파이썬 기초: 프로그램 흐름 제어

#리스트 #시퀀스 자료형 #조건문

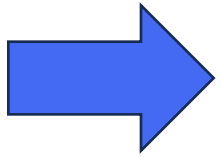
목차

리스트(List) 자료형

기초 자료형의 한계

기초 자료형의 경우 값을 단 한 개만 저장이 가능! 만약에 여러 개를 담아야 한다면?..

```
name1 = "홍길동"
name2 = "김철수"
name3 = "박영희"
...

name100 = "최이썬"
```



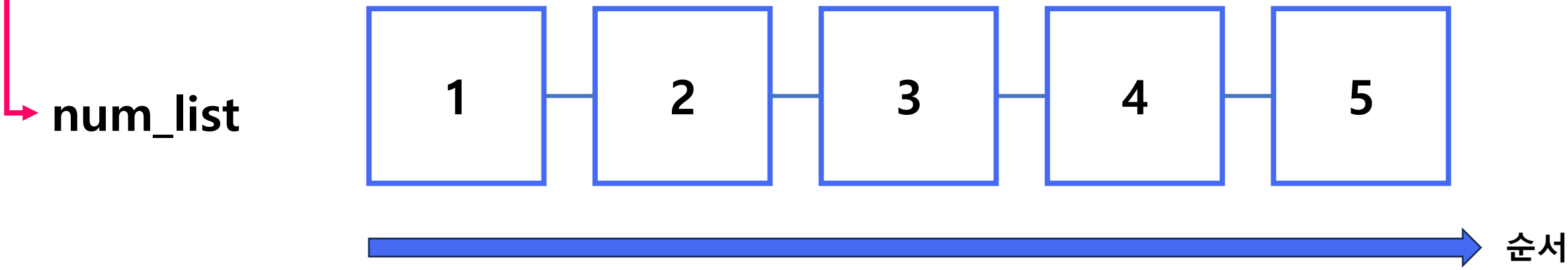
```
names = ["홍길동", "김철수", "박영희", ... , "최이썬"]
```

리스트(List) 자료형의 등장

여러 개의 값을 담을 수 있는 자료형

- **대괄호 쌍**으로 표현하며, 각각의 원소(element)마다 **,(콤마)**로 구분한다.
- **각각의 원소에는 순서가 존재한다.**

```
empty_list = [] # 빈 리스트
num_list = [1, 2, 3, 4, 5]
mix_list = ["a", 2, False, 27.15] # 서로 다른 자료형끼리 가능
```



시퀀스 자료형

시퀀스 자료형

순서가 있으면서, **여러 원소**(element)들을 담고 있는 자료형

물리적으로 별도의 존재하는 자료형은 아니며, 공통된 특징을 일컫는 자료형

자료형	예시
문자열(String)	“Hello World”
리스트(List)	[“a”, 1, “Python”]
튜플(Tuple)	(1, 2, 3, 4, 5)

시퀀스 자료형에는 여러 특징 및 기능들이 있다!

인덱스(Index)

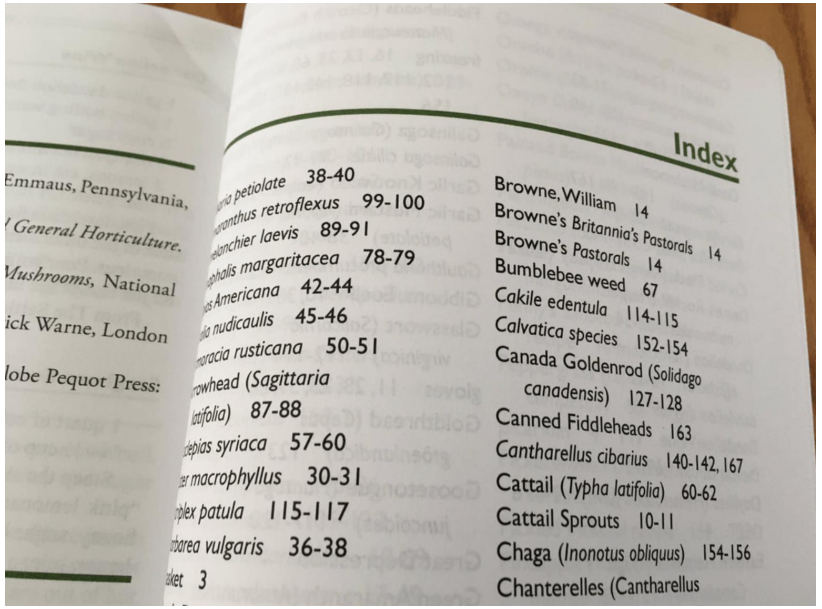
시퀀스 자료형의 각 원소의 위치(순서, 순번)를 인덱스(index)라고 표현한다.
첫번째 위치는 0부터 시작한다.

```
num_list = [1, 2, 3, 4, 5]
```

index → 0 1 2 3 4

```
str_var = "Hi! Python"
```

index → 0123456789



원소 접근하기 - (1) 인덱싱(Indexing): 값 읽기

인덱스(index)를 바탕으로 해당 위치에 접근하는 기법

문자열/리스트[인덱스_번호]

```
alpha = [1, 3, 5, 7, 9, 11]
print(alpha[2])
```

2번째 인덱스의 값을 가져온다

```
beta = "Hello Python!"
print(beta[4])
```

4번째 인덱스의 값을 가져온다

원소 접근하기 - (1) 인덱싱(Indexing): 값 수정

인덱스(index)를 바탕으로 해당 위치에 접근하는 기법

리스트[인덱스_번호] = 값

```
num_list = [1, 2, 3, 4, 5]

num_list[1] = 20
num_list[4] = 50

print(num_list)
```

1번째 인덱스에 값을 20으로 수정
4번째 인덱스에 값을 50으로 수정

문자열은 수정이 불가능! 문자열은 **Immutable** 타입이다

```
word = "pyThoN"

word[2] = 't'
word[5] = 'n'
print(word)
```

```
TypeError: 'str' object does not support item assignment
```

원소 접근하기 - (2) 슬라이싱(Slicing)

인덱스(index) 범위를 통해 연속된 범위의 일부분을 가져오는 기법

문자열/리스트[시작_인덱스 : 끝_인덱스]

끝 범위의 인덱스는 포함하지 않는다!

```
num_list = [1, 2, 3, 4, 5]

print(num_list[1:4])
```

인덱스 1부터 4미만의 원소 가져오기

```
str_var = "Hi! Python"

print(str_var[0:3])
```

인덱스 0부터 3미만의 원소 가져오기

*리스트의 경우 슬라이싱 결과는 리스트 형태

원소 접근하기 - (3) 인덱싱(Indexing)과 슬라이싱(Slicing) 확장

음수의 인덱싱, 시작 범위 또는 끝 범위를 비울 수 있다!

```
a = "Hello"
b = [3, 7, 10, 15, 20]

print(a[-1]) # 뒤에서 첫번째
print(b[ :3]) # 처음부터 3번째미만
print(b[1: ]) # 1번째부터 마지막까지
```

출력 결과

```
20
[3, 7, 10]
[7, 10, 15, 20]
```

시퀀스 자료형

Quiz! 아래의 출력 결과는 어떤 결과가 나올까요?

```
a = "Hello World"
b = [4, 5, 3, 2, 1]

print(a[-2:])
print(b[1:-3])
```

시퀀스 자료형 활용: in 연산자

시퀀스 자료형 안에 해당 원소가 포함되어 있는지 검사하는 연산자

원소 in 시퀀스 자료형

```
sentence = "Hello World"
num_list = [1, 3, 5, 7, 9]

print("E" in sentence)
print(3 in num_list)
```

출력 결과

```
False
True
```

시퀀스 자료형 활용: len() 함수

시퀀스 자료형의 길이(length)를 구하는 함수 (= 총 원소의 개수)

len(시퀀스 자료형)

```
num_list = [1, 2, 3, 4, 5]
word = "Python"

print(len(num_list))
print(len(word)
```

출력 결과

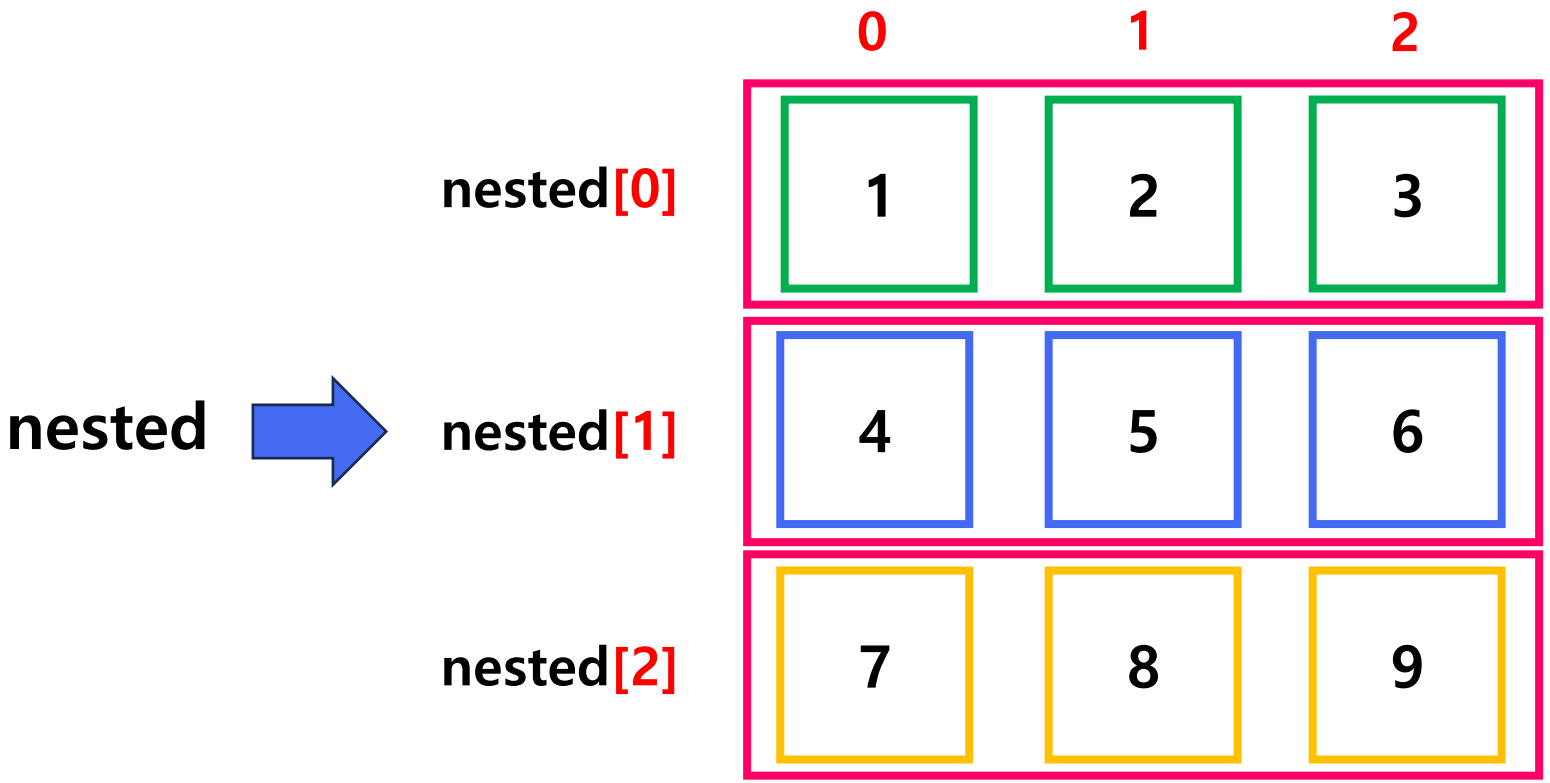
```
5
6
```

리스트 자료형의 다양한 기능 살펴보기

중첩 리스트(Nested List)

리스트 내의 원소가 리스트인 구조(= 2차원 리스트)

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



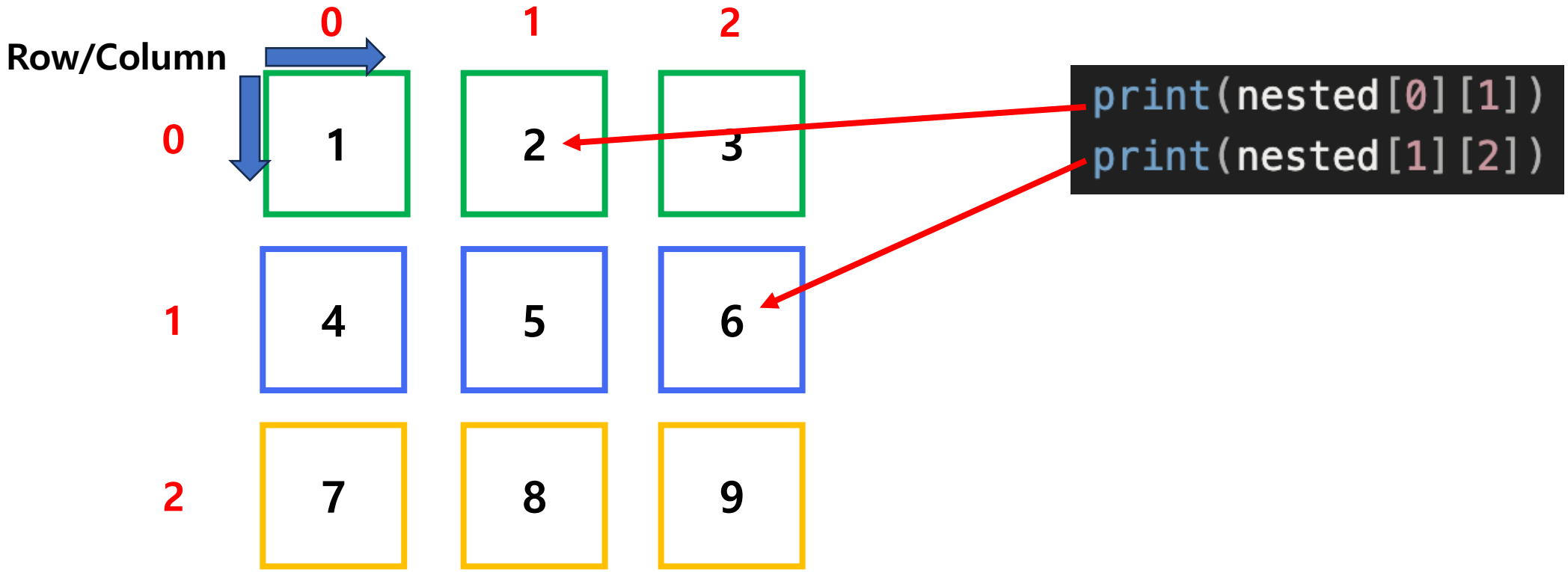

```
len(nested)
len(nested[0])
```

?

중첩 리스트(Nested List)

리스트 내의 원소가 리스트인 구조(= 2차원 리스트)

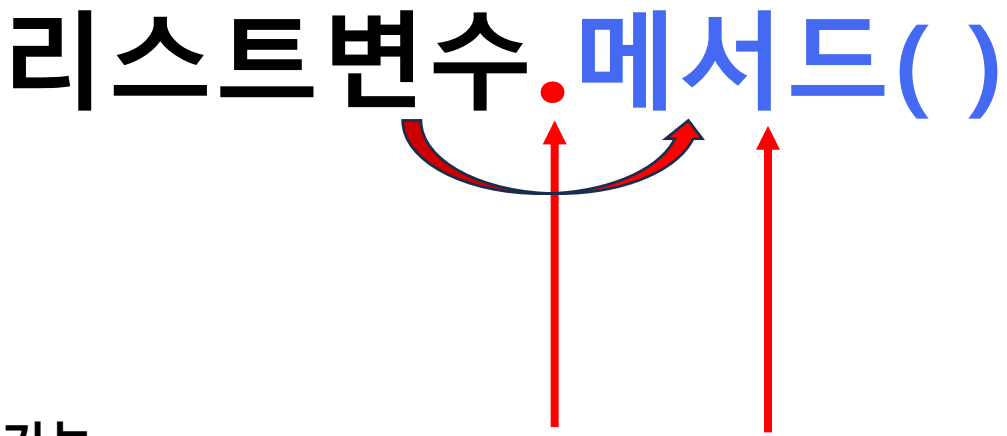
```
nested = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```



리스트의 다양한 메서드

리스트 자료형마다의 기능(메서드)를 살펴보자!

- .append(x)
- .insert(i, x)
- .remove(x)
- .pop(i)
- .sort()
- .index(x)
- .count(x)
- .clear()



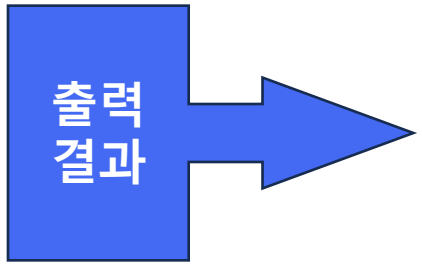
리스트 자료형에 존재하는 리스트 활용을 위한 탑재된 기능

리스트 자료형의 다양한 메서드

list.append(x) → None

현재 리스트에 원소 x를 가장 마지막에 추가

```
a = []
b = ['p', 'y', 't']
a.append(20)
a.append(10)

b.append('h')
b.append('o')
b.append('n')

print(a)
print(b)
```



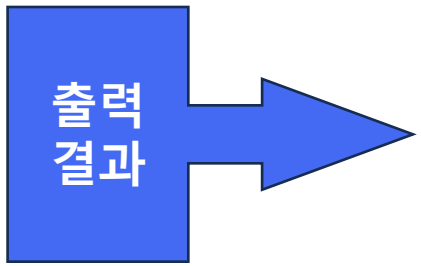

```
[20, 10]
['p', 'y', 't', 'h', 'o', 'n']
```

리스트 자료형의 다양한 메서드

list.insert(i, x) → None

현재 리스트에 i번째 인덱스에 원소 x를 추가(삽입)

```
a = [1, 2, 4, 5]
a.insert(2, 3)
print(a)
```




```
[1, 2, 3, 4, 5]
```

2번째 인덱스에 3 삽입

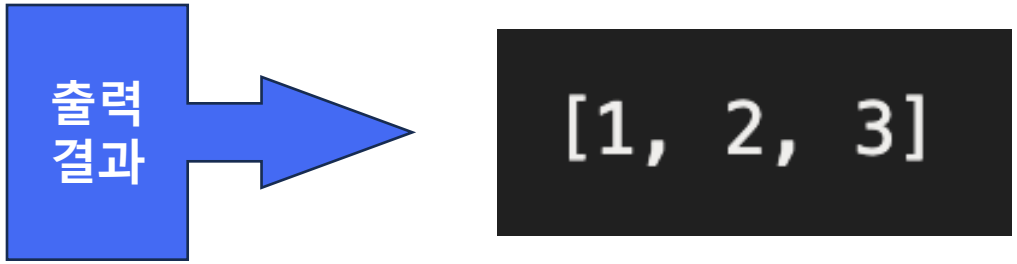
리스트 자료형의 다양한 메서드

list.remove(x) → None

현재 리스트에서 처음 검색된 원소 x를 삭제

- *앞에서부터 일치하는 값 검색 후, 처음 검색된 값을 삭제*

```
num_list = [3, 1, 2, 3]

num_list.remove(3)
print(num_list)
```

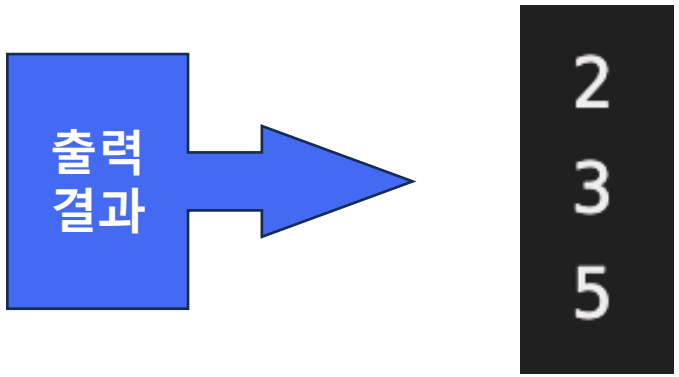


리스트 자료형의 다양한 메서드

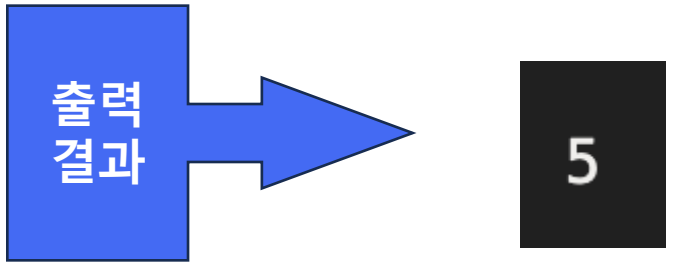
list.pop(i) → object

현재 리스트에서 i번째 인덱스의 원소를 삭제 후, 반환(= 원소 빼내기)

```
num_list = [1, 2, 3, 4, 5]

print(num_list.pop(1))
print(num_list.pop(1))
print(num_list.pop(2))
```




```
num_list = [1, 2, 3, 4, 5]

print(num_list.pop())
```



인자에 아무 것도 넣지 않으면 Default로 마지막 원소를 대상으로 수행

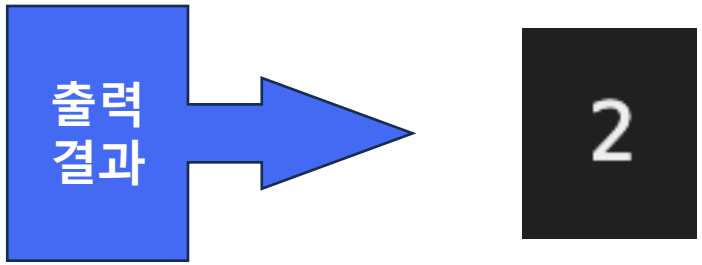
리스트 자료형의 다양한 메서드

list.index(x) → int

현재 리스트에서 원소 x에 해당하는 인덱스 검색

- *앞에서부터 일치하는 값 검색 후, 처음 검색된 위치(인덱스)를 반환*

```
num_list = [3, 1, 4, 1, 2]

print(num_list.index(4))
```



리스트 자료형의 다양한 메서드

※주의할 점! - remove(), pop(), index()

검색/삭제 대상의 원소나 인덱스가 존재하지 않거나 범위를 초과하면 에러 발생!

```
num_list = [1, 2, 3, 4, 5]

print(num_list.pop(7))
print(num_list.remove(10))
print(num_list.index(0))
```

출력 결과

```
IndexError: pop index out of range

ValueError: list.remove(x): x not in list

ValueError: 0 is not in list
```

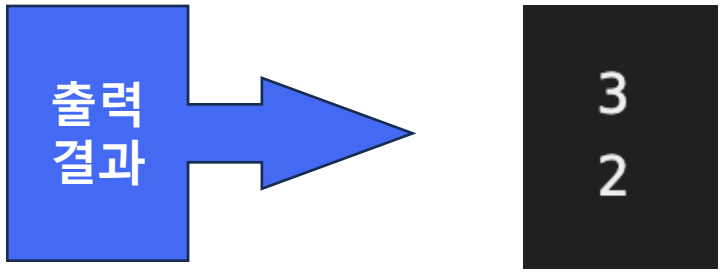
해당 값이 포함되어 있는지 먼저 확인하는 것이 안전한 방법!

리스트 자료형의 다양한 메서드

list.count(x) → int

현재 리스트에서 원소 x의 개수를 반환

```
num_list = [3, 1, 4, 1, 2, 1, 3]

print(num_list.count(1))
print(num_list.count(3))
```



리스트 자료형의 다양한 메서드

list.sort() → None

현재 리스트의 원소들을 내부적으로 정렬

- Default는 오름차순 정렬. reverse 매개변수에 True 전달 시, 내림차순 정렬

```
num_list = [4, 2, 3, 1, 5]

num_list.sort()
print(num_list)

num_list.sort(reverse=True)
print(num_list)
```

출력 결과

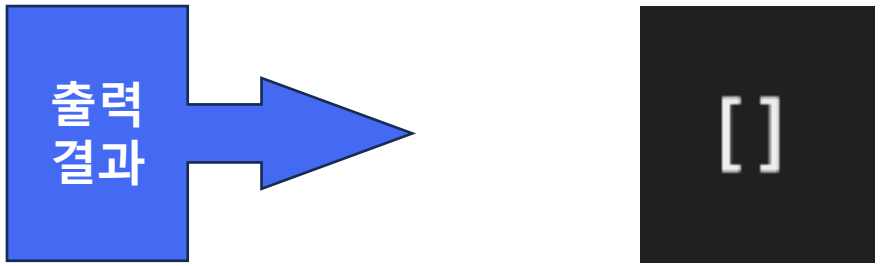
```
[1, 2, 3, 4, 5]
[5, 4, 3, 2, 1]
```

리스트 자료형의 다양한 메서드

list.clear() → None

현재 리스트의 모든 원소들을 내부적으로 제거하는 기능(= 빈 리스트로 만드는 기능)

```
a = [5, 4, 2, 1, 6]
a.clear()

print(a)
```

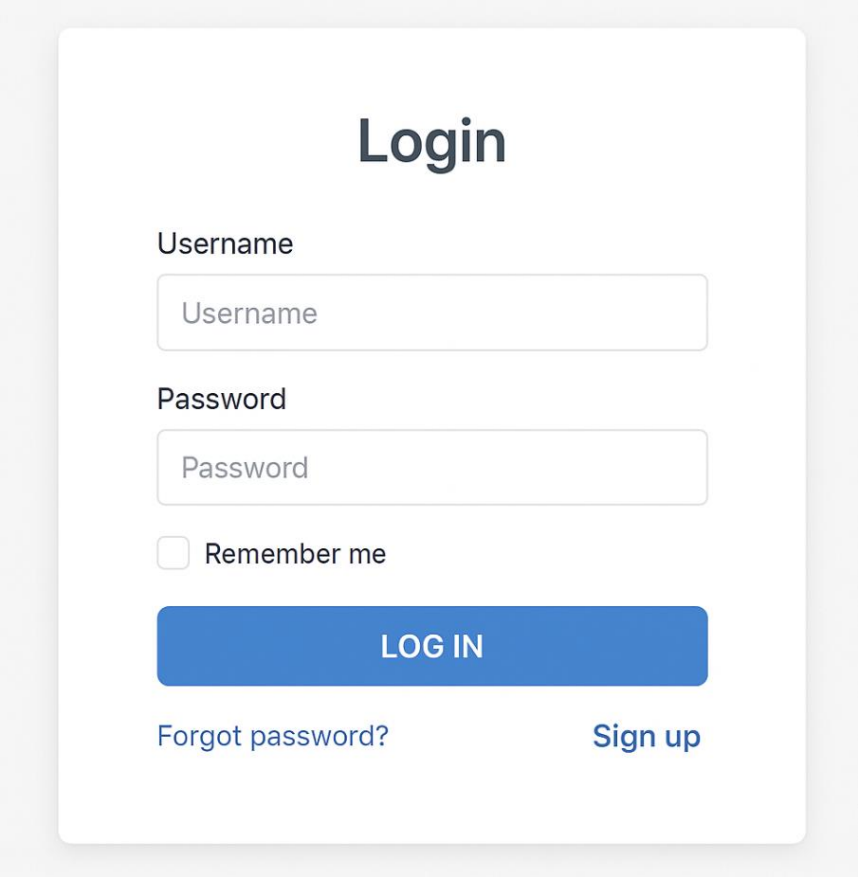


조건문

조건문

지금까지 순서대로 명령을 모두 실행했었다. 그렇지 않고 특정 조건에 따라서 명령을 수행하려면?

우리의 삶은 조건에 따라서 해야 할 일이 다른 경우가 비일비재



메인 페이지 이동

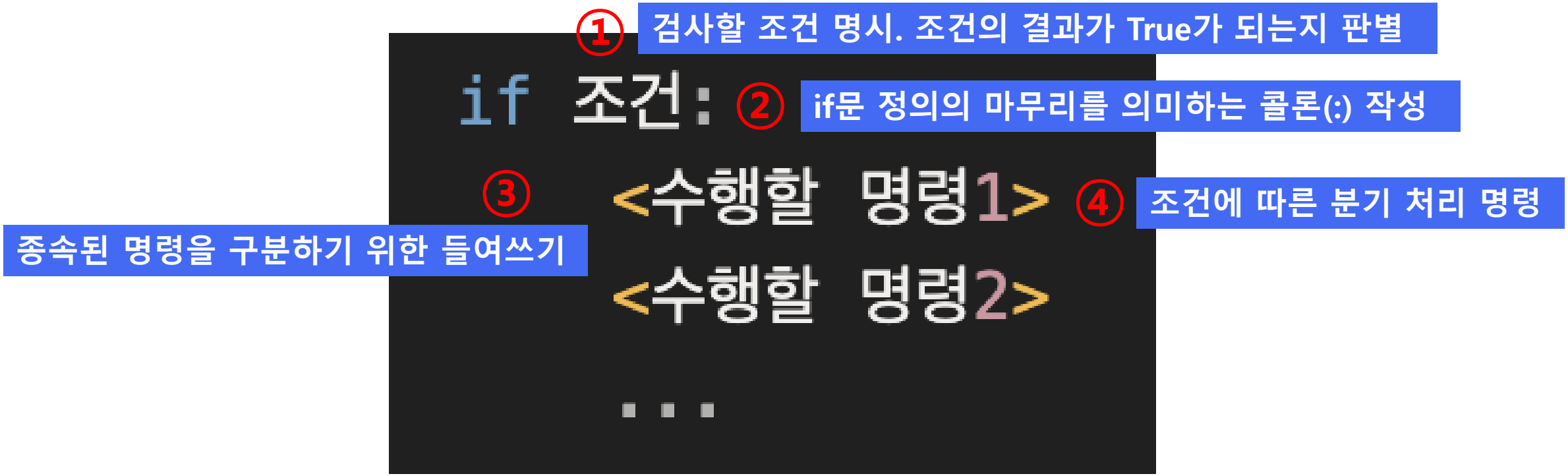
회원가입 정보와 입력한 로그인 정보와 **일치하는 경우**

경고 메시지 출력

회원가입 정보와 입력한 로그인 정보와 **다른 경우**

if 문

만약에 특정 조건이 참(True)이라면 명령을 실행



만약에 조건이 참이면, '수행할 명령'들을 수행해라

if 문

만약에 특정 조건이 참(True)이라면 명령을 실행

비교연산, 논리연산 등의 조건식을 바탕으로 결과에 따라서 실행

```
var1 = 1

if var1 > 0:
    print(var1, "은 양수입니다.")

print("조건문 종료")
```

```
1 은 양수입니다.
조건문 종료
```

```
var1 = -1

if var1 > 0:
    print(var1, "은 양수입니다.")

print("조건문 종료")
```

```
조건문 종료
```

들여쓰기(Indentation)

파이썬에서는 공백도 일종의 문법이다

if/else/for/while 등등 블럭의 들여쓰기는
공백 4칸 또는 탭 1칸을 의미한다!

```
num1 = 10
num2 = 20

if num1 + num2 > 30:
    print(num1, num2)
```

*들여쓰기된 명령이 해당 블럭에 ***종속***된 명령

```
IndentationError: unexpected indent
```

```
var1 = 1

if var1 > 0:
print(var1, "은 양수입니다.")
```

```
var1 = 1

if var1 > 0:
  print(var1, "은 양수입니다.")
```

```
num1 = 10
num2 = 20

if num1 + num2 > 30:
        print(num1, num2)
```

Example

```
radius = int(input("반지름을 입력하세요: "))

if radius > 0:
    print('넓이 = ', (3.14 * radius ** 2))
    print('둘레 = ', (2 * 3.14 * radius))
```

```
num = int(input('정수를 입력하세요: '))

if num % 2 == 0:
    print(num, '은 짝수입니다.')
```

```
반지름을 입력하세요: 3

넓이 =  28.26
둘레 =  18.84
```

```
정수를 입력하세요: 8

8 은 짝수입니다.
```

if-else 문

만약에 조건이 맞지 않을 때는? else 구문!

만약에 조건이 참이면 A 실행, 그렇지 않으면 B 실행

if문에 종속된 명령

else문에 종속된 명령

```
if 조건:
    print('A')
else:
    print('B')
```

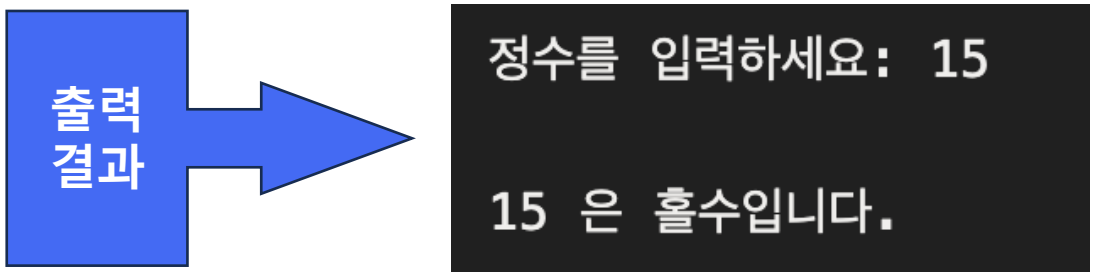
```
if 회원가입 정보와 로그인 정보가 일치:
    메인 페이지 이동
else:
    경고 메시지 출력
```

if-else 문

Example

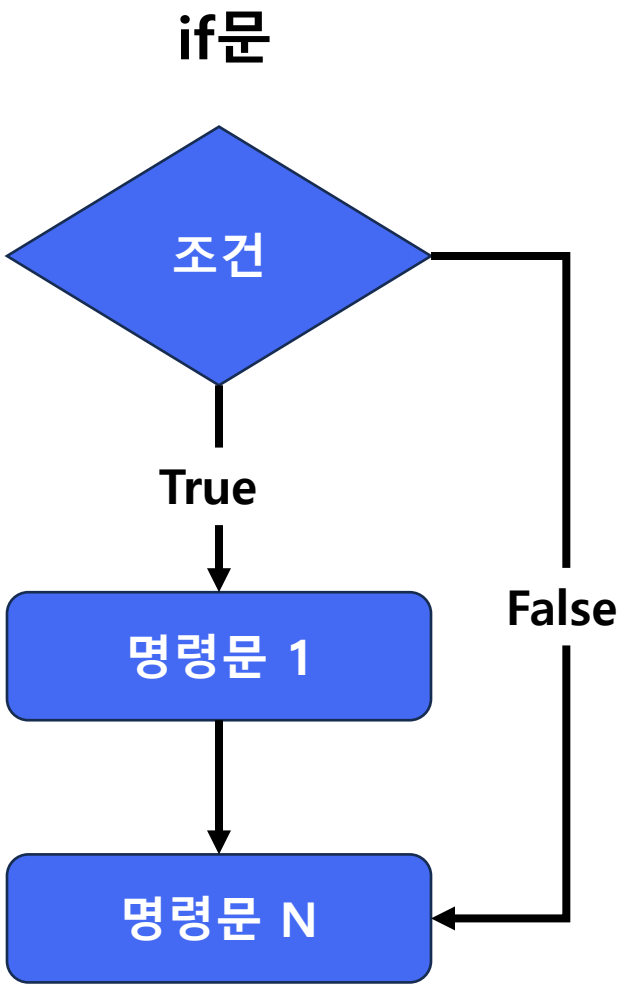
if-else 문: 만약에 조건이 참이면 명령1, 그렇지 않으면 명령2 실행

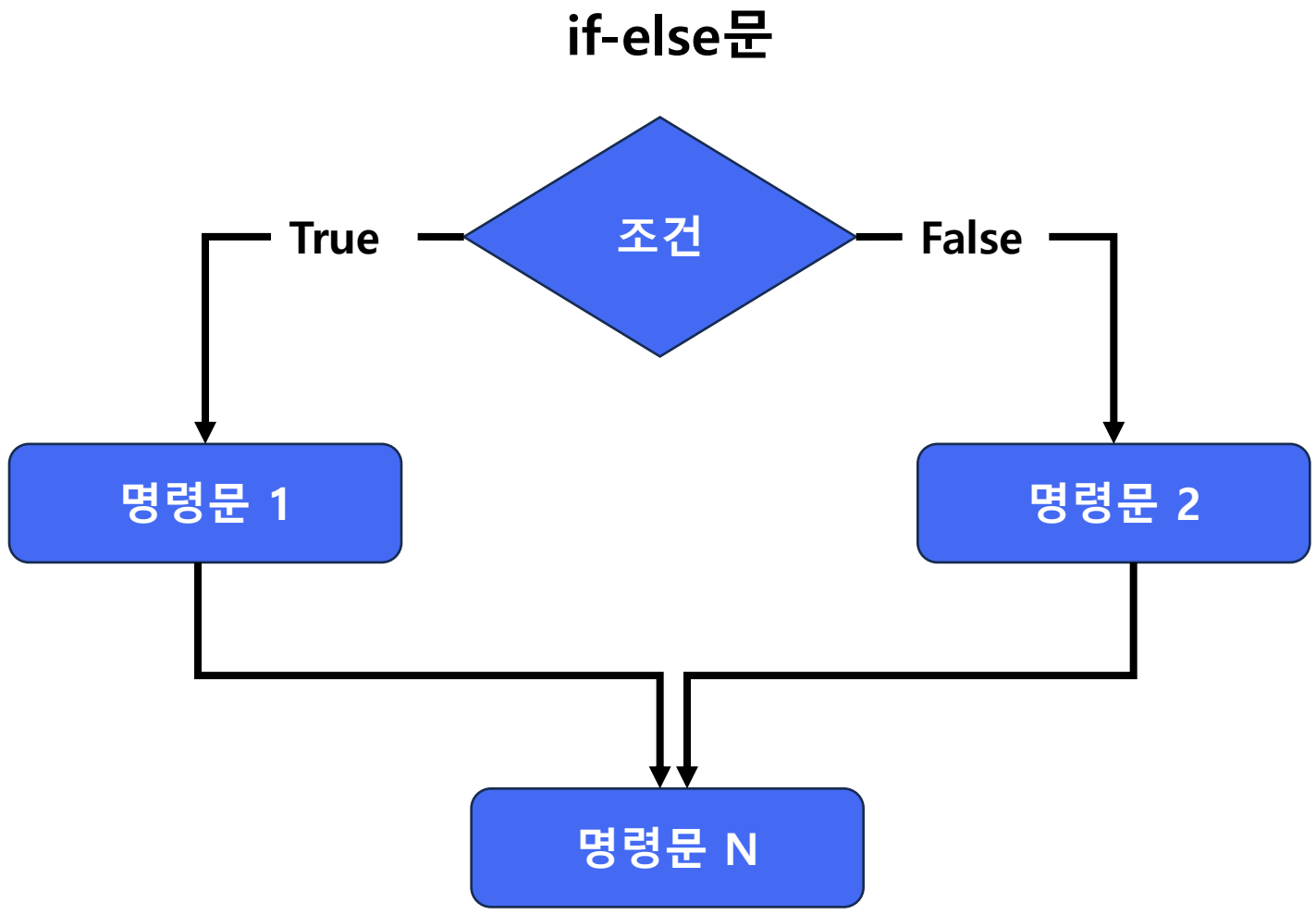
```
num = int(input('정수를 입력하세요: '))

if num % 2 == 0:
    print(num, '은 짝수입니다.')
else:
    print(num, '은 홀수입니다.')
```



*else의 조건은 정확히 if 조건의 정반대의 의미가 된다.

if문과 if-else문





if-elif-else 문

if-else 구문은 하나의 조건, 그리고 그렇지 않은 경우로 이분법적이다

그렇지 않고, 그 이상의 여러 개의 조건을 걸고 싶을 때는? → **if-elif-else 문!**

```
if 조건1:
    <if문 종속 명령>
elif 조건2:
    <elif 종속 명령>
else:
    <else 종속 명령>
```

- else if의 약자 (*"그렇지 않고, 만약 ~라면"* 의 의미)
- 상위의 조건들이 **False**인 경우, 또 다른 조건을 거는 것
- elif 문은 여러 번 사용이 가능하다.

if-elif-else 문

if-else 구문은 하나의 조건, 그리고 그렇지 않은 경우로 이분법적이다

그렇지 않고, 그 이상의 여러 개의 조건을 걸고 싶을 때는? → **if-elif-else 문!**

```
if 조건1:
    <if문 종속 명령>
elif 조건2:
    <elif 종속 명령>
else:
    <else 종속 명령>
```

만약, <조건1>이 참이라면, <명령1>을 실행하고,

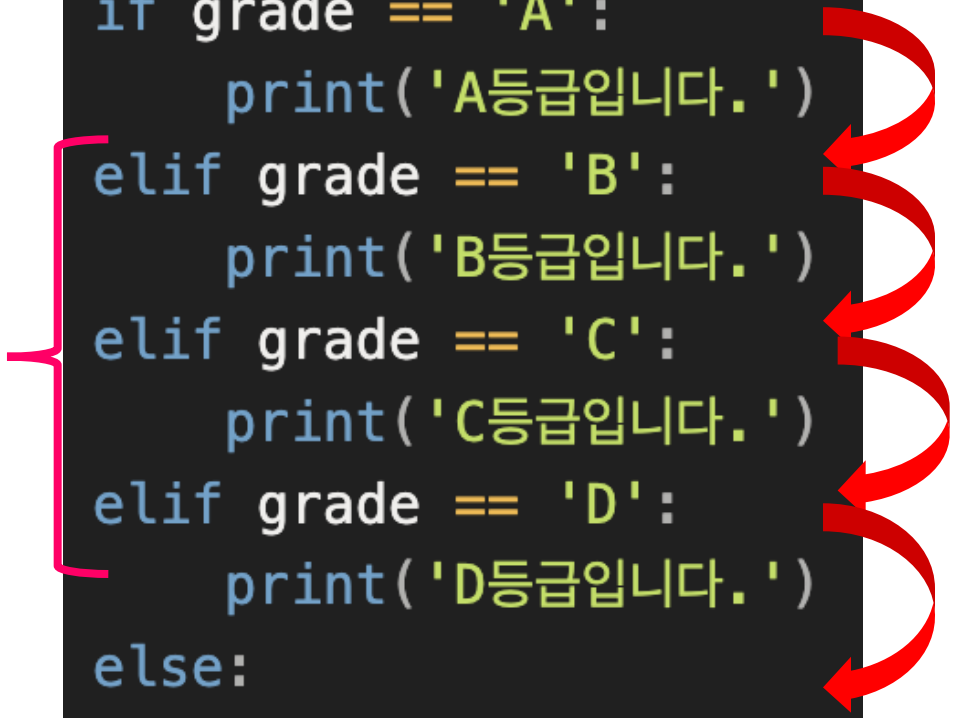
그렇지 않고 만약, <조건2>가 참이라면, <명령2>를 실행하고,

그 이외의 경우에는 <명령3>을 실행해라

if-elif-else 문

그렇지 않고, 그 이상의 여러 개의 조건을 걸고 싶을 때는?

```
if grade == 'A':
    print('A등급입니다.')
elif grade == 'B':
    print('B등급입니다.')
elif grade == 'C':
    print('C등급입니다.')
elif grade == 'D':
    print('D등급입니다.')
else:
    print('F등급입니다.')
```



if부터 차례대로 조건을 하나씩 검사
조건들 중 하나만 참이어도, 해당 명령 실행 후, if-elif-else 종료

```
grade = 'A'

grade = 'C'

grade = 'F'
```

각각 값이 다를 때 어떠한 결과?

중첩 if문(Nested if)

두 숫자가 양수인지 음수인지 출력?

조건을 세부적으로 걸고 싶다면?

```
num1 = 10
num2 = -10

if num1 > 0:
    if num2 > 0:
        print('둘다 양수입니다.')
    elif num2 < 0:
        print('num1은 양수, num2은 음수입니다.')
elif num1 < 0:
    if num2 > 0:
        print('num1은 음수, num2은 양수입니다.')
    elif num2 < 0:
        print('둘다 음수입니다.')
```

여러 조건이 동시에 참이어야 내부 실행

비교 연산자 체이닝(Chained Comparison)

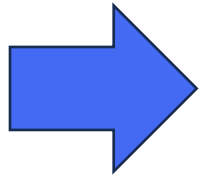
비교 연산자를 한 줄에 이어서 쓸 수 있는 기능 (조건 연결하기)

① num이 10이상이면서, 90이하

```
if num >= 10 and num <= 90:
```

② num은 10부터 90이하

```
if 10 <= num <= 90:
```



동일한 의미

다른 일반 프로그래밍 언어에서는 ①과 같은 방법을 사용해야 하는데, Python의 편리한 문법!

조건문의 다양한 예시

Example 1

```
a = ["H", "h", "e", "l", "l", "o"]

if "H" in a:
    a.remove("H")
else:
    print("H가 없습니다.")
```

```
word = "Python"

if len(word) >= 5:
    print(word * 2)
else:
    print(word)
```

조건문의 다양한 예시

Example 2

```
word = input('단어를 입력하세요: ')

if len(word) >= 5 and 'P' in word:
    print(word)
else:
    print('조건에 맞지 않는 단어입니다.')
```

```
score = int(input("시험 점수를 입력하세요: "))
attendance = int(input("출석 점수를 입력하세요: "))

if score >= 70 and attendance >= 90:
    print("통과")
elif score >= 70 and attendance < 90:
    print("조건부 통과(출석 미달)")
else:
    print("미흡")
```

감사합니다!