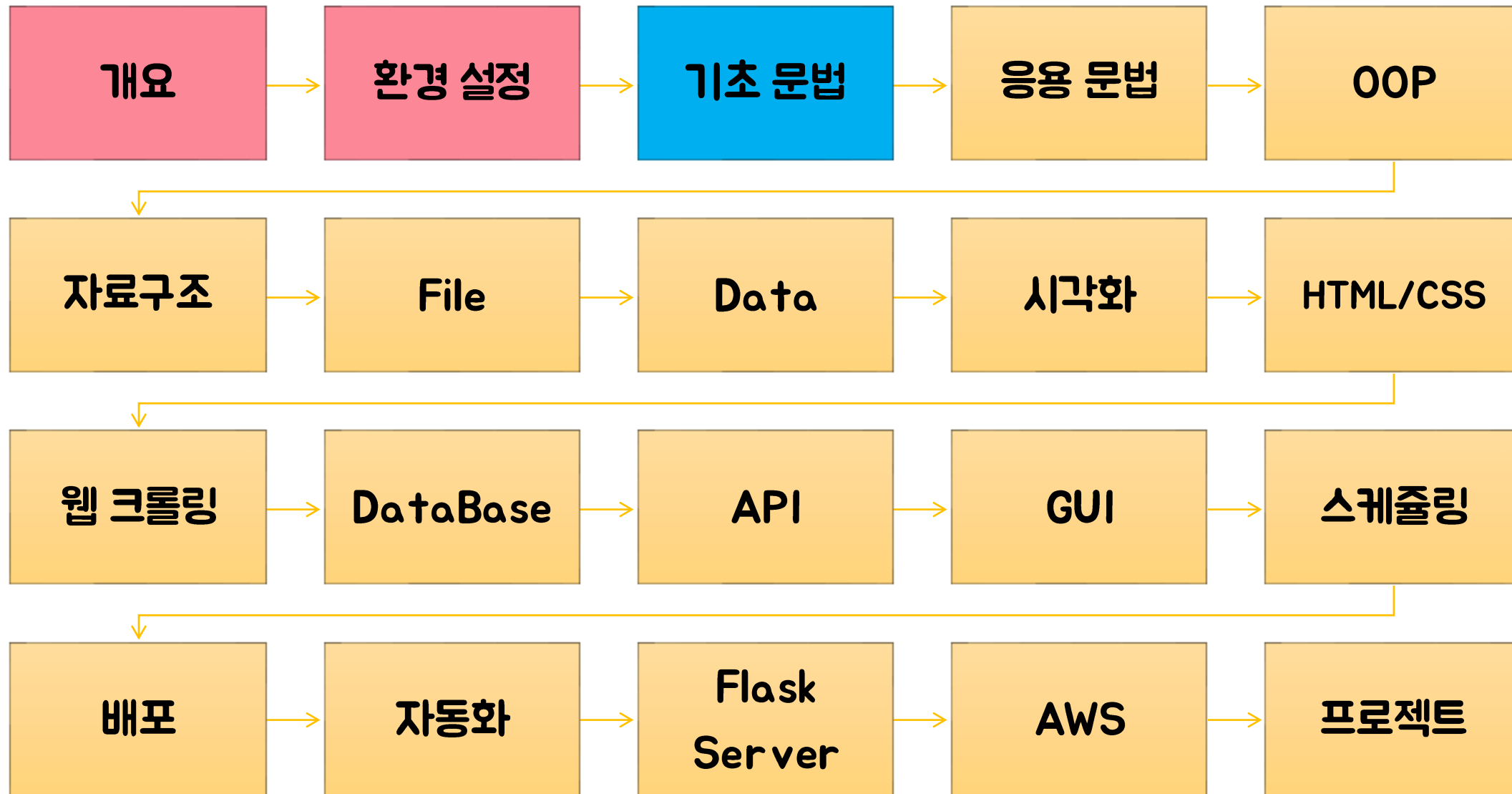




대한상공회의소
서울기술교육센터

나 예 호 교수

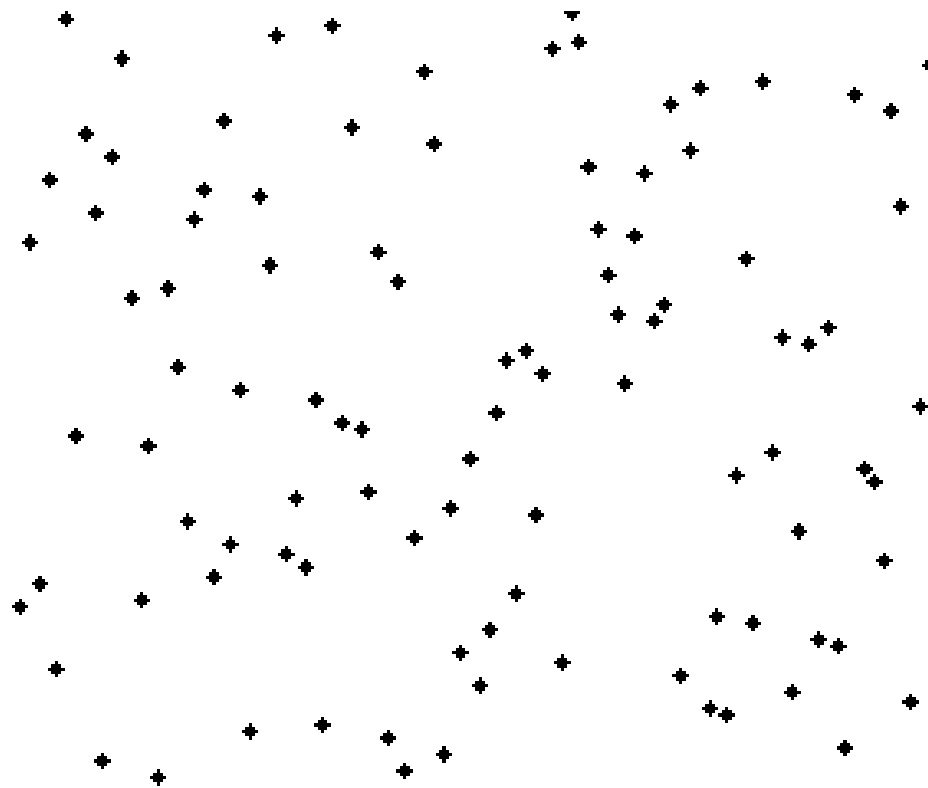


정렬 알고리즘

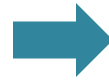
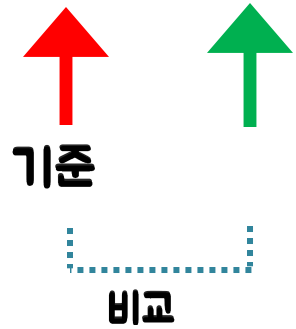
원소들을 일정한 순서대로 열거하는 알고리즘

Bubble sort

두 인접한 원소를 비교하여 정렬하는 방법
속도는 느리지만 코드가 단순하다.

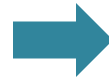
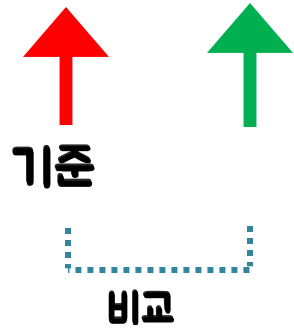


[0]	[1]	[2]	[3]	[4]
45	7	12	82	25



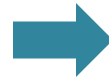
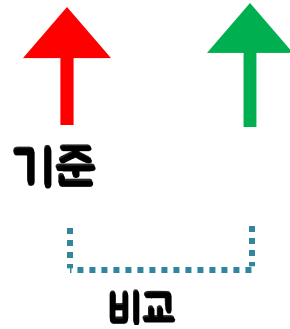
[0]	[1]	[2]	[3]	[4]
7	45	12	82	25

[0]	[1]	[2]	[3]	[4]
7	45	12	82	25



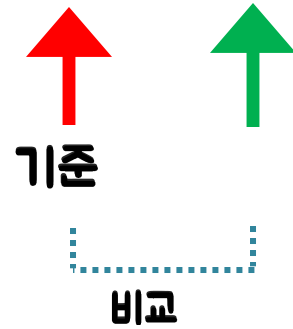
[0]	[1]	[2]	[3]	[4]
7	12	45	82	25

[0]	[1]	[2]	[3]	[4]
7	12	45	82	25

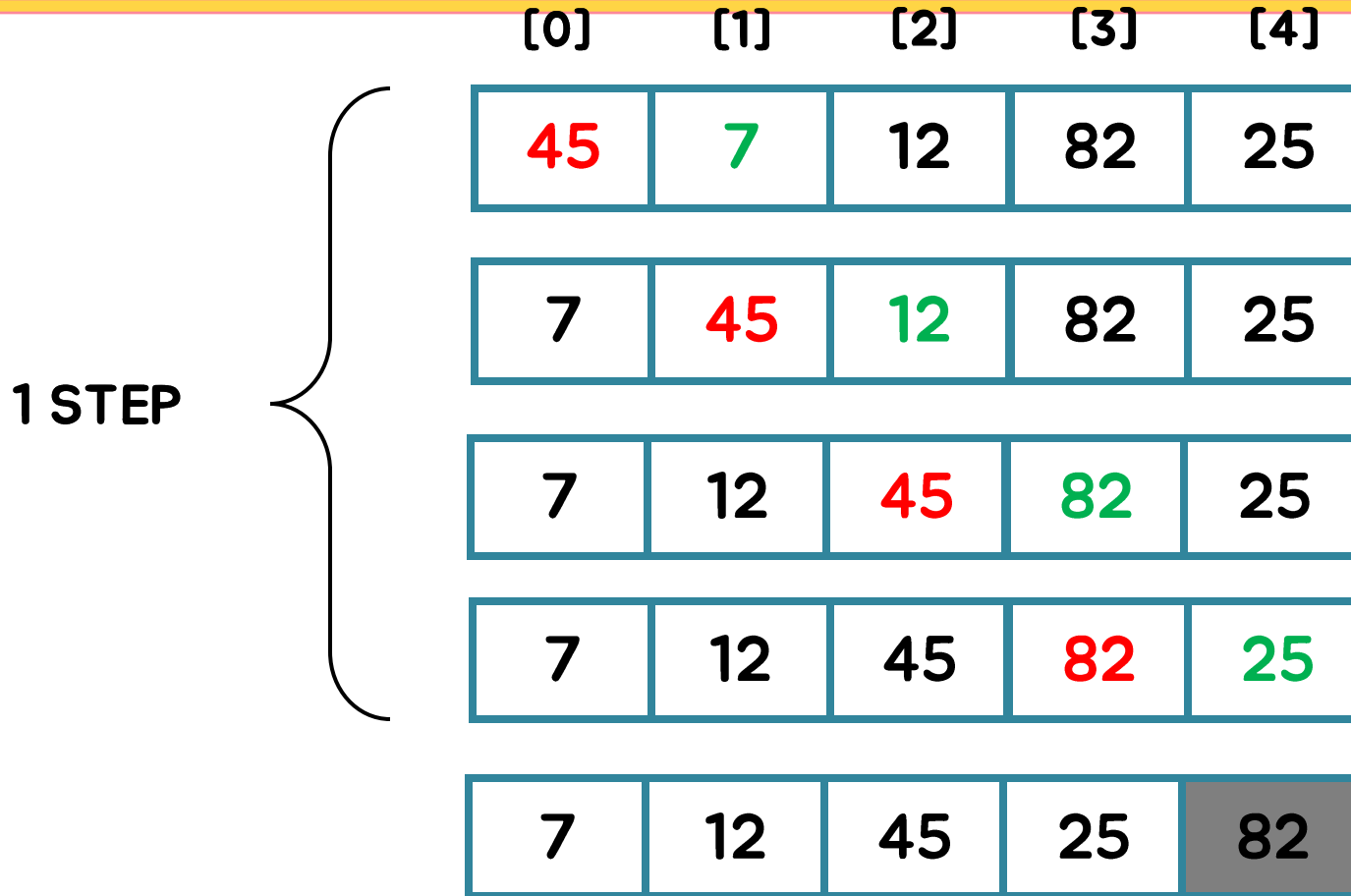


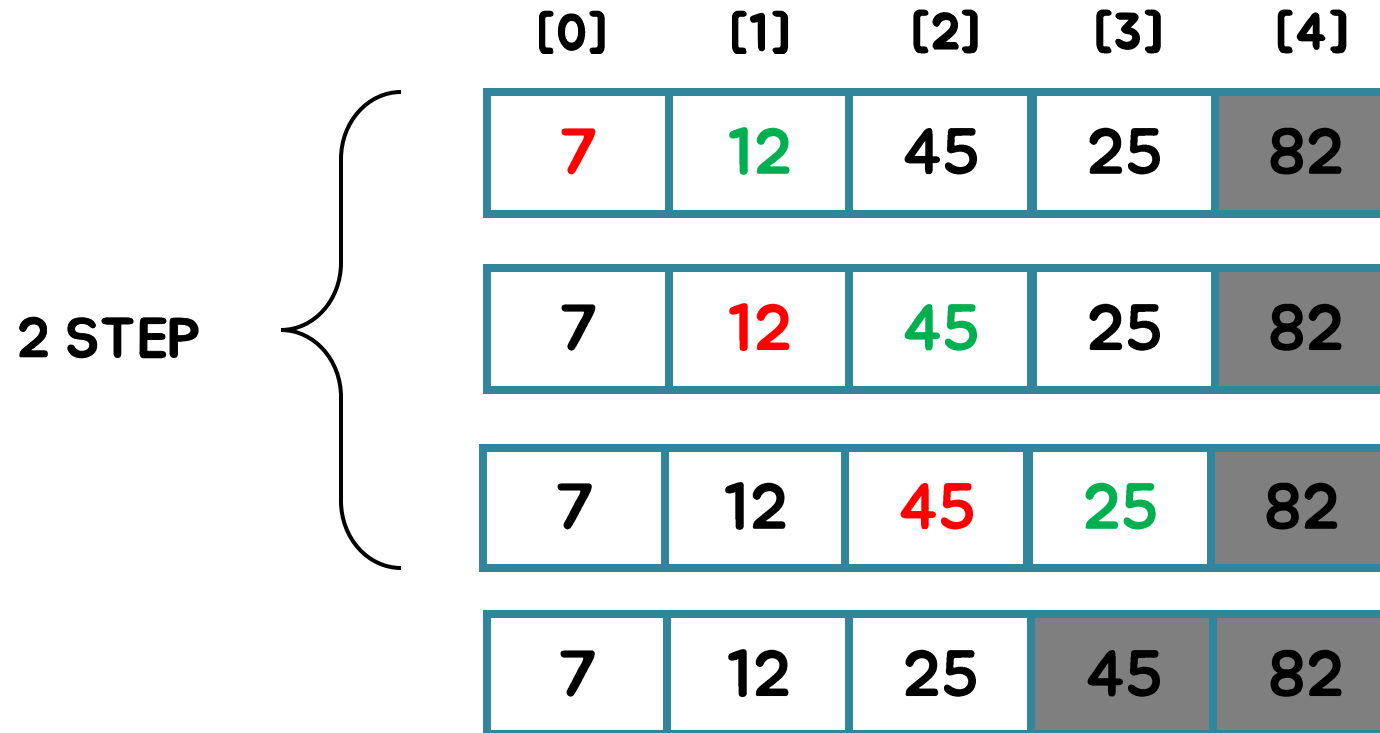
[0]	[1]	[2]	[3]	[4]
7	12	45	82	25

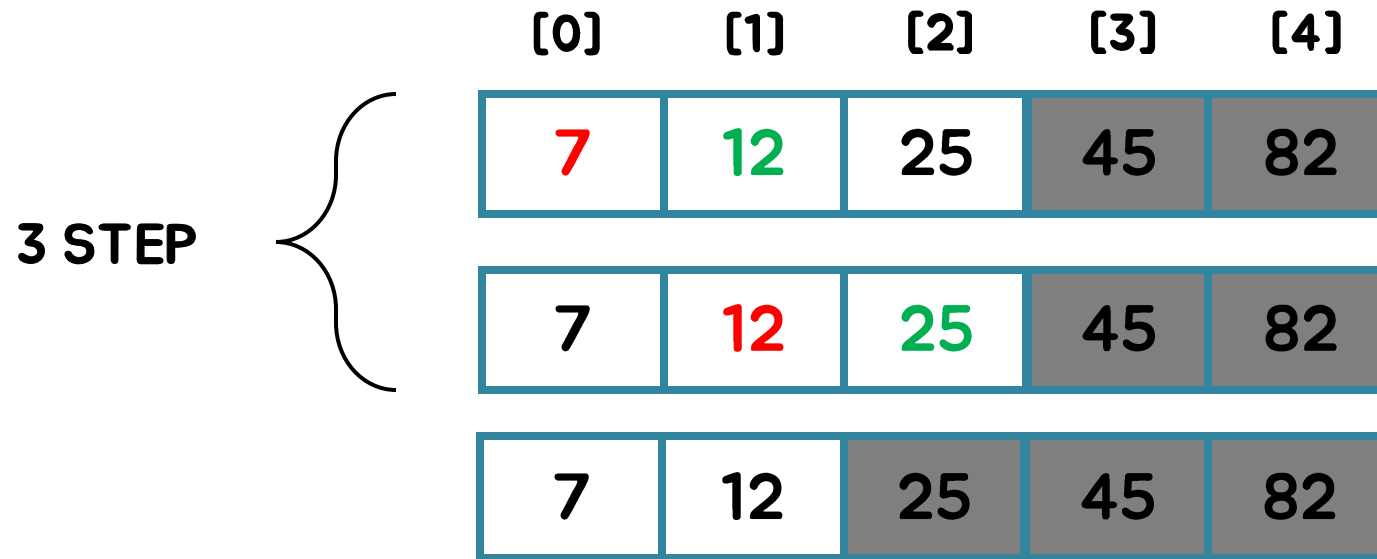
[0]	[1]	[2]	[3]	[4]
7	12	45	82	25

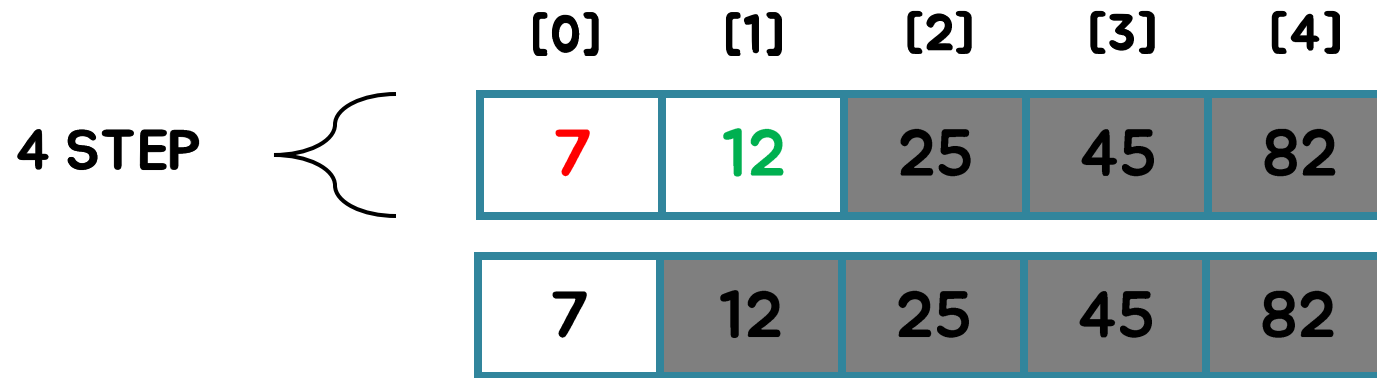


[0]	[1]	[2]	[3]	[4]
7	12	45	25	82



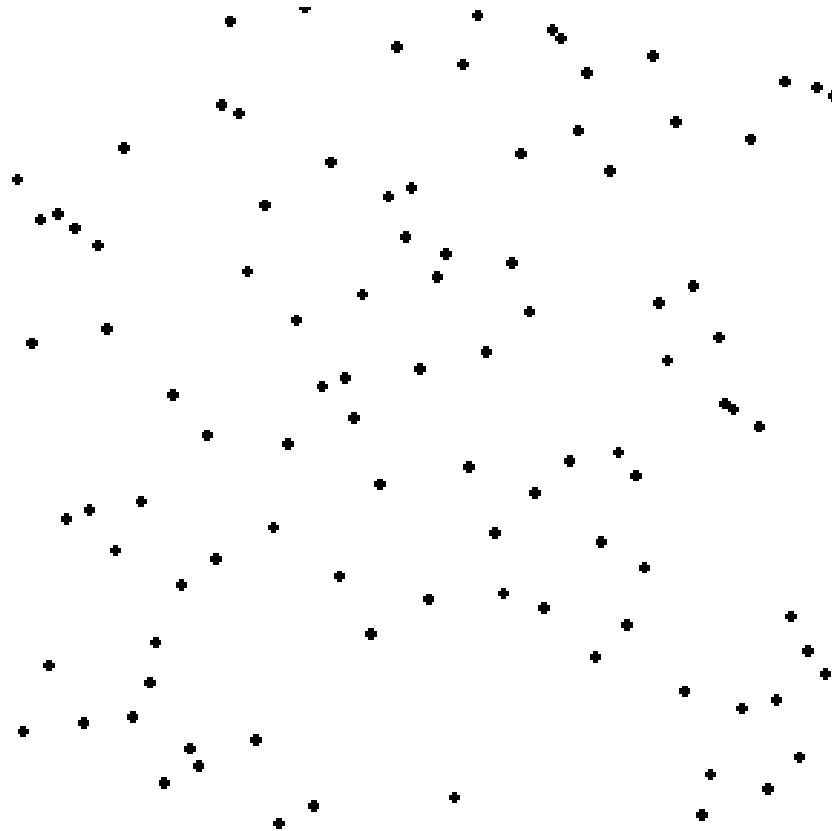






Selection sort

가장 큰 원소 또는 작은 원소를 찾아
주어진 위치(리스트 처음~끝)를 교체해 나가는 정렬 방법



기준 선택



[0] [1] [2] [3] [4]

98	7	70	13	24
----	---	----	----	----

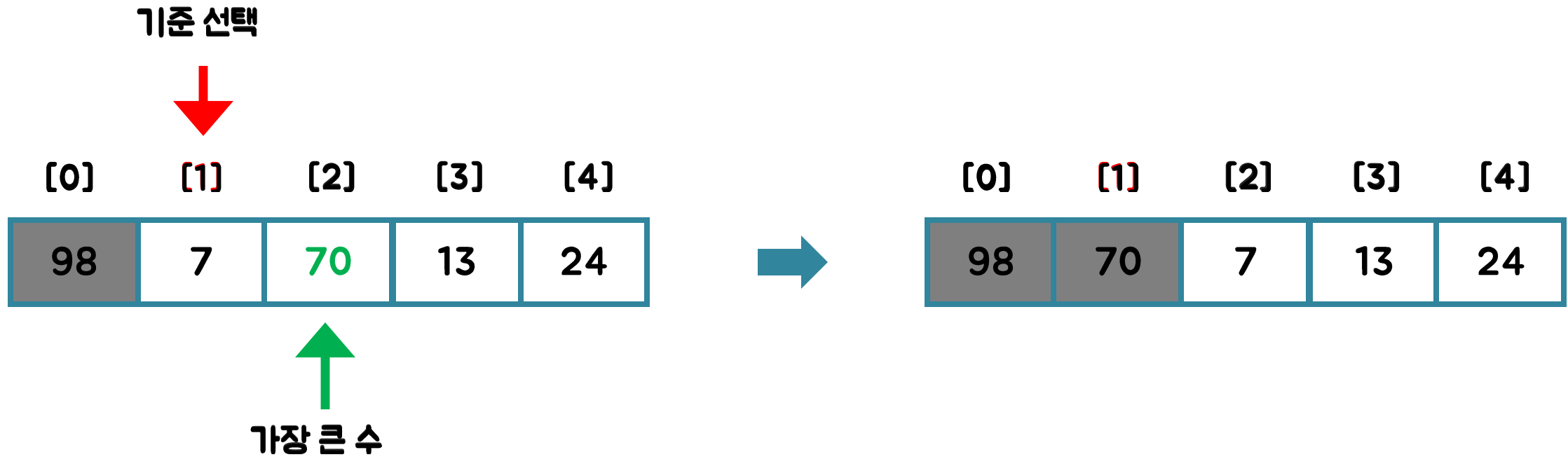


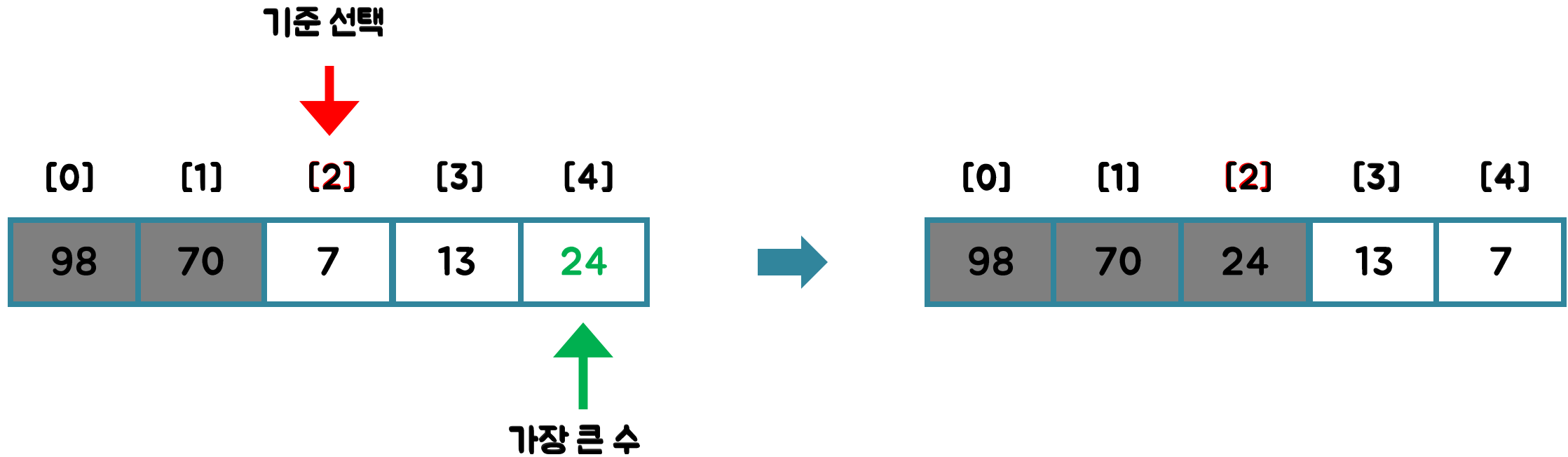
[0] [1] [2] [3] [4]

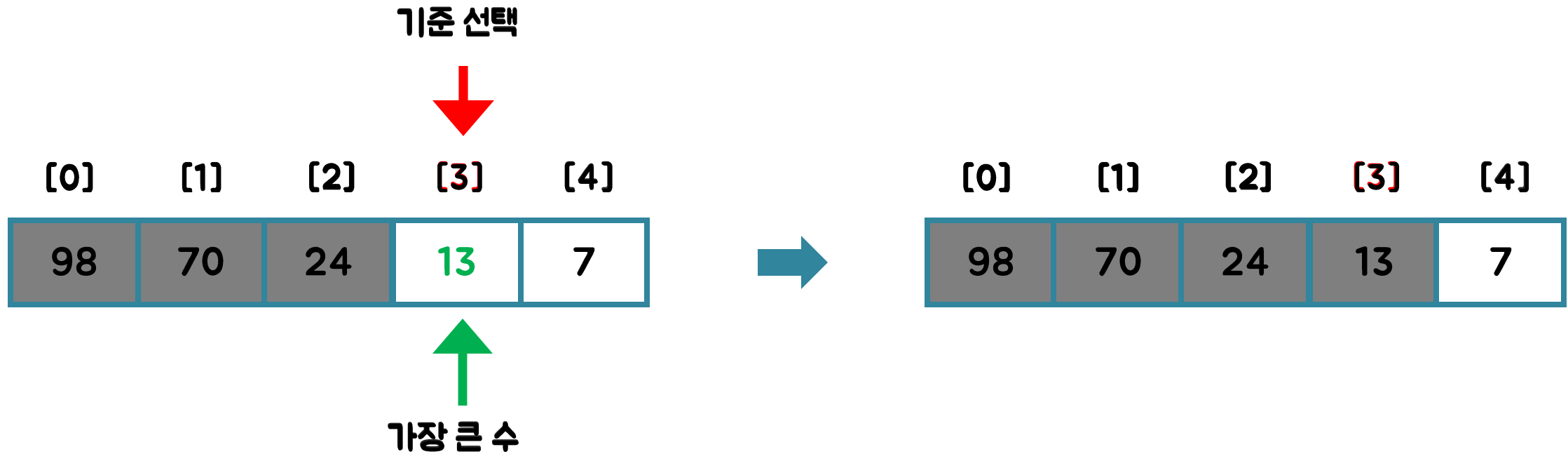
98	7	70	13	24
----	---	----	----	----



가장 큰 수







검색 알고리즘

특정 원소를 검색하는 알고리즘

Sequential search

가장 단순한 검색 방법으로 원소의 정렬이 필요 없다.
하지만 리스트 길이가 길면 비효율적

찾고자 하는 수

78

[0]

[1]

[2]

[3]

[4]

[5]

[6]

[7]

[8]

[9]

13

35

15

11

26

72

78

13

61

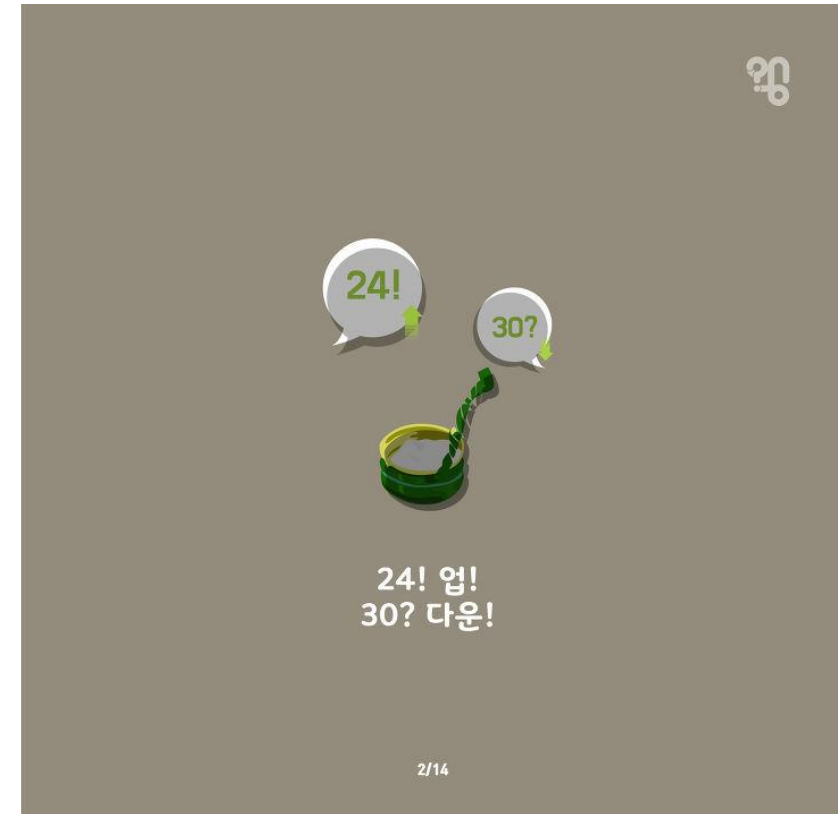
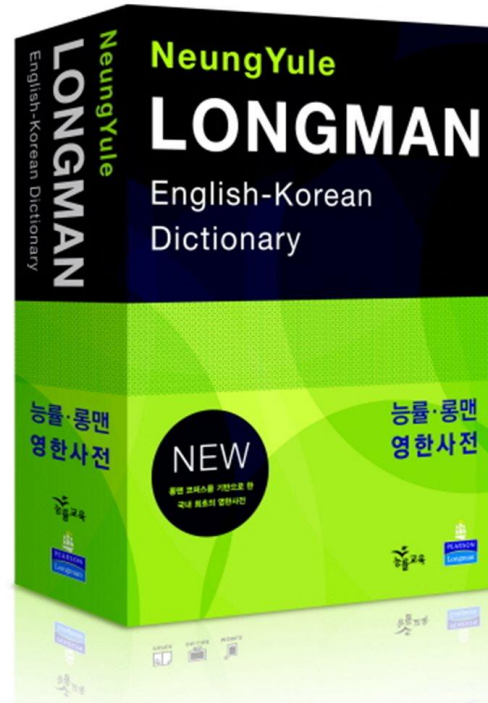
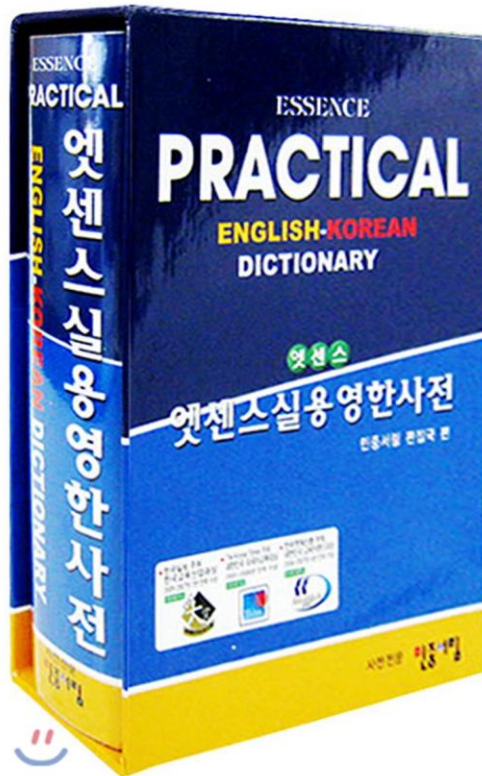
90

비교

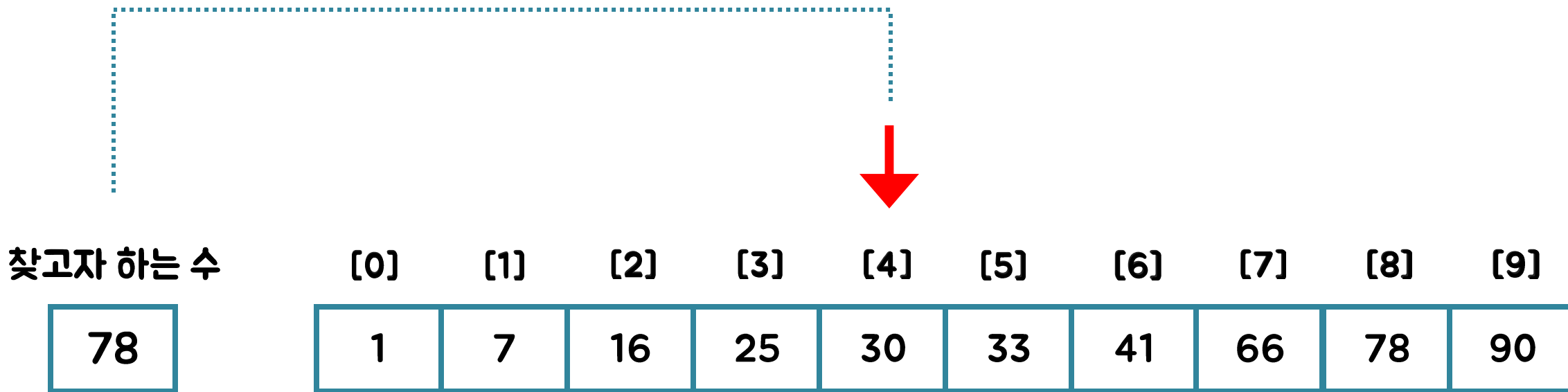
처음부터 끝까지 순차적으로 비교

Binary search

리스트의 **중간 값을** 정해 **크고 작음을** 비교해
검색하는 알고리즘 **정렬된 리스트**에 사용 할 수 있다.

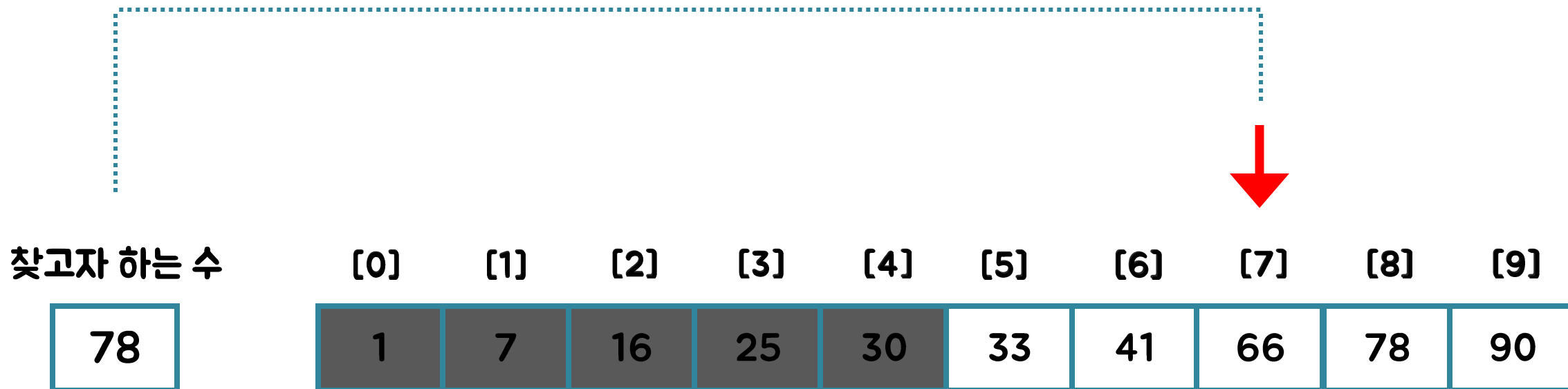


비교



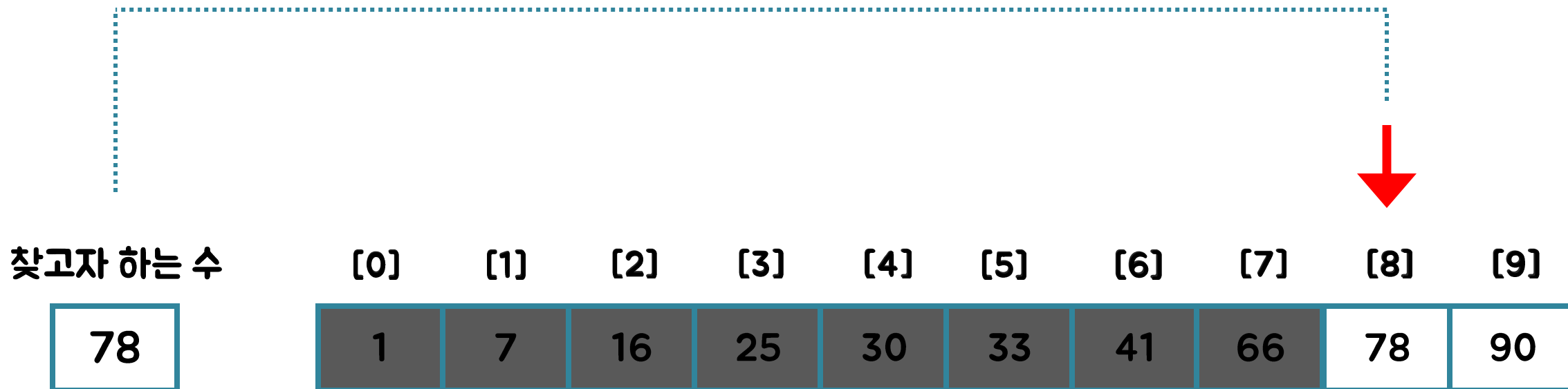
lowIndex = 0, highIndex = 9

비교



lowIndex = 5, highIndex = 9

비교



lowIndex = 8, highIndex = 9

- dictionary는 단어 그대로 해석하면 사전이라는 뜻
- “people”이라는 단어는 “사람”, “baseball”이라는 단어는 “야구”에 부합되
듯이 dictionary는 Key와 Value를 한쌍으로 갖는 자료형
- 딕셔너리 타입은 immutable한 key와 mutable한 value로 맵핑되어
있는 순서가 없는 집합

딕셔너리명 = {Key : Value, Key : Value, ... }

※ Key에는 변하지 않는 값을 사용하고,
Value에는 변하는 값과 변하지 않는 값 모두 사용할 수 있다.

```
a = {}
```

```
b = { "name" : "YH" }
```

```
c = { 1 : 5, 2 : 3 }
```

```
dic1 = {"name": "YH", "age": 20, "mbti": "ENFJ"}  
print(dic1)
```

```
{'name': 'YH', 'age': 20, 'mbti': 'ENFJ'}
```

key	value
name	YH
age	20
mbti	ENFJ

딕셔너리 정보

딕셔너리명[key] = value

```
dic1 = {"name": "YH", "age": 20, "mbti": "ENFJ"}  
print(dic1)
```

```
{'name': 'YH', 'age': 20, 'mbti': 'ENFJ'}
```

```
dic1["birth"] = "08/01"
```

```
print(dic1)
```

```
{'name': 'YH', 'age': 20, 'mbti': 'ENFJ', 'birth': '08/01'}
```

key value	
name	YH
age	20
mbti	ENFJ
birth	08/01

딕셔너리 정보

1. 변수 `dic_song`를 다음과 같이 만드시오.

key	value
노래제목	HOME SWEET HOME

2. 딕셔너리 추가를 통해 다음과 같이 정보를 저장 시키시오.

key	value
노래제목	HOME SWEET HOME
가수	G-DRAGON
날짜	2025.02.06

```
print(dic_song)
```

```
{'노래제목': 'HOME SWEET HOME', '가수': 'G-DRAGON', '날짜': '2025.02.06'}
```


del 딕셔너리명[key]

```
dic2 = {"name": "YH", "age": 20, "mbti": "ENFJ"}  
del dic2["age"]  
print(dic2)
```

```
{'name': 'YH', 'mbti': 'ENFJ'}
```

key	value
name	YH
mbti	ENFJ

딕셔너리 정보

딕셔너리명[Key]

```
print(dic2)
```

```
{'name': 'YH', 'mbti': 'ENFJ'}
```

```
print(dic2["name"])
```

```
YH
```

```
print(dic2["mbti"])
```

```
ENFJ
```

key	value
name	YH
mbti	ENFJ

딕셔너리 정보

딕셔너리명.get(Key)

```
print(dic2)
```

```
{'name': 'YH', 'mbti': 'ENFJ'}
```

```
print(dic2.get("name"))
```

```
YH
```

```
print(dic2.get("mbti"))
```

```
ENFJ
```

key	value
name	YH
mbti	ENFJ

딕셔너리 정보

딕셔너리명[key] VS 딕셔너리명.get(Key)

```
dic2["성별"]
```

--

KeyError

t)

<ipython-input-31-e10b9ca4

----> 1 dic2["성별"]

KeyError: '성별'

```
temp = dic2.get("성별")  
print(temp)
```

None

False

딕셔너리명.keys()

```
dic3 = {"name": "YH", "age": 20, "mbti": "ENFJ"}  
print(dic3.keys())
```

```
dict_keys(['name', 'age', 'mbti'])
```

```
print(type(dic3.keys()))  
list3 = list(dic3.keys())  
print(list3)  
print(type(list3))
```

```
<class 'dict_keys'>  
['name', 'age', 'mbti']  
<class 'list'>
```

key	value
name	YH
age	20
mbti	ENFJ

딕셔너리 정보

딕셔너리명.values()

```
dic3 = {"name": "YH", "age": 20, "mbti": "ENFJ"}  
print(dic3.values())
```

```
dict_values(['YH', 20, 'ENFJ'])
```

key	value
name	YH
age	20
mbti	ENFJ

딕셔너리 정보

딕셔너리 for문 활용

```
for key in dic3.keys():  
    print(key)
```

name
age
mbti

```
for value in dic3.values():  
    print(value)
```

YH
20
ENFJ

```
for key, value in dic3.items():  
    print(key, value)
```

name YH
age 20
mbti ENFJ

key in 딕셔너리명

- in은 딕셔너리의 키에 한에서 동작한다.

```
print("name" in dic3)
```

True

```
print("성별" in dic3)
```

False

```
print("YH" in dic3)
```

False

딕셔너리명.clear()

```
print(dic3)
```

```
{'name': 'YH', 'age': 20, 'mbti': 'ENFJ'}
```

```
dic3.clear()
```

```
print(dic3)
```

```
{}
```

여러 학생의 성적점수가 Dictionary로 아래와 같이 구성되어있다.
과목 별 합을 구하여 새로운 Dictionary로 구성하시오.

```
dic_score = {"나예호" : {"수학" : 55, "영어" : 23, "국어" : 41},  
             "공유" : {"영어" : 67, "국어" : 87, "수학" : 67},  
             "수지" : {"수학" : 99, "국어" : 75, "영어" : 80}}
```



```
{ '수학' : 221, '국어' : 203, '영어' : 170 }
```

내장함수

외장함수

사용자 정의
함수

```
pr random.randint(1,10)
```

NameError

<ipython-input-7-60206fa06a2a> in <modu
----> 1 random.randint(1,10)

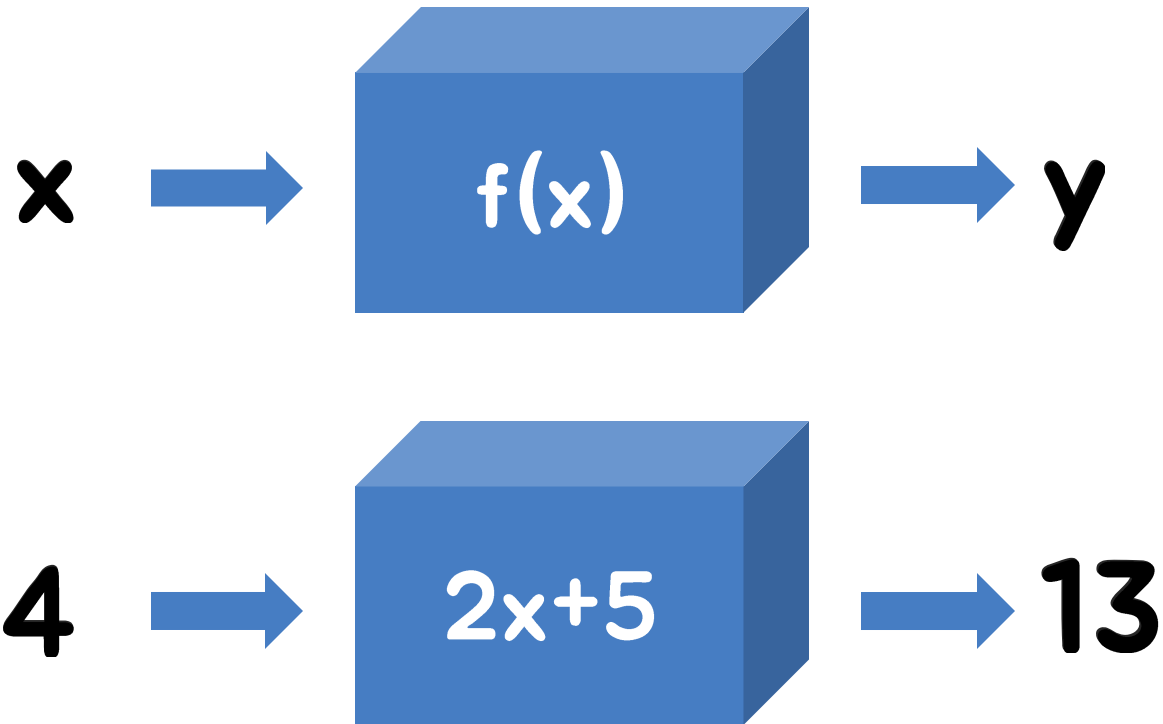
NameError: name 'random' is not defined

```
import random
```

```
random.randint(1,10)
```

4





함수란?

- 하나의 특별한 목적의 작업을 수행하기 위해 독립적으로 설계된 코드의 집합

함수를 사용하는 이유

- 반복적인 프로그래밍을 피할 수 있다.
- 모듈화로 인해 전체적인 코드의 가독성이 좋아진다.
- 프로그램에 문제가 발생하거나 기능의 변경이 필요할 때에도 손쉽게 유지보수가 가능하다.

데이터 전처리로 텍스트의 “ㅋ”을 모두 제거하고 싶을 때

- 데이터는 review_list1, review_list2에 대입되어 있음

```
review_list1 = ["월 스미스가 하드캐리ㅋㅋㅋㅋㅋㅋ",  
                "자스민 너무 멋지고 팬 엄청나게 생길듯"]  
review_list2 = ["색감도 노래도 너무 화려하고 재밌었어요ㅋㅋ",  
                "오늘부터 디즈니 팬입니다ㅋㅋ",  
                "디즈니 의 새로운해석도 놀랍고. 월스미스도 신의한수!"]
```

```
s = "월 스미스가 하드캐리ㅋㅋㅋㅋㅋㅋ"
s.
```

'월 스미스가 하드캐리'

함수	설명
<code>count('문자')</code>	문자열에 포함된 문자 개수 세기
<code>find('문자')</code>	문자 위치 알려주기
<code>index('문자')</code>	문자 위치 알려주기
<code>join('문자')</code>	각각의 문자 사이에 '문자' 삽입하기
<code>upper()</code>	소문자를 대문자로 바꾸기
<code>lower()</code>	대문자를 소문자로 바꾸기
<code>lstrip()</code>	왼쪽 공백 지우기
<code>rstrip()</code>	오른쪽 공백 지우기
<code>strip()</code>	양쪽 공백 지우기
<code>replace('문자1', '문자2')</code>	문자열1을 문자열 2로 바꾸기
<code>split()</code>	문자열 나누기

define : 정의를 내리다

def

함수명(매개 변수):

(colon, 콜론)

실행문장

return 반환 변수

들여쓰기 (Tab, Space*4)

함수호출

- 함수명(인수1, 인수2)

```
def 함수명(입력인수):  
    실행문장  
    return 반환변수
```

```
def number_sum(num1, num2):  
    result = num1 + num2  
    return result
```

```
number_sum(3, 10)
```

13

```
number = number_sum(3, 10)  
number
```

13

두 수를 입력 받아서 뺀 결과를 return하는 함수를 정의하시오.

```
num1 = int(input("첫 번째 정수 입력 >> "))  
num2 = int(input("두 번째 정수 입력 >> "))  
result = number_minus(num1, num2)  
result
```

```
첫 번째 정수 입력 >> 10  
두 번째 정수 입력 >> 3
```

```
7
```

문자열을 입력 받아 'ㅋ'을 제거하고 return하는 함수를 정의하시오.

```
s = input("문자열 입력 >> ")  
result = s_replace(s)  
result
```

문자열 입력 >> ㅋ을 모두 지워주세요ㅋㅋㅋㅋㅋㅋ

'을 모두 지워주세요'

두 수를 입력 받아서 원하는 연산을 수행하는 함수를 정의하시오.

```
num1 = int(input("첫 번째 정수 입력 >> "))  
num2 = int(input("두 번째 정수 입력 >> "))  
op = input("연산자 입력(+, -) >> ")  
result = cal(num1, num2, op)  
print("결과 : {}".format(result))
```

```
첫 번째 정수 입력 >> 5  
두 번째 정수 입력 >> 3  
연산자 입력(+, -) >> +  
결과 : 8
```

```
첫 번째 정수 입력 >> 5  
두 번째 정수 입력 >> 3  
연산자 입력(+, -) >> -  
결과 : 2
```

2개의 정수를 받아 2개의 숫자 중 10에 더 가까운 수를
반환하는 함수 close10을 만들어보세요
(만약 두 숫자 모두 10과의 차이가 같다면 첫 번째 수를 반환)

```
num1 = int(input("첫 번째 정수 입력 : "))  
num2 = int(input("두 번째 정수 입력 : "))  
result = # close10 함수 호출  
  
print("10에 가까운 수 : %d"%result)
```

첫 번째 정수 입력 : 2
두 번째 정수 입력 : 7
10에 가까운 수 : 7

첫 번째 정수 입력 : 8
두 번째 정수 입력 : 5
10에 가까운 수 : 8

첫 번째 정수 입력 : 2
두 번째 정수 입력 : 13
10에 가까운 수 : 13

독스트링(docstring)

- 함수의 설명을 작성 (Shift + <Tab>)

```
def cal(num1, num2, op):  
    """덧셈과 뺄셈을 계산하는 함수"""  
    if op=='+':  
        return num1+num2  
    else:  
        return num1-num2
```

cal()

Signature: cal(num1, num2, op)

Docstring: 덧셈과 뺄셈을 계산하는 함수

약수를 구하는 함수를 정의하시오.

```
divisor(10)
```

```
1 2 5 10
```

```
divisor(32)
```

```
1 2 4 8 16 32
```

```
divisor(100)
```

```
1 2 4 5 10 20 25 50 100
```


가변 매개변수(variable parameters)

- 함수 호출 시 몇 개의 인수가 전달될지 알 수 없다면, 사용자가 직접 매개변수의 개수를 정할 수 있도록 선언

```
def 함수명(*매개변수):  
    실행문장  
    return 반환변수
```

가변 매개변수(variable parameters)

- 전달된 모든 인수는 튜플(tuple)의 형태로 저장

```
def add(*args):  
    print(args)
```

args → arguments

```
add(1,2,3)
```

```
(1, 2, 3)
```

```
add(1,2,3,4,5,6,7,8,9,10)
```

```
(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

가변 매개변수를 활용해 모든 숫자를 더해서 반환하는 함수를 작성하십시오.

```
def add(*args):  
    ?
```

```
add(1,2,3)
```

6

```
add(1,2,3,4,5,6,7,8,9,10)
```

55

함수의 결과값은 언제나 하나이다.

```
def add_sub(num1, num2):  
    return num1+num2, num1-num2
```

```
add_sub(10, 7)
```

함수의 결과값은 언제나 하나이다.

```
def add_sub(num1, num2):  
    return num1+num2, num1-num2
```

```
result_add, result_sub = add_sub(10,7)  
print(result_add)  
print(result_sub)
```

```
17  
3
```

기본값 설정(default parameters)

```
def power_of_N(num, power=2):  
    return num**power
```

```
power_of_N(|)
```

Signature: power_of_N(num, power=2)

기본값 설정(default parameters)

```
def power_of_N(num, power=2):  
    return num**power
```

```
power_of_N(3)
```

9

```
power_of_N(3,3)
```

27

```
power_of_N(3,power=5)
```

243

인수로 넘겨받은 숫자를 더하여 반환하는 함수를 작성하시오.

```
1 result = sum_many(1,2,3,4,5,6,7,8,9,10)
2 print(result)
```

55

```
1 result = sum_many(3,6,9,10)
2 print(result)
```

28

```
1 result = sum_many(1,1,2,2)
2 print(result)
```

6

'sum'을 넘겼을 때는 숫자의 합을 'mul'을 넘겼을 때는 숫자의 곱을 반환하는

sum_mul()메소드를 작성하시오.

```
1 result = sum_mul('sum', 1, 2, 3, 4)
2 print(result)
```

10

```
1 result = sum_mul('mul', 1, 2, 3, 4)
2 print(result)
```

24

함수의 결과값은 언제나 하나이다.

```
1 def sum_and_mul(a,b):  
2     return a+b, a*b
```

```
1 result = sum_and_mul(3,4)
```

```
1 print(result)
```

```
(7, 12)
```

하나의 튜플 값을 두 개의 결과 값처럼 받고 싶다면?

```
1 def sum_and_mul(a,b):  
2     return a+b, a*b
```

```
1 resultSum,resultMul = sum_and_mul(3, 4)
```

```
1 print(resultSum)  
2 print(resultMul)
```

7

12

가변 매개변수(variable parameters)

- 딕셔너리 형태로 함수 내부에서 처리하고 싶을 때

```
def 함수명(**매개변수):  
    실행문장  
    return 반환변수
```

가변 매개변수(variable parameters)

- 딕셔너리 형태로 함수 내부에서 처리하고 싶을 때

```
def print_map(**kwargs):  
    print(kwargs)  
    for key,value in kwargs.items():  
        print(key, "/", value)
```

kwargs → keyword arguments

```
print_map(하나=1)
```

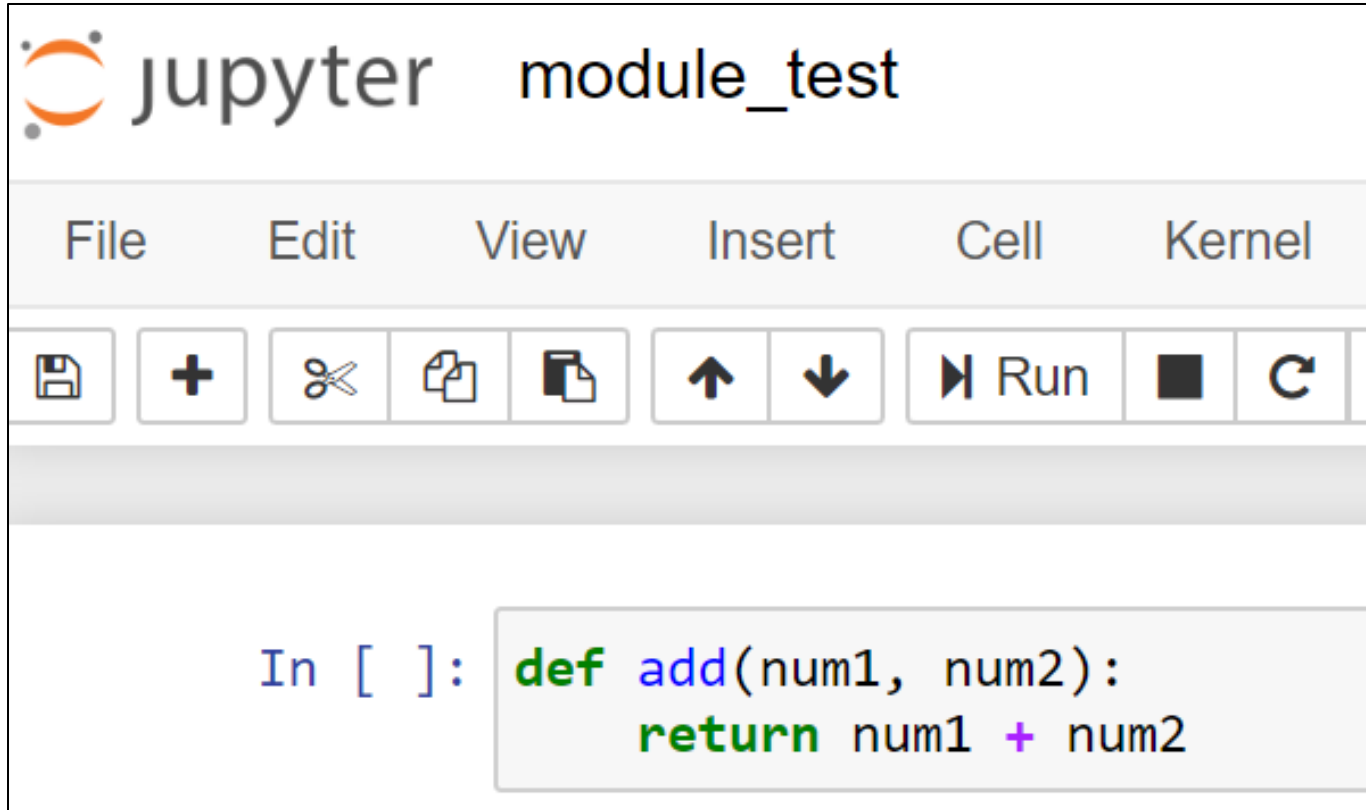
```
{'하나': 1}  
하나 / 1
```

```
print_map(one=1, two=2)
```

```
{'one': 1, 'two': 2}  
one / 1  
two / 2
```

- 변수나 함수 또는 클래스를 모아 놓은 **파일**
- 다른 파이썬 프로그램에서 불러와 사용할 수 있게끔 만든 파이썬 파일
- 모듈은 다른 사람이 이미 만들어 놓은 모듈을 사용할 수도 있고 직접 만들어서 사용할 수도 있다.
- 파이썬에서 사용할 수 있는 모듈은 **확장자가 .py파일** 이다.

- 모듈의 의미는 구성단위, 라이브러리는 도서관이라는 뜻을 가지고있다.
- 하지만, 개발에서는 모듈과 라이브러리는 **동일한 의미**라고 생각하면 된다.



The image shows a Jupyter Notebook interface. At the top, the Jupyter logo and the text "module_test" are visible. Below this is a menu bar with options: File, Edit, View, Insert, Cell, and Kernel. Under the menu bar is a toolbar with icons for saving, adding new cells, cutting, copying, pasting, moving cells up and down, running the cell, and refreshing. The main area of the notebook contains a single code cell. The prompt "In []:" is followed by a Python function definition:

```
def add(num1, num2):  
    return num1 + num2
```


- File → Download as → Python (.py) → 같은 경로로 옮기기

1. Click the **File** menu.

2. Click **Download as**.

3. Select **Python (.py)**.

4. The file **module_test.py** is saved in the same directory as the notebook.

import + 모듈이름

```
import module_test
```

```
module_test.add(10, 20)
```

30

from 모듈이름 import + 함수(or클래스)

```
from module_test import add
```

```
add(20, 50)
```

70