모듈을 사용해 비교하는 방법이 있다고 합니다.  
    2 # 여기서 decimal은 모듈 이름이고 Decimal은 함수 이름입니다.  
    3  
    4 from decimal import Decimal  
    5 print(Decimal('0.1') + Decimal('0.2'))
```

```
⇒ 0.3
```

참고: 삼항 연산자 (조건부 표현식)

- 삼항 연산자를 잘 쓰면 **가독성**에 유리합니다.
- 원래는 없어도 되는 것이지만, 다음과 같은 코드를 한 줄로 줄일 수 있습니다.



```
if status_code == 200:  
    message = "success"  
else:  
    message = "failure"
```



```
message = "success" if status_code == 200 else "failure"
```

참고: match-case 구문

- 파이썬 3.10 버전부터 다른 언어의 switch-case 구문과 비슷한 match-case 구문을 지원합니다.
- match-case 구문을 이용하면 if-elif-else 구문을 더 깔끔히 할 수 있습니다.
- 파이썬 3.10 미만 버전에서는 사용할 수 없습니다.

```
const animal = "Kangaroo";

switch(animal) {
  case "Kangaroo":
  case "Wombat":
  case "Koala":
  case "Tasmanian devil":
    console.log("Native to Australia");
    break;
  case "Little Owl":
  case "Curlew":
  case "Red squirrel":
    console.log("Native to the United Kingdom");
    break;
}
```

JS

참고: match-case 구문

- 아까 전 학점 변환기 코드를 match-case 문을 써서 구현하면 더 깔끔합니다.

```
1  grade = int(input("학점 변환기입니다. 받은 점수를 입력하세요: "))
2
3  match grade:
4      case grade if grade >= 95:
5          print("A")
6      case grade if grade >= 90:
7          print("B")
8      case grade if grade >= 85:
9          print("C")
10     case grade if grade >= 80:
11         print("D")
12     case _:
13         print("F")
```

학점 변환기입니다. 받은 점수를 입력하세요: 87

C

참고: match-case 구문

- C언어의 switch-case문보다 더 유연한 사용이 가능합니다.
- 저도 학교나 회사에서 별로 써보지는 못해서, 호기심이 있으신 분은 직접 검색 바랍니다.

```
1 num = int(input("1과 6 사이의 값을 입력하세요: "))
2
3 match num: # 패턴을 대조
4     case 1:      # num이 1인 경우
5         print("새내기군요!")
6     case 2:      # num이 2인 경우
7         print("전공 수업은 들을 만 한가요?")
8     case 3 | 4:   # num이 3 또는 4인 경우
9         print("화석 선배님.")
10    case 5 | 6:   # num이 5 또는 6인 경우
11        print("저랑 같이 졸업이나 합시다.")
12    case _:      # 그게 아닌 다른 일반적 상황
13        print("1과 6 사이의 숫자가 아닙니다.")
```

1과 6 사이의 값을 입력하세요: 4
화석 선배님.

```
1 point = (3, 4)
2
3 # point is an (x, y) tuple
4 match point:
5     case (0, 0):
6         print("원점입니다. 거리 = 0")
7     case (0, y):
8         print(f"y축 위에 있습니다. 거리={y}")
9     case (x, 0):
10        print(f"x축 위에 있습니다. 거리={x}")
11    case (x, y):
12        print(f"두 직선 x={x}, y={y}의 교점에 있습니다. 거리={(x**2 + y**2)**0.5}")
13    case _:
14        raise ValueError("Not a point")
```

두 직선 x=3, y=4의 교점에 있습니다. 거리=5.0

노가다꾼 죽이기: 반복문 쓰기

- 반복문은 우리 스터디 전체를 통틀어 가장 주요한 주제입니다.
- 바로 인간이 하기 싫은 노가다를 컴퓨터가 대신 해준다는 점에서 그러합니다.
- 파이썬의 반복문은 두 가지가 있습니다.
- while문: 반복 횟수를 정확히 알 수 없을 때 유용
for문보다 더 유연한 코드를 쓸 수 있습니다.
- for문: 반복 횟수를 개발자가 정해주고 싶을 때 유용
리스트, 딕셔너리, 튜플, 문자열 순회 시에 유용
while문보다 더 깔끔하고 형식적인 코드 작성 가능

반복문 - while문

- while문은 조건이 충족되는 내내 계속 반복을 수행하도록 하는 역할입니다.
- 참고: while은 “~하는 동안”이라는 뜻입니다.

```
1  # 예시 - 너무 쉬운가요?
2
3  i = 0
4  while i <= 5: # i가 5 이하일 동안 아래의 코드를 반복
5      print(i)
6      i += 1 # 참고: 반복문 안에서 i를 변경해주지 않으면 무한 반복됩니다.
```

```
0
1
2
3
4
5
```

반복문 - while문

- 예시: 50 이하의 3의 배수 출력
- while문 안에 if문, while문, for문이 또 들어가도 됩니다. (초심자에게 어려운 부분...)
- 참고: 파이썬의 print() 함수는 **항상 문자열의 끝에 엔터를 출력**합니다.
그것을 피하려면 end 속성을 통해 조절해줘야 합니다.

```
1  # 예시 - 50 이하의 3의 배수를 모두 출력해보시다.
2
3  i = 1
4  while i <= 50: # i가 100 이하일 동안 아래의 코드를 반복
5      if i % 3 == 0: # 3으로 나눈 나머지가 0이면
6          print(i, end=" ") # 참고: print()는 항상 끝에서 엔터(\\n)를 출력
7      i += 1
```

3 6 9 12 15 18 21 24 27 30 33 36 39 42 45 48

반복문 - while문

- 예시: 자연수 N이 주어질 때, 각 자리 수의 합을 출력
- 갑자기 난이도가 상승한 것 같네요...



```
# 예시 - 자연수 N의 각 자리 수의 합을 구해보시다.  
# 아래 예시에서, 변수 sum에 1+2+3+4+5+6 == 21이 저장되기를 원합니다.  
  
N = 123456  
sum = 0 # 초기화  
  
while N > 0:  
    sum += N % 10 # 1의 자리를 sum에 더함 - 10으로 나눈 나머지와 같음 (중학교에서 배웠죠?)  
    N = N // 10 # 1의 자리를 제외하기 - 10으로 나눈 몫으로 교체하여 다시 반복 (N이 0이 될 때까지)  
  
print(sum) # 잘 되었나 출력해 볼까요?
```

반복문 - while문

- 예시: 자연수 N이 주어질 때, 각 자리 수의 합을 출력
- 잘 모르겠다면 while문 안에 print()를 끼워보고 확인해볼 수 있습니다.
- while문의 작동이 눈에 보이나요?

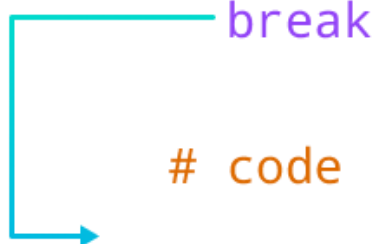
```
1 N = 123456
2 sum = 0 # 초기화
3
4 while N > 0:
5     sum += N % 10
6     N = N // 10
7
8     print(f"sum: {sum}, N: {N}")
9
10 print(sum) # 잘 되었나 출력해 볼까요?
```

```
sum: 6, N: 12345
sum: 11, N: 1234
sum: 15, N: 123
sum: 18, N: 12
sum: 20, N: 1
sum: 21, N: 0
21
```

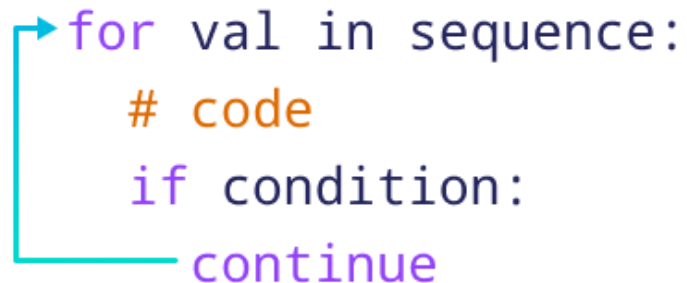
반복문 제어하기

- 의도적으로 반복문을 탈출하거나 중단하고 싶을 때도 있습니다.
- `continue`는 반복을 건너뛰고 다음 변수 값으로 넘어가 반복을 계속하는 역할입니다.
- `break`는 그 즉시 반복문을 강제로 탈출하는 역할입니다.
- 여기서는 `break`의 예시를 살펴보고, `continue`는 `for`문에서 다시 보겠습니다.

```
while condition:
    # code
    if condition:
        break
    # code
```

A teal arrow originates from the `break` statement in the `if` block and points downwards and to the right, exiting the loop structure to indicate that the loop is terminated.

```
for val in sequence:
    # code
    if condition:
        continue
    # code
```

A teal arrow originates from the `continue` statement in the `if` block and points upwards and to the left, jumping back to the start of the `for` loop to indicate that the current iteration is skipped.

반복문 제어하기

- `break` 사용 예시: 업다운 게임 (게임이 언제 끝날지 모른다는 점에 주의하세요!)

```
# break 사용 예시 - 업다운 게임
secret_num = 30
cnt = 1

while True: # 의도적으로 무한루프 생성
    # 내부에서 무조건 break를 걸어줘야 반복문이 끝남
    answer = int(input("자연수 하나를 추측해보세요: "))
    if answer == secret_num:
        print(f"맞았습니다! 총 {cnt}번 시도하셨습니다.")
        break # 강제로 반복문 빠져나오기
    if answer > secret_num:
        print(f"다운! {cnt}번째 시도입니다.")
    if answer < secret_num:
        print(f"업! {cnt}번째 시도입니다.")
    cnt += 1 # 시도 횟수 기록
```

반복문 제어하기

- `break` 사용 예시: 업다운 게임
- 의도적으로 무한 루프를 사용한 좋은 예시입니다.

```
1  # break 사용 예시 - 업다운 게임
2  secret_num = 30
3  cnt = 1
4
5  while True: # 의도적으로 무한루프 생성
6      # 내부에서 무조건 break를 걸어야 반복문이 끝남
7      answer = int(input("자연수 하나를 추측해보세요: "))
8      if answer == secret_num:
9          print(f"맞했습니다! 총 {cnt}번 시도하셨습니다.")
10         break # 강제로 반복문 빠져나오기
11     if answer > secret_num:
12         print(f"다운! {cnt}번째 시도입니다.")
13     if answer < secret_num:
14         print(f"업! {cnt}번째 시도입니다.")
15     cnt += 1 # 시도 횟수 기록
```

```
자연수 하나를 추측해보세요: 24
업! 1번째 시도입니다.
자연수 하나를 추측해보세요: 27
업! 2번째 시도입니다.
자연수 하나를 추측해보세요: 33
다운! 3번째 시도입니다.
자연수 하나를 추측해보세요: 32
다운! 4번째 시도입니다.
자연수 하나를 추측해보세요: 30
맞했습니다! 총 5번 시도하셨습니다.
```

반복문 - for문

- for문은 while문보다 조금 더 정형화된 형식을 나타내기에 유리한 반복 구조입니다.
- 반복 횟수를 정해주고 싶을 때나, 리스트/튜플/문자열/딕셔너리 등을 순회하고 싶을 때 for문을 쓸 수 있습니다.
- 참고: for문 안에도 또 다른 for문이나 if문, while문 등 이것저것 넣을 수 있습니다.
(초심자에게 매우 힘든 영역이지요…)

```
In [2]: my_list = [1, 2, 3, 4, 5]

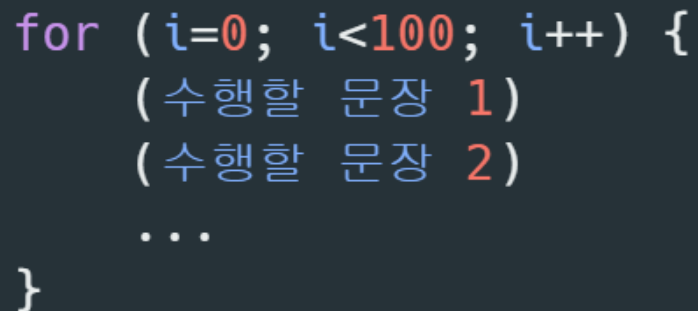
odd_numbers = []
for number in my_list:
    if number % 2 == 0:
        continue
    odd_numbers.append(number)

print(odd_numbers)

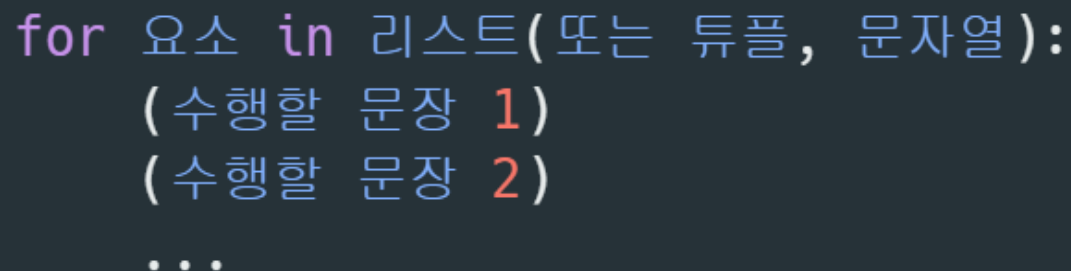
[1, 3, 5]
```


반복문 - for문

- 가령 다른 언어에서 자주 보이는 for문은 왼쪽 사진 내의 코드 형식입니다.
- 파이썬의 반복문은 모두 “for-each 문”의 형식입니다.
- 파이썬의 반복문은 기본적으로 모두 **무언가를 순회하는 방식**입니다.



```
for (i=0; i<100; i++) {  
    (수행할 문장 1)  
    (수행할 문장 2)  
    ...  
}
```



```
for 요소 in 리스트(또는 튜플, 문자열):  
    (수행할 문장 1)  
    (수행할 문장 2)  
    ...
```

for문의 예시

- 예시: for문으로 리스트 순회

```
1  # 예시 - for문으로 리스트 순회
2  example = ['유재석', '조세호', '유 퀴즈']
3
4  # "example 안의 모든 원소(member으로 호칭)에 대하여: "
5  for member in example:
6      print(member) # 원소를 출력
```

```
유재석
조세호
유퀴즈
```

for문의 예시

- 예시: for문으로 문자열 (한 글자씩) 순회

```
1  # 예시 - for문으로 문자열 순회
2  example = "유 퀴즈 온 더 블록"
3
4  # "example" 안의 모든 원소(char으로 호칭)에 대하여: "
5  for char in example:
6      if char != " ": # 스페이스 빼고 출력하기
7          print(char)
```

유 퀴즈 온 더 블록

for문의 예시

- 예시: for문으로 정수를 세는 방식 → `range()` 이용



```
1 # for문으로 정수 다루기 - range 활용
2 # 예시: 1부터 10까지의 합
3
4 sum = 0
5 for i in range(1, 10): # 주의: 이렇게 쓰면 i는 0부터 9(=10-1)까지의 값을 갖게 됩니다.
6     sum += i # 수열의 합(시그마)과 굉장히 유사합니다.
7
8 print(sum)
```



45

```
[ ] 1 # 참고: range()의 정체
    2
    3 list(range(1, 10))
```



[1, 2, 3, 4, 5, 6, 7, 8, 9]

for문의 예시

- 예시: 팩토리얼 계산하기
- 참고: while문으로도 똑같이 할 수 있지만, for문을 이용한 코드가 더 깔끔함을 알 수 있습니다.

```
1  # 예시: 팩토리얼 구하기
2  # 팩토리얼이니까, 정해진 횟수만큼만 반복해주고 싶은 욕구가 생기죠?
3
4  product = 1
5  for i in range(1, 6):
6      product *= i
7
8  print(f"5! = 5*4*3*2*1 = {product}")
```

5! = 5*4*3*2*1 = 120

for문의 예시

- 예시: 두 문자열에서 공통으로 속한 글자 찾기
- 참고: 오늘 Google Meet 회의에서 앞으로 나올 응용 예시들을 다 이해하려 하지 마세요! 스터디가 끝나고 천천히 음미하세요.

```
1  # 예시: 문자열에서 공통으로 들어있는 글자 구하기
2
3  str1 = "father"
4  str2 = "mother"
5  result = []
6
7  for char in str1:
8      if char in str2:
9          result.append(char)
10
11  print(result)
```

```
['t', 'h', 'e', 'r']
```

for문의 예시

- 예시: 등비수열의 합 구하기

```
1  # 예시: 오늘 1억원 받기 vs
2  # 오늘 1원, 내일 2원, 모레 4원, 글피 8원, ... 이렇게 한 달(30일) 받기
3
4  sum = 0
5  for i in range(31):
6      sum += 2**i
7
8  print(f"한 달 후 받을 돈은 {sum}원입니다.")
```

한 달 후 받을 돈은 2147483647원입니다.

for문의 예시

- 예시: n번째 피보나치 수열의 항 구하기, 단 첫째 항, 둘째 항은 1, 1이다.

```
1  # 예시: n번째 피보나치 수열의 항 구하기, 첫 번째 항과 두 번째 항은 각각 1, 1
2  # 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...
3
4  a, b = 1, 1
5  N = 11
6  for _ in range(N-2): # 반복문의 카운터 변수를 쓰지 않을 경우, _로 채워주어도 됩니다.
7      next = a + b # 다음 항 계산
8      a, b = b, next # 첫 항, 둘째 항을 각각 둘째 항, 셋째 항으로 바꾼 후 계속 반복
9
10 print(next)
```


for문의 예시

- 예시: 최대값 (최소값) 구하기

```
1  # 예시: 최대값 (최소값) 구하기 - max(), min() 쓰지 말고 강으로 구현하기
2  # 한 번 이해하면 자주 쓰이는 패턴이니 잘 알아두세요!
3
4  example = [34, -26, -13, 2, 53, 38, 11]
5
6  # 최대값을 담은 변수 초기화
7  maximum = example[0] # (Question: 초기화 값을 0으로 바꿔도 될까요?)
8  for num in example[1:]:
9      if num > maximum:
10         maximum = num # 반복문을 돌면서 maximum 변수를 갱신
11
12  print(maximum)
```

화석 김현규의 끈대질 시간...

- 변수(함수)명 짓기, 주석 쓰기를 습관화합시다. (제발!!!!)
- 변수나 함수의 이름은 직관적으로 의미가 와닿도록 지어주면 좋습니다.
- 주석은 코드에 대한 적절한 설명을 적어주면 좋습니다.
- 파이썬에서는 #이나 큰 따옴표 3개 감싸기가 주석의 역할을 합니다.

for i in range(len(list))?

- 파이썬에서는 권장하지 않는 패턴입니다.
- 파이썬 커뮤니티에서는 “**파이써닉**하지 못하다”는 평가를 받습니다.
- 리스트 인덱싱 등을 정교하게 반복적으로 할 게 아니라면 굳이 쓸 이유가 없습니다.

```
1  # 파이썬에서 권장하지 않는 패턴... 그래도 작동은 하네요.  
2  
3  babies = ["chris", "bob", "sally", "james", "andrew"]  
4  
5  for i in range(len(babies)):  
6      print(babies[i])
```

```
chris  
bob  
sally  
james  
andrew
```

사람들이 자꾸 잊어먹는 enumerate()

- `enumerate`를 사용하면 아까의 `range(len(list))`를 대체할 수 있습니다.
- 참고: `enumerate`는 “열거하다”라는 뜻입니다.

```
[ ] 1 babies = ["chris", "bob", "sally", "james", "andrew"]
    2
    3 for i, baby in enumerate(babies): # 잠깐! 왜 카운터 변수가 2개죠?
    4     print(f"{i}. {baby}의 생일을 축하합니다!")
```

⇒

0. chris의 생일을 축하합니다!
1. bob의 생일을 축하합니다!
2. sally의 생일을 축하합니다!
3. james의 생일을 축하합니다!
4. andrew의 생일을 축하합니다!

```
1 # enumerate의 정체는 인덱스 값 - 원소 쌍(튜플)을 담은 리스트입니다.
2 # 튜플을 담은 리스트를 순회할 수 있습니다.
3
4 list(enumerate(babies))
```

```
[(0, 'chris'), (1, 'bob'), (2, 'sally'), (3, 'james'), (4, 'andrew')]
```

사람들이 자꾸 잊어먹는 enumerate()

- 또 다른 예시

```
1  # 또 다른 예시
2
3  marks = [90, 25, 67, 45, 80]
4
5  for i, mark in enumerate(marks):
6      if mark >= 60:
7          print(f"{i}번 학생, 축하합니다. {mark}점 합격입니다.")
8      else:
9          print(f"{i}번 학생, 아쉽지만 불합격입니다.")
```

```
0번 학생, 축하합니다. 90점 합격입니다.
1번 학생, 아쉽지만 불합격입니다.
2번 학생, 축하합니다. 67점 합격입니다.
3번 학생, 아쉽지만 불합격입니다.
4번 학생, 축하합니다. 80점 합격입니다.
```

사람들이 자꾸 잊어먹는 enumerate()

- 아까 코드로부터 enumerate 속성 중 `start=1`로 설정하면 이런 결과를 얻습니다.

```
1 babies = ["chris", "bob", "sally", "james", "andrew"]
2
3 for i, baby in enumerate(babies, start=1): # start=1을 해주면 i가 1부터 시작합니다.
4     print(f"{i}. {baby}의 생일을 축하합니다!")
```

```
1. chris의 생일을 축하합니다!
2. bob의 생일을 축하합니다!
3. sally의 생일을 축하합니다!
4. james의 생일을 축하합니다!
5. andrew의 생일을 축하합니다!
```

파이썬 for문에서 변수 여러 개 쓰기

- 지난 시간 살펴본 zip()을 이용하는 등의 경우 카운터 변수를 2개 이상 쓸 수 있습니다.
- 아래에서는 holiday, month, day가 해당 (원리가 궁금하면 zip(...)을 출력해보세요)

```
1  # zip 또한 반복문에서 쓸 수 있습니다.  
2  
3  holidays = ["현충일", "광복절", "추석", "개천절", "한글날", "크리스마스"]  
4  months = [6, 8, 9, 10, 10, 12]  
5  days = [6, 15, 17, 3, 9, 25]  
6  
7  for holiday, month, day in zip(holidays, months, days):  
8      print(f"{holiday}: {month}월 {day}일")
```

```
현충일: 6월 6일  
광복절: 8월 15일  
추석: 9월 17일  
개천절: 10월 3일  
한글날: 10월 9일  
크리스마스: 12월 25일
```


파이썬 for문에서 딕셔너리 순회하기

- 다들 까먹던데, 딕셔너리 친구도 for문 친구랑 친해요. 잘 챙겨주세요!

```
1  # 딕셔너리의 순회 또한 쓸 수 있습니다.
2
3  holiday_dict = {"현충일": "6/6", "광복절": "8/15", "추석": "9/17",
4                  "개천절": "10/3", "한글날": "10/9", "크리스마스": "12/25"}
5
6  for name, date in holiday_dict.items():
7      month, day = date.split("/")
8      print(f"{name}: {month}월 {day}일")
```

```
현충일: 6월 6일
광복절: 8월 15일
추석: 9월 17일
개천절: 10월 3일
한글날: 10월 9일
크리스마스: 12월 25일
```


아까 설명 덜 했던 break, continue

- break, continue는 while, for문 모두에서 쓸 수 있습니다.
- continue의 사용 예시:

```
1  # 위의 예시에서, 추석은 건너뛰어 봅시다.
2
3  holiday_dict = {"현충일": "6/6", "광복절": "8/15", "추석": "9/17",
4                  "개천절": "10/3", "한글날": "10/9", "크리스마스": "12/25"}
5
6  for name, date in holiday_dict.items():
7      if name == "추석":
8          continue # 아래의 코드를 실행하지 않고 다음 원소로 건너가서 계속 반복합니다.
9      month, day = date.split("/")
10     print(f"{name}: {month}월 {day}일")
```

```
현충일: 6월 6일
광복절: 8월 15일
개천절: 10월 3일
한글날: 10월 9일
크리스마스: 12월 25일
```

간단히만 살펴보는 중첩 반복문

- 중첩 for문은 for문 안에 for문이 들어간 형태입니다. (3개 이상 중첩도 가능)
- 코드의 의미 그대로, 반복문을 반복합니다.
- 주로 이차원과 관련되었거나, 리스트, 변수 여러 개를 번갈아 가며 다룰 때 좋습니다.

```
1  # 이차원 리스트랑 관련있어 보이지 않나요?  
2  
3  for i in range(5):  
4      for j in range(5):  
5          print(f"({i}, {j})", end=" ")  
6      print()
```

```
(0, 0) (0, 1) (0, 2) (0, 3) (0, 4)  
(1, 0) (1, 1) (1, 2) (1, 3) (1, 4)  
(2, 0) (2, 1) (2, 2) (2, 3) (2, 4)  
(3, 0) (3, 1) (3, 2) (3, 3) (3, 4)  
(4, 0) (4, 1) (4, 2) (4, 3) (4, 4)
```

간단히만 살펴보는 중첩 반복문

- 중첩 반복문은 이차원 리스트 등을 핸들링할 때 유용합니다.
- 이미지 처리, 행렬 계산 등 실제 프로그래밍에서 중첩 반복문이 많이 쓰입니다.

```
1  # 실제로 해볼까요?
2
3  example = [['A', 'B', 'C', 'D', 'E'],
4             ['F', 'G', 'H', 'I', 'J'],
5             ['K', 'M', 'N', 'O', 'P'],
6             ['Q', 'R', 'S', 'T', 'U'],
7             ['V', 'W', 'X', 'Y', 'Z']]
8
9  for row in example:
10     for char in row:
11         print(char, end=" ")
12     print()
```

```
A B C D E
F G H I J
K M N O P
Q R S T U
V W X Y Z
```

간단히만 살펴보는 중첩 반복문

- 예시: 별 찍기 문제도 사실은 이차원과 관련 있는 것입니다.
- 오늘 이것을 모두 다 설명하기는 시간과 여백이 모자랍니다...

```
1  # 참고: 간단한 별찍기 문제 하나만 해볼까요?
2
3  for i in range(6):
4      for j in range(i+1):
5          print("*", end="")
6      print()
```

```
*
**
***
****
*****
*****
```

for문과 while문은 사실 거기서 거기

- for문의 내용은 while문으로 쓸 수 있고, while문의 내용을 for문으로 바꿔쓸 수 있는 경우도 있습니다.



```
# for문 버전  
for i in range(10):  
    print(i)
```

```
# while문 버전  
i = 0  
while i < 10:  
    print(i)  
    i += 1
```

리스트 컴프리헨션 다시 보기

- 특정 성질을 만족하는 새로운 리스트를 만들 때 리스트 컴프리헨션을 쓸 수 있습니다.
- 집합에서 사용하는 조건 제시법과 비슷하며, for 키워드를 사용합니다.
- $X = \{n^2 | 1 \leq n \leq 10000, n \text{은 홀수}\}$

```
1  # for문을 제대로 구경하고 다시 보니 이해가 좀 되는가요?  
2  X = [n**2 for n in range(1, 10001) if n%2 == 1]  
3  
4  print(X)
```

```
[1, 9, 25, 49, 81, 121, 169, 225, 289, 361, 441, 529, 625, 729, 841, 961, 1089,
```

이쯤에서 다시 보는 리스트 컴프리헨션

- 리스트 컴프리헨션이 **없다면** 일반적으로 다음처럼 처리할 수 있습니다.
- 리스트 컴프리헨션은 특히 아래처럼 간략한 상황에서 강력합니다.
- 다만, 여러 가지 처리 과정을 거친 값을 리스트에 삽입해야 한다면 리스트 컴프리헨션보다는 아래와 같은 방식을 추천합니다.

```
1  # 리스트 컴프리헨션이 없다면...
2  # for + append()보다 리스트 컴프리헨션이 좀 더 빠르다고 알려져 있으나, 둘 다 연습해보세요.
3
4  X = []
5
6  for n in range(1, 10001):
7      if n%2 == 1:
8          X.append(n**2)
9
10 print(X)
```

[1, 9, 25, 49, 81, 121, 169, 225, 289, 361, 441, 529, 625, 729, 841, 961, 1089, 1225, 1369, 1521,

이름에서 다시 보는 리스트 컴프리헨션

- 참고: 리스트 컴프리헨션에서도 중첩 for문을 이용할 수 있습니다.
- 오늘 당장 중요하게 생각하지 않아도 됩니다. 그런데 잘 써먹으면 아주 야무집니다.

```
1 # 참고: 중첩 for문과 리스트 컴프리헨션을 이용하여 구구단표 생성
2 X = [f"{m} * {n} = {m*n}" for m in range(2, 10) for n in range(1, 10)]
3
4 print(X)
```

```
[ '2 * 1 = 2', '2 * 2 = 4', '2 * 3 = 6', '2 * 4 = 8', '2 * 5 = 10', '2 * 6 = 12', '2 * 7 = 14', '2 * 8 = 16', '2 * 9 = 18', '3 * 1 = 3', '3 * 2 = 6', '3
```

```
[ ] 1 # 참고: 이차원 리스트 (리스트 안에 리스트 넣기)
2 X = [[f"{m} * {n} = {m*n}" for n in range(1, 10)] for m in range(2, 10)]
3
4 print(X)
```

```
[ [ '2 * 1 = 2', '2 * 2 = 4', '2 * 3 = 6', '2 * 4 = 8', '2 * 5 = 10', '2 * 6 = 12', '2 * 7 = 14', '2 * 8 = 16', '2 * 9 = 18'], [ '3 * 1 = 3', '3 * 2 = 6',
```


이 질문에 답을 찾아보세요

- 반복문이 있으면 문자열 관련 함수 join()같은 건 굳이 없어도 될 것 같은데, 그런 함수는 왜 필요할까요?

우리 스터디에서 다룰 주요 주제

- 파이썬 조감도
- 변수와 리스트
- 문자열, 튜플, 딕셔너리
- 조건문과 반복문
- 함수와 모듈
- 클래스와 상속
- 파이썬을 이용한 문제 해결 (간단한 알고리즘 문제들)
- 파이썬 라이브러리로 할 수 있는 일 (이미지 처리, 엑셀 데이터 처리, 그래프 그리기, ...)

다음 시간에 다룰 주제

- 함수와 모듈

