학습 내용

1부. 프로그래밍 언어 기본

1장. 파이썬 개요 및 개발환경 구성

2장. 자료형과 연산자

3장. 데이터 구조

4장. 제어문

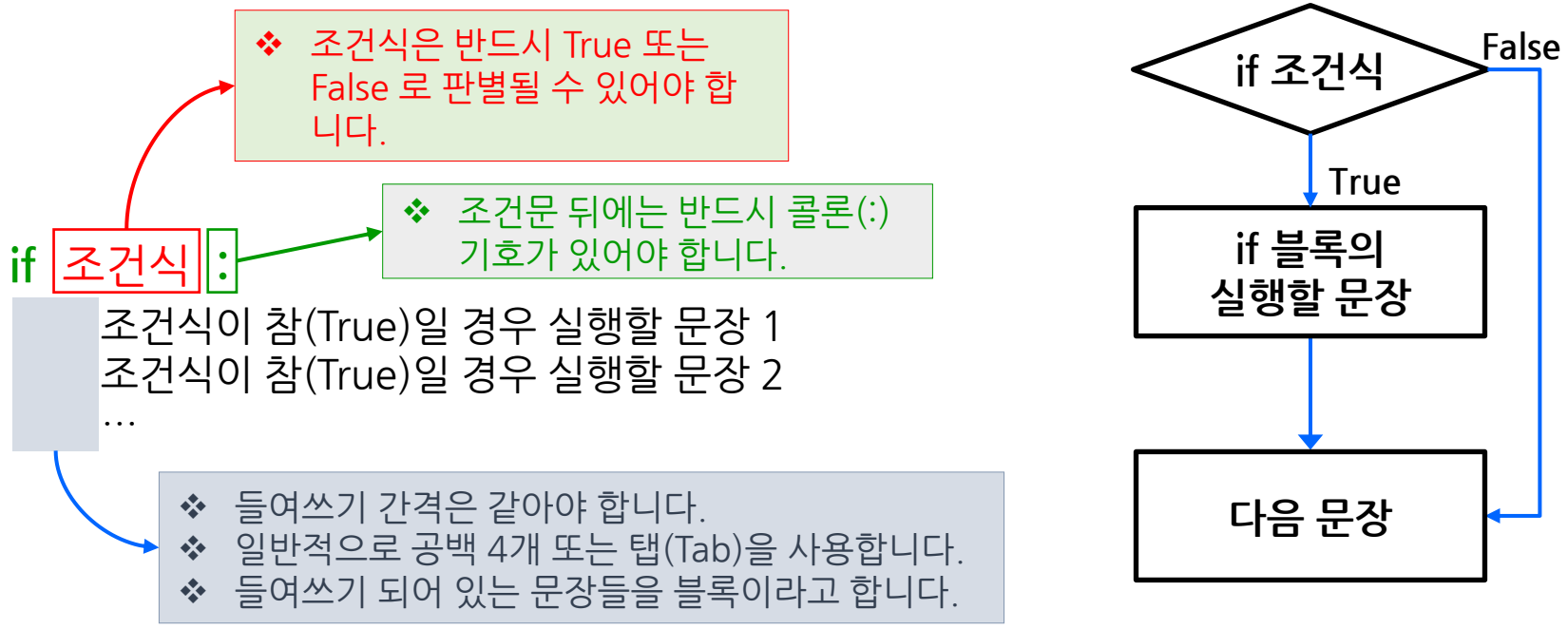
- **1. 조건문**
- **2. 반복문**
- **3. 중첩 루프**
- **4. 중첩 루프 탈출**

5장. 함수

1.1. if

1절. 조건문

- 조건문에는 if 라는 키워드를 사용
- if 다음에는 '조건식'이 존재하는데 이 '조건식'이 참(True)이면 들여쓰기 한 문장 실행
- if 문장 끝에는 콜론(:) 을 입력
  - 콜론은 블록의 시작을 의미
- if 문의 '조건식'이 참(True)일 때 실행되는 문장은 들여쓰기를 해야 합니다.



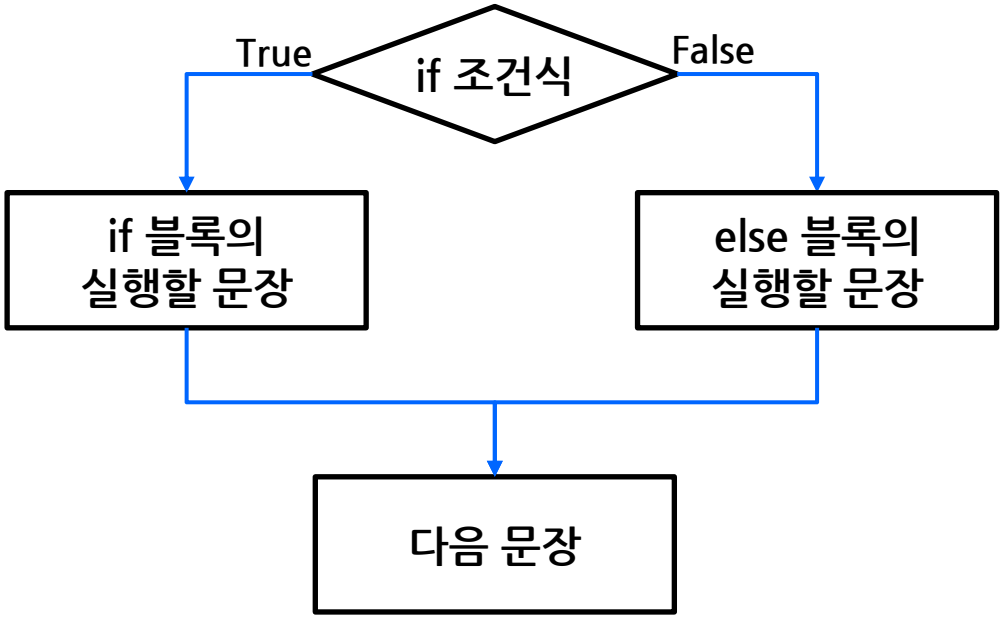
2) if ~ else

- 조건식이 참일 경우에 실행할 문장과 거짓일 경우 실행할 문장이 다를 경우 if 구문에 else 구문을 추가
- else 구문은 if 문의 조건식이 False일 경우 실행하는 블록을 정의
- else는 단독으로 사용될 수 없으며 반드시 if와 같이 사용돼야 함

```
if 조건식 :
    조건식이 참(True)일 경우 실행할 문장
    …
else :
    조건식이 거짓(False) 일 경우 실행할 문장
    …
```

❖ else 뒤에도 반드시 콜론(:) 기호가 있어야 합니다.

❖ if 구문 다음의 들여쓰기 간격과 else 구문 다음의 들여쓰기 간격은 같아야 합니다.



3) if ~ elif ~ else

- 여러 개 조건식을 사용하려면 elif 구문을 이용
- elif는 단독으로 사용 안됨

```
if 조건식1 :
    조건식1이 참(True)일 경우 실행할 문장
    …
elif 조건식2 :
    조건식1이 거짓(False)이고 조건식2가 참(True)일 경우 실행할 문장
    …
elif 조건식3 :
    조건식1, 2가 거짓(False)이고 조건식3이 참(True)일 경우 실행할 문장
    …
else :
    모든 조건식이 거짓(False)일 경우 실행할 문장
```

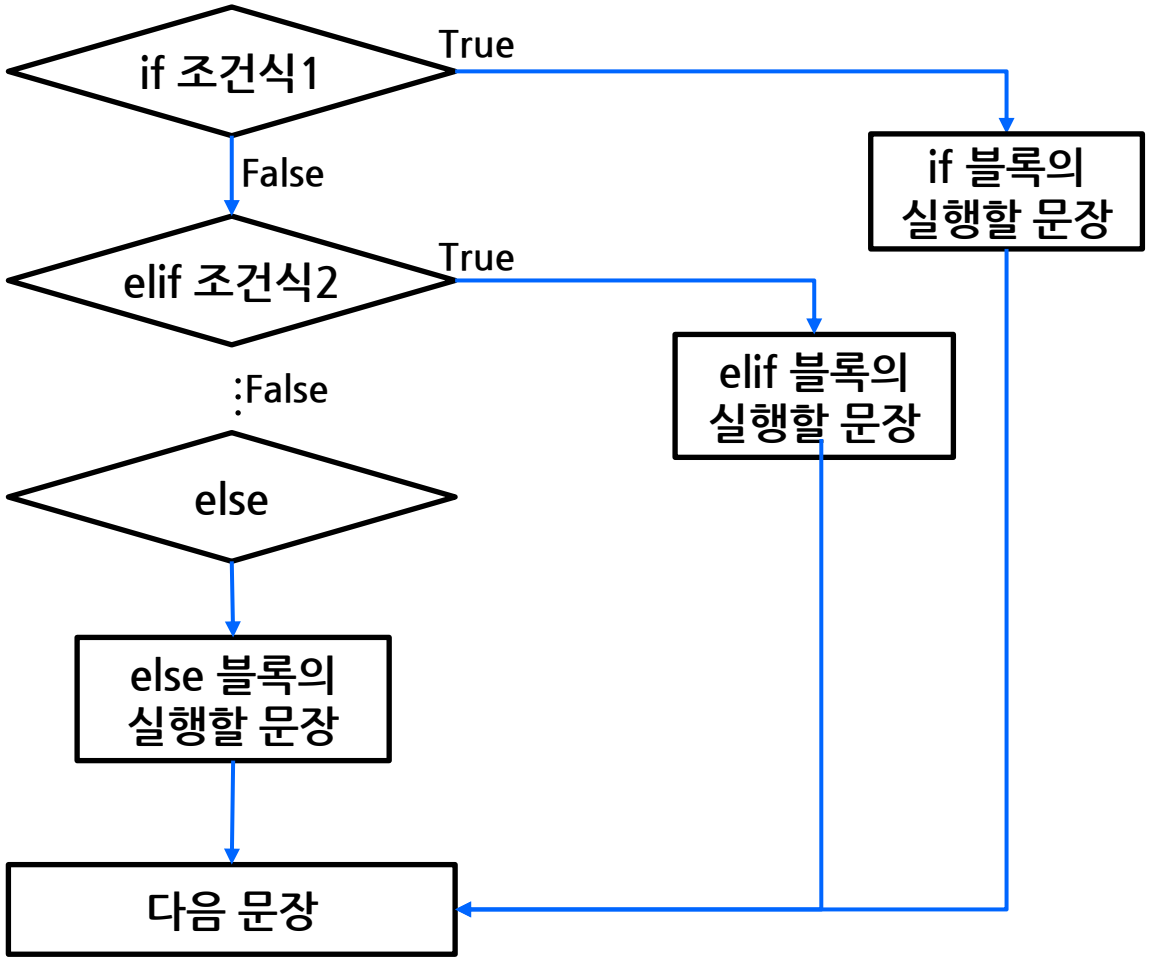
❖ elif 구문은 여러 개 넣을 수 있습니다.
❖ if 구문 없이 elif 구문만 사용할 수 없습니다.

❖ else 구문은 선택사항 입니다.

❖ if 구문 다음의 들여쓰기 간격과 elif 구문 다음의 들여쓰기 간격은 같아야 합니다.

3) if ~ elif ~ else

- if~elif~else 순서도



2.1. for

2절. 반복문

- for ~ in 반복문
- 나열 가능한 자료에서 자료를 모두 소비할 때까지 처리

❖ 나열 가능한 자료에서 하나씩 빼내서 저장할 변수 또는 변수의 목록입니다.
❖ for 구문 블록에서 사용합니다.
❖ 단일 변수 또는 여러 개 변수가 올 수 있습니다.

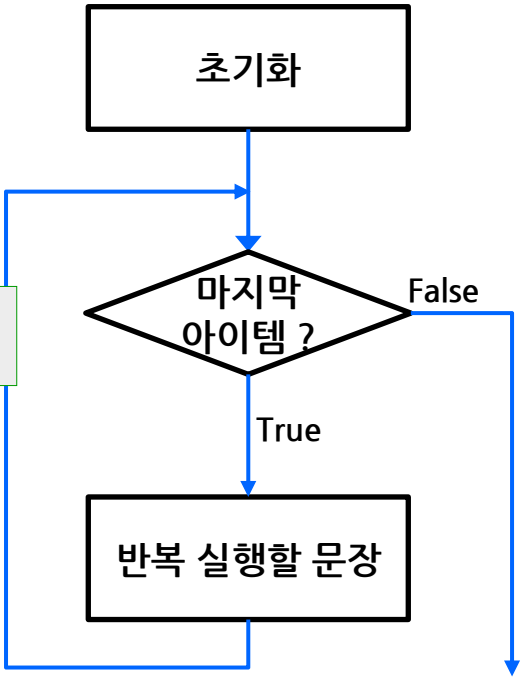
❖ 나열 가능한 자료를 저장한 변수의 이름 또는 실행 결과입니다.
❖ 리스트, 튜플, 딕셔너리, 셋 등이 될 수 있습니다.

```
for 변수 in 나열가능한자료 :
    변수의 값을 처리할 문장 1
    변수의 값을 처리할 문장 2
    …
else :
    변수에 값이 할당되지 않는 경우 실행할 문장
```

❖ for ~ in 반복문 뒤에는 반드시 콜론(:) 기호가 있어야 합니다.

❖ else 구분은 선택사항 입니다.
❖ 필요 없을 경우에는 사용하지 않아도 됩니다.

❖ 들여쓰기 되어 있는 문장들을 블록이라고 합니다.
❖ 들여쓰기 간격은 같아야 합니다.
❖ 일반적으로 공백 4개 또는 탭(Tab)을 사용합니다.



range(from, to, by)

- for 문장의 items 객체 위치에 range(*start*, *stop*, *step*)함수를 이용하여 반복문을 실행시킬 수 있음. range() 함수를 이용하면 인덱스 위주의 반복을 실행시킬 수 있음

```
1  for i in range(0, 10):
2      print(i, end='₩t')
```

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

- range() 함수의 *start*가 생략되면 0부터 시작.
- range() 함수의 *step*은 얼마씩 증가시킨 값을 갖게 할 것인지 결정
  - range(0, 10, 3)으로 수정하면 출력되는 결과는 [0, 3, 6, 9]

```
1  favorite_hobby = ['reading', 'fishing', 'shopping']
2  for hobby in favorite_hobby :
3      print('%s is my favorite hobby' % hobby)
```

```
reading is my favorite hobby
fishing is my favorite hobby
shopping is my favorite hobby
```

2.2. while

2절. 반복문

- while 반복문
- 조건문이 참일 동안 계속 실행

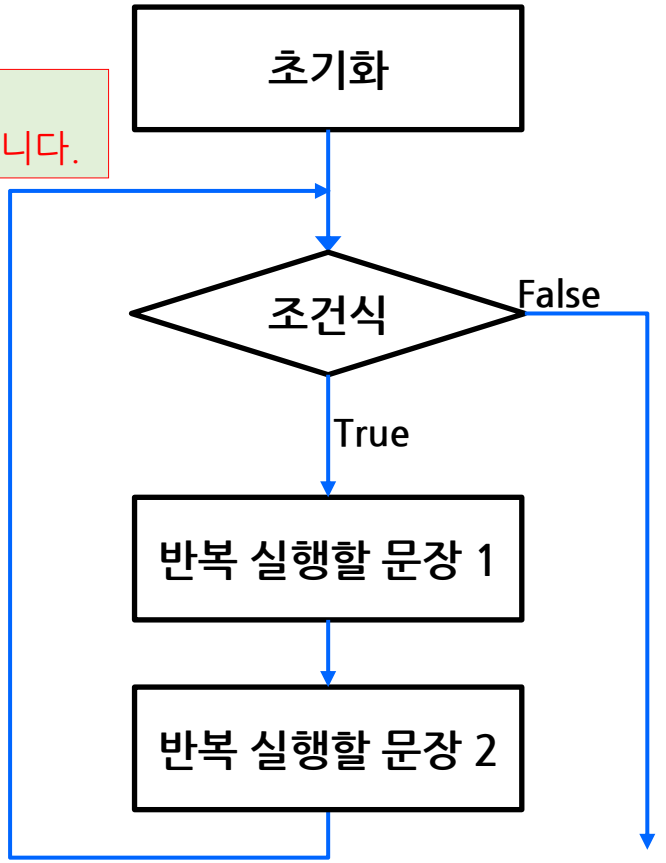
❖ 조건문을 검사해서 결과가 참(True)일 동안 블록을 반복 실행합니다.
❖ 조건문의 결과가 거짓(False)이면 else 블록을 1회 실행하고 빠져 나옵니다.

```
while 조건문 :
    조건이 참(True)일 동안 반복 실행할 구문
    …
else :
    조건이 거짓(False)일 경우 실행할 문장
```

❖ while 조건문 뒤에는 반드시 콜론(:) 기호가 있어야 합니다.

❖ else 구문은 선택사항 입니다.
❖ 필요 없을 경우에는 사용하지 않아도 됩니다.

❖ 들여쓰기 되어 있는 문장들을 블록이라고 합니다.
❖ 들여쓰기 간격은 같아야 합니다.
❖ 일반적으로 공백 4개 또는 탭(Tab)을 사용합니다.



2.2. while

2절. 반복문

```
1  i = 0
2  while i <= 10:
3      print(i, end=' ')
4      i = i + 1
```

```
0 1 2 3 4 5 6 7 8 9 10
```

```
1  num = 0
2  while num <= 10:
3      if num % 2 == 1:
4          print(num, end=' ')
5      num += 1
```

```
1 3 5 7 9
```

2.3. break & continue

2절. 반복문

- break를 만나면 break를 포함하는 반복문을 완전히 탈출
- continue는 반복문 내에서 continue 이후의 문장을 건너뜀

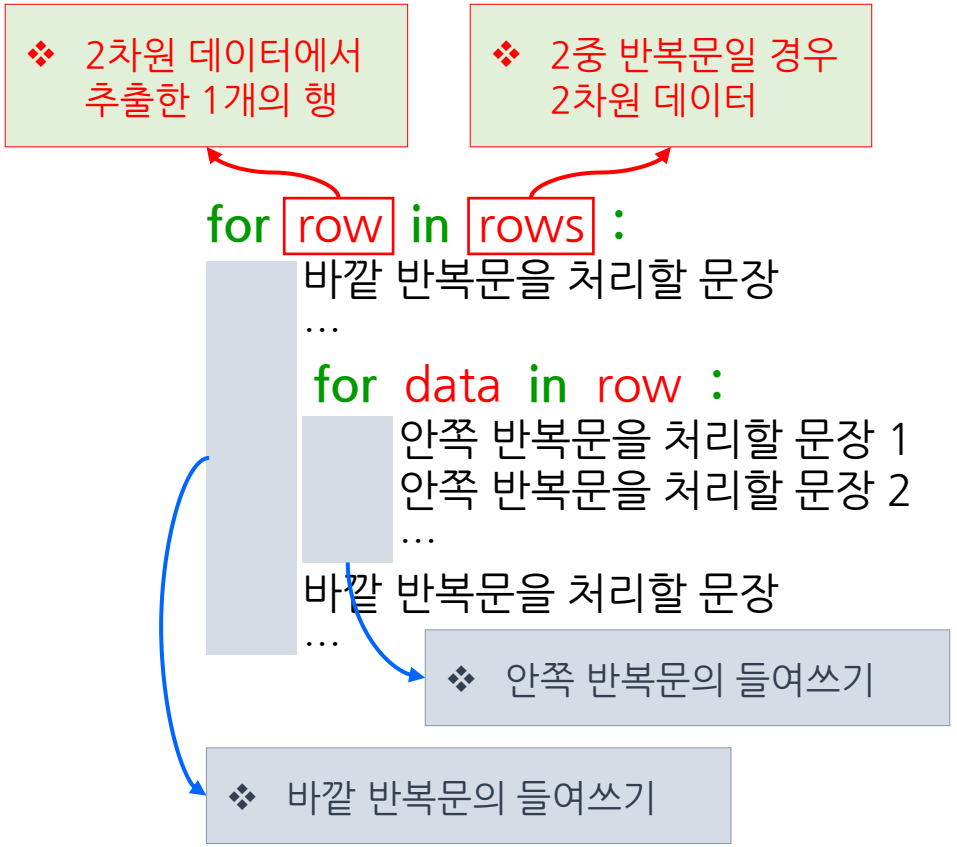
```
1  num = 0
2  while num < 10:
3      num += 1
4      if num == 5:
5          break
6      print(num, end=' ')
```

```
1 2 3 4
```

```
1  num = 0
2  while num < 10:
3      num += 1
4      if num == 5:
5          continue
6      print(num, end=' ')
```

```
1 2 3 4 6 7 8 9 10
```

3절. 중첩 루프


❖ 2차원 데이터에서 추출한 1개의 행
❖ 2중 반복문일 경우 2차원 데이터
for row in rows :
바깥 반복문을 처리할 문장
…
for data in row :
안쪽 반복문을 처리할 문장 1
안쪽 반복문을 처리할 문장 2
…
바깥 반복문을 처리할 문장
…
❖ 안쪽 반복문의 들여쓰기
❖ 바깥 반복문의 들여쓰기

3.1. 2차원 리스트 인덱싱

3절. 중첩 루프

```
1  list_2d = [[1,2,3,4,5], [11,12,13,14,15], [21,22,23,24,25]]
2  print(list_2d)
```

```
[[1, 2, 3, 4, 5], [11, 12, 13, 14, 15], [21, 22, 23, 24, 25]]
```

2차원 리스트에 대한 인덱싱은 '변수명[행][열]' 형식으로 지정

```
1  list_2d[1][2]
```

```
13
```

2차원 리스트를 하나의 반복문으로 처리하면 리스트 안의 리스트를 받음

```
1  for data in list_2d :
2      print(data)
```

```
[1, 2, 3, 4, 5]
[11, 12, 13, 14, 15]
[21, 22, 23, 24, 25]
```

2차원 리스트에서 데이터를 조회하려면 중첩 루프를 이용

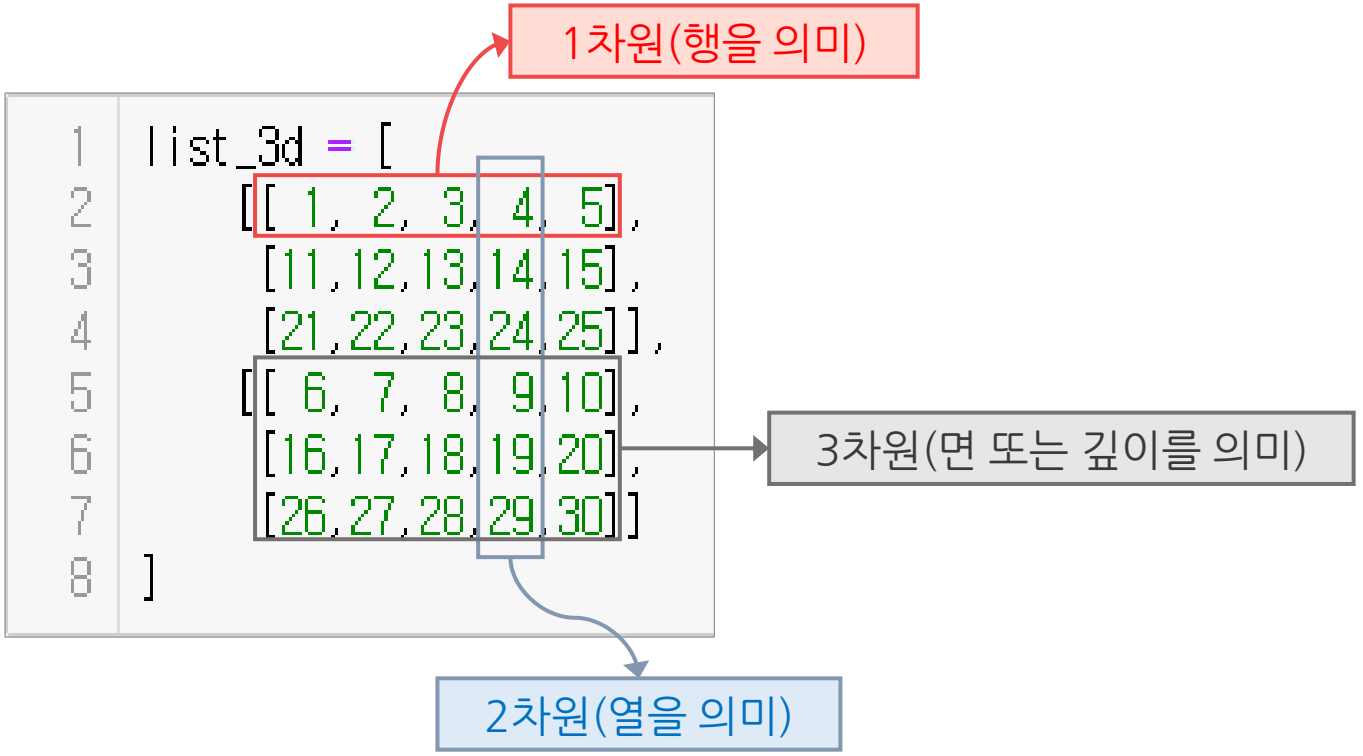
```
1  for row in list_2d:
2      for data in row:
3          print(data, end=' ')
4      print()
```

```
1 2 3 4 5
11 12 13 14 15
21 22 23 24 25
```

3.2. 3차원 리스트 인덱싱

3절. 중첩 루프

- 중첩 루프(Nested Loop)는 2차원 이상 구조 데이터를 다룰 때 사용
- 2차원 리스트에 대한 인덱싱은 '변수명[행][열]' 형식으로 지정
- 3차원 리스트에 대한 인덱싱은 '변수명[면][행][열]' 형식으로 지정




```
1  list_3d = [
2      [[ 1,  2,  3,  4,  5],
3       [11,12,13,14,15],
4       [21,22,23,24,25]],
5      [[ 6,  7,  8,  9,10],
6       [16,17,18,19,20],
7       [26,27,28,29,30]]
8  ]
```

3.2. 3차원 리스트 인덱싱

3절. 중첩 루프

```
list_3d = [
    [[ 1, 2, 3, 4, 5],
     [11,12,13,14,15],
     [21,22,23,24,25]],
    [[ 6, 7, 8, 9,10],
     [16,17,18,19,20],
     [26,27,28,29,30]]
]
```

```
print(list_3d)
```

```
[[[1, 2, 3, 4, 5], [11, 12, 13, 14, 15], [21, 22, 23, 24, 25]], [[6, 7, 8, 9, 10], [16, 17, 18, 19, 20], [26, 27, 28, 29, 30]]]
```

```
list_3d[0][2][3]
```

```
24
```

3차원 리스트에서의 인덱싱은 '변수명[면][행][열]' 형식으로 지정

```
list_3d[1][2][3]
```

```
29
```

```
for face in list_3d :
    for row in face :
        for data in row:
            print(data, end=' ')
```

```
1 2 3 4 5 11 12 13 14 15 21 22 23 24 25 6 7 8 9 10 16 17 18 19 20 26 27 28 29 30
```

3.3. 구구단 출력하기

3절. 중첩 루프

```
1  for i in range(2, 10):
2      for j in range(1, 10):
3          print(i*j, end=' ')
4      print()
```

```
2 4 6 8 10 12 14 16 18
3 6 9 12 15 18 21 24 27
4 8 12 16 20 24 28 32 36
5 10 15 20 25 30 35 40 45
6 12 18 24 30 36 42 48 54
7 14 21 28 35 42 49 56 63
8 16 24 32 40 48 56 64 72
9 18 27 36 45 54 63 72 81
```

위의 코드는 단이 가로방향으로 출력

```
1  for i in range(1, 10):
2      for j in range(2, 10):
3          print(i*j, end=' ')
4      print()
```

```
2 3 4 5 6 7 8 9
4 6 8 10 12 14 16 18
6 9 12 15 18 21 24 27
8 12 16 20 24 28 32 36
10 15 20 25 30 35 40 45
12 18 24 30 36 42 48 54
14 21 28 35 42 49 56 63
16 24 32 40 48 56 64 72
18 27 36 45 54 63 72 81
```

단이 세로방향으로 출력되게 하려면 반복문의 인덱스 실행 범위만 바꿔주면 됨

3.3. 구구단 출력하기

3절. 중첩 루프

```
1  for i in range(1, 10):
2      for j in range(2, 10):
3          print("{}x{}={:>2}".format(j, i, i*j), end='  ')
4      print()
```

```
2x1= 2  3x1= 3  4x1= 4  5x1= 5  6x1= 6  7x1= 7  8x1= 8  9x1= 9
2x2= 4  3x2= 6  4x2= 8  5x2=10  6x2=12  7x2=14  8x2=16  9x2=18
2x3= 6  3x3= 9  4x3=12  5x3=15  6x3=18  7x3=21  8x3=24  9x3=27
2x4= 8  3x4=12  4x4=16  5x4=20  6x4=24  7x4=28  8x4=32  9x4=36
2x5=10  3x5=15  4x5=20  5x5=25  6x5=30  7x5=35  8x5=40  9x5=45
2x6=12  3x6=18  4x6=24  5x6=30  6x6=36  7x6=42  8x6=48  9x6=54
2x7=14  3x7=21  4x7=28  5x7=35  6x7=42  7x7=49  8x7=56  9x7=63
2x8=16  3x8=24  4x8=32  5x8=40  6x8=48  7x8=56  8x8=64  9x8=72
2x9=18  3x9=27  4x9=36  5x9=45  6x9=54  7x9=63  8x9=72  9x9=81
```

3.4. 반복문 실행 상태 표시기

- print() 함수의 end 속성을 '₩r'로 출력 로그를 덮어 쓸 수 있음.

```
1  import time
2  progress = ''
3  percent = 0
4  num = 100
5  for i in range(1, num+1) :
6      percent = (i/num)*100
7      progress = int(percent*5/10) * '#'
8      time.sleep(1)
9      print("{:5.1f}%[{:<50s}] {}".format(percent, progress, i),
10           end='₩r')
```

```
37.0%[##################                                ] 37
```

break와 continue의 중첩루프 탈출

4절. 중첩 루프 탈출

- break 문을 사용하면 조건에 따라서 반복문을 종료
- continue 문을 사용하면 현재 반복문 블록을 실행시키지 않고 다음 반복으로 이동
- 파이썬에서 반복문의 실행을 제어하는 break와 continue는 가장 안쪽에 있는 루프에만 적용

```
1  for a in range(0, 3):
2      for b in range(1, 3):
3          if a==b:
4              break
5          print(a, b)
```

```
0 1
0 2
2 1
```

4.1. 플래그를 이용한 중첩루프 탈출

4절. 중첩 루프 탈출

- 프로그래머가 바깥 쪽을 둘러싸는 루프의 다음 반복으로 이동하거나 한 번에 여러 루프를 종료하려는 경우에는 레이블이 지정된 break를 모방하는 일반적인 방법으로 플래그 값을 지정

```
1  for a in range(0, 3):
2      break_out_flag = False
3      for b in range(1, 3):
4          if a==b:
5              break_out_flag = True
6              break
7          print(a, b)
8      if break_out_flag:
9          break;
```

```
0 1
0 2
```

4.2. 예외처리를 이용한 중첩루프 탈출

4절. 중첩 루프 탈출

- 바깥 반복문에서 안쪽 반복문을 실행시키기 위해 예외처리 코드를 작성

```
1  class BreakOutOfALoop(Exception): pass
```

```
1  for a in range(0, 3):
2      try:
3          for b in range(1, 3):
4              if a==b:
5                  raise BreakOutOfALoop
6              print(a, b)
7      except BreakOutOfALoop:
8          break
```

```
0 1
0 2
```

5. 연습문제

1. 양의 정수를 입력 받아 홀수인지 짝수인지를 판별하는 프로그램을 작성하세요. 양의 정수가 아니면 숫자를 다시 입력 받아야 합니다.
2. 1~50까지 자연수 중 3의 배수의 총합을 출력하는 프로그램을 작성하세요
3. 아래 패턴의 별을 출력하는 프로그램을 작성하세요.

a	b	c	d	e
*****	*	*****	*	*****
*****	**	****	**	****
*****	***	***	***	***
*****	****	**	****	**
*****	*****	*	*****	*

5. 연습문제

1. 다음 코드는 1부터 30까지(30포함) 자연수 중에서 3의 배수의 총 합을 출력하는 프로그램입니다. 빈칸의 코드를 완성하시오.

```
sum = 0
for i in _____________:
    if _______________:
            sum = sum + i
    else:
            pass
print(sum)
```

2. 다음 코드의 실행결과가 “1 3 5 7 9”가 되도록 빈칸을 완성하시오

```
num = 0
_________ num <= 10:
    if num%2 == 1:
            print(num, end=‘ ’)
    num += 1
```

5. 연습문제

3. 다음 2차원 리스트의 모든 값을 출력하는 코드입니다. 빈칸을 완성하시오.

```
list2d = [ [1,2,3],[4,5,6,7],[8,9]]
for row in list2d:
    ____________________:
            print(data, end=‘ ’)
    print()
```

결과 :

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

4. 다음 빈칸에 들어갈 함수 이름은?

```
colors = {"red":'apple', "yello":'banana'}
for i, v in ____________________(colors.values()):
    print(I, k)
```

```
0 apple
1 banana
```

5. 연습문제

5. 다음 코드의 실행결과는?

```
for i in range(0, 2):
    for j in range(0,2):
            if i==j:
                    break
            print(I, j)
```

6. 다음 코드의 실행 결과로 출력될 수 없는 것은?

```
for i in range(0, 2):
    for j in range(0,2):
             print(I, j)
            if i==j:
                    break
```

① 0 0　② 0 1　③ 1 0　④ 1 1

5. 연습문제

7. 다음 코드의 실행결과는?

```
L = [3, 4, 5, 6, 7, 8, 9, 10]
for i, data in enumerate(L):
    if (i%2 == 0):
            print(data, end=‘ ’)
```