

# 외모 데이터 **체크!**

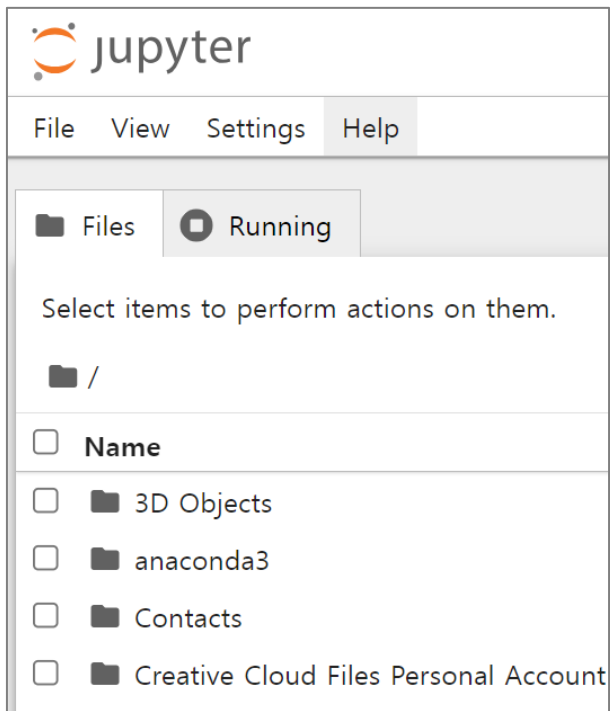
WEEK2 Python 기초 리뷰



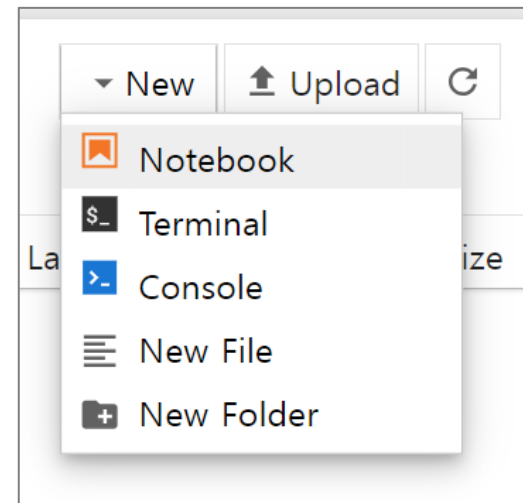
# 주피터 노트북 파일 만들기

외모  
데이터 **체크!**

- 적절한 폴더 진입하기



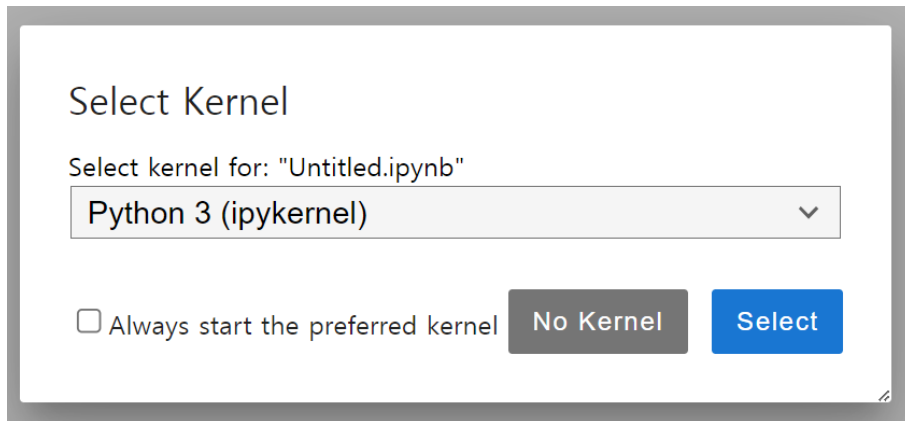
- 오른쪽 'New' > 'Notebook'



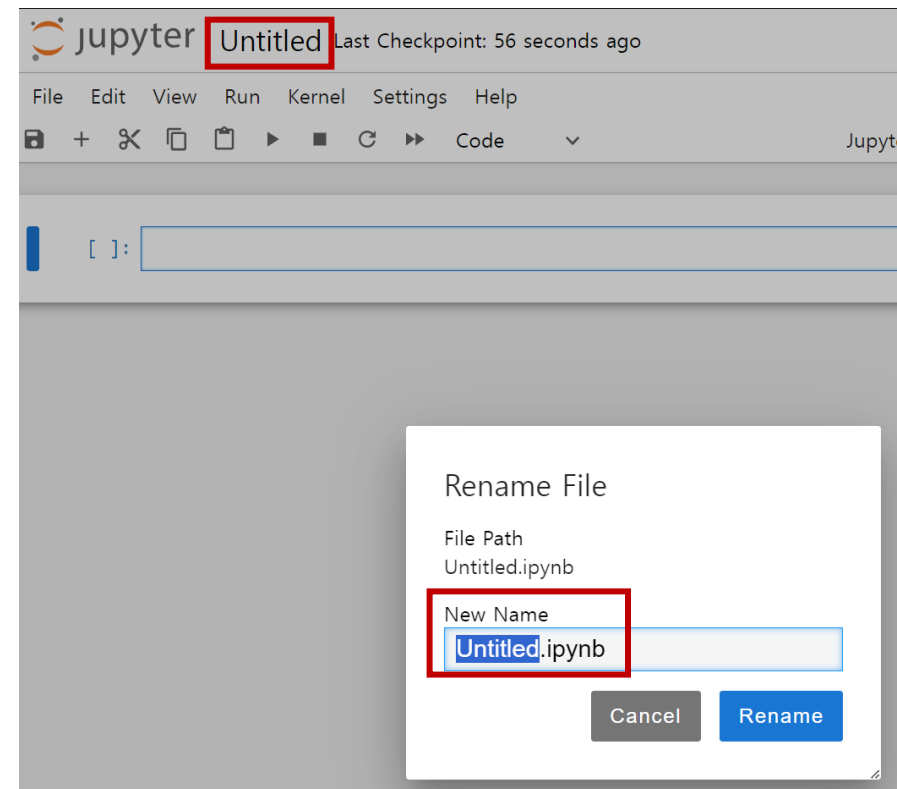
※ Notebook 대신  
ipykernel 등 다른 이름일 수도 있어요.

# 주피터 노트북 파일 만들기

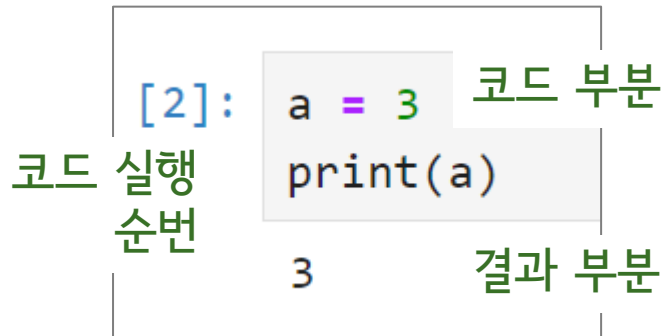
- Python3 선택  
(아래 창 안 뜰 경우 생략)



- 제목 클릭하여 파일명 바꾸기



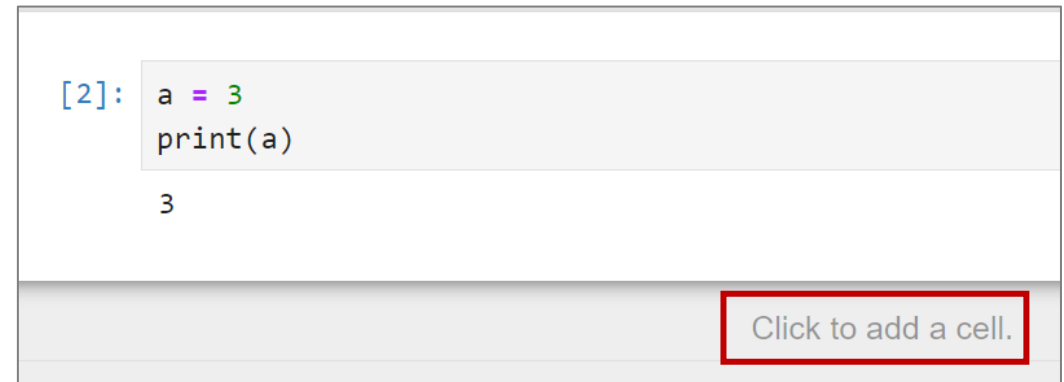
- 칸에 코드 기입 후  
Ctrl+Enter(Cmd+Enter)하면  
코드가 실행돼요.



A diagram of a Jupyter Notebook cell. The cell contains the code `[2]: a = 3` and `print(a)`. The code is highlighted in a light blue box, and the output `3` is shown below it. Labels with arrows point to different parts: '코드 부분' (Code part) points to the code, '결과 부분' (Result part) points to the output, '코드 실행' (Code execution) points to the prompt `[2]:`, and '순번' (Number) points to the prompt `[2]:`.

```
[2]: a = 3
     print(a)
3
```

- 하단 버튼을 누르거나  
칸 선택 없이 B를 누르면  
아래에 칸이 추가돼요.



A screenshot of a Jupyter Notebook cell. The cell contains the code `[2]: a = 3` and `print(a)`, with the output `3` shown below. At the bottom of the cell, there is a button labeled 'Click to add a cell.' which is highlighted with a red box.

```
[2]: a = 3
     print(a)
3
```

Click to add a cell.

※ Shift+Enter는 현재 칸을 실행하면서 아래 칸으로 이동해요.

## ■ 변수

- 변수명 = 값의 형태로 값을 변수에 저장할 수 있어요.

## ■ 자료형

- 값의 형태는 정수(int), 실수(float), 문자열(str), 불린(bool)이 있어요.
- type(값)으로 값의 자료형을 확인할 수 있어요.

# 뒤의 글은 주석이라 부르며, 코드로 실행되지 않아요.

```
# S02-01
a = 1
b = 1.5
print(a, b)

c = 'check' # 문자열은 '~' 또는 "~"로 표현
d = True   # 불린은 True 또는 False
print(c, d)

print(type(a), type(b))
print(type(c), type(d))
```

코드

```
1 1.5
check True
<class 'int'> <class 'float'>
<class 'str'> <class 'bool'>
```

출력값

- 캐스트 (형 변환)
  - `int(~)`, `float(~)`,  
`str(~)`, `bool(~)`을 통해  
자료형을 바꿀 수 있어요.
  - 수 → 불린의 경우,  
0이면 `False`, 그 외는 `True`가 돼요.
  - 불린 → 수의 경우,  
`False`는 0, `True`는 1이 돼요.

```
# S02-02
a = 5.8
print(int(a), str(a))

a = bool(a)
print(a)

a = int(a)
print(a)
```

5 5.8  
True  
1

## ■ 산술 연산

- 사칙연산 순서 적용됨
- +, -, \* 곱하기, / 나누기, // 몫, % 나머지

## ■ 비교 연산

- == 같다, != 다르다, <, >, <=, >=

## ■ 논리 연산

- 불린끼리의 연산
- and 둘 다 True여야 True,  
or 하나라도 True이면 True, not 반대로 바꾸기

```
# S02-03
```

```
a = 5
```

```
b = 3 - a * 2
```

```
print(b)
```

```
c = (a == b)
```

```
d = (a >= b)
```

```
print(c, d)
```

```
print(c and d, c or d, not c)
```

```
-7
```

```
False True
```

```
False True True
```

## ■ 복합연산자

- 연산에 사용한 변수에 연산 결과를 덮어씌울 때 사용합니다.
- `a = a + 2`와 `a += 2`는 같은 결과를 보여요.
- 모든 산술/비교 연산에 적용 가능합니다.

```
# S02-04
a = 5
a += 1
print(a)

a *= 2
print(a)

a %= 5
print(a)
```

```
6
12
2
```



## ■ 문자열 연산

- 문자열 + 문자열을 통해 문자열을 이을 수 있어요.
- 문자열 \* 자연수를 통해 문자열을 반복할 수 있어요.
- len(문자열)을 통해 문자열의 길이를 구할 수 있어요.

```
# S02-05
a = 'data'
b = 'check'
print(a + b)
print(b * 3)

c = '35'
d = '35' + '3'
print(c, d)
```

```
datacheck
checkcheckcheck
35 353
```

## 체크! EXERCISE 02-01

아래 코드의 출력은 무엇일까요?

```
a = 3
b = 5
a += b
b *= a
print(a, b)

a %= b
b **= 2
print(a, b)
```

## 체크! EXERCISE 02-02

아래 코드의 출력은 무엇일까요?

```
a = 3
a = str(a)
a += '2'
print(a)

a = float(a)
print(a)
```

## ■ 리스트

- 리스트를 통해 여러 값을 하나로 묶어 표현할 수 있어요.
- `[값, 값, ...]`과 같이 표현해요.
- 리스트의 값 순서를 인덱스라 하고, 0부터 시작해요.

## ■ 인덱싱

- `리스트[인덱스]`로 값을 선택해요.

```
# S02-06
list1 = ['a', 'b', 'c', 1, 2]
print(list1, list1[0], list1[3])
print(list1[-1]) # 뒤에서 첫번째
```

```
list1[0] = list1[4] + 3
print(list1[0])
```

```
['a', 'b', 'c', 1, 2] a 1
2
5
```

list1

'a'	'b'	'c'	1	2
-----	-----	-----	---	---

인덱스    0        1        2        3        4

- 슬라이싱
  - `리스트[시작:끝]`으로  
시작 인덱스부터 끝 인덱스 전까지를  
선택할 수 있어요.
  - 시작 또는 끝을 생략하면  
처음과 마지막을 의미해요.

```
# S02-07
list1 = ['a', 'b', 'c', 1, 2]
print(list1[1:4]) # 4는 미포함
print(list1[:3]) # 인덱스 0~2
print(list1[2:]) # 인덱스 2~4
```

```
['b', 'c', 1]
['a', 'b', 'c']
['c', 1, 2]
```

## ■ 리스트 메서드

- `리스트.append(값)`

리스트 끝에 값 추가하기

- `리스트.extend([값, 값, ...])`

리스트 끝에 값들 추가하기

- `리스트.pop(인덱스)`

리스트에서 주어진 인덱스 순번 값 삭제  
(인덱스 생략 시 마지막 값 삭제)

- `리스트.remove(값)`

리스트에서 주어진 값 삭제 (여러 개면 첫 번째 삭제)

- `len(리스트)`

리스트 길이

```
# S02-08
the_list = ['a', 1, 2]

the_list.append(3)
print(the_list, len(list))
the_list.extend([10, 11])
print(the_list)
the_list.pop()
print(the_list)
the_list.remove(2)
print(the_list)
```

```
['a', 1, 2, 3] 4
['a', 1, 2, 3, 10, 11]
['a', 1, 2, 3, 10]
['a', 1, 3, 10]
```

## ■ 리스트 메서드

- `리스트.sort()`

리스트 정렬하기 [오름차순]

- `sorted(리스트)`

리스트를 정렬시켜 새로운 리스트 만들기  
(주어진 리스트는 그대로 유지됨)

- `리스트.reverse()`

리스트 순서 뒤집기

```
# S02-09
the_list = [53, 24, 74, 16, 31]

the_list.sort()
print(the_list)

the_list.reverse()
print(the_list)

sorted_list = sorted(the_list)
print(the_list, sorted_list)
```

```
[16, 24, 31, 53, 74]
[74, 53, 31, 24, 16]
[74, 53, 31, 24, 16] [16, 24, 31, 53, 74]
```

## ▪ in

- 값 in 리스트

리스트 안에 값이 있는지 여부 (True/False)

## ▪ range

- range(시작, 끝, 간격)

시작 자연수부터 끝 직전 자연수까지  
간격만큼 연속하여 반복

- range(끝)

0부터 끝 직전 자연수까지 1 간격으로 반복

- range가 리스트는 아니지만,  
list(range(~))로 리스트화할 수 있어요.

```
# S02-10
print(range(5))
print(range(2, 10, 2))
```

```
r3 = list(range(3))
print(r3)
print(2 in r3)
print(3 in r3)
```

```
range(0, 5)
range(2, 10, 2)
[0, 1, 2]
True
False
```

## ■ 리스트 통계

- `sum(리스트)`

리스트 내부 값들의 합

- `max(리스트)`

리스트 내부 값들 중 최댓값

- `min(리스트)`

리스트 내부 값들 중 최솟값

- `round(값, 소수점자리수)`

값을 반올림하여 소수점자리수까지 나타내기

```
# S02-11
num_data = [12, 34, 32, 5, 53]
print(sum(num_data))
print(max(num_data), min(num_data))

# 평균
mean = sum(num_data) / len(num_data)
mean = round(mean, 2)
print(mean)
```

```
136
53 5
27.2
```

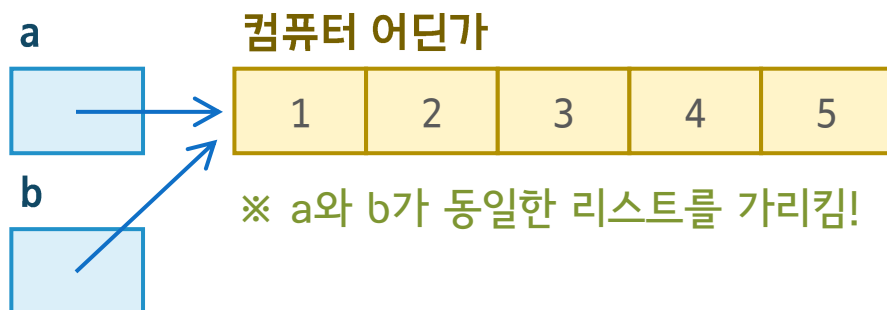


## ■ 리스트 변수

- 리스트 변수는 다음과 같이  
리스트로 향하는 화살표로 저장돼요.



- 리스트 변수를 다른 변수에 저장하면,  
변수에 저장된 화살표가 복사돼요.



```
# S02-12
```

```
a = [1, 2, 3, 4, 5] # a에 화살표 저장됨
```

```
b = a # a에 저장된 화살표가 b에 복사됨
```

```
print(a, b)
```

```
b[3] = 10
```

```
print(a, b)
```

```
a.append(0)
```

```
print(a, b)
```

```
c = a[:] # 슬라이싱은 리스트를 새로 만듦
```

```
c[0] = -1
```

```
print(a, c)
```

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

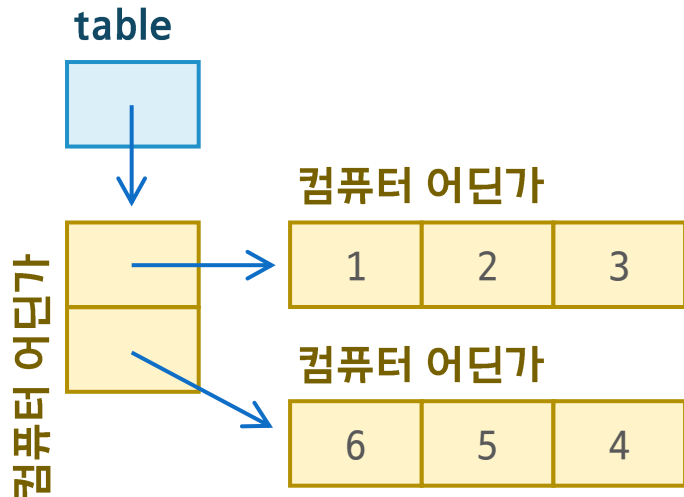
```
[1, 2, 3, 10, 5] [1, 2, 3, 10, 5]
```

```
[1, 2, 3, 10, 5, 0] [1, 2, 3, 10, 5, 0]
```

```
[1, 2, 3, 10, 5, 0] [-1, 2, 3, 10, 5, 0]
```

## ■ 2차원 리스트

- 리스트 안에 리스트를 넣어 2차원 표와 유사한 형태를 만들 수 있어요.



```
# S02-13
table = [[1, 2, 3], [6, 5, 4]]

print(table[1][2], table[0][-1])
print(table[1])
print(len(table), len(table[0]))

table[1].sort()
print(table[1])
```

```
4 3
[6, 5, 4]
2 3
[4, 5, 6]
```

- 문자열 인덱싱/슬라이싱
  - 문자열도 리스트와 유사하게 인덱싱/슬라이싱이 가능해요.
  - 똑같이 0번째 부터 시작함에 유의!

```
# S02-14
string = 'data check'
print(string[2], string[:6])
print()

string = 'data\ncheck' # \n은 줄바꿈(엔터)
print(string)
print(string[3:6])
```

```
t data c
```

```
data
check
a
c
```

## ■ 딕셔너리

- 리스트와 유사하되,  
인덱스가 수가 아니라 문자열인  
자료형을 딕셔너리라 한다.
- 문자열인 인덱스를 **key**라 하고,  
그 값을 **value**라 한다.
- `score.keys()` 키 정보를 담은 것  
`score.values()` 밸류 정보를 담은 것

```
# S02-15
score = {'Alice':95, 'Jackson':90, 'Kate':85}
print(score['Jackson'])
```

```
key = list(score.keys())
value = list(score.values())
print(key[2], value[0])
```

```
90
Kate 95
```

- 딕셔너리 안 리스트
  - 딕셔너리 안에  
value를 리스트로 두어  
표 형태를 구현할 수 있다.

```
# S02-16
# \는 코드 줄바꿈 표시
student = {'name':['Alice', 'Jackson', 'Kate'], \
           'score':[95, 90, 85]}
print(student['name'][2])
print(sum(student['score']))
```

```
Kate
270
```

## 체크! EXERCISE 02-03

아래 코드의 출력은 무엇일까요?

```
contents = [1, 2, 3]
print(contents[1], contents[-1])

contents.append(5)
contents.extend([4, 5])
print(contents[3:])

contents.pop()
print(contents[-1])
```

## 체크! EXERCISE 02-04

아래를 수행하는 코드를 작성해봅시다.

- range를 이용하여,  
두 자리 홀수를 담은 리스트를 만듭니다.
- 만든 리스트를 활용하여,  
두 자리 홀수의 개수를 구해 출력합니다.
- 만든 리스트를 활용하여,  
두 자리 홀수의 평균을 구해 출력합니다.

## ▪ if, if-else

- if를 통해 **특정 조건을 만족할 때만** 정해진 코드를 실행할 수 있어요.
- else를 통해 **특정 조건이 아닐 때만** 정해진 코드를 실행할 수 있어요.

```
if ...condition...:  
    ...code_if_true...
```

들여쓰기는 4칸!  
들여쓰기가 유지될 때까지  
if 등이 영향을 줌

```
if ...condition...:  
    ...code_if_true...  
else:  
    ...code_if_false...
```

```
# S02-17  
a = [1, 2, 3]  
if a[2] != 5:  
    print('first')  
else:  
    print('second')  
  
if 2 in a:  
    print('third')
```

```
first  
third
```

## ▪ if-elif-else

- elif를 통해 이전 조건이 아닐 때 또 다른 조건을 줄 수 있어요.
- elif를 여러 개 둘 수도 있어요.

```
if ...condition1...:
    ...code_if_condition1...
elif ...condition2...:
    ...code_if_condition2...
else:
    ...code_if_false...
```

```
# S02-18
a = 52
if a < 10 or a % 10 == 3:
    print('first')
elif str(a)[1] == 2:
    print('second')
else:
    print('third')
```

third



## ■ for

- for를 통해  
반복변수의 값을 바꿔가며  
주어진 코드를 반복할 수 있어요.

```
for ...var... in ...list...:  
    ...repeat_code...
```

```
# S02-19  
alphabet = ['j', 'd', 'o']  
for a in alphabet:  
    print(a, end = ' ') # 줄바꿈 없도록 설정
```

jdo

```
# S02-20  
points = [2, 2, 2, 4, 4]  
for i in range(len(points)):  
    print(points[i])
```

22244

## ▪ for과 range

- range를 적절히 활용하여  
각종 연산을 수행할 수 있어요.

```
# S02-21 : 100 미만의 자연수의 합  
s = 0  
for n in range(1, 100):  
    s += n  
print(s)
```

4950

```
# S02-22 : 93의 약수의 개수  
count = 0  
for n in range(1, 94):  
    if 93 % n == 0:  
        count += 1  
print(count)
```

4

## ▪ break, continue

- **break**가 나오면  
가장 가까운 반복문이 중단돼요.

```
# S02-23
points = [2, 2, 2, 4, 4]
for i in range(len(points)):
    if points[i] > 3:
        break
print(i)
```

3

- **continue**가 나오면  
현재 반복의 잔여 부분은 생략하고  
다음 반복이 시작돼요.

```
# S02-24
divisor_20 = []
for n in range(1, 20):
    if 20 % n != 0:
        continue
    divisor_20.append(n)
print(divisor_20[3])
```

5

## 체크! EXERCISE 02-05

아래를 수행하는 코드를 작성해봅시다.

- a에 ['45', '-34.5', '24.3', '-4']를 저장한다.
- for를 활용하여,  
a의 각 문자열을 수로 바꾸고,  
음수인 경우 -1을 곱한다.
- a를 출력한다.

## 체크! EXERCISE 02-06

아래를 수행하는 코드를 작성해봅시다.

- for를 활용하여,  
288와 432의 최대공약수를 찾아 출력한다.

## ■ 함수

- 코드들을 묶어 하나의 기능을 하는 함수로 정의할 수 있어요.
- 함수의 **매개변수** parameter에 함수 호출에서 전달된 **인자** argument가 저장된 채로 내부 코드가 실행돼요.

함수 정의  
부분

```
def ...func_name...(...prm1..., ...prm2...):  
    ...code_in_func...  
    return ...return_value...
```

함수 호출  
부분

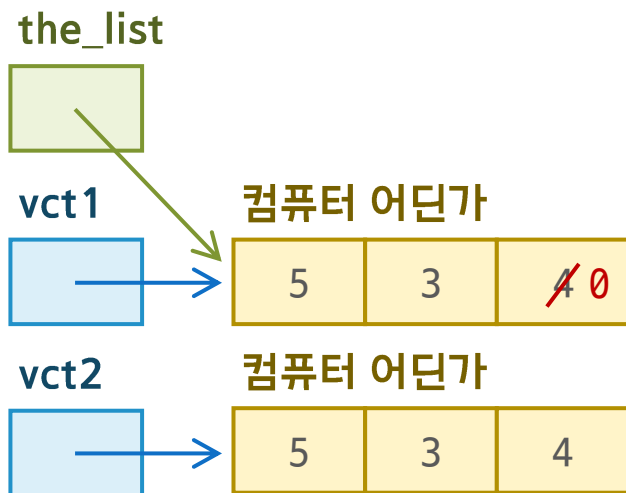
```
...func_name...(...arg1..., ...arg2...)
```

```
# S02-25  
def big(x, y):  
    if x > y:  
        return x  
    else:  
        return y  
  
n1 = 10  
n2 = 12  
print(big(n1, n2))
```

12

## ■ 리스트 파라미터

- 인자에 리스트 변수를 넣으면,  
리스트로 향하는 화살표가 전달돼요.



```
# S02-26
def zero(the_list):
    the_list[-1] = 0 # 마지막 값 0으로 바꾸기

vct1 = [5, 3, 4]
vct2 = vct1[:] # 리스트 복제

zero(vct1)
print(vct1)
print(vct2)
```

```
[5, 3, 0]
[5, 3, 4]
```

- 지역변수
  - 함수 안에서 만든 변수는  
함수 안에서만 작동하는  
지역변수예요.

```
# S02-27
def fct(x):
    a = x + 5
    a += 5
    return a

no = 16
no = fct(no)
no += a 오류!
print(no)
```

## 체크! EXERCISE 02-07

아래를 수행하는 코드를 작성해봅시다.

- 두 자연수를 매개변수로 받아  
두 값의 차이를 반환하는  
함수 `dif(num1, num2)`를 정의한다.
- `dif(10, 15)`를 출력한다.

## 체크! EXERCISE 02-08

아래를 수행하는 코드를 작성해봅시다.

- 수가 포함된 리스트를 파라미터로 받아  
리스트 내 값의 평균(소수 둘째자리 반올림)을  
반환하는 함수 `avg(the_list)`를 정의한다.  
예) `[20, 10, 7] → 12.33`
- 리스트 `[7, 10, 3, 13, 5, 17]`를  
변수 `li`에 저장한다.
- `avg(li)`를 출력한다.



## ■ 모듈

- 편리한 기능을 만들어둔 **모듈**을 우리가 사용하는 코드로 불러와 해당 기능을 활용할 수 있다.

```
import 모듈명  
모듈명.함수(~~)
```

```
import 모듈명 as 약어  
약어.함수(~~)
```

```
# 모듈 내 특정 함수만 불러오기  
from 모듈명 import 함수  
함수(~~)
```

```
# S02-28  
import random  
# randint: 1~100 정수 중 랜덤  
print(random.randint(1, 100))
```

```
# S02-29  
import random as rd  
print(rd.randint(1, 100))
```

```
# S02-30  
from random import randint  
print(randint(1, 100))
```

# 외모 데이터 **체크!**



WEEK2 Python 기초 리뷰

## Further Topics

Python: Class 문법

## References

-

제작자의 동의 없이 FORIF 외부에서 복제·활용할 수 없습니다.

## ■ f-string

- 문자열 안에 변수 또는 연산값을 그때그때 넣는 포매팅 방법이에요.
- 문자열 앞에 f를 붙여 표시하고, 안에 {변수 또는 연산}으로 포매팅할 수 있어요.

f'문자문자 {연산} 문자문자'

```
# S02-31
num1 = 10
num2 = 2
print(f'sum is {num1 + num2}')
```

```
string = 'hello'
print(f'the second character is {string[1]}')
```

```
sum is 12
the second character is e
```

## ■ 리스트 컴프리헨션

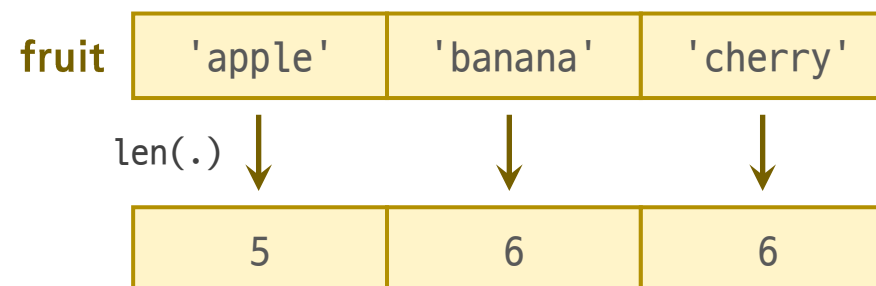
- 리스트 컴프리헨션은 반복을 통해 새로운 리스트를 만들 때 유용해요.  
복잡한 for를 한 줄로 쓸 수 있어요.

# for를 한 줄로 표현하여 새로운 리스트 만들기  
[연산내용 for 반복변수 in 리스트]

```
# S02-32
fruit = ['apple', 'banana', 'cherry']

length = [len(s) for s in fruit]
print(length)
```

[5, 6, 6]



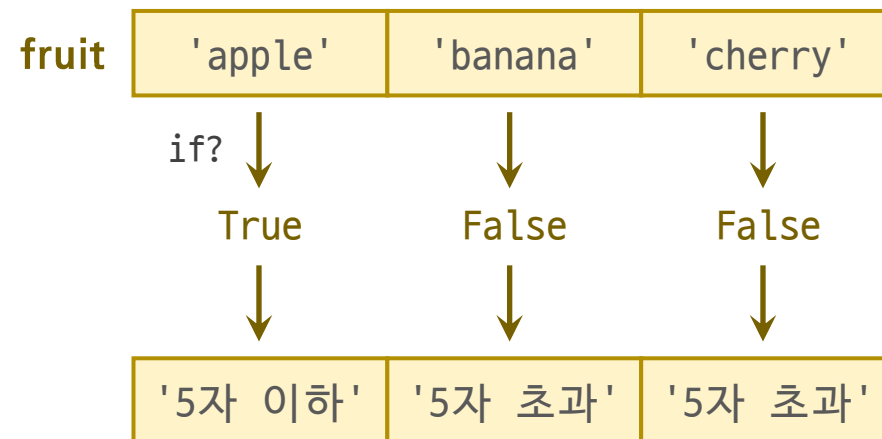
## ■ 리스트 컴프리헨션

# 반복할 때, 조건이 참일 때와 거짓일 때 다른 결과 보이기  
[조건이\_참일\_때\_연산 if 조건 else 조건이\_거짓일\_때\_연산 for 반복변수 in 리스트]

```
# S02-33
fruit = ['apple', 'banana', 'cherry']

length = ['5자 이하' if len(s) <= 5 \
          else '5자 초과' for s in fruit]
print(length)
```

```
['5자 이하', '5자 초과', '5자 초과']
```



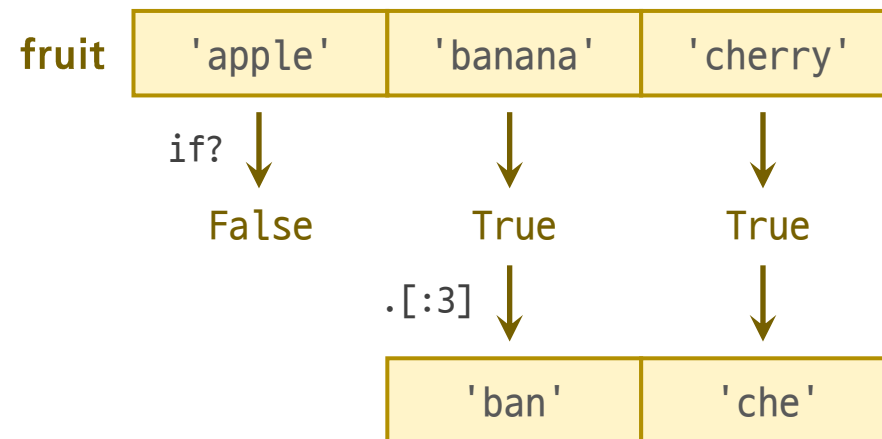
## ■ 리스트 컴프리헨션

# 반복할 때, 조건이 거짓인 경우는 아예 빼기  
[조건이\_참일\_때\_연산 for 반복변수 in 리스트 if 조건]

```
# S02-34
fruit = ['apple', 'banana', 'cherry']

over5 = [s[:3] for s in fruit \
        if len(s) > 5]
print(over5)

['ban', 'che']
```

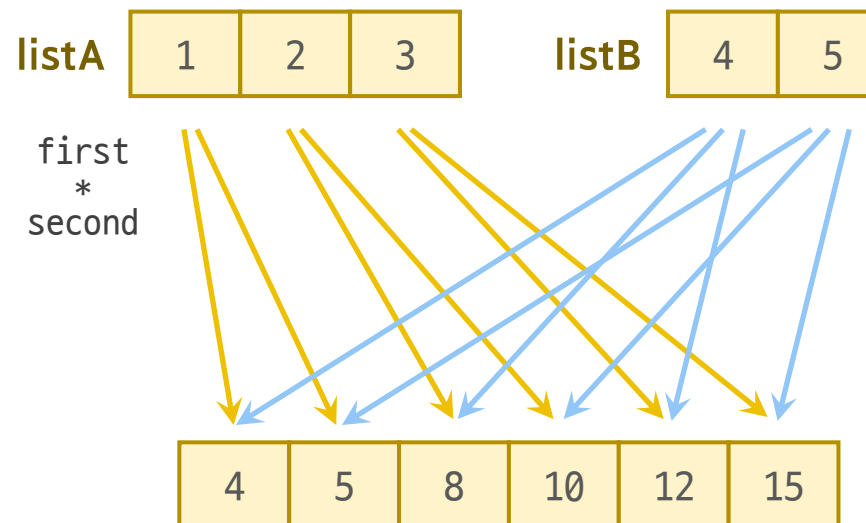


## ■ 리스트 컴프리헨션

# 이중 for를 한 줄로 표현하여 새로운 리스트 만들기  
[연산내용 for 반복변수1 in 리스트1 for 반복변수2 in 리스트2]

```
# S02-35
listA = [1, 2, 3]
listB = [4, 5]
listC = [first * second \
         for first in listA \
         for second in listB]
print(listC)
```

[4, 5, 8, 10, 12, 15]



## ■ 딕셔너리 컴프리헨션

- 리스트 컴프리헨션과 유사하나, **키:**를 통해 딕셔너리 형태를 만듭니다.

```
{키 : 밸류연산 for 반복변수 in 리스트}
```

```
{키 : 조건이_참일_때_밸류 if 조건 else 조건이_거짓일_때_밸류 for 반복변수 in 리스트}
```

```
{키 : 조건이_참일_때_밸류 for 반복변수 in 리스트 if 조건}
```

```
{키 : 밸류연산 for 반복변수1 in 리스트1 for 반복변수2 in 리스트2}
```

```
# S02-36
squared = {f'{n}의 제곱' : n ** 2 for n in range(5)}
print(squared)
```

```
{'0의 제곱': 0, '1의 제곱': 1, '2의 제곱': 4, '3의 제곱': 9, '4의 제곱': 16}
```



# 체크! LEARN MORE : 딕셔너리.items()

외모 체크!  
데이터

- 딕셔너리.items()
  - 딕셔너리의 키와 밸류를 모두 for의 반복변수로 넣고 싶으면 `딕셔너리.items()`를 활용해요.

```
for key, value in 딕셔너리.items():  
    ~~
```

```
# S02-37  
score = {'Alice':95, 'Jackson':90, 'Kate':85}  
# 90점 이상인 학생의 이름을 passed로 넣기  
passed = []  
for key, value in score.items():  
    if value >= 90:  
        passed.append(key)  
print(passed)
```

```
['Alice', 'Jackson']
```

```
# S02-38  
score = {'Alice':95, 'Jackson':90, 'Kate':85}  
# 컴프리헨션으로 같은 결과 내기  
passed = [k for k, v in score.items() if v >= 90]  
print(passed)
```

## ■ enumerate

- `enumerate(리스트)`를 활용하면 `for`에서 리스트의 인덱스와 값을 동시에 활용할 수 있어요.

```
for i, v in enumerate(리스트):
    ~~
```

```
# S02-39
fruit = ['apple', 'banana', 'cherry']
for i, v in enumerate(fruit):
    print(f'{i}: {v}')
```

```
0: apple
1: banana
2: cherry
```

## ■ map

- `map(함수, 리스트)`를 사용하면  
리스트 각 요소 값에  
함수를 적용할 수 있어요.
- `map`의 결과를 리스트로 활용하려면  
`list(map(~~))`으로 해줘야 해요.

```
# S02-40
def to_positive(num):
    # 수로 바꾸고 양수로 만들기
    num = int(num)
    # if-else를 줄여쓸 수 있어요!
    return num if num >= 0 else num * -1
```

```
number = ['11', '-12', '-13', '14']
number1 = list(map(int, number))
number2 = list(map(to_positive, number))
print(number1, number2)
```

```
[11, -12, -13, 14] [11, 12, 13, 14]
```

## ■ 람다 함수

- 특정한 순간에만 간단히 사용할 함수가 필요한 경우, 이름 없이 기능만 하는 **람다 함수**를 설정할 수 있다.

lambda 파라미터 : 반환값(연산)

```
# S02-41
# int로 바꾸고 6으로 나눈 나머지
num = (lambda x : int(x) % 6)('345')
print(num)
```

3

```
# S02-42 : map에 람다 함수 활용
# 수로 바꾸고 6으로 나눈 나머지
number = ['345', '842', '234', '539']
number3 = list(map(lambda x : int(x) % 6, number))
print(number3)
```

[3, 2, 0, 5]