

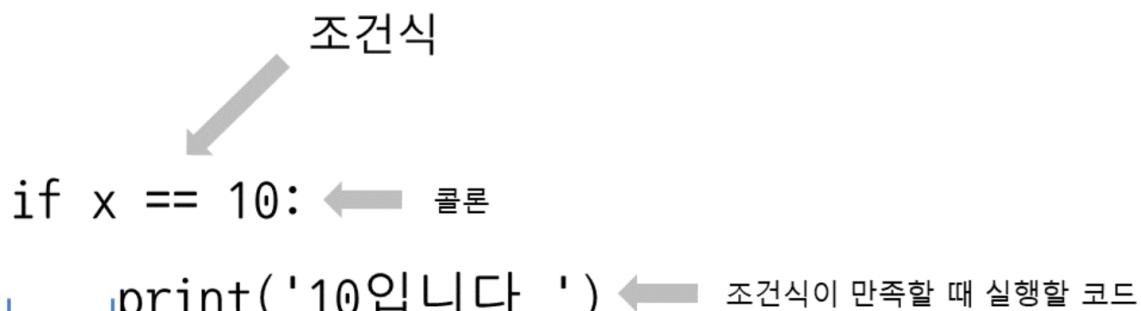


## 1 파이썬 조건문(Condition Statements): 입력이나 처리에 조건을 달자

- 목적: 입력을 하거나 데이터를 처리할 때, 특정 조건인 경우 실행되게 하는 기능

- 조건이 참(True) 또는 거짓(False)으로 나뉨
- 주로 논리 연산자와 산술 연산자와 함께 사용
- 조건문 뒤에 반드시 콜론(:) 붙이는 것이 문법
  - 콜론이 등장하면 그 다음 줄부터 무조건 들여쓰기 함
  - 다른 프로그래밍과의 차이는 괄호로 묶여 구분하지 않고 들여쓰기로 영역 구분
  - 조건에 따라 해당 조건에 맞는 상황을 실행 할 수 있게 됨

### 1.1 if / elif / else



if 조건식:  
문장

if 내일 비가 온다:  
집에서 TV본다

```
In [1]: money = 1000
      if money >= 500:
          print("돈이 500원 이상 있습니다.")
executed in 13ms, finished 20:13:44 2021-11-04
```

돈이 500원 이상 있습니다.

```
In [2]: money = 100
      if money >= 500:
          print("돈이 500원 이상 있습니다.")
executed in 15ms, finished 20:13:44 2021-11-04
```

```
In [3]: money >= 500
executed in 14ms, finished 20:13:44 2021-11-04
```

Out[3]: False

if 조건식:  
문장1  
else:  
문장2

if 내일 비가 온다:  
집에서 TV본다  
else:  
밖에서 산책한다

```
In [4]: # 돈이 100원일 때 "돈이 500원 미만입니다." 출력하기
```

```

money = 100

if money >= 500:
    print("돈이 500원 이상 있습니다.")
else: # if money < 500:
    print("돈이 500원 미만입니다.")

executed in 13ms, finished 20:13:44 2021-11-04

```

돈이 500원 미만입니다.

```

In [5]: # # 명령어 사용 후 들여쓰기 기준 할당 내용이 없으면 오류
# money = 1000

# if money >= 500:
#     print("돈이 500원 이상 있습니다.")
# else:
#     print("돈이 500원 미만입니다.")

executed in 12ms, finished 20:13:44 2021-11-04

```

```

In [6]: money = 100

if money >= 500:
    print("돈이 500원 이상 있습니다.")
else:
    print("돈이 500원 미만입니다.")
    print("허허..")

executed in 14ms, finished 20:13:44 2021-11-04

```

돈이 500원 미만입니다.  
허허..

```

In [7]: money = 500

if money >= 500:
    print("돈이 500원 이상 있습니다.")
else:
    print("돈이 500원 미만입니다.")
    print("허허..")

executed in 11ms, finished 20:13:44 2021-11-04

```

돈이 500원 이상 있습니다.  
허허..

```

In [8]: # 돈이 500일때와 아닐때를 실행해서 비교
money = 500

if money >= 500:
    print("돈이 500원 이상 있습니다.")
else:
    print("돈이 500원 미만입니다.")
    print("허허..")

executed in 13ms, finished 20:13:44 2021-11-04

```

돈이 500원 이상 있습니다.

- if와 else는 한번만 사용할 수 있지만, elif는 여러번 사용 가능
- elif는 단독으로 사용할 수 없고 if와 else 사이에만 와야 함

```

if 조건식1:
    문장1
elif 조건식2:
    문장2
else:
    문장3

```

```

if 내일 비가 온다:
    집에서 TV본다
elif (비는 안오는데) 내일 덥다:
    집에서 에어컨 켜고 TV본다
else: (비도 안오고 안더우면)
    밖에서 산책한다

```

```

In [9]: # 돈이 500원이 있을 때 "돈이 딱 500원 있습니다." 출력하기
money = 500

if money > 500:
    print("돈이 500원 이상 있습니다.")
if money == 500:
    print("돈이 딱 500원 있습니다.")
else:
    print("돈이 500원 미만입니다.")

executed in 15ms, finished 20:13:44 2021-11-04

```

```

In [10]: money > 500
executed in 13ms, finished 20:13:44 2021-11-04

```

Out[10]: False

```

In [11]: # 들여쓰기 위치에 따른 출력 확인
money = 500

if money > 500:
    print("돈이 500원 이상 있습니다.")

executed in 13ms, finished 20:13:44 2021-11-04

```

```
    if money == 500:
        print("돈이 딱 500원 있습니다.")
    else: # money <= 500
        print("돈이 500원 미만입니다.")
executed in 14ms, finished 20:13:44 2021-11-04
```

돈이 500원 미만입니다.

```
In [12]: money = 500

if money > 500:
    print("돈이 500원 이상 있습니다.")
if money == 500:
    print("돈이 딱 500원 있습니다.")
else: # money != 500
    print("돈이 500원 미만입니다.")
executed in 14ms, finished 20:13:44 2021-11-04
```

돈이 딱 500원 있습니다.

```
In [13]: # 중괄호 들어쓰기 위치에 if문이 있으면 반복적으로 if문 실행
money = 5000

if money > 500:
    print("돈이 500원 이상 있습니다.")
if money == 500:
    print("돈이 딱 500원 있습니다.")
else: # money != 500
    print("돈이 500원 미만입니다.")
executed in 12ms, finished 20:13:44 2021-11-04
```

돈이 500원 이상 있습니다.  
돈이 500원 미만입니다.

```
In [14]: # if ~ elif ~ else 사용하여 돈이 5000원 있을 때 "돈이 500원 이상 있습니다." 출력하기
money = 5000

if money > 500:
    print("돈이 500원 이상 있습니다.")
elif money == 500:
    print("돈이 딱 500원 있습니다.")
else:
    print("돈이 500원 미만입니다.")
executed in 14ms, finished 20:13:44 2021-11-04
```

돈이 500원 이상 있습니다.

```
In [15]: # if ~ elif ~ else 사용하여 돈이 500원 있을 때 "돈이 딱 500원 있습니다." 출력하기
money = 500

if money > 500:
    print("돈이 500원 이상 있습니다.")
elif money == 500:
    print("돈이 딱 500원 있습니다.")
else:
    print("돈이 500원 미만입니다.")
executed in 12ms, finished 20:13:44 2021-11-04
```

돈이 딱 500원 있습니다.

```
In [16]: # if ~ elif ~ else 사용하여 돈이 50원 있을 때 "돈이 500원 미만입니다." 출력하기
money = 50

if money > 500:
    print("돈이 500원 이상 있습니다.")
elif money == 500:
    print("돈이 딱 500원 있습니다.")
else: # money < 500 and money != 500
    print("돈이 500원 미만입니다.")
executed in 13ms, finished 20:13:44 2021-11-04
```

돈이 500원 미만입니다.

```
In [17]: # elif를 사용하지 않고 돈이 500원 있을 때 "돈이 딱 500원 있습니다." 출력하기
# 위에 있는 코드를 동일하게 복제
money = 500

if money > 500:
    print("돈이 500원 이상 있습니다.")
else:
    if money == 500:
        print("돈이 딱 500원 있습니다.")
    else: # money < 500 and money != 500 (money != 500)
        print("돈이 500원 미만입니다.")
executed in 14ms, finished 20:13:44 2021-11-04
```

돈이 딱 500원 있습니다.

```
In [18]: # # elif로 시작할 수 없음
# money = 500

# elif money == 500:
#     print("돈이 딱 500원 있습니다.")
# else:
#     print("돈이 500원 미만입니다.")
executed in 14ms, finished 20:13:44 2021-11-04
```

```
In [19]: # # 조건문 사용 후 명령이 없으면 오류
```

```
# money = 5000
# if money == 5000:

executed in 11ms, finished 20:13:44 2021-11-04
```

```
In [20]: # 조건문 사용 후 명령이 없어나 나중에 반영하고 싶을 때 pass 사용하면 오류 미발생
money = 5000

if money == 5000:
    pass      # TODO: 나중에 이슈 생성시 진행

executed in 14ms, finished 20:13:44 2021-11-04
```

```
In [21]: money = 5000

if money == 5000:
    pass      # TODO: 나중에 이슈 생성시 진행
else:
    print('5000은 아닌가봐!')

executed in 14ms, finished 20:13:44 2021-11-04
```

```
In [22]: money = 5000

if money == 5000:
    pass      # TODO: 나중에 이슈 생성시 진행
else:
    print('5000은 아닌가봐!')

executed in 11ms, finished 20:13:44 2021-11-04
```

5000은 아닌가봐!

```
In [23]: # 조건문과 명령을 한줄에 줄여서 사용해도 실행은 가능
# (초급인 경우) 실사용은 권장하지 않음
money = 500

if money == 5000: pass
else: print('5000은 아닌가봐!')

executed in 13ms, finished 20:13:44 2021-11-04
```

5000은 아닌가봐!

```
In [24]: # # 진행할 명령어도 없는데 들여쓰기를 하면 에러 발생
# x = 10

# if x == 10:
#     print('x에 들어있는 숫자는')
#     print('10입니다. ')      # unexpected indent 에러 발생

executed in 14ms, finished 20:13:44 2021-11-04
```

```
In [25]: # 조건문 아래 명령은 여러줄 실행 가능
x = 10

if x == 10:
    print('x에 들어있는 숫자는')
    print('10입니다.')

executed in 14ms, finished 20:13:44 2021-11-04
```

x에 들어있는 숫자는  
10입니다.

```
In [26]: # 조건문이 관할하는 명령문은 오로지 들여쓰기 기준으로만 판단
x = 10

if x == 10:
    print('x에 들어있는 숫자는')
    print ('10입니다.')

executed in 14ms, finished 20:13:44 2021-11-04
```

x에 들어있는 숫자는  
10입니다.

```
In [27]: x = 5

if x == 10:
    print('x에 들어있는 숫자는')
    print ('10입니다.')

executed in 13ms, finished 20:13:44 2021-11-04
```

10입니다.

- **Question:** 아래 코드에서 x의 값이 10이 아닐때 'ok'가 출력되도록 하시오

```
x = 5

if _____:
    print(____)
```

```
In [ ]: _____
```

executed in 11ms, finished 15:26:32 2021-10-06

- if, elif는 여러개 있더라도 하나만 참인 경우만 실행되며, elif는 같은 if절로 취급

```
if 조건문1:
    문자11
```

```

문장21
elif 조건문2:
    문장21
    문장22
elif 조건문3:
    문장31
    문장32
else:
    문장41
    문장42

```

```
In [28]: # if문을 동일 들여쓰기에서 반복하면 모두 실행됨
a, b, c = 10, 20, 30
```

```

if a == 10:      # 조건식이 참
    print('10')   # 출력
if b == 20:      # 조건식이 참
    print('20')
if c == 30:      # 조건식이 참
    print('30')   # 출력

```

executed in 13ms, finished 20:13:44 2021-11-04

10  
20  
30

```
In [29]: # elif는 여러번 사용가능
a, b, c = 10, 20, 30
```

```

if a == 10:      # 조건식이 참
    print('10')   # 출력
elif b == 20:    # 조건식을 검사하지 않고 건너뜀
    print('20')
elif c == 30:    # 조건식을 검사하지 않고 건너뜀
    print('30')

```

executed in 14ms, finished 20:13:44 2021-11-04

10

```
In [30]: a, b, c = 5, 20, 30
```

```

if a == 10:      # 조건식이 거짓
    print('10')
elif b == 20:    # 조건식이 참
    print('20')
elif c == 30:    # 조건식을 검사하지 않고 건너뜀
    print('30')

```

executed in 13ms, finished 20:13:44 2021-11-04

20

## 1.2 조건식의 특수경우

```
In [31]: # 조건식에 True / False에 대응되는 불리언 사용가능
if True:
    print('참')   # True는 참
else:
    print('거짓')

if False:
    print('참')
else:
    print('거짓')   # False는 거짓

if None:   # None은 False로 취급 (실수든 아니든 계산상 None이 될 경우 다수)
    print('참')
else:
    print('거짓')
```

executed in 13ms, finished 20:13:44 2021-11-04

참  
거짓  
거짓

```
In [32]: None == False
```

executed in 13ms, finished 20:13:44 2021-11-04

Out[32]: False

- 0이면 거짓, 0이 아닌 수는 참

```
In [33]: # 조건식에 True / False에 대응되는 숫자 사용가능
```

```

if 0:
    print('참')
else:
    print('거짓')   # 0은 거짓

if 1:
    print('참')   # 1은 참
else:
    print('거짓')

```

```

▼ if 0x1F: # 16진수
    print('참') # 0x1F는 참
▼ else:
    print('거짓')

▼ if 0b1000: # 2진수
    print('참') # 0b1000은 참
▼ else:
    print('거짓')

▼ if 13.5: # 실수
    print('참') # 13.5는 참
▼ else:
    print('거짓')

```

executed in 14ms, finished 20:13:44 2021-11-04

거짓  
참  
참  
참  
참

- 빈 문자열은 거짓, 그렇지 않으면 모두 참

```

In [34]: ▼ # 조건식에 True / False에 대응되는 문자열 사용 가능
▼ if 'Hello': # 문자열
    print('참') # 문자열은 참
▼ else:
    print('거짓')

▼ if '': # 공백 문자열
    print('참')
▼ else:
    print('거짓') # 빈 문자열은 거짓

▼ if '': # 빈 문자열
    print('참')
▼ else:
    print('거짓') # 빈 문자열은 거짓

```

executed in 13ms, finished 20:13:44 2021-11-04

참  
참  
거짓

- (주의) 0, None, "은 False로 취급하므로 else가 동작할 수 있음!

```

In [35]: ▼ if not 0: # not A
    print('참') # not 0은 참

▼ if not None:
    print('참') # None은 참

▼ if not '':
    print('참') # not 빈 문자열은 참

```

executed in 14ms, finished 20:13:44 2021-11-04

참  
참  
참

- 요약: False로 취급하는 것들

1. None
2. False
3. 0인 숫자: 0, 0.0, 0j
4. 비어있는 문자열/리스트/튜플/세트/딕셔너리: "", [], (), set(), {}
5. bool(), len() 등의 함수가 False 또는 0을 반환하는 경우들

### 1.3 여러개의 조건식 결합

```

In [36]: ▼ # 조건식은 여러개를 사용하여 판단해 할 수 있음
x = 10
y = 20

▼ if x == 10 and y == 20: # x가 10이면서 y가 20일 때
    print('참')
▼ else:
    print('거짓')

```

executed in 13ms, finished 20:13:44 2021-11-04

참

```

In [37]: ▼ # if문을 반복적용하여 예시 생성
▼ if x > 0:
    if x < 20: # x > 0 and x < 20
        print('20보다 작은 양수입니다.')

```

executed in 12ms, finished 20:13:44 2021-11-04

20보다 작은 양수입니다.

```
In [38]: # 20보다 작은 양수입니다.
if x > 0 and x < 20:
    print('20보다 작은 양수입니다.')
executed in 14ms, finished 20:13:44 2021-11-04
```

20보다 작은 양수입니다.

```
In [39]: # if문만 사용해서 위 코드를 변환
# 다른 언어와 물리 부동호 여러개 동시에 적용 가능
if 0 < x < 20:
    print('20보다 작은 양수입니다.')
executed in 14ms, finished 20:13:44 2021-11-04
```

20보다 작은 양수입니다.

- **Question:** 필기 시험 점수가 80점 이상이며 동시에 코딩 시험을 통과(True)하면 합격인 아래의 조건식을 완성하여 합격/불합격 여부를 출력하시오

```
written_test = 75
coding_test = True

if _____:
    print('합격')
else:
    print('불합격')
```

```
In [40]: written_test = 75
coding_test = True

# 불합격 출력
# if
executed in 13ms, finished 20:13:44 2021-11-04
```

- **Question:** 변수 x가 11과 20 사이면 '11~20', 21과 30 사이면, '21~30', 어디도 해당하지 않으면 '아무것도 해당하지 않음'을 출력하는 조건문을 만들어 아래의 변수값에 따른 결과를 출력하시오

```
In [ ]: x = 5
# 조건문 작성 및 결과 출력
executed in 6ms, finished 15:25:20 2021-10-06
```

```
In [ ]: x = 15
# 조건문 작성 및 결과 출력
```

```
In [ ]: x = 25
# 조건문 작성 및 결과 출력
```

## 2 파이썬 반복문(Loop): 처리를 반복하자

- 비슷한 작업을 반복해야 한다면 반복문을 사용할 것

- **while문:** 조건문이 참/거짓 여부에 따라 반복을 진행
- **break/continue문:** 반복문을 제어하기
- **for문:** 값이나 요소를 하나씩 가져오면서 반복을 진행

### 2.1 while문

- 조건식을 품고 있으며, 조건식이 참이면 문장을 반복하고 그렇지 않으면 반복문을 빠져나옴
- if문을 반복적으로 수행하는 것이 while문

**if** 조건식:  
문장

**while** 조건식:  
문장

초기식  
**while** 조건식:  
문장  
변화식

```
In [41]: i = 0                      # 초기식
while i < 10:                     # while 조건식
    print('Hello, world!')        # 반복할 코드
    i = i + 1                     # 변화식
executed in 12ms, finished 20:13:44 2021-11-04
```

Hello, world!

```
Hello, world!
```

i	조건식	조건판단	수행 문장	while문
0	0 < 10	참	Hello, world!	반복
1	1 < 10	참	Hello, world!	반복
2	2 < 10	참	Hello, world!	반복
3	3 < 10	참	Hello, world!	반복
4	4 < 10	참	Hello, world!	반복
5	5 < 10	참	Hello, world!	반복
6	6 < 10	참	Hello, world!	반복
7	7 < 10	참	Hello, world!	반복
8	8 < 10	참	Hello, world!	반복
9	9 < 10	참	Hello, world!	반복
10	10 < 10	거짓		종료

```
In [42]: # 반복할 명령과 반복변수 동시에 출력
i = 1 # 초기식
while i < 10: # while 조건식
    print('Hello, world!', i) # 반복할 코드
    i = i + 1 # 변화식
```

executed in 14ms, finished 20:13:45 2021-11-04

```
Hello, world! 1
Hello, world! 2
Hello, world! 3
Hello, world! 4
Hello, world! 5
Hello, world! 6
Hello, world! 7
Hello, world! 8
Hello, world! 9
```

```
In [43]: # 반복할 명령과 반복변수 동시에 출력
i = 9 # 초기식
while i > 0: # while 조건식
    print('Hello, world!', i) # 반복할 코드
    i = i - 1 # 변화식
```

executed in 12ms, finished 20:13:45 2021-11-04

```
Hello, world! 9
Hello, world! 8
Hello, world! 7
Hello, world! 6
Hello, world! 5
Hello, world! 4
Hello, world! 3
Hello, world! 2
Hello, world! 1
```

```
In [44]: # 변수를 입력받아 반복문 사용 가능
# iteration = int(input('반복 횟수 입력: '))
# i = 0 # 초기식
# while i < iteration: # while 조건식
#     print('Hello, world!', i) # 반복할 코드
#     i = i + 1 # 변화식
```

executed in 15ms, finished 20:13:45 2021-11-04

```
In [45]: # 변수를 입력받아 반복문 사용 가능
# iteration = int(input('반복 횟수 입력: '))
# i = 0 # 초기식
# while iteration > 0: # while 조건식
#     print('Hello, world!', iteration) # 반복할 코드
#     iteration = iteration - 1 # 변화식
```

executed in 15ms, finished 20:13:45 2021-11-04

## 2.2 while문의 유용한 케이스

- 일반적으로 while문은 반복 횟수는 정해지지 않은 경우 유용
- 우리가 사용하는 프로그램 중 무한 루프 개념을 사용하지 않는 프로그램은 거의 없음(ex. 자판기)

```
In [46]: import random # random 모듈을 가져옴
executed in 11ms, finished 20:13:45 2021-11-04
```

```
In [47]: # 1과 6사이의 랜덤한 수 하나를 출력하며 실행할때마다 랜덤하게 출력
random.randint(1, 6)
```

```
executed in 14ms, finished 20:13:45 2021-11-04
```

```
Out[47]: 5
```

```
In [48]: # 반복 갯수 30! 나올때까지 멈추지 않는 반복문 실행
import random # random 모듈을 가져옴

i = 0
while i != 30: # 30! 아닐 때 계속 반복
    i = random.randint(1, 6) # randint를 사용하여 1과 6 사이의 난수를 생성
    print(i)
executed in 14ms, finished 20:13:45 2021-11-04
```

```
2
6
1
2
5
2
1
2
6
6
4
1
5
2
3
```

```
In [49]: # # 무한루프 실행 가능 (무한루프는 대부분의 전자기기에 필수)
# while True: # while에 True를 지정하면 무한 루프
#     print('Hello, world!')
executed in 13ms, finished 20:13:45 2021-11-04
```

```
In [50]: # # True에 대응되는 숫자 사용 가능
# while 1: # 0이 아닌 숫자는 True로 취급하여 무한 루프로 동작
#     print('Hello, world!')
executed in 14ms, finished 20:13:45 2021-11-04
```

```
In [51]: # # True에 대응되는 문자 사용 가능
# while 'Hello': # 내용이 있는 문자열은 True로 취급하여 무한 루프로 동작
#     print('Hello, world!')
executed in 16ms, finished 20:13:45 2021-11-04
```

- **Question:** 2개의 변수 x와 y를 변화시키는 반복문을 작성하시오. 출력되는 결과는 아래와 같고 x와 y가 함께 출력되어야 합니다.

```
x = 2
y = 5

----- or -----
print(x, y)
-----
-----
2 5
4 4
8 3
16 2
32 1
```

```
In [52]: x = 2
y = 5

# while
executed in 10ms, finished 20:13:45 2021-11-04
```

## 2.3 break/continue문

- **break:** 반복문을 제어/중단 시키고 반복문을 빠져나옴, 무한루프 강제종료에 제격
- **continue:** 반복문의 흐름은 유지한채, 조건식 내 문장 실행을 건너뛰고 다시 조건식으로 돌아감

```
In [53]: i = 0
while True: # 무한 루프
    print(i)
    i = i + 1
    if i == 10: # i가 10일 때
        break # 반복문을 끝냄. while의 제어흐름을 뺏어남
executed in 14ms, finished 20:13:45 2021-11-04
```

```
0
1
2
3
4
5
6
7
8
9
```

```
In [54]: i = 0
```

```

while i < 10:          # i가 10보다 작을 때 반복. 0부터 9까지 증가하면서 10번 반복
    i = i + 1          # i를 1씩 증가시킴
    if i % 2 == 0:      # i를 2로 나누었을 때 나머지가 0이면 짝수
        continue         # 아래 코드를 실행하지 않고 조건식 이하를 건너뜀
    print(i)

```

executed in 14ms, finished 20:13:45 2021-11-04

1  
3  
5  
7  
9

```

# # 멈추기 위해서는 kernel - interrupt
# i = 1
# while i:          # i가 10보다 작을 때 반복. 0부터 9까지 증가하면서 10번 반복
#     i = i + 1      # i를 1씩 증가시킴
#     print(i)
#     continue        # 아래 코드를 실행하지 않고 조건식 이하를 건너뜀

```

executed in 13ms, finished 20:13:45 2021-11-04

```

# i = 1
# while i:          # i가 10보다 작을 때 반복. 0부터 9까지 증가하면서 10번 반복
#     i = i + 1      # i를 1씩 증가시킴
#     continue        # 아래 코드를 실행하지 않고 조건식 이하를 건너뜀
#     print(i)

```

executed in 14ms, finished 20:13:45 2021-11-04

```

# # 반복횟수를 입력받아 i를 출력하시오
# count = int(input('반복할 횟수를 입력하세요: '))
# i = 0
# while True:        # 무한 루프
#     print(i)
#     i = i + 1
#     if i == count:  # i가 입력받은 값과 같을 때
#         break        # 반복문을 끝냄

```

executed in 13ms, finished 20:13:45 2021-11-04

- **Question:** 입력한 숫자까지 해당하는 홀수를 출력하시오

```
count = int(input('반복할 횟수를 입력하세요: '))
```

```
i = 0
while True:
    i = i + 1
    -----
    -----
    -----
    -----
```

In [ ]:

executed in 2.72s, finished 15:50:25 2021-10-06

- **Question:** 0과 100사이의 숫자 중 끝자리가 6으로 끝나는 숫자만 출력하시오

```
i = 0
while True:
    if _____
        _____
    if _____
        break
    print(i)
    -----
```

In [ ]:

executed in 10ms, finished 15:54:26 2021-10-06

```

# # 무인자판기 생성
# # 커피 1잔이 판매되면 자동 종료
# # 커피 1잔은 300원
# # 잔돈 계산까지 진행
# coffee = 10

# while True:
#     money = int(input("돈을 넣어 주세요: "))
#     if money == 300:
#         print("커피를 줍니다.")
#         coffee = coffee -1
#     elif money > 300:
#         print("거스름돈 %d를 주고 커피를 줍니다." % (money -300))
#         coffee = coffee -1
#     else:
#         print("돈을 다시 돌려주고 커피를 주지 않습니다.")
#         print("남은 커피의 양은 %d개입니다." % coffee)
#     if coffee == 0:
#         print("커피가 다 떨어졌습니다. 판매를 중지 합니다.")
#         break

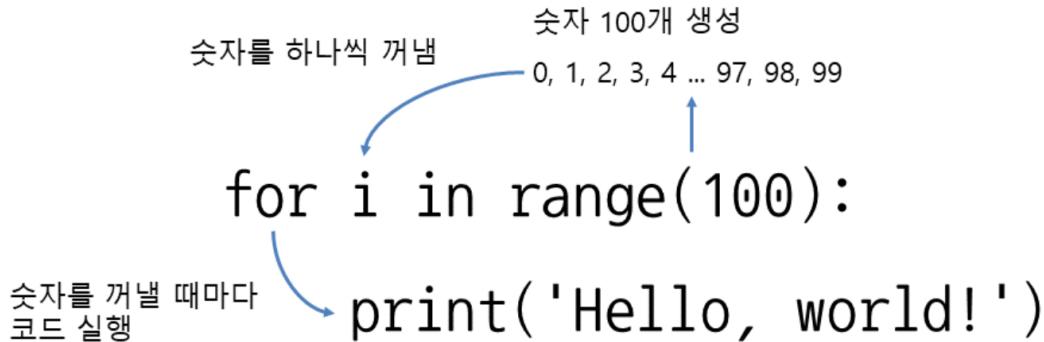
```

executed in 13ms, finished 20:13:45 2021-11-04

## 2.4 for문

- 일반적으로 for문은 반복 횟수가 정해진 경우 유용
- while문과 비슷하지만 직관적이며 문장 구조가 심플한 편
- 조건식이 없는 유한개의 반복을 실행

for 변수 in 반복횟수:  
    문장



```
In [59]: # range 시퀀스 내장함수를 사용하여 'Hello, world!' 10회 출력
for i in range(10):
    print('Hello, world!')
executed in 14ms, finished 20:13:45 2021-11-04
```

```
Hello, world!
```

```
In [60]: # range 시퀀스 내장함수를 사용하여 'Hello, world!'와 반복횟수 출력
for i in range(10):
    print('Hello, world!', i)
executed in 14ms, finished 20:13:45 2021-11-04
```

```
Hello, world! 0
Hello, world! 1
Hello, world! 2
Hello, world! 3
Hello, world! 4
Hello, world! 5
Hello, world! 6
Hello, world! 7
Hello, world! 8
Hello, world! 9
```

```
In [61]: # 50 이상 100 미만 까지 'Hello, world!' 와 반복횟수 출력
for i in range(5,10):
    print('Hello, world!', i)
executed in 13ms, finished 20:13:45 2021-11-04
```

```
Hello, world! 5
Hello, world! 6
Hello, world! 7
Hello, world! 8
Hello, world! 9
```

```
In [62]: # 0이상 10미만 2간격 'Hello, world!' 와 반복횟수 출력
for i in range(0,10,2):
    print('Hello, world!', i)
executed in 14ms, finished 20:13:45 2021-11-04
```

```
Hello, world! 0
Hello, world! 2
Hello, world! 4
Hello, world! 6
Hello, world! 8
```

```
In [63]: # range 함수의 간격은 증가가 기본이기 때문에 미실행
for i in range(10,0): # range는 숫자가 증가하는 기본 값이 양수 1
    print('Hello, world!', i)
executed in 14ms, finished 20:13:45 2021-11-04
```

```
# 10이상 0미만 -1간격 'Hello, world!' 와 반복횟수 출력
for i in range(10,0,-1):
    print('Hello, world!', i)
executed in 13ms, finished 20:13:45 2021-11-04
```

```
Hello, world! 10
Hello, world! 9
Hello, world! 8
Hello, world! 7
Hello, world! 6
```

```
~> ~> world!
Hello, world! 5
Hello, world! 4
Hello, world! 3
Hello, world! 2
Hello, world! 1
```

```
In [65]: # 90이상 -1미만 -1간격 'Hello, world!'와 반복횟수 출력
for i in range(9,-1,-1):
    print('Hello, world!', i)
executed in 14ms, finished 20:13:45 2021-11-04
```

```
Hello, world! 9
Hello, world! 8
Hello, world! 7
Hello, world! 6
Hello, world! 5
Hello, world! 4
Hello, world! 3
Hello, world! 2
Hello, world! 1
Hello, world! 0
```

```
In [66]: # reversed 함수 사용 90이상 -1미만 -1간격 'Hello, world!'와 반복횟수 출력
for i in reversed(range(10)):
    print('Hello, world!', i)
executed in 10ms, finished 20:13:45 2021-11-04
```

```
Hello, world! 9
Hello, world! 8
Hello, world! 7
Hello, world! 6
Hello, world! 5
Hello, world! 4
Hello, world! 3
Hello, world! 2
Hello, world! 1
Hello, world! 0
```

```
In [67]: # # 반복횟수를 입력받아 'Hello, world!'와 반복횟수 출력
# count = int(input('반복할 횟수를 입력하세요: '))
# for i in range(count):
#     print('Hello, world!', i)
executed in 7ms, finished 20:13:45 2021-11-04
```

- `range` 대신에 `sequence` 데이터를 반영해도 가능!

```
In [68]: # 시퀀스 숫자 반복 가능
a = [10, 20, 30, 40, 50]
for i in a:
    print(i)
executed in 11ms, finished 20:13:45 2021-11-04
```

```
10
20
30
40
50
```

```
In [69]: # 시퀀스 튜플 반복 가능
fruits = ('apple', 'orange', 'grape')
for fruit in fruits:
    print(fruit)
executed in 14ms, finished 20:13:45 2021-11-04
```

```
apple
orange
grape
```

```
In [70]: # 시퀀스 리스트 내 2개 출력씩 반복 가능
a = [(1,2), (3,4), (5,6)]
for (first, last) in a:
    print(first + last)
executed in 13ms, finished 20:13:45 2021-11-04
```

```
3
7
11
```

```
In [71]: # 시퀀스 문자열 반복 가능
for letter in 'Python':
    print(letter, end=' ')
executed in 14ms, finished 20:13:45 2021-11-04
```

```
P y t h o n
```

```
In [72]: # 시퀀스 역 문자열 반복 가능
for letter in reversed('Python'):
    print(letter, end=' ')
executed in 13ms, finished 20:13:45 2021-11-04
```

```
n o h t y P
```

- **Question:** `for`문을 사용하여 아래 리스트 각 요소에 10을 곱한 값을 한 줄로 출력하시오

```
x = [49, -17, 25, 102, 8, 62, 21]
```

```
for _____:  
    print(_____  
_____)
```

```
In [73]: x = [49, -17, 25, 102, 8, 62, 21]  
# for  
_____
```

executed in 12ms, finished 20:13:45 2021-11-04

- **Question:** for문과 if~else문을 사용하여 아래 리스트 각 요소가 홀수인 경우 2배를 짹수인 경우 -2배를 출력하시오

```
x = [1, 2, 3, 4, 5]  
  
for _____:  
    if _____:  
        _____  
        print(i)  
    else:  
        _____  
        print(i)
```

```
In [74]: # 예상출력: 2 -4 6 -8 10  
x = [1, 2, 3, 4, 5]  
# for  
_____
```

executed in 13ms, finished 20:13:45 2021-11-04

- **Question:** 총 5명의 학생 시험 점수가 있다. 60점이 합격 커트라인이라고 할때 5명의 학생이 합격인지 불합격인지 결과를 출력하라. 학생의 이름은 1번부터 5번까지이며 아래와 같은 결과가 출력되어야 한다.

```
scores = [90, 25, 67, 45, 80]
```

(자유작성)

```
1번 학생은 합격입니다.  
2번 학생은 불합격입니다.  
3번 학생은 합격입니다.  
4번 학생은 불합격입니다.  
5번 학생은 합격입니다.
```

```
In [75]: scores = [90, 25, 67, 45, 80]
```

executed in 14ms, finished 20:13:45 2021-11-04

## 2.5 반복문의 중첩

- 반복문에 반복문이 들어가는 것 중첩루프 또는 다중루프라고 함
- for문에 while문이 또는 while문에 for문이 들어갈 수 있음
- 2차원 이상의 데이터 입력이나 연산은 아주 자주 사용됨

```
In [76]: for i in range(5):          # 바깥쪽 루프  
    for j in range(5):          # 안쪽 루프  
        print('i:', i, j, sep=' ') # j값 출력, end에 ''를 지정하여 줄바꿈 대신 한 칸 띄움  
        print('j:', j, '##n', sep='') # i값 출력, 개행 문자 모양도 출력  
_____
```

executed in 12ms, finished 20:13:45 2021-11-04

```
j:0 j:1 j:2 j:3 j:4 i:0##n  
j:0 j:1 j:2 j:3 j:4 i:1##n  
j:0 j:1 j:2 j:3 j:4 i:2##n  
j:0 j:1 j:2 j:3 j:4 i:3##n  
j:0 j:1 j:2 j:3 j:4 i:4##n
```

```
In [77]: # 5개의 행 7개의 열에 별을 출력  
for i in range(5):          # 바깥쪽 루프  
    for j in range(7):          # 안쪽 루프  
        print('*', end='') # 별 출력, end에 ''를 지정하여 줄바꿈을 하지 않음  
        print()  
# 가로 방향으로 별을 다 그린 뒤 다음 줄로 넘어감  
_____
```

executed in 13ms, finished 20:13:45 2021-11-04

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

```
In [78]: # 5개의 행 별의 갯수가 증가되도록 출력  
for i in range(5):          # 바깥쪽 루프  
    for j in range(7):          # 안쪽 루프  
        if j <= i:  
            print('*', end='') # 별 출력, end에 ''를 지정하여 줄바꿈을 하지 않음  
            print() # 가로 방향으로 별을 다 그린 뒤 다음 줄로 넘어감  
_____
```

executed in 14ms, finished 20:13:45 2021-11-04

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

- **Question:** 위 예를 참고하여 대각선 방향으로 별이 1개씩만 출력하시오

```
In [ ]:
```

```
executed in 8ms, finished 16:11:02 2021-10-06
```

- **Question:** 반복문을 2개 중첩하여 구구단 2단부터 9단까지 실행하시오

(출력결과 예시)

```
2 * 1 = 2
```

```
...
```

```
9 * 9 = 81
```

```
In [ ]:
```

```
executed in 21ms, finished 16:12:11 2021-10-06
```

- **Question:** FizzBuzz 문제

1. 1에서 100까지 출력
2. 3의 배수는 Fizz 출력
3. 5의 배수는 Buzz 출력
4. 3과 5의 공배수는 FizzBuzz 출력

```
In [79]: # 예상출력: 1 2 Fizz 4 Buzz Fizz 7 8 Fizz Buzz 11 Fizz 13 14 FizzBuzz 16 17 ...
```

```
executed in 12ms, finished 20:13:45 2021-11-04
```

## 2.6 List Comprehension

- 파이썬에선 리스트 내포를 이용해서 함수를 훨씬 직관적이고 짧은 코드로 변환 가능(가독성 좋아짐)
- List 내부에 for문과 if문을 사용할 수 있기에 가능
- 식으로 지정되어 생성할 출력을 List로 표현하는 것

[출력식 for 변수 in 반복]  
list(출력식 for 변수 in 반복)

```
In [19]: # 0부터 9까지의 숫자를 생성 및 저장  
result = []  
for i in range(10):  
    result.append(i)  
  
result
```

```
executed in 17ms, finished 15:23:36 2021-11-09
```

```
Out[19]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
In [81]: # 파이썬 언어의 특수 기능으로 이런 효율적 코드가 파이썬 활용의 핵심  
# 영어의 어순 참고  
# (I need) i which is in 0~9  
[i for i in range(10)]
```

```
executed in 14ms, finished 20:13:45 2021-11-04
```

```
Out[81]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
In [82]: # 대괄호와 list() 방식 중 대괄호 방식이 훨씬 빠름고 list()는 C언어 스타일에서 따옴  
list(i for i in range(10))
```

```
executed in 12ms, finished 20:13:45 2021-11-04
```

```
Out[82]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
In [83]: # 변수의 연산 값 생성 가능  
[i + 5 for i in range(10)]
```

```
executed in 13ms, finished 20:13:45 2021-11-04
```

```
Out[83]: [5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```

```
In [84]: result = []  
for i in range(10):  
    result.append(i+5)  
  
result
```

```
executed in 14ms, finished 20:13:45 2021-11-04
```

```
Out[84]: [5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```

```
In [85]: # 위와 동일한 출력을 list comprehension 방식으로 출력  
result = [i + 5 for i in range(10)]
```

```
executed in 9ms, finished 20:13:45 2021-11-04
```

```
Out[85]: [5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```

```
In [86]: # 0이상 10미만 짝들을 2를 곱하여 리스트 출력  
d = [i * 2 for i in range(10)]  
d
```

executed in 14ms, finished 20:13:45 2021-11-04

Out[86]: [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]

```
for 변수 in 반복:  
    if 조건식:  
        출력식  
    else:  
        다른출력  
  
[출력식 for 변수 in 반복 if 조건식 else 다른출력]  
list(출력식 for 변수 in 반복 if 조건식 else 다른출력)
```

```
In [87]: # 0~9 숫자 중 2의 배수인 숫자(짝수)로 리스트 생성  
result = [i for i in range(10) if i % 2 == 0]  
result
```

executed in 14ms, finished 20:13:45 2021-11-04

Out[87]: [0, 2, 4, 6, 8]

```
In [88]: # 위의 결과를 일반적인 반복문 및 조건문으로 구현  
result = []  
for i in range(10):  
    if i % 2 == 0:  
        result.append(i)  
  
result
```

executed in 13ms, finished 20:13:45 2021-11-04

Out[88]: [0, 2, 4, 6, 8]

```
In [89]: # 0~9 숫자 중 홀수에 5를 더하여 리스트 생성  
b = [i + 5 for i in range(10) if i % 2 == 1]  
b
```

executed in 14ms, finished 20:13:45 2021-11-04

Out[89]: [6, 8, 10, 12, 14]

```
[출력식 for 변수1 in 반복1  
      for 변수2 in 반복2 if 조건식 else 다른출력]  
  
[출력식 for 변수1 in 반복1  
      for 변수2 in 반복2  
      ...  
      for 변수n in 반복n if 조건식 else 다른출력]  
  
[변수1 + 변수2 + ... + 변수n for 변수1 in 반복1  
      for 변수2 in 반복2  
      ...  
      for 변수n in 반복n if 조건식 else 다른출력]  
  
list(출력식 for 변수1 in 반복1  
      for 변수2 in 반복2  
      ...  
      for 변수n in 반복n if 조건식 else 다른출력)
```

- **Question:** 다음 리스트 중 홀수에만 2를 곱하여 저장하는 코드가 있다 이를 리스트 내포를 사용하여 변환하라.

```
numbers = [1, 2, 3, 4, 5]  
result = []  
for n in numbers:  
    if n % 2 == 1:  
        result.append(n*2)
```

```
In [90]: # 예상결과: [2, 6, 10]
```

executed in 14ms, finished 20:13:45 2021-11-04

- **Question:** 리스트 표현식을 사용하여 구구단을 출력하시오 리스트 내부 요소 출력 형식은 '2 \* 1 = 2'

```
In [ ]:
```

executed in 24ms, finished 10:15:19 2021-10-07

## 2.7 차원증가

```
리스트 = [[값, 값], [값, 값], [값, 값]]
```

```
In [91]: # 1차원 데이터
```

```

a = [10, 20]
a
executed in 14ms, finished 20:13:45 2021-11-04
Out[91]: [10, 20]

In [92]: # 2차원 데이터의 세팅
a = [[10, 20], [30, 40], [50, 60]]
a
executed in 13ms, finished 20:13:45 2021-11-04
Out[92]: [[10, 20], [30, 40], [50, 60]]

In [93]: # 2차원 데이터의 세팅
a = [[10, 20],
      [30, 40],
      [50, 60]]
a
executed in 13ms, finished 20:13:45 2021-11-04
Out[93]: [[10, 20], [30, 40], [50, 60]]

In [94]: # 데이터의 행의수와 열의수 출력
import numpy as np

np.shape(a)
executed in 119ms, finished 20:13:45 2021-11-04
Out[94]: (3, 2)

In [95]: # 행렬의 내부 값 선택
a[0]
executed in 14ms, finished 20:13:45 2021-11-04
Out[95]: [10, 20]

In [96]: # 행렬의 내부 값 선택
a[1]
executed in 12ms, finished 20:13:45 2021-11-04
Out[96]: [30, 40]

In [97]: a[0]
executed in 12ms, finished 20:13:45 2021-11-04
Out[97]: [10, 20]

In [98]: # 행렬의 내부 값 선택
a[0][0] # 행렬(Matrix): A(1, 1)
executed in 14ms, finished 20:13:45 2021-11-04
Out[98]: 10

In [99]: a[0][1]
executed in 13ms, finished 20:13:45 2021-11-04
Out[99]: 20

In [100]: # 행렬의 내부 값 치환
a[0][1] = 1000
a
executed in 13ms, finished 20:13:45 2021-11-04
Out[100]: [[10, 2000], [30, 40], [50, 60]]


• 톰니형 리스트(Jagged List) 생성 가능

In [101]: a = [[10, 20],
           [500, 600, 700],
           [9],
           [30, 40],
           [8],
           [800, 900, 1000]]
a
executed in 14ms, finished 20:13:46 2021-11-04
Out[101]: [[10, 20], [500, 600, 700], [9], [30, 40], [8], [800, 900, 1000]]

In [102]: a[0]
executed in 13ms, finished 20:13:46 2021-11-04
Out[102]: [10, 20]

In [103]: # 행렬의 내부 값 추가
a[0].append(100)
a
executed in 13ms, finished 20:13:46 2021-11-04
Out[103]: [[10, 20, 100], [500, 600, 700], [9], [30, 40], [8], [800, 900, 1000]]
```

리스트 = [(값, 값), (값, 값), (값, 값)]  
 튜플 = ([값, 값], [값, 값], [값, 값])  
 뮤플 = ((값, 값), (값, 값), (값, 값))

```
In [104]: a = ((10, 20), (30, 40), (50, 60)) # 튜플 안에 튜플을 넣은 2차원 튜플
          b = ([10, 20], [30, 40], [50, 60]) # 튜플 안에 리스트를 넣음
          c = [(10, 20), (30, 40), (50, 60)] # 리스트 안에 튜플을 넣음
executed in 14ms, finished 20:13:46 2021-11-04
```

```
In [105]: # a[0][0] = 500 # 안쪽 튜플은 변경할 수 없음. TypeError 발생
executed in 12ms, finished 20:13:46 2021-11-04
```

```
In [106]: # a[0] = (500, 600) # 바깥쪽 튜플은 변경할 수 없음. TypeError 발생
executed in 14ms, finished 20:13:46 2021-11-04
```

```
In [107]: b[0]
executed in 15ms, finished 20:13:46 2021-11-04
```

```
Out[107]: [10, 20]
```

```
In [108]: b[0][0] = 500 # 안쪽 리스트는 변경할 수 있음
b
executed in 12ms, finished 20:13:46 2021-11-04
```

```
Out[108]: ([500, 20], [30, 40], [50, 60])
```

```
In [109]: # b[0] = (500, 600) # 바깥쪽 튜플은 변경할 수 없음. TypeError 발생
executed in 14ms, finished 20:13:46 2021-11-04
```

```
In [110]: c[0]
executed in 14ms, finished 20:13:46 2021-11-04
```

```
Out[110]: (10, 20)
```

```
In [111]: # c[0][0] = 500 # 안쪽 튜플은 변경할 수 없음. TypeError 발생
executed in 14ms, finished 20:13:46 2021-11-04
```

```
In [112]: c[0] = (500, 600) # 바깥쪽 리스트는 변경할 수 있음
c
executed in 13ms, finished 20:13:46 2021-11-04
```

```
Out[112]: [(500, 600), (30, 40), (50, 60)]
```

```
In [113]: a = [[10, 20], [30, 40], [50, 60]]
a
executed in 14ms, finished 20:13:46 2021-11-04
```

```
Out[113]: [[10, 20], [30, 40], [50, 60]]
```

```
In [ ]: !pip install pprintpp
```

```
In [114]: # pprint 외장함수로 데이터를 차원에 맞게 출력 가능
import pprint
pprint pprint(a, width=10)
executed in 13ms, finished 20:13:46 2021-11-04
[[10, 20],
 [30, 40],
 [50, 60]]
```

```
In [115]: # pprint 외장함수로 데이터를 차원에 맞게 출력 가능
# 사용 함수의 코드를 짧게 작성하기 위해 from 모듈 import 함수 사용
from pprint import pprint
pprint(a, width=10)
executed in 14ms, finished 20:13:46 2021-11-04
[[10, 20],
 [30, 40],
 [50, 60]]
```

## 2.8 2차원에서의 반복문

- 추출하는 값의 길이와 반복문에서의 변수 갯수는 일치해야 함

리스트[세로인덱스][가로인덱스]

```
In [116]: import numpy as np
a = [[10, 20], [30, 40], [50, 60]]
print(a, np.shape(a), len(a), len(a[0]))
executed in 13ms, finished 20:13:46 2021-11-04
[[10, 20], [30, 40], [50, 60]] (3, 2) 3 2
```

```
In [117]: len(a)
executed in 13ms, finished 20:13:46 2021-11-04
```

```
Out[117]: 3
```

```
In [118]: a[0]
executed in 14ms, finished 20:13:46 2021-11-04
```

```
Out[118]: [10, 20]
```

```
In [119]: len(a[0])
```

executed in 13ms, finished 20:13:46 2021-11-04

```
Out[119]: 2
```

```
In [120]: # 기존 프로그램의 인덱스 접근
a = [[10, 20], [30, 40], [50, 60]]
for i in range(3):
    for j in range(2):
        print(a[i][j], end=' ')
print()
```

executed in 13ms, finished 20:13:46 2021-11-04

```
10 20
30 40
50 60
```

```
In [121]: # 데이터가 변경되어도 실행되지 않는 값이 존재
a = [[10, 20], [30, 40], [50, 60], [70, 80]]
for i in range(3):
    for j in range(2):
        print(a[i][j], end=' ')
print()
```

executed in 14ms, finished 20:13:46 2021-11-04

```
10 20
30 40
50 60
```

```
In [122]: # 기존 프로그램의 인덱스 접근
a = [[10, 20], [30, 40], [50, 60]]
for i in range(len(a)):
    for j in range(len(a[i])):
        print(a[i][j], end=' ')
print()
```

executed in 13ms, finished 20:13:46 2021-11-04

```
10 20
30 40
50 60
```

```
In [123]: # 데이터가 변경되어도 모두 실행
a = [[10, 20], [30, 40], [50, 60], [70, 80]]
for i in range(len(a)):
    for j in range(len(a[i])):
        print(a[i][j], end=' ')
print()
```

executed in 13ms, finished 20:13:46 2021-11-04

```
10 20
30 40
50 60
70 80
```

```
In [124]: # 데이터의 각 row 길이가 달라도 모두 실행
a = [[10, 20], [30, 40], [50, 60], [70, 80, 90, 100]]
for i in range(len(a)):
    for j in range(len(a[i])):
        print(a[i][j], end=' ')
print()
```

executed in 14ms, finished 20:13:46 2021-11-04

```
10 20
30 40
50 60
70 80 90 100
```

```
In [125]: # 2차원 데이터를 반복문의 중첩을 사용해서 연산도 가능
a = [[10, 20], [30, 40], [50, 60]]
for i in range(len(a)):
    for j in range(len(a[i])):
        print(a[i][j]+5, end=' ')
print()
```

executed in 15ms, finished 20:13:46 2021-11-04

```
15 25
35 45
55 65
```

```
In [126]: # while문을 사용해서 동일결과 출력
a = [[10, 20], [30, 40], [50, 60]]
i = 0
while i < len(a):
    x, y = a[i]           # 반복할 때 리스트의 크기 활용(세로 크기)
    print(x, y)           # 요소 두 개를 한꺼번에 가져오기
    i = i + 1             # 인덱스를 1 증가시킴
```

executed in 13ms, finished 20:13:46 2021-11-04

```
10 20
30 40
50 60
```

```
In [127]: # while문을 사용해서 동일결과 출력
a = [[10, 20], [30, 40], [50, 60]]

i = 0
while i < len(a):          # 세로 크기
    i = i + 1
```

```

        j = 0
        while j < len(a[i]): # 가로 크기
            print(a[i][j], end=' ')
            j = j + 1 # 가로 인덱스를 1 증가시킴
        print() # 세로 인덱스를 1 증가시킴
    i = i + 1

```

executed in 13ms, finished 20:13:46 2021-11-04

10 20  
30 40  
50 60

In [128]: a = [[10, 20], [30, 40], [50, 60]]  
for i in a:  
 print(i)

executed in 15ms, finished 20:13:46 2021-11-04

[10, 20]  
[30, 40]  
[50, 60]

In [129]: # 인덱스 위치를 신경을 필요가 없이 값 자체를 하나씩 반복 가능  
a = [[10, 20], [30, 40], [50, 60]]  
for i in a:  
 for j in i:  
 print(j, end=' ')  
 print()

executed in 13ms, finished 20:13:46 2021-11-04

10 20  
30 40  
50 60

In [130]: # 파이썬의 간접으로 반복 동안 2개(그 이상도 가능)의 출력값을 활용 가능  
a = [[10, 20], [30, 40], [50, 60]]  
for x, y in a:  
 print(x, y)

executed in 14ms, finished 20:13:46 2021-11-04

10 20  
30 40  
50 60

In [131]: line = [] # 안쪽 리스트로 사용할 빈 리스트 생성  
for j in range(2):  
 line.append(1) # 안쪽 리스트에 0 추가  
  
line

executed in 14ms, finished 20:13:46 2021-11-04

Out[131]: [1, 1]

In [132]: a = [] # 빈 리스트 생성  
for i in range(3):  
 line = [] # 안쪽 리스트로 사용할 빈 리스트 생성  
 for j in range(2):  
 line.append(1) # 안쪽 리스트에 0 추가  
 a.append(line) # 전체 리스트에 안쪽 리스트를 추가  
  
pprint(a, width=10)

executed in 13ms, finished 20:13:46 2021-11-04

[[1, 1],  
 [1, 1],  
 [1, 1]]

In [133]: # 리스트 표현식으로 재현  
# line = [] # 안쪽 리스트로 사용할 빈 리스트 생성  
# for j in range(2):  
# line.append(1) # 안쪽 리스트에 0 추가  
# line  
  
row = [1 for j in range(2)]  
row

executed in 15ms, finished 20:13:46 2021-11-04

Out[133]: [1, 1]

In [134]: # 리스트 표현식으로 위 코드 결과를 재현  
row = [1 for j in range(2)]  
a = [row for i in range(3)]  
  
from pprint import pprint  
pprint(a, width=10)

executed in 14ms, finished 20:13:46 2021-11-04

[[1, 1],  
 [1, 1],  
 [1, 1]]

In [135]: # 반복문의 중첩도 한줄의 list comprehension으로 전환 가능  
a = [[1 for j in range(2)] for i in range(3)]  
pprint(a, width=10)

executed in 13ms, finished 20:13:46 2021-11-04

[[1, 1],  
 [1, 1],  
 [1, 1]]

```

In [136]: # 대괄호를 하나 삭제한다면?
[1 for j in range(2) for i in range(3)]
executed in 13ms, finished 20:13:46 2021-11-04
Out[136]: [1, 1, 1, 1, 1, 1]

In [137]: [1 for j in range(2)]
executed in 15ms, finished 20:13:46 2021-11-04
Out[137]: [1, 1]

In [138]: [1]*2
executed in 12ms, finished 20:13:46 2021-11-04
Out[138]: [1, 1]

In [139]: # 반복문 중첩도 효율적 연산으로 반복문을 줄여 동일결과 출력 가능
a = [[1]*2 for i in range(3)]
pprint(a, width=10)
executed in 14ms, finished 20:13:46 2021-11-04
[[1, 1],
 [1, 1],
 [1, 1]]

In [140]: a = []      # 빈 리스트 생성
for i in range(3):
    line = []          # 안쪽 리스트로 사용할 빈 리스트 생성
    for j in range(2):
        line.append(1)  # 안쪽 리스트에 0 추가
    a.append(line)      # 전체 리스트에 안쪽 리스트를 추가
pprint(a, width=10)
executed in 14ms, finished 20:13:46 2021-11-04
[[1, 1],
 [1, 1],
 [1, 1]]

In [141]: list(range(3))
executed in 13ms, finished 20:13:46 2021-11-04
Out[141]: [0, 1, 2]

In [142]: # 가로의 길이를 임의의 값으로 입력받아 각 행별 출력 길이 조절 가능
a = [3, 1, 3, 2, 5]      # 가로 크기를 저장한 리스트
b = []                    # 빈 리스트 생성

for i in a:              # 가로 크기를 저장한 리스트로 반복
    line = []            # 안쪽 리스트로 사용할 빈 리스트 생성
    for j in range(i):   # 리스트 a에 저장된 가로 크기만큼 반복
        line.append(1)
    b.append(line)        # 리스트 b에 안쪽 리스트를 추가
pprint(b, width=20)
executed in 13ms, finished 20:13:46 2021-11-04
[[1, 1, 1],
 [1],
 [1, 1, 1],
 [1, 1],
 [1, 1, 1, 1, 1]]

In [143]: # 위와 동일한 결과를 list comprehension으로 전환 가능
a = [[1]*i for i in [3, 1, 3, 2, 5]]
pprint(a, width=20)
executed in 15ms, finished 20:13:46 2021-11-04
[[1, 1, 1],
 [1],
 [1, 1, 1],
 [1, 1],
 [1, 1, 1, 1, 1]]

In [144]: # 변수끼리의 = 기호는 할당이 아니라 동일한 변수로 반영
a = [[10, 20], [30, 40]]
b = a
b[0][0] = 500
print(a, b)
executed in 13ms, finished 20:13:46 2021-11-04
[[500, 20], [30, 40]] [[500, 20], [30, 40]]

In [145]: # 변수끼리의 = 기호는 할당이 아니라 동일한 변수로 반영
# 복사를 해서 값만 할당해줘야 함
# 2차원 이상에서는 이 방법도 적용 불가
a = [[10, 20], [30, 40]]
b = a.copy()
b[0][0] = 500
print(a, b)
executed in 14ms, finished 20:13:46 2021-11-04
[[500, 20], [30, 40]] [[500, 20], [30, 40]]

```

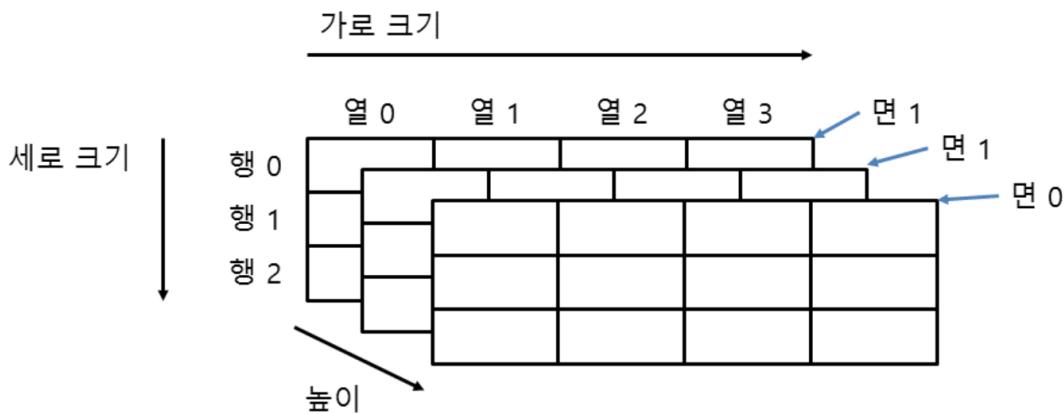
```
# 딥코피는 같은 객체를 가진 다른 객체를 만드는 기법으로서 쓰는 주제를 살펴보자
# deepcopy 외장함수로 내부 값들을 복사하여 전달
a = [[10, 20], [30, 40]]
import copy
b = copy.deepcopy(a)
b[0][0] = 500
print(a, b)

executed in 14ms, finished 20:13:46 2021-11-04
[[10, 20], [30, 40]] [[500, 20], [30, 40]]
```

## 2.9 반복문으로 3차원 리스트 생성

- 3차원 리스트 생성 및 값 반영:

```
리스트 = [[[값, 값], [값, 값]], [[값, 값], [값, 값]], [[값, 값], [값, 값]]]
리스트[높이인덱스][세로인덱스][가로인덱스] = 값
```



```
In [147]: [0 for col in range(3)]
executed in 12ms, finished 20:13:46 2021-11-04
```

```
Out[147]: [0, 0, 0]
```

```
In [148]: # list comprehension 활용 2차원 데이터 출력
a = [[0 for col in range(3)] for row in range(4)]
pprint(a, width=20)

executed in 14ms, finished 20:13:46 2021-11-04
[[0, 0, 0],
 [0, 0, 0],
 [0, 0, 0],
 [0, 0, 0]]
```

```
In [149]: # list comprehension 활용 2차원 데이터 출력을 중복적용하여 3차원 데이터 출력
a = [[0 for col in range(3)] for row in range(4)]
mat3d = [a for depth in range(2)]
pprint(mat3d, width=20)

executed in 14ms, finished 20:13:46 2021-11-04
[[[0, 0, 0],
 [0, 0, 0],
 [0, 0, 0]],
 [[0, 0, 0],
 [0, 0, 0],
 [0, 0, 0]],
 [[0, 0, 0],
 [0, 0, 0],
 [0, 0, 0]],
 [[0, 0, 0],
 [0, 0, 0],
 [0, 0, 0]]]
```

```
In [150]: # list comprehension 활용 2차원 데이터 출력을 중복적용을 한줄로 표현하여 3차원 데이터 출력
mat3d = [[[0 for col in range(3)] for row in range(4)] for depth in range(2)]
pprint(mat3d, width=20)

executed in 13ms, finished 20:13:46 2021-11-04
[[[0, 0, 0],
 [0, 0, 0],
 [0, 0, 0]],
 [[0, 0, 0],
 [0, 0, 0],
 [0, 0, 0]],
 [[0, 0, 0],
 [0, 0, 0],
 [0, 0, 0]],
 [[0, 0, 0],
 [0, 0, 0],
 [0, 0, 0]]]]
```

```
In [151]: # 3차원 데이터의 시각화는 어려운 부분이라, 2차원 데이터를 반복시켜 확인하는게 일반적
import pandas as pd

for df in [[[0 for col in range(3)] for row in range(4)] for depth in range(2)]:
    display(pd.DataFrame(df))

executed in 329ms, finished 20:13:47 2021-11-04
```

0	1	2
0	0	0
1	0	0
2	0	0

3 0 0 0

0 1 2

0 0 0 0

1 0 0 0

2 0 0 0

3 0 0 0