

**PCCP**

압축 해방 속성 강의

# 강사 소개

- 유태영
- 멋쟁이 사자처럼 (**likelion**)
  - 5기 ~ 6기(2017 ~ 2018) 전국 대학생 대상 강의 전담 진행
- SSAFY(**삼성 청년 SW 아카데미**)
  - 1기 ~ 7기(2019 ~ 2022) 전담 강사 - python, web, django, algorithms, js, vue
  - COVID 비대면 교육 이후 전국 수강생 대상 Youtube 라이브 진행
- 기타 장기강의/단기특강 다수

# PCCP

- Programmers Certified Coding Professional
- 코딩 전문 역량 인증

## 기본 프로그램 구현

제시된 조건과 요구 사항을 만족하는 프로그램을 개발 하는 능력을 평가합니다.

## 초급 자료구조/알고리즘 활용

문자열(String), 배열(Array), 그리디(Greedy), 정렬(Sort) 등을 활용한 프로그램 개발하는 능력을 평가합니다.

## 중급 자료구조/알고리즘 활용

스택(Stack), 큐(Queue), 덱(Deque), 해시(Hash), 이진 탐색(Binary Search), 깊이 우선 탐색(DFS), 너비 우선 탐색(BFS) 등을 활용한 프로그램 개발하는 능력을 평가합니다.

## 고급 자료구조/알고리즘 활용

그래프(Graph), 트리(Tree), 힙(Heap), 다이나믹 프로그래밍(Dynamic Programming) 등을 활용한 프로그램 개발하는 능력을 평가합니다.

## 정확하고 효율적인 프로그램 작성

빠른 판단력으로 주어진 문제에 알맞은 자료구조와 알고리즘을 선택하여 오류없이 프로그램을 개발하는 능력을 평가합니다.

# 만약 시간이 많다면

실시간 비대면	8	Python 기초 프로그래밍 1 (변수와 식별자)
실시간 비대면	8	Python 기초 프로그래밍 2 (제어 - 조건)
실시간 비대면	8	Python 기초 프로그래밍 3 (제어 - 반복)
집합 오프라인	8	Python 기초 프로그래밍 (문제풀이 1)
집합 오프라인	8	Python 기초 프로그래밍 (문제풀이 2)
실시간 비대면	8	Python 기초 프로그래밍 4-1 (함수)
실시간 비대면	8	Python 기초 프로그래밍 4-2 (함수)
실시간 비대면	8	Python 기초 프로그래밍 5 (데이터 구조의 활용과 메서드)
집합 오프라인	8	Python 기초 프로그래밍 6 (객체지향프로그래밍)
집합 오프라인	8	Python 기초 프로그래밍 (문제풀이 3)

+ 28 \* 8 시간

↑  
80 시간

우리는?

20 시간

# 이번 20시간의 목표

최고 효율  
최단 시간  
으로 짹먹

+

자생하고  
지속 가능  
장기 계획

# 데이터 (값)

values

# 데이터 타입

(value types)

- 숫자
  - 정수 (integer): -2, -1, 0, 1, 2, 123, 1000, 0b11, 0o11, 0x11
  - 실수(float): 1.7, 3.14, -100.0, 5e10,
  - 허수, 무한대 등..
- 문자 (string)
  - ‘python’, ‘안녕’, ‘123’, ‘(ง\:\_)ง’, ‘’
  - 주의! 123 과 ‘123’은 컴퓨터에서 전혀 다르다!! (숫자 vs 문자)

# 데이터 타입

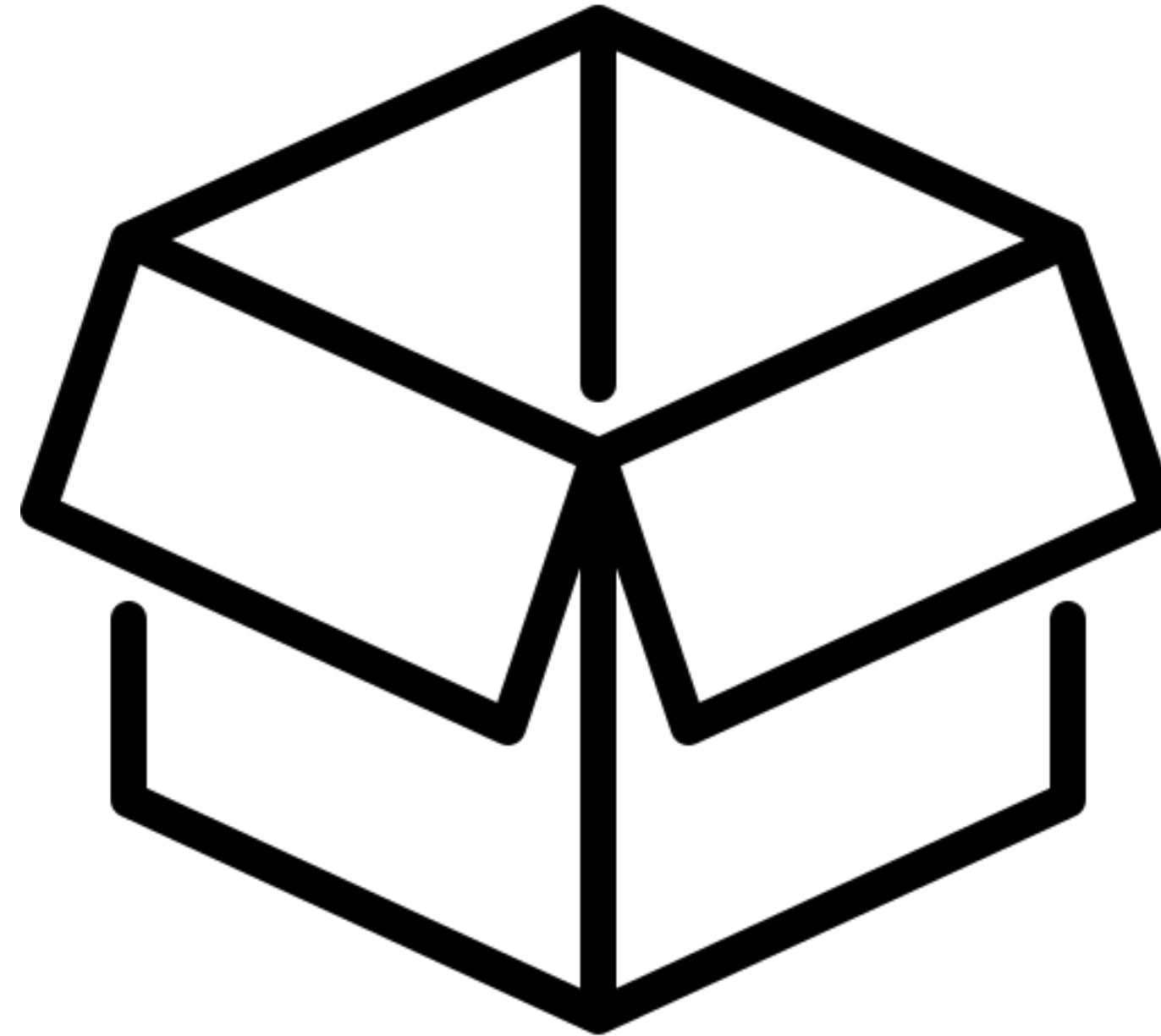
(value types)

- 참/거짓 (boolean)
  - True
  - False
- 없음 (None)
  - None

변수

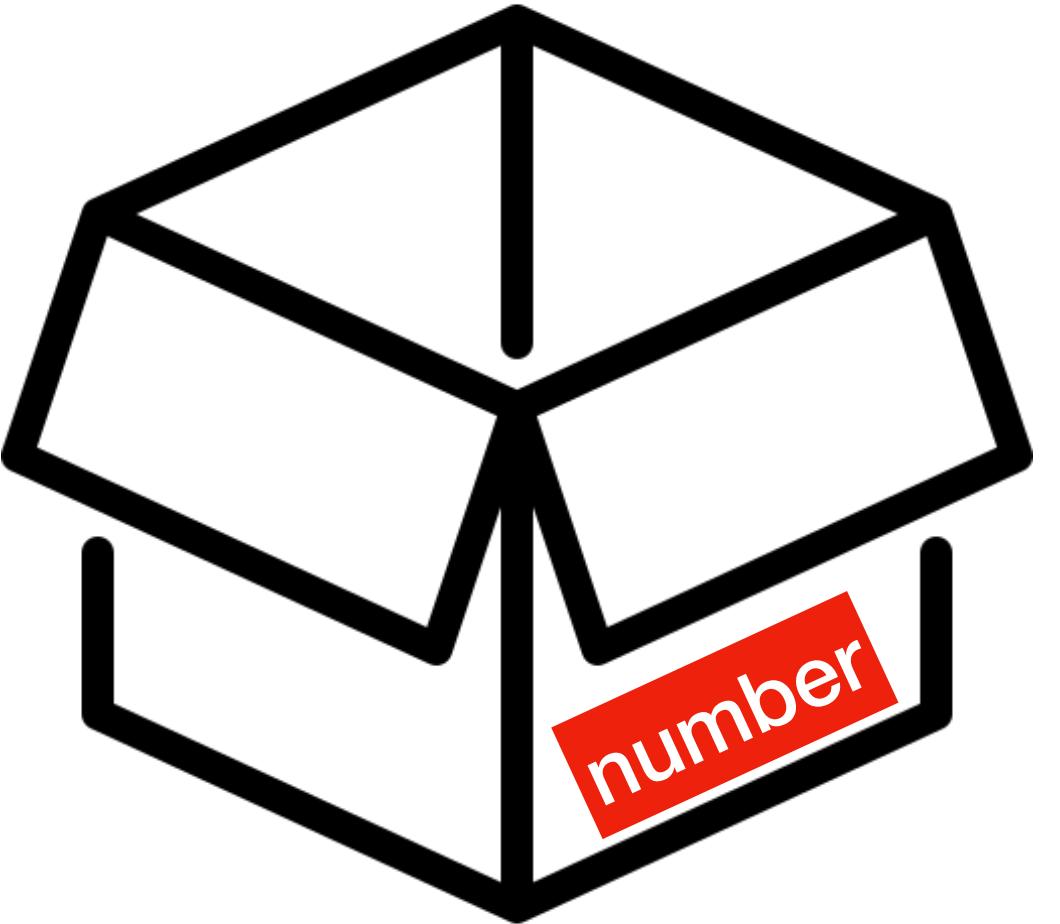
variables

# 변수(Variable)



A.K.A 박스

# 변수



박스

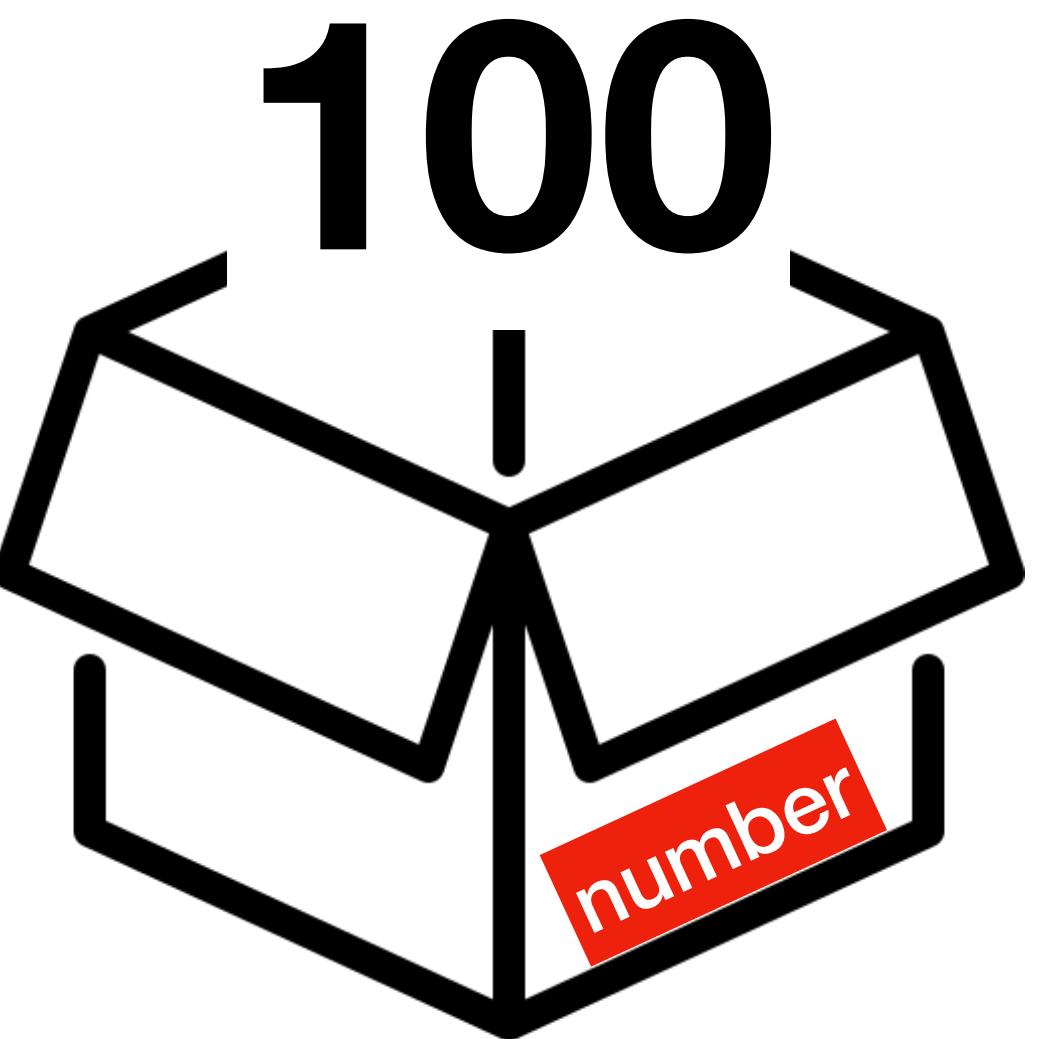
100

값

# 할당 (assignment)



# 할당



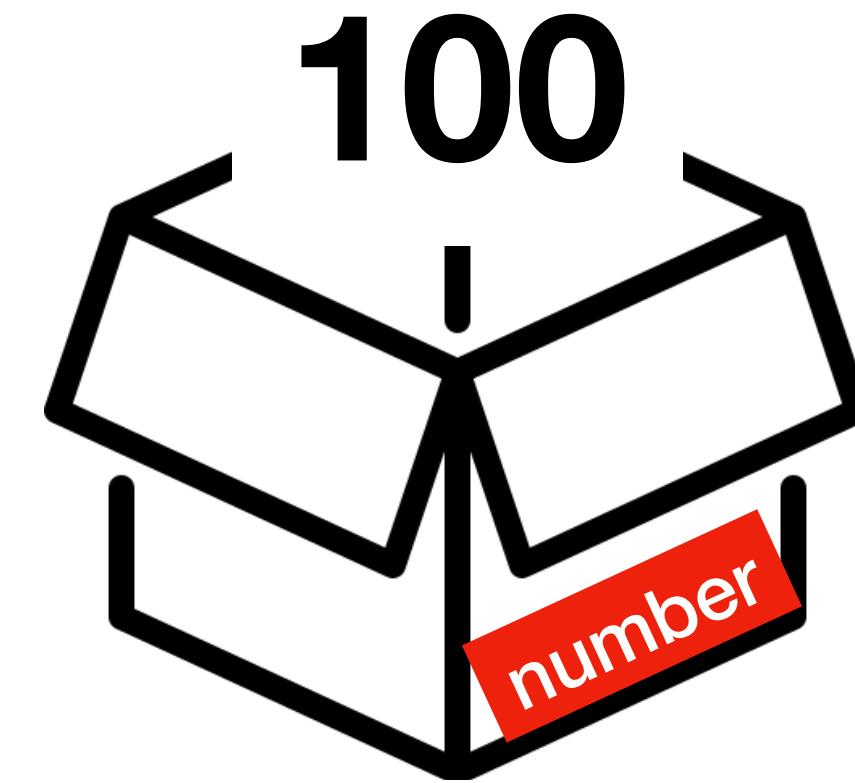
number = 100



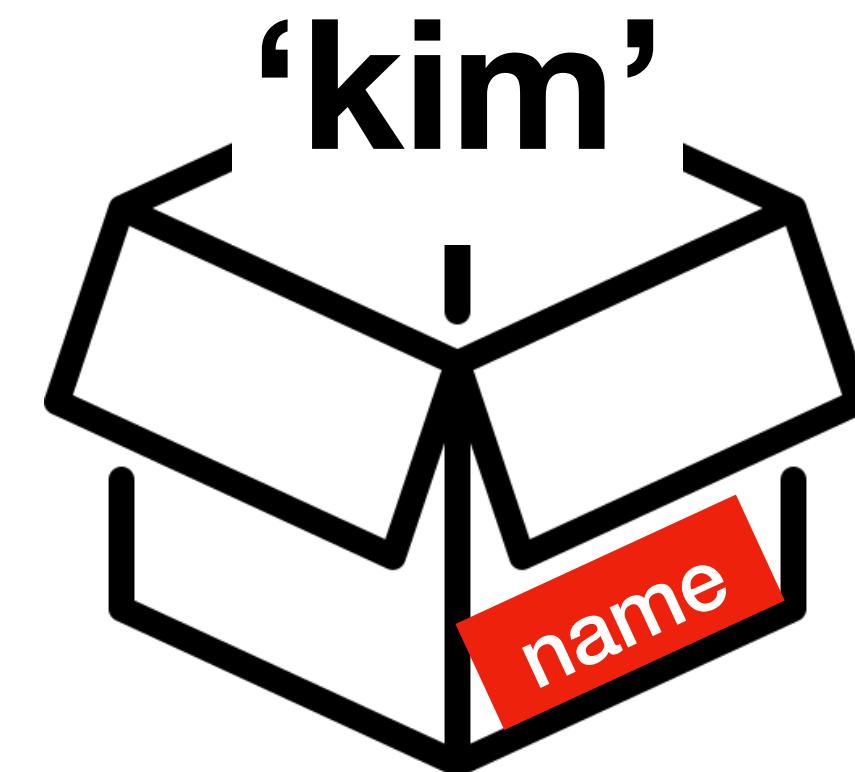
좌와 우가 같다는 뜻이 아님!  
우를 좌에 할당(저장)한다!!

할당

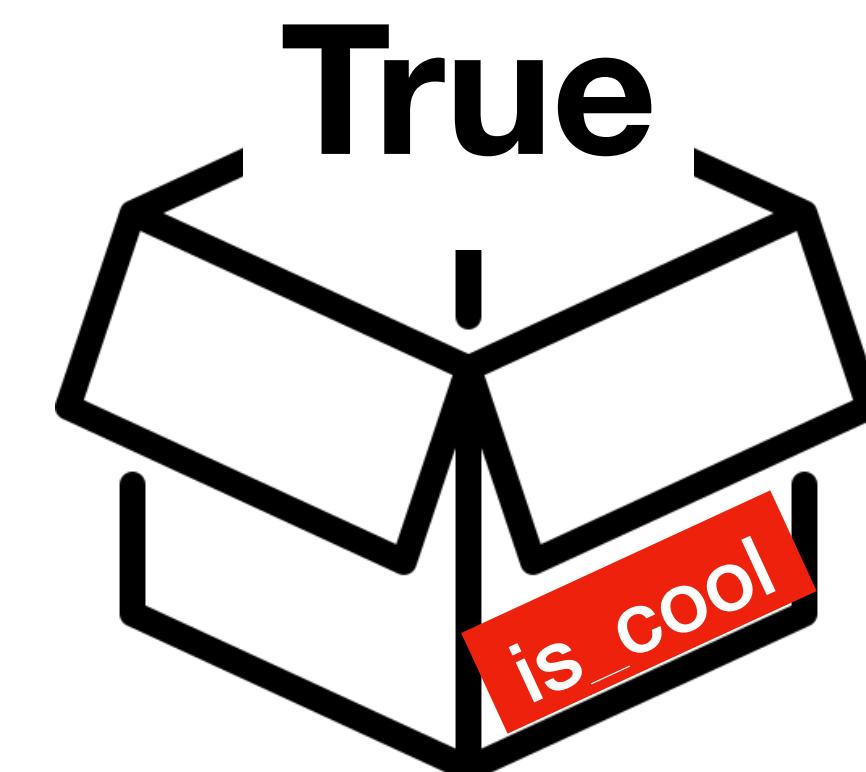
number = 100



name = 'kim'



is\_cool = True

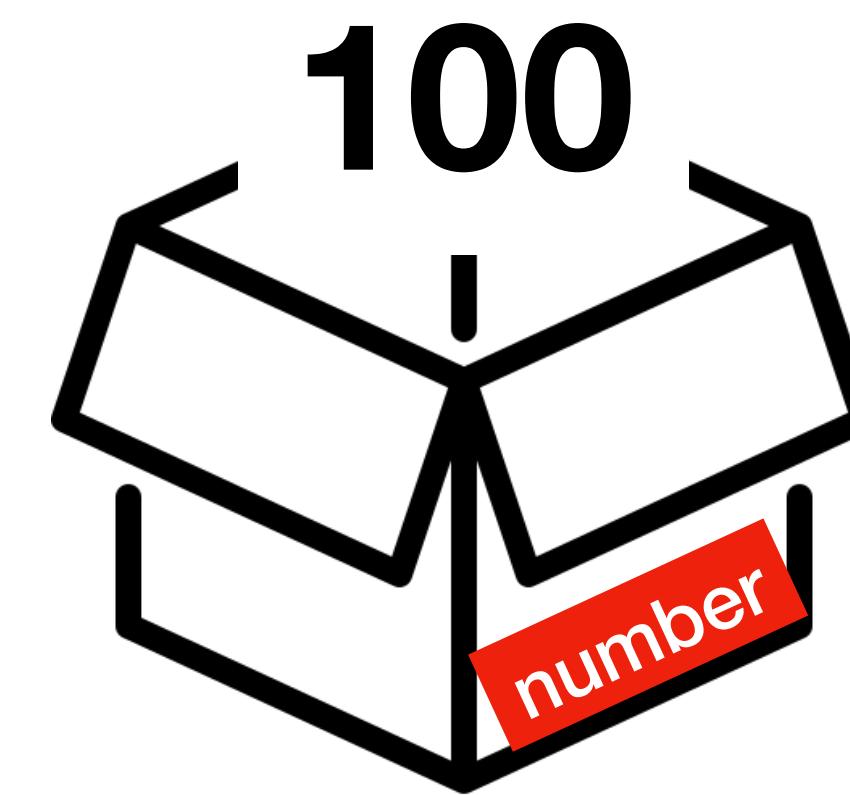


# 재할당

number = 1

number = 10

number = 100



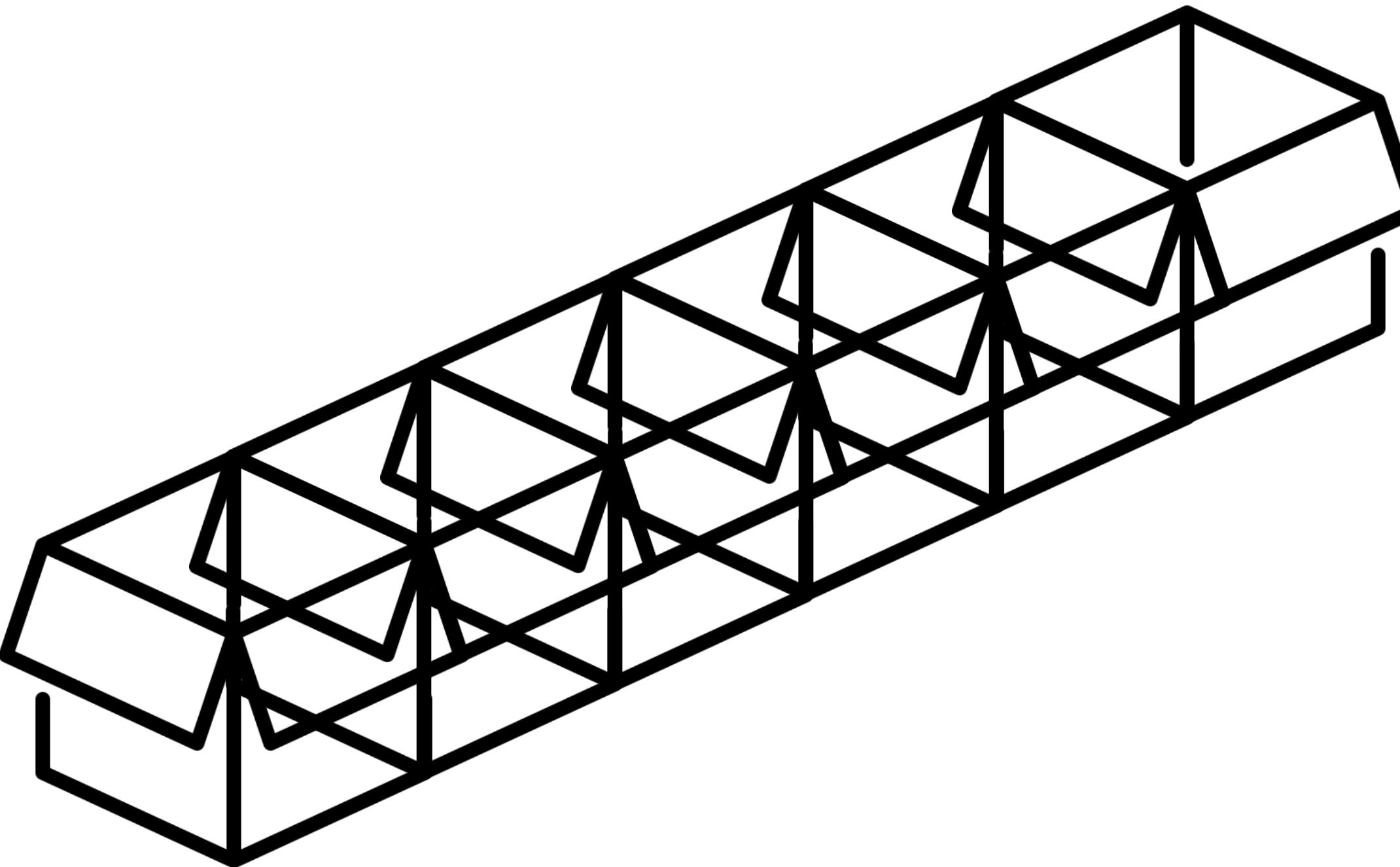
# 변수명 짓는 규칙

- 문자(a ~ z, \_)로 시작(숫자로 시작 불가) / 첫 글자만 제외하면 숫자 사용 가능
- 사전 예약어 사용 불가(if, else, for, while, True, False, class, def 등)
- 대/소문자 구분 (Num 과 num은 다른 변수명. 클래스 이름과 상수 제외 모두 소문자로 구성)
- 띄어쓰기가 필요한 경우 \_ 로 구분 (snake\_case)
- 명확하고 의미 있는 단어로 지을 것!
  - Good ) numbers, maximum, num, count, is\_odd
  - Bad ) a, b, c, x, y, z, x1, x2, aa, bbb

**리스트**

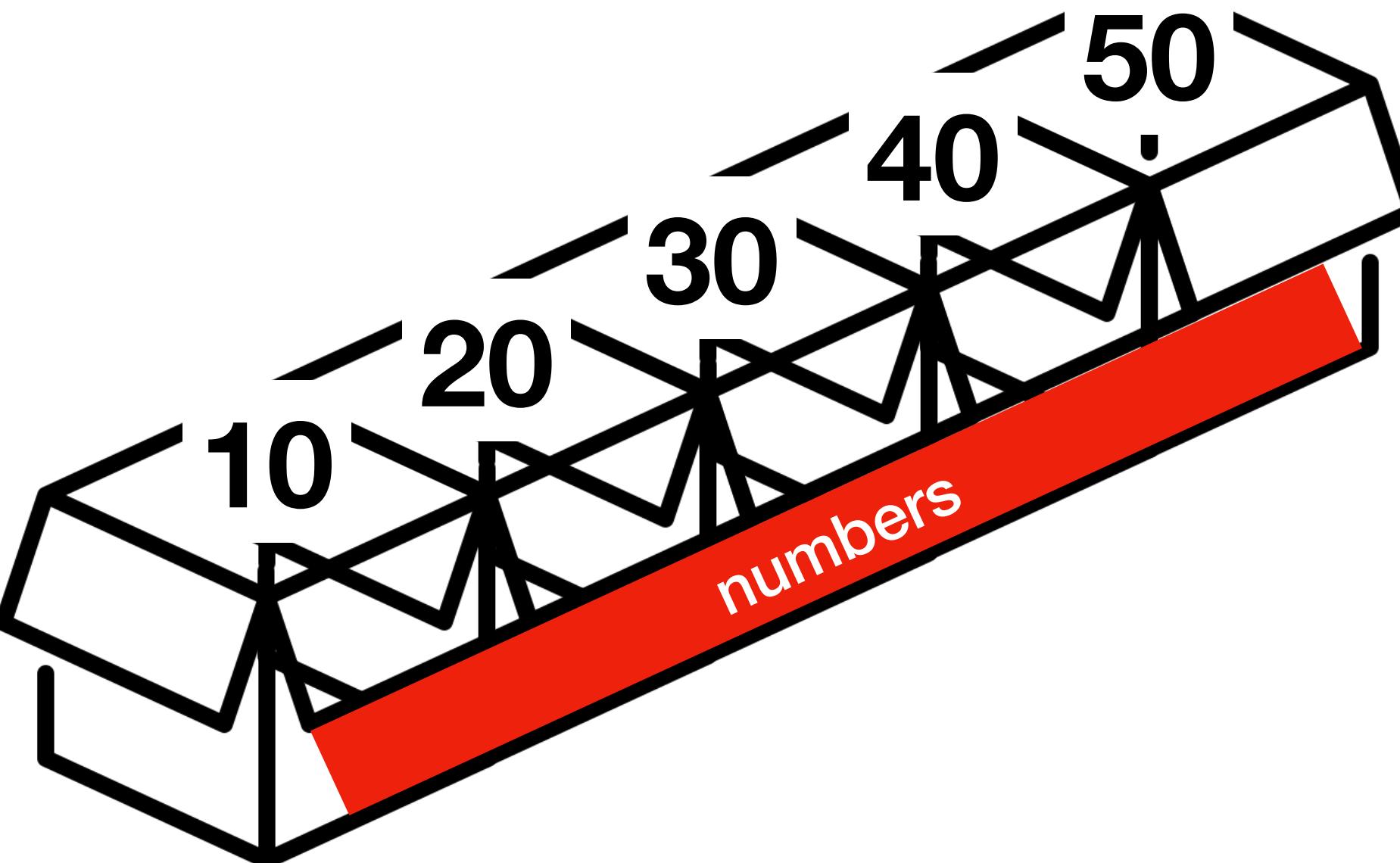
list

# 리스트(List)

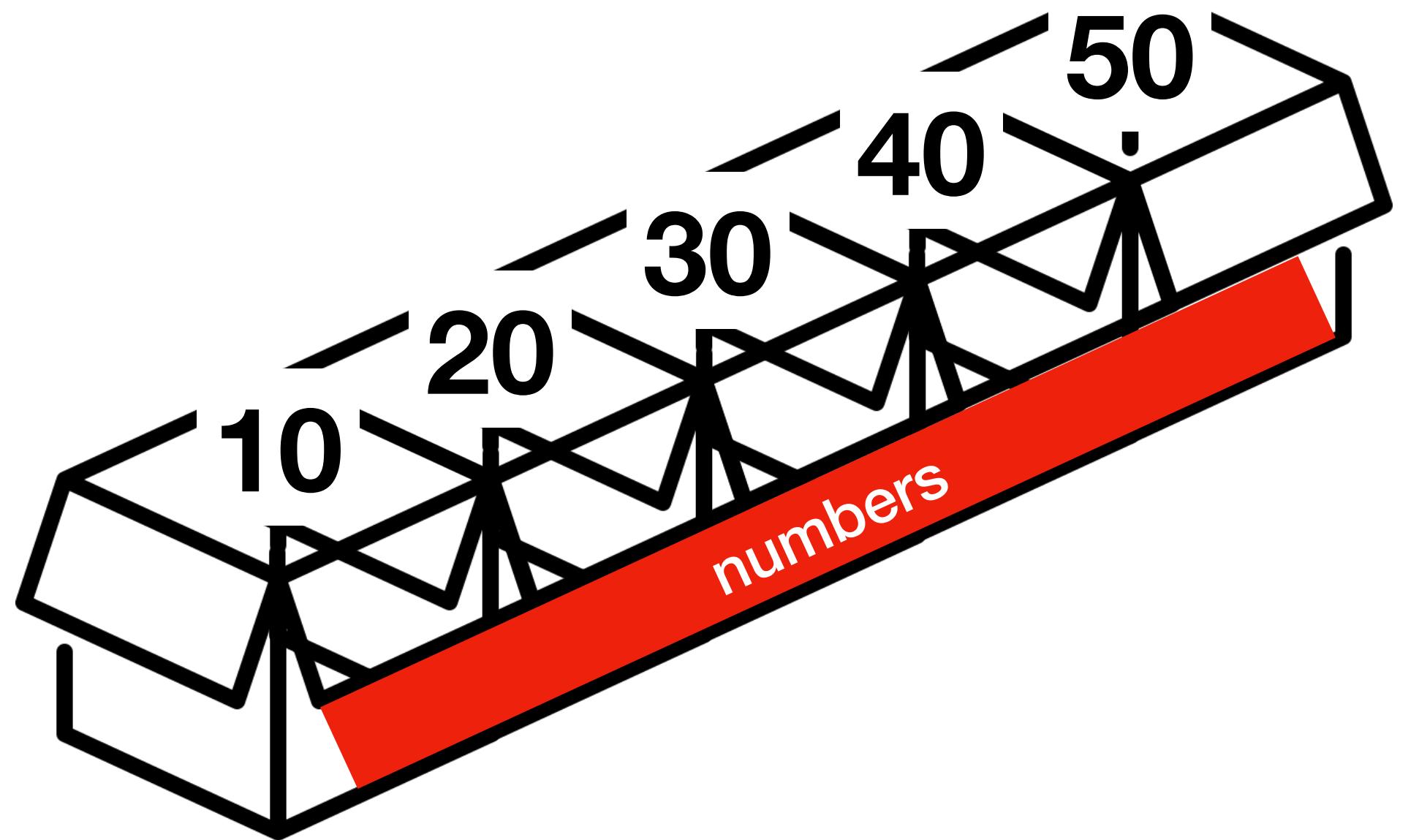


A.K.A 박스 여러개

# 할당



# 할당

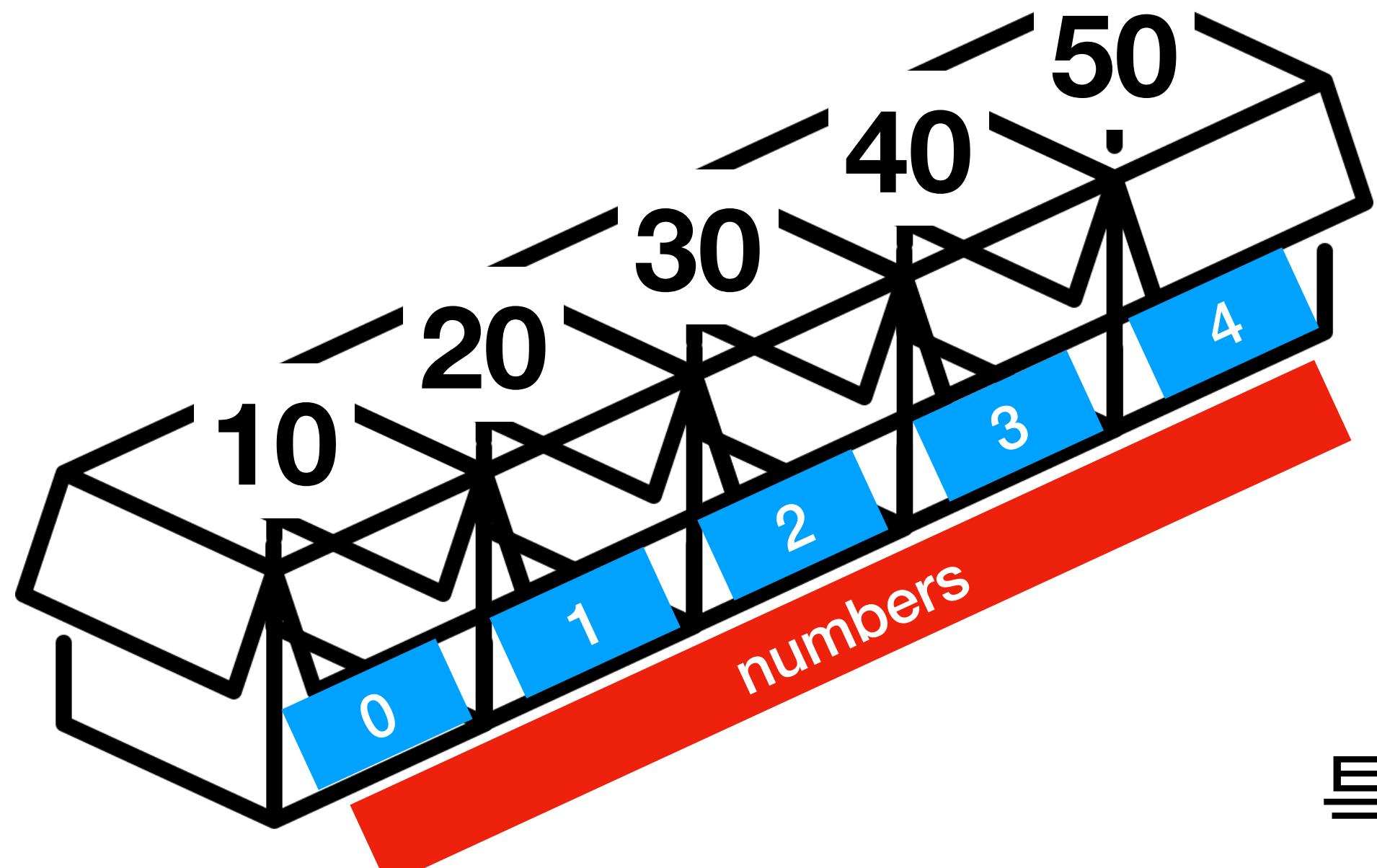


```
numbers = [10, 20, 30, 40, 50]
```



대괄호 ([ ])로 감싸고  
쉼표로 구분한다!

# 접근



numbers

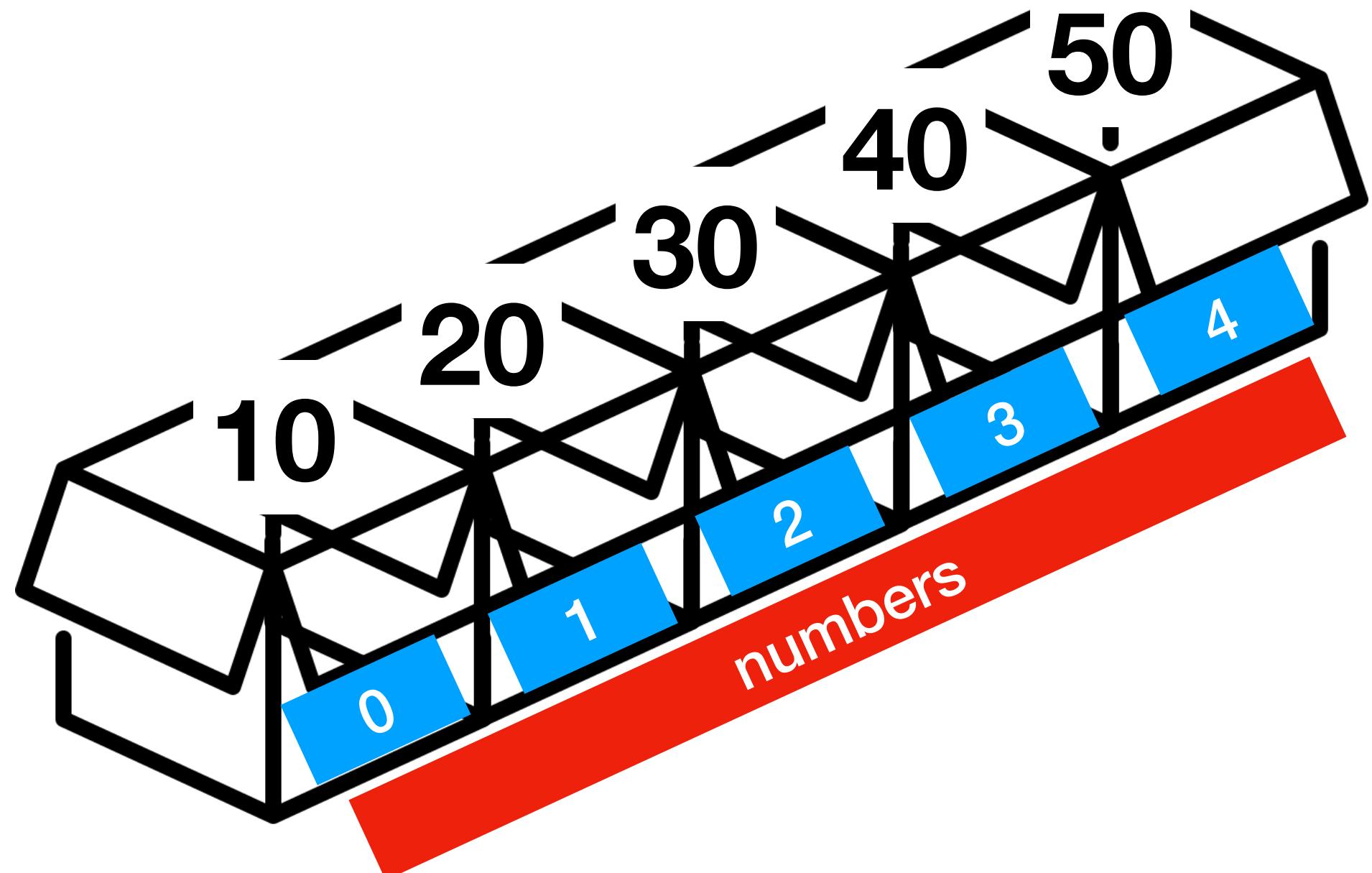
numbers[0]

# 리스트 전체

# 10

특정 요소에 접근할 때 대괄호 ([ ]) 안에  
사용하는 숫자를 인덱스(index)라고 한다!  
인덱스는 0부터 시작한다!

# 접근



`numbers[3]` # 40

`numbers[-1]` # 50

`numbers[-3]` # 30

파이썬은 음수 인덱싱도 가능!  
뒤에서부터 보면 된다!

`numbers[0:3]` # [10, 20, 30]

이건 슬라이싱! `:`으로 구분한다!  
인덱스 0 이상 3 미만(0, 1, 2)의  
요소들로 구성된 리스트

# 초기 할당

```
numbers = [10, 20, 30, 40, 50]
```

```
chars = ['a', 'b', 'c']
```

```
names = ['kim', 'lee', 'park']
```

```
heights = [171.0, 162.3, 158.7, 181.1]
```

# 접근

```
numbers = [10, 20, 30, 40, 50]
```

```
numbers[3] # 40
```

```
chars = ['a', 'b', 'c']
```

```
chars[-1] # 'c'
```

```
menus = ['피자', '짜장면', '백반']
```

```
menus[1:3] # ['짜장면', '백반']
```

# 요소 재할당 및 추가

```
numbers = [10, 20, 30, 40, 50]
```

```
numbers[3] = 100 # [10, 20, 30, 100, 50]
```

```
chars = ['a', 'b', 'c']
```

```
chars[-1] = 'c' # ['a', 'b', 'c']
```

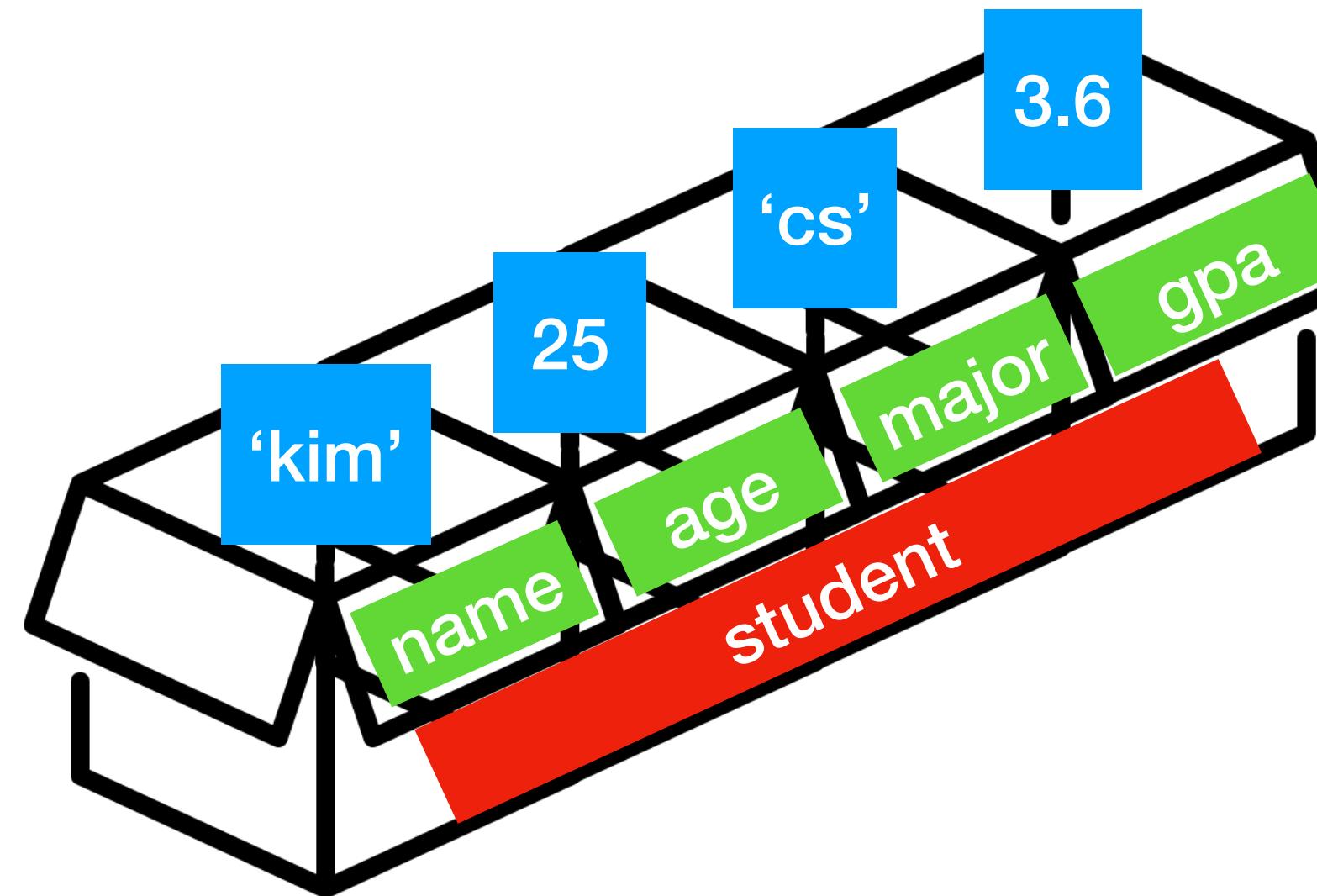
```
menus = ['피자', '짜장면']
```

```
menus.append('샐러드') # ['피자', '짜장면', '샐러드']
```

딕셔너리

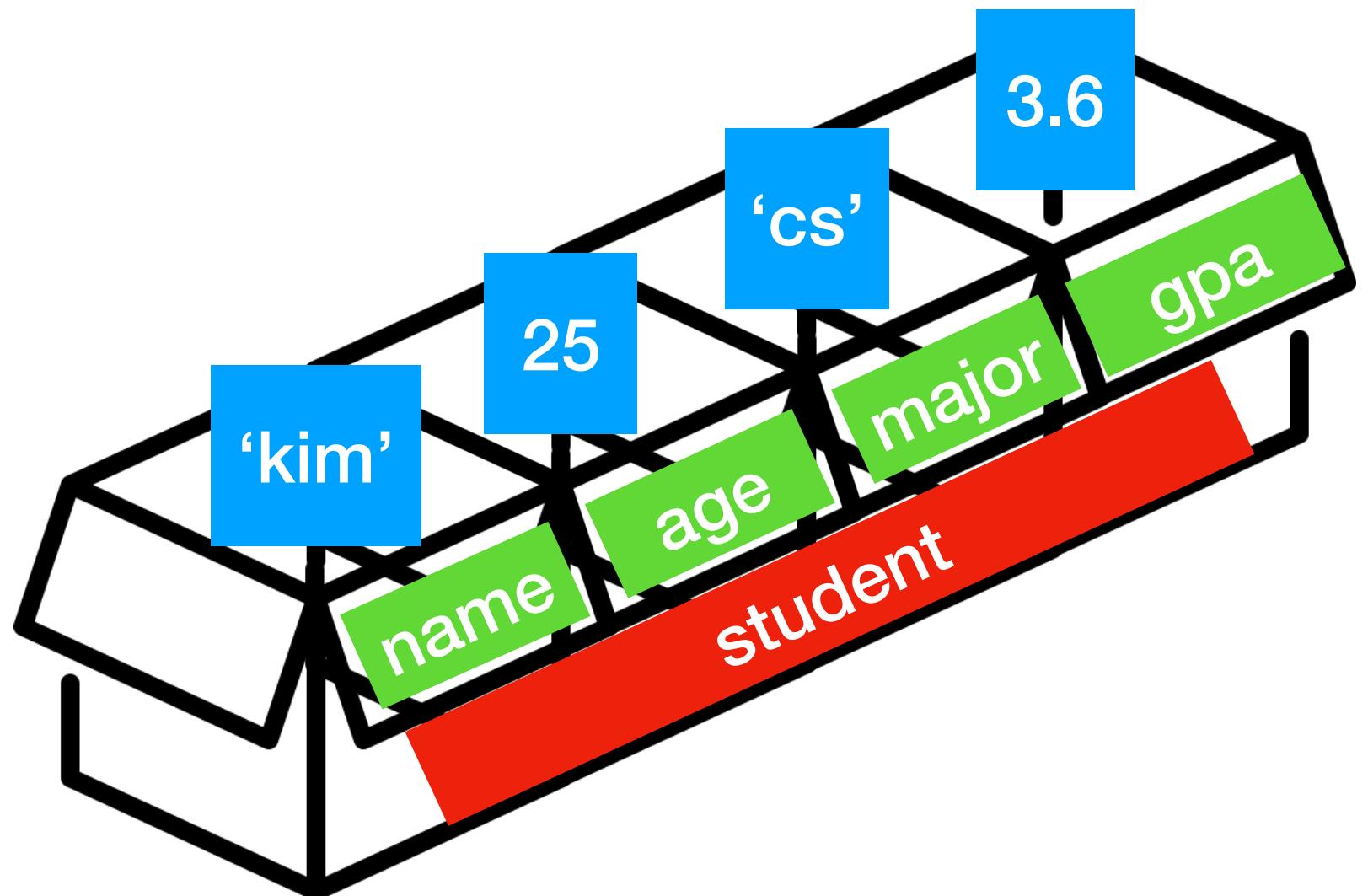
dict

# 딕셔너리(dict)



A.K.A 이름있는 박스 모음

# 할당

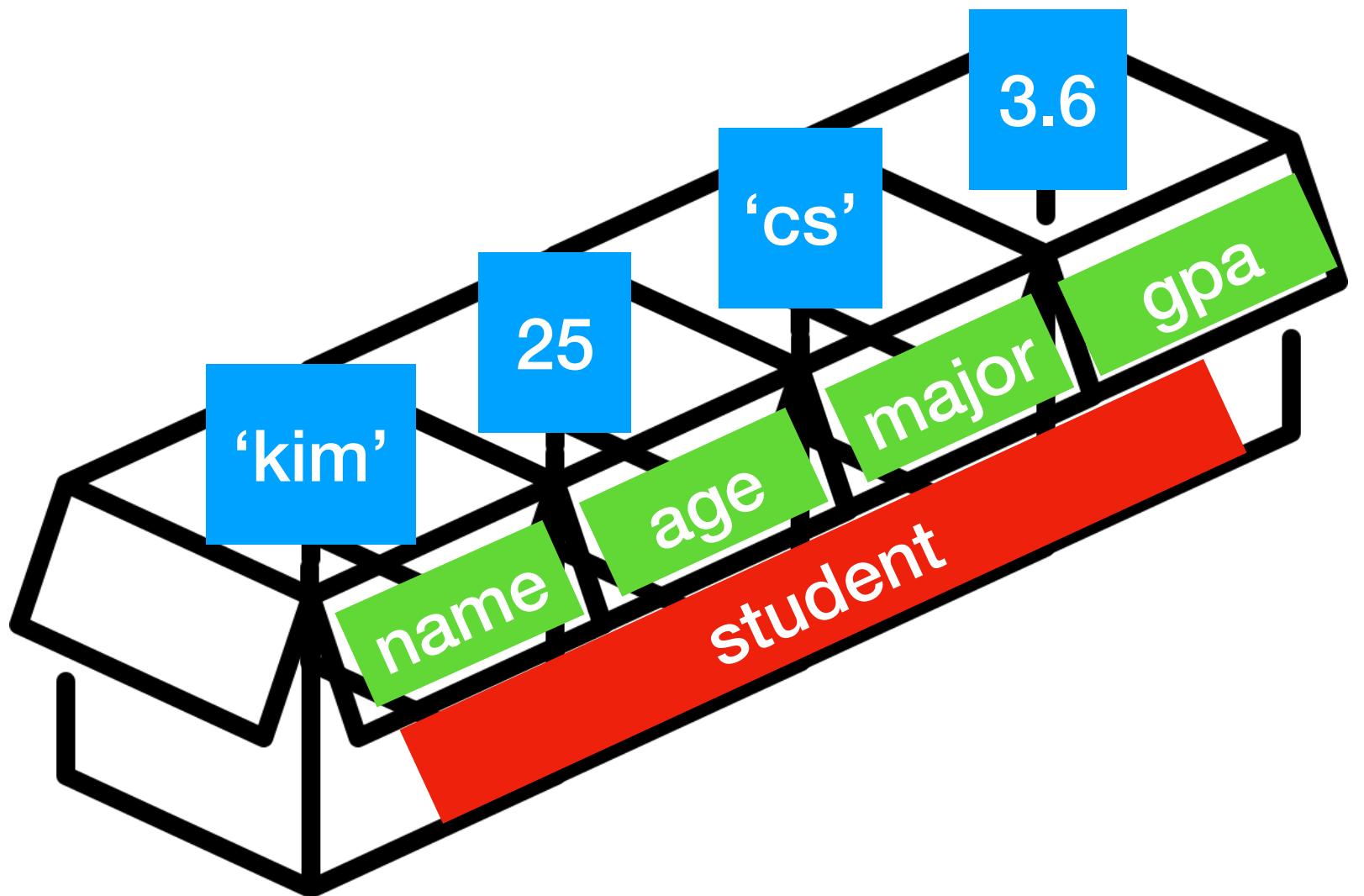


```
student = {  
    'name': 'kim',  
    'age': 25,  
    'major': 'cs',  
    'gpa': 3.6,  
}
```

key →      ← value

중괄호 ({ })로 감싸고  
key와 value를 콜론(:)으로 구분하고  
key-value쌍을 쉼표로 구분한다!

# 접근



```
student
```

# 딕셔너리 전체

```
student['name']
```

# 'kim'

```
student['age']
```

# 25

```
student['major']
```

# 'cs'

```
student['gpa']
```

# 3.6

# 요소 접근, 재할당, 추가

```
char_dict = {'a': 1, 'b': 2}
```

```
char_dict['b'] # 2
```

```
char_dict.get('b') # 2
```

```
char_dict['b'] = 20 # {'a': 1, 'b': 20}
```

```
char_dict['c'] # KeyError 접근 불가
```

```
char_dict.get('c') # None
```

```
char_dict['c'] = 30 # {'a': 1, 'b': 20, 'c': 30}
```

범위

range

# 범위(range)

1. 이름과는 다르게, 실제로는 범위 안의 정수만을 의미한다.
2. 리스트와는 다르게 우리 눈에 직관적으로 보이지 않는다.
3. 반복문에서 아주 많이 사용된다!

# 범위(range)

```
r1 = range(5) # 0, 1, 2, 3, 4
```

```
r2 = range(1, 3) # 1, 2
```

```
r3 = range(0, 10, 2) # 0, 2, 4, 6, 8
```

```
r4 = range(5, 1, -1) # 5, 4, 3, 2
```

# 연산자 & 자료 조작

## operators

# 산술 연산

기호	설명	예시	결과	비고
+	더하기	1 + 2	3	
-	빼기	4 - 2	2	사칙연산 순서 주의!
*	곱하기	3 * 4	12	
**	제곱	2 ** 4	16	우선순위가 높다!
/	나누기	5 / 2	2.5	정수끼리 나눠도 결과는 실수!
//	몫	7 // 3	2	
%	나머지	7 % 3	1	짝수/홀수와 배수 판별에 많이 쓰임!

# 산술 + 할당 연산

기호	설명	예시	결과	비고
<code>+ =</code>	더하기	<code>num = 1 num += 1</code>	<code>num은 2</code>	<code>num = num + 1</code> 과 같다!
<code>- =</code>	빼기	<code>num = 20 num -= 10</code>	<code>num은 10</code>	<code>num = num - 10</code> 과 같다!
<code>* =</code>	곱하기	<code>num = 10 num *= 3</code>	<code>num은 30</code>	<code>num = num * 3</code> 과 같다!
<code>/ =</code>	나누기	<code>num = 3 num /= 2</code>	<code>num은 1.5</code>	<code>num = num / 2</code> 와 같다!
<code>** =</code>	제곱	<code>num = 2 num **= 2</code>	<code>num은 4</code>	쓸일이 잘 없다..!
<code>// =</code>	몫	<code>num = 12 num // = 4</code>	<code>num은 3</code>	
<code>% =</code>	나머지	<code>num = 9 num %= 4</code>	<code>num은 1</code>	

# 비교 / 포함 연산

기호	설명	예시	결과	비고
>, <	초과, 미만	1 < 2	True	
>=, <=	이상, 이하	3 <= 2	False	
==	같다	10 == 2*5	True	연산자가 2개 이상의 기호로 이루어져 있어서 순서가 헷갈린다면, 항상 = 이 뒤에 온다고 기억하면 된다! (<=, >=, !=, +=, -=, *=, ...)
!=	같지 않다	100 != '100'	True	
in	포함 여부	'a' in 'apple' 1 in [1, 2, 3]	True	in 뒤에는 문자열과 리스트 (그리고 더 많은 것들이) 올 수 있다!

# 논리 연산

기호	설명	예시	결과	비고
and	논리적 and (논리곱)	True and True	True	and로 연결된 논리연산은 모두가 True여야 전체가 True 하나라도 False라면 전체가 False
		True and False	False	
		False and True	False	
		False and False	False	
or	논리적 or (논리합)	True or True	True	or로 연결된 논리연산은 모두가 False여야 전체가 False 하나라도 True라면 전체가 True
		True or False	True	
		False or True	True	
		False or False	False	

# 논리 연산 예시

이 강의를 진행하려면

컴퓨터 공학 전공 **and** 경력 n년 이상 **and** 강의 평가 4.5 이상 **and** 오늘 일정이 비어있어야 한다.  
하나라도 만족하지 않으면 강의를 진행할 수 없다 (하나라도 False => 전체 False)

이 강의에 참여하려면

학생 **or** 강사 **or** 매니저 **or** 담당자 여야 한다.  
하나라도 만족하면 강의에 참여할 수 있다 (하나라도 True => 전체 True)

# 문자 연산 & 인덱싱 & 포매팅

기호	설명	예시	결과	비고
+	연결하기	'py' + 'thon'	'python'	다른 타입(ex. 숫자)과는 연결 불가!
*	반복하기	'hi' * 3	'hihihi'	양의 정수만 곱할 수 있다!
[ ]	인덱스 접근	'python'[0]	'p'	인덱스는 0부터 시작! 음수 인덱스는 뒤에서부터! 인덱스가 문자열 길이를 넘어가면 에러!
[ : ]	슬라이싱	'python'[0:3]	'pyt'	[x:y] 일 경우 인덱스 x이상 y미만! y는 포함되지 않아!
[ :: ]	슬라이싱 + 스텝	'python'[0:5:2]	'pto'	[x:y:z] 는 범위 x이상 y미만에서 z만큼 점프!
f'{}'	f-string	mood = 'good' f'I feel {mood}'	'I feel good'	변수와 조합하면 강력해진다!
\n, \t, \ ...	이스케이프 문자열	'*\n**\n***'	* ** ***	역 슬래시(\)와 다른 문자가 합쳐져서 하나의 역할을 한다! 둘이 합쳐 하나의 문자인게 포인트!

# 리스트 연산 & 인덱싱 & 슬라이싱 & 추가

기호	설명	예시	결과	비고
+	연결하기	[1, 'a'] + [0.5, True]	[1, 'a', 0.5, True]	
*	반복하기	[0] * 3	[0, 0, 0]	
[]	인덱스 접근	[1, 2, 3][0]	1	인덱스는 0부터 시작! 음수는 뒤에서부터! 인덱스가 문자열 길이를 넘어가면 에러!
[:]	슬라이싱	['a', 'b', 'c'][1:2]	['b']	[x:y]일 경우 인덱스 x이상 y미만! y는 포함되지 않아!
[::]	슬라이싱 + 스텝	['a', 'b', 'c'][0:3:2]	['a', 'c']	[x:y:z]는 범위 x이상 y미만에서 z만큼 점프!
.append()	요소 추가	menus = [] menus.append('빵')	menus는 ['빵']	

**조건**

Condition

# 조건문 작성법



**if 조건:**



조건이 참(True)일 경우  
실행되는 코드 블럭

**if 조건:**

조건이 참(True)일 경우  
실행되는 코드 블럭

**else:**

조건이 거짓(False)일 경우  
실행되는 코드 블럭

# 조건문 작성법

if 조건1:

    조건1이 True일 경우 실행

elif 조건2:

    조건1은 아니지만, 조건2는 True일 경우 실행

elif 조건3:

    조건1도 조건2도 아니지만, 조건3은 True일 경우 실행

else:

    위의 조건이 모두 아닐 경우 실행

# 조건문 작성법

등급	획득점수	등급별 역량
LV.1	400 ~ 599점	프로그래밍 개념과 기본적인 문법 지식을 가지고 간단한 문제 해결 및 프로그램을 작성하는 능력이 있습니다.
LV.2	600 ~ 749점	프로그래밍 기본 지식을 가지고 복잡하지 않은 프로그램 설계·구현, 디버깅 문제를 해결할 수 있는 능력이 있습니다.
LV.3	750 ~ 899점	프로그래밍 중급 지식을 가지고 간단한 프로그램을 요구사항에 맞게 설계·구현, 디버깅, 수정 등 테스트 케이스를 설계할 수 있는 능력이 있습니다.
LV.4	900 ~ 1000점	프로그래밍 중급 지식을 가지고 프로그램을 요구사항에 맞게 설계·구현할 수 있는 능력이 있습니다.

# 조건문 작성법

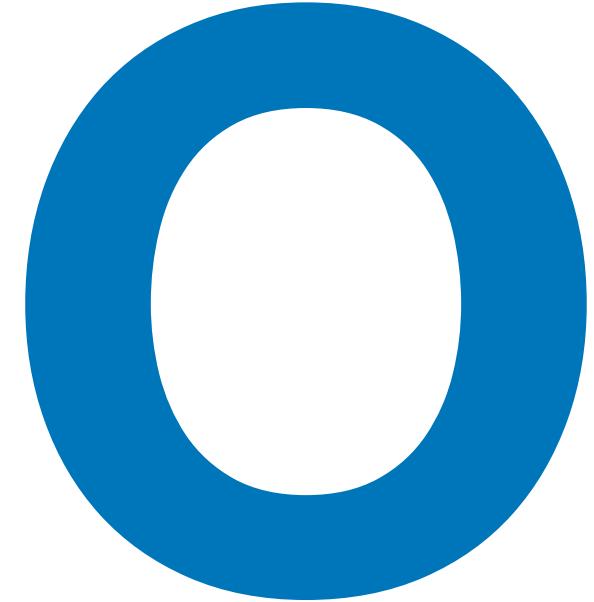
```
score = 678
```

```
if 1000 >= score >= 900:  
    level = 4  
elif 900 > score >= 750:  
    level = 3  
elif 750 > score >= 600:  
    level = 2  
elif 600 > score >= 400:  
    level = 1  
else:  
    level = '(:( )'
```

# 조건문 작성법

```
score = 678
```

```
if score >= 900:  
    level = 4  
elif score >= 750:  
    level = 3  
elif score >= 600:  
    level = 2  
elif score >= 400:  
    level = 1  
else:  
    level = '(:( )'
```



# 조건문 작성법

```
score = 678
```

```
if 1000 >= score:  
    level = 4  
elif 900 > score:  
    level = 3  
elif 750 > score:  
    level = 2  
elif 600 > score:  
    level = 1  
else:  
    level = '(◑◑)'
```



# 조건을 잘 짜자!

```
banana_milk = 2
```

```
is_melon_bread = True
```

```
if is_melon_bread:  
    melon_bread = 4
```



나 너 우리



```
banana_milk = 2
```

```
is_melon_bread = True
```

```
if is_melon_bread:  
    banana_milk = 4
```



컴퓨터

반복

Iteration

# 반복문 종류

수동

while

자동

for ... in

# while 구문

while 조건:



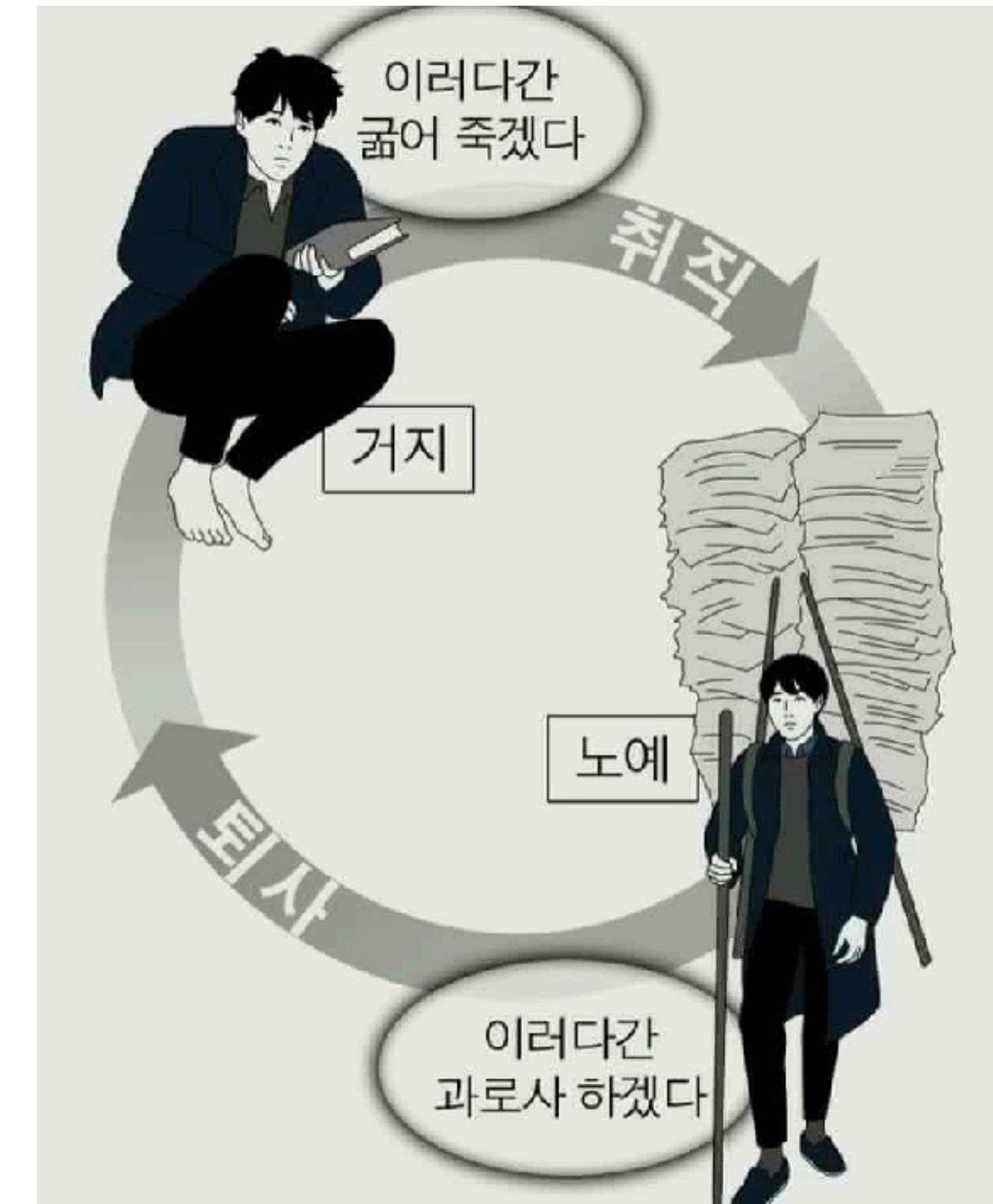
조건이 참(True)일 경우  
반복적으로 실행되는 코드 블럭

# while 구문

```
while True:  
    print('INFINITE')
```

무한루프

Inifinite Loop



# while 구문

# 5 번만 반복하고 싶다면?

```
n = 0
while n < 5:
    print('FIVE TIMES')
    n += 1
```

조건 설정과 관리가 매우 중요하다!

# while 구문

# numbers 의 요소들을 모두 더하고 싶다면?

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

```
idx = 0
```

```
total = 0
```

```
while idx < len(numbers):
```

```
    total += numbers[idx]
```

```
    idx += 1
```

# for ... in 구문

for **임시변수** in **리스트, 문자열 등등:**

리스트나 문자열의 요소들이  
처음부터 끝까지 차례로 임시변수에 할당되고  
모든 요소들이 임시변수에 들어가면 종료

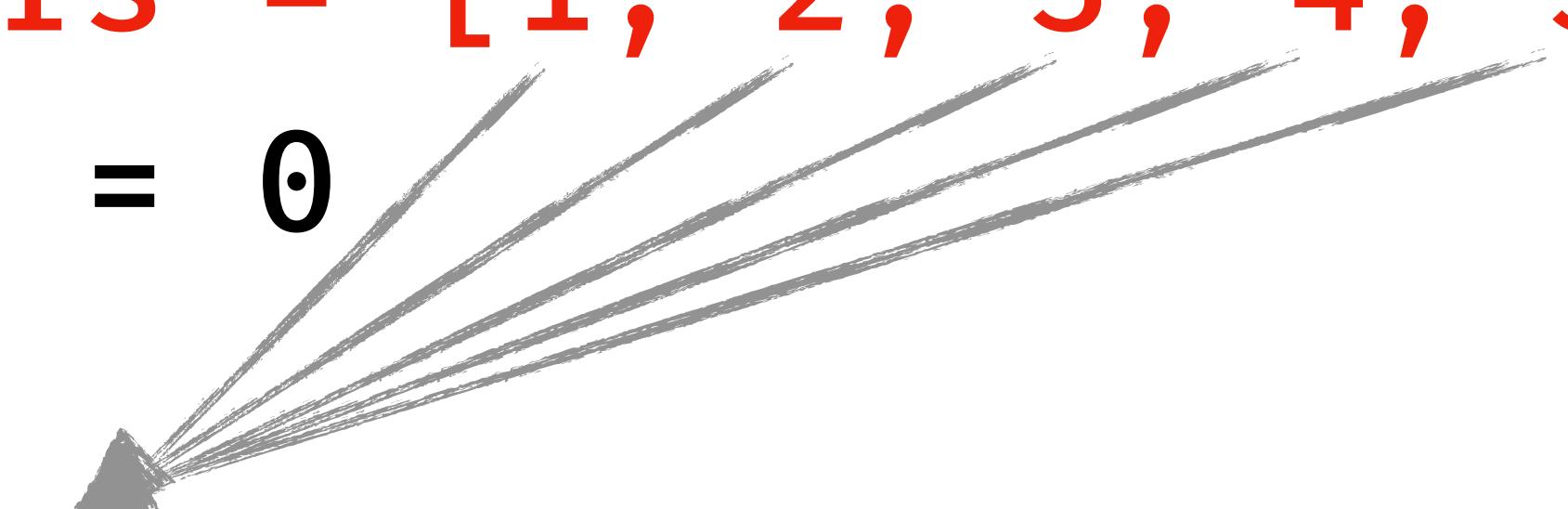
# for ... in 구문

# numbers 의 요소들을 모두 더하고 싶다면?

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

```
total = 0
```

```
for num in numbers:  
    total += num
```



# while 과 for 비교

# numbers 의 요소들을 모두 더하고 싶다면?

```
numbers = [1, 2, 3, 4, 5]
total = 0
idx = 0

while idx < len(numbers):
    total += numbers[idx]
    idx += 1
```

조건 설정 & 관리

```
numbers = [1, 2, 3, 4, 5]
total = 0

for num in numbers:
    total += num
```

임시변수

# for & range

# 5 번만 반복하고 싶다면?

```
for n in range(5):  
    print('FIVE TIMES')
```

```
FIVE TIMES  
FIVE TIMES  
FIVE TIMES  
FIVE TIMES  
FIVE TIMES
```

범위(range)

1. 이름과는 다르게, 실제로는 범위 안의 정수만을 의미한다.
2. 리스트와는 다르게 우리 눈에 직관적으로 보이지 않는다.
3. 반복문에서 아주 많이 사용된다!

# for & string

# 문자열을 for로 반복하면 한글자씩 임시변수에 들어간다!

```
for char in 'python':  
    print(f'!{char}!')
```

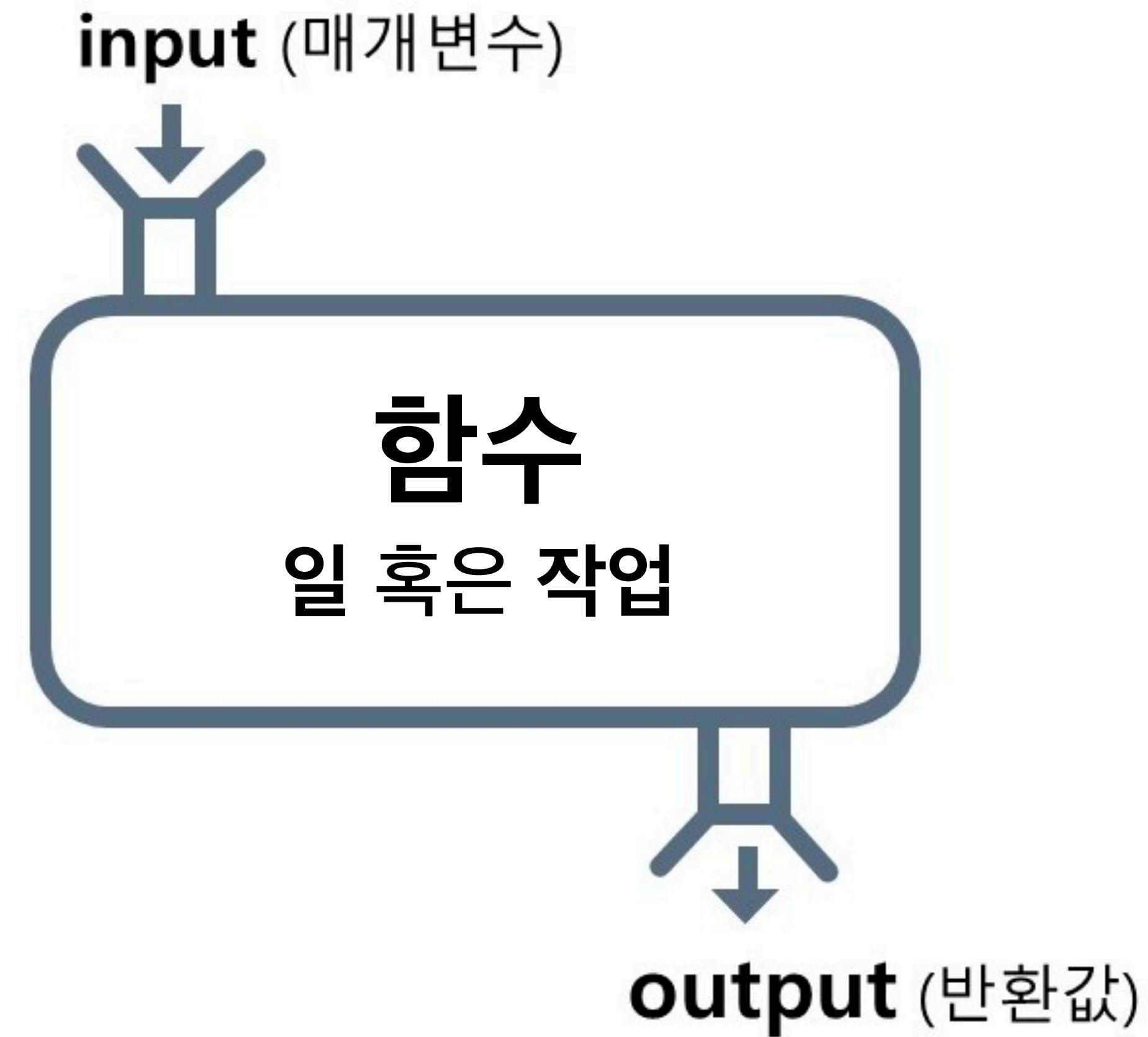
```
!p!  
!y!  
!t!  
!h!  
!o!  
!n!
```

f'{}'	f-string	mood = 'good' f'I feel {mood}'	'I feel good'	변수와 조합하면 강력해진다!
-------	----------	-----------------------------------	---------------	-----------------

함수

function

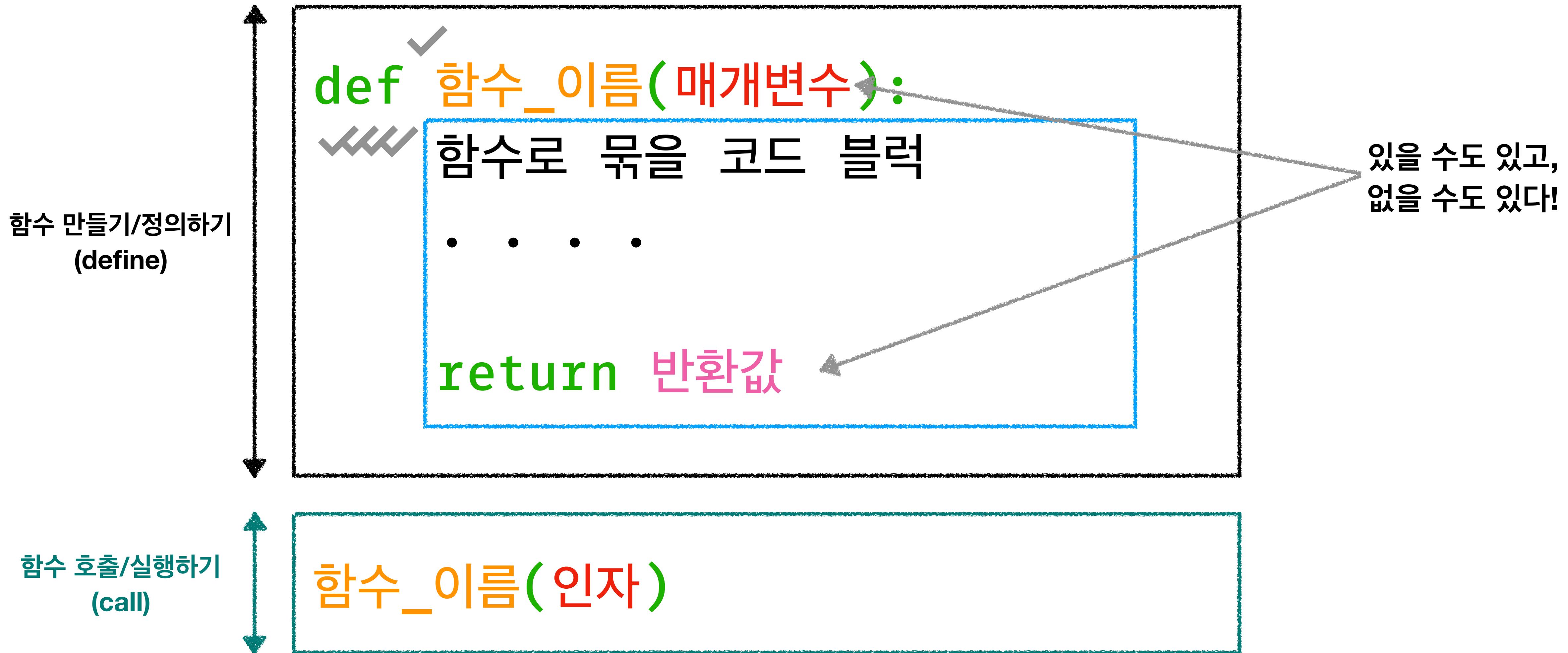
# 함수



# 내장 함수

내장함수	설명	예시	결과	비고
<code>print()</code>	콘솔 출력	<code>print('Python!')</code>	<code>Python!</code> 이 출력	
<code>len()</code>	길이 반환	<code>len('Python!')</code> <code>len([1, 2, 3])</code>	7 3	반복문에서 잠깐 봤듯이 문제풀이에 많이 쓰인다!
<code>str()</code>	문자열 생성	<code>str(987)</code>	'987'	모든 숫자들은 문자열로 변환 가능!
<code>int()</code>	정수 생성	<code>int('123')</code>	123	숫자로 구성되지 않은 문자열('abc')은 정수로 변환할 수 없다!
<code>list()</code>	리스트 생성	<code>list(range(3))</code> <code>list('abc')</code>	[0, 1, 2] ['a', 'b', 'c']	
<code>bool()</code>	참/거짓 생성	<code>bool(0)</code> <code>bool('')</code> <code>bool([])</code>	<code>False</code>	0, '', [] 과 같이 “없다/비었다”라고 해석할 수 있는 모든 값들은 <code>False</code> 로 변환된다!
		<code>bool(1)</code> <code>bool('qwer')</code> <code>bool([1, 2, 3])</code>	<code>True</code>	위의 경우가 아닌 나머지 모든 경우는 <code>True</code> 로 변환된다!

# 함수 만들기 & 호출하기



# 함수 만들고 사용하기

```
def hello_n_times(num):  
    for i in range(num):  
        print(f'hello - {i}')
```

```
hello_n_times(5)
```

```
hello - 0  
hello - 1  
hello - 2  
hello - 3  
hello - 4
```

# 함수 만들고 사용하기

```
def get_circle_area(r):  
    pi = 3.14  
    area = pi * r**2  
    return area  
  
print(get_circle_area(3))
```

28.26

# 함수를 사용하는 이유?

DRY (Don't repeat yourself)

같은 기능을 구현하기 위해 반복적으로 같은 코드를 쓰지 않을 수 있다.

Easy To Read

(함수 이름을 잘 짓는다면) 전체 코드의 가독성이 좋아진다.

Maintain

프로그램에 문제가 발생하거나 기능의 변경이 필요할 때에도 손쉽게 유지보수를 할 수 있다.