데이터 구조(Data Structure) I

데이터 구조(Data Structure)란 데이터에 편리하게 접근하고, 변경하기 위해서 데이터를 저장하거나 조작하는 방법을 말한다.

> **Program = Data Structure + Algorithm**
>
> - Niklaus Wirth

- 알고리즘에 빈번히 활용되는 순서가 있는(ordered) 데이터 구조
  - 문자열(String)
  - 리스트(List)
- 데이터 구조에 적용 가능한 Built-in Function

문자열(String)

> 변경할 수 없고(immutable), 순서가 있고(ordered), 순회 가능한(iterable) // 순회 가능한 : 반복문을 쓸 수 있다

문자열의 다양한 조작법(method)

https://docs.python.org/ko/3/library/stdtypes.html#string-methods

조회/탐색

.find(x)

x의 **첫 번째 위치**를 반환합니다. 없으면, `-1` 을 반환합니다.

In [ ]:
```
# 아래에 코드를 작성하세요.
```

In [ ]:
```
a = 'apple'
```

In [ ]:
```
a.find('a')
```

In [ ]:
```
a.find('p')  # 더 먼저 등장한 index 값을 반환함.
```

In [ ]:
```
a.find('f')
```

.index(x)

x의 **첫번째 위치**를 반환합니다. 없으면, 오류가 발생합니다.

In [ ]:
```

```

```
# 아래에 코드를 작성하세요.
```

In [ ]:
```
a = 'apple'
```

In [ ]:
```
a.index('a')
```

In [ ]:
```
a.index('p')
```

In [ ]:
```
a.index('f')
```

find와 index 차이

- find : 없으면 -1 반환
- index : 없으면 오류 발생

값 변경,,

.replace(old, new[, count])

바꿀 대상 글자를 새로운 글자로 바꿔서 반환합니다.

count를 지정하면 해당 갯수만큼만 시행합니다.

In [ ]:
```
# 아래에 코드를 작성하세요.
```

In [ ]:
```
z = 'zoo!yoyo!'
```

In [ ]:
```
z.replace('o', '')   # 알파벳o는 공백으로 치환해줘!
```

In [ ]:
```
z.replace('o','',2)   # 2번만큼만 시행
```

In [ ]:
```
'banana'.replace('a','',2)
```

.strip([chars])

특정한 문자들을 지정하면, 양쪽을 제거하거나 왼쪽을 제거하거나(lstrip), 오른쪽을 제거합니다(rstrip).

지정하지 않으면 공백을 제거합니다.

In [ ]:
```
# 아래에 코드를 작성하세요.
```

In [ ]:
```
oh = '    oh!\n'   # \n도 공백 취급
```

In [ ]:
```
oh.strip()   # 앞 뒤 공백 모두 제거
```

In [ ]:
```
oh.lstrip()  # 왼쪽 공백 제거
```

In [ ]:
```
oh.rstrip()  # 오른쪽 공백 제거
```

In [ ]:
```
'홍길동 '=='홍길동'   # 공백이 있어서 둘이 다름
```

In [ ]:
```
escape = '\t\n'
```

In [ ]:
```
escape.strip()
```

In [ ]:
```
'hehehihihihihi'.rstrip('hi')
```

.split() 중요!!!!!

문자열을 특정한 단위로 나누어 리스트로 반환합니다.

In [ ]:
```
# 아래에 코드를 작성하세요.
```

In [ ]:
```
csv = '1,홍길동,01012344567'
```

In [ ]:
```
csv.split(',')     # ,가 구분자가 되어 하나의 리스트로 만들어준다.
```

In [ ]:
```
numbers = '1 5 6'
# 기본은 공백을 기준으로 끊어준다!!! 리스트로 반환
numbers.split()
```

'separator'.join(iterable)

특정한 문자열로 만들어 반환합니다.

반복가능한(iterable) 컨테이너의 요소들을 separator를 구분자로 합쳐(`join()`) 문자열로 반환합니다.

In [ ]:
```
word = '배고파'
words = ['안녕', 'hello']

# 아래에 코드를 작성하세요.
```

In [ ]:
```
'!'.join(word)
```

In [ ]:
```
','.join(words)
```

In [ ]:
```
' '.join(word)
```

In [ ]:
```
''.join(word)
```

In [ ]:
```
words = ['1','2','3']
''.join(words)
```

문자 변형

.capitalize(), .title(), .upper()

- .capitalize() : 앞글자를 대문자로 만들어 반환한다.
- .title() : 어포스트로피나 공백 이후를 대문자로 만들어 반환한다.
- .upper() : 모두 대문자로 만들어 반환한다.

In [ ]:
```
a = 'hI! Everyone, I\'m kim'

# 아래에 코드를 작성하세요.
```

In [ ]:
```
# 처음만 대문자
a.capitalize()
```

In [ ]:
```
# 특정조건에서만 대문자
a.title()       # 어포스트로피는 ' 이다.
```

In [ ]:
```
# 모두 대문자
a.upper()
```

In [ ]:
```

```

.lower(), .swapcase()

- lower() : 모두 소문자로 만들어 반환한다.
- swapcase() : 대 <-> 소문자로 변경하여 반환한다.

In [ ]:
```
a = 'hI! Everyone, I\'m kim'

# 아래에 코드를 작성하세요.
```

In [ ]:
```
a.lower()
```

In [ ]:
```
a.swapcase()
```

In [ ]:

기타 문자열 관련 검증 메소드 : 참/거짓 반환

`.isalpha()`, `.isdecimal()`, `.isdigit()`, `.isnumeric()`, `.isspace()`, `.isupper()`, `.istitle()`, `.islower()`

In [ ]:
```
# 다음 명령어로 문자열 메소드를 확인할 수 있습니다.
3.isalpha()
```

In [ ]:
```
'a'.isalpha()
```

In [ ]:
```
'3.5'.isdecimal()
```

In [ ]:
```
'3'.isnumeric()
```

In [ ]:
```
'0o23'.isdecimal()
```

In [ ]:
```
'0o23'.isdigit()
```

리스트(List)

> 변경 가능하고(mutable), 순서가 있고(ordered), 순회 가능한(iterable)

데이터 구조로서의 리스트(list)와 조작법(method)

- https://docs.python.org/ko/3/tutorial/datastructures.html#more-on-lists

값 추가 및 삭제

.append(x)

리스트에 값을 추가할 수 있습니다.

In [ ]:
```
# 카페 리스트를 만들어봅시다.
cafe = ['starbucks', 'tomntoms', 'hollys']
print(cafe)
```

In [ ]:
```
# 값을 추가해봅시다.
cafe.append('espresso')
```

In [ ]:
```
# 코드 입력
print(cafe)
```

In [ ]:
```
cafe = ['starbucks', 'tomntoms', 'hollys']
cafe.append(['cococo'])
print(cafe)
print('-----')
cafe.extend(['wefgwg'])
print(cafe)
```

In [ ]:
```
# append 정리
cafe = ['starbucks', 'tomntoms', 'hollys']
cafe.append('혜인커피')
cafe
```

In [ ]:
```
cafe.append(('혜인커피','마시러올래?'))   # 튜플형태로 들어감
cafe
```

In [ ]:
```
cafe.append(['라떼'])   # 리스트 안에 리스트
cafe
```

In [ ]:
```
cafe.append(['제일좋아','진짜로!'])
cafe
```

In [ ]:
```
cafe.extend(['우중커피'])
cafe
```

.extend(iterable)

리스트에 iterable(list, range, tuple, string**[주의]**) 값을 붙일 수가 있습니다.

In [ ]:
```
# 앞서 만든 리스트에 추가해봅시다.
```

In [ ]:
```
# 코드 입력
cafe.extend(['이디야'])
print(cafe)
```

In [ ]:
```
# 앞서 배운 list concatenate와 동일합니다.
```

In [ ]:
```
# 코드 입력
cafe = cafe + ['스벅']
print(cafe)
```

In [ ]:
```
cafe.extend(['이디야','던킨도너츠','커피와빵'])
```

In [ ]:
```
cafe
```

In [ ]:
```
# append와 비교해봅시다.
```

```
# append는 하나 넣는거고 extend는 여러개 넣는 것
a = []
a += ['김밥천국','김밥카페']
```

In [ ]:
```
a
```

In [ ]:
```
# 코드 입력
print(cafe)
```

In [ ]:
```
# 문자열도 활용해봅시다.
```

In [ ]:
```
print(cafe)
cafe.extend('ediya')   # 따라서 하나로 넣고자 할 때는 리스트로 묶어서 넣어줘야함
print(cafe)
```

.insert(i, x)

정해진 위치 i 에 값을 추가합니다.

In [ ]:
```
# 앞서 만든 리스트의 가장 앞에 'hi'를 넣어봅시다.
```

In [ ]:
```
# 코드 입력
cafe.insert(0,'hi')
print(cafe)
```

In [ ]:
```
# 앞서 만든 리스트의 가장 뒤에 'bye'를 넣어봅시다
```

In [ ]:
```
# 코드 입력
cafe.insert(-1,'bye')   # -1하면 한칸 앞에 들어간다.
print(cafe)
```

In [ ]:
```
cafe.insert(len(cafe),'진짜안녕')   # length로 해야 가장 마지막에 들어간다
print(cafe)
```

In [ ]:
```
# 리스트의 길이를 넘어서는 인덱스는 마지막에 아이템이 추가됩니다.
len(cafe)
```

In [ ]:
```
# 코드 입력
cafe.insert(200,'이만안녕')
print(cafe)
```

.remove(x)

리스트에서 값이 x인 것을 삭제합니다.

In [ ]:
```
# remove를 사용해봅시다.
```

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

In [ ]:
```
# 중복된 값 1을 삭제 해봅시다.
```

In [ ]:
```
# 코드 입력
numbers.remove(1)   # 하나만 삭제됨
print(numbers)
```

In [ ]:
```
# 한번 더 삭제해봅시다.
```

In [ ]:
```
# 코드 입력
numbers.remove(1)
print(numbers)
```

In [ ]:
```
# remove는 값이 없으면 오류가 발생합니다. 확인해봅시다.
```

In [ ]:
```
numbers.remove(1)
print(numbers)
```

.pop(i)

정해진 위치 `i` 에 있는 값을 삭제하며, 그 항목을 반환합니다.

`i` 가 지정되지 않으면 마지막 항목을 삭제하고 되돌려줍니다.

- remove는 값을 찾아서 삭제, pop은 index를 찾아서 삭제

In [ ]:
```
# pop을 사용해봅시다.
a = [1, 2, 3, 4, 5, 6]
```

In [ ]:
```
# 가장 앞에 있는 것을 삭제해봅시다. return도 확인해보세요.
```

In [ ]:
```
a.pop()
```

In [ ]:
```
print(a)
```

In [ ]:
```
a.pop(0)
```

In [ ]:
```
print(a)
```

In [ ]:
```
# 값이 return이 된다는 것은 별도의 변수에 저장할 수 있다는 것입니다.
```

In [ ]:
```
students = ['홍길동','유재석']
bye = students.pop()
```

```
print(f'{bye} 학생이 떠나갔습니다.{students}')
```

.clear()

리스트의 모든 항목을 삭제합니다.

In [ ]:
```
# clear를 사용해봅시다.
```

In [ ]:
```
a = [1,2,3,4,5]
a.clear()
print(a)
```

탐색 및 정렬

.index(x)

x 값을 찾아 해당 index 값을 반환합니다.

In [ ]:
```
# index를 사용해봅시다.
a = [1, 2, 3, 4, 5]
```

In [ ]:
```
a.index(1)
```

In [ ]:
```
# index는 없을 시 오류가 발생합니다. 확인해봅시다.
# 앞서 remove 역시도 같은 에러가 발생하였습니다. (ValueError)
a.index('a')
```

.count(x)

원하는 값의 개수를 확인할 수 있습니다.

In [ ]:
```
# count를 사용해봅시다.
a = [1, 2, 5, 1, 5, 1]
```

In [ ]:
```
a.count(1)
```

In [ ]:
```
# 따라서 원하는 값을 모두 삭제하려면 다음과 같이 할 수 있습니다.
```

In [ ]:
```
a = [1, 2, 5, 1, 5, 1]
for i in range(a.count(1)):
    a.remove(1)
print(a)
```

In [ ]:
```
# 모두 삭제되었는지 검증해봅시다.
```

In [ ]:
```
```

```
1 in a
```

In [ ]:
```
# replace는 문자열에만 쓸 수 있는 것
```

.sort()

정렬을 합니다.

내장함수 `sorted()` 와는 다르게 **원본 list를 변형**시키고, **None** 을 리턴합니다.

In [ ]:
```
import random
lotto = random.sample(range(1, 46), 6)
print(lotto)
```

In [ ]:
```
# sort() 를 사용해봅시다.
# 원본을 바꾼다
lotto.sort()
# 이렇게 하면 아무것도 반환이 안되는데 원본을 바꾼것
```

In [ ]:
```
lotto
```

In [ ]:
```
# 원본은 바꾸지 않고 sorted된 것을 반환한다.
sorted(random.sample(range(1, 46), 6))
```

.reverse()

반대로 뒤집습니다. **(정렬 아님)**

In [ ]:
```
classroom = ['Tom', 'David', 'Justin']
print(classroom)
```

In [ ]:
```
classroom.reverse()
```

In [ ]:
```
classroom
```

리스트 복사

In [ ]:
```
# 리스트 복사를 해봅시다.
original_list = [1, 2, 3]
```

In [ ]:
```
copy_list = original_list
print(copy_list)
```

In [ ]:
```
# copy_list의 값을 바꾸고 original_list를 출력해봅시다.
```

In [ ]:
```
# 둘다 바뀐다  # 주의!!!!
copy_list[0] = 'A!!'
print(copy_list,original_list)
```

In [ ]:
```
# id 값을 확안해봅시다.
```

In [ ]:
```
# 같은 주소를 가리키고 있다.
print(id(copy_list))
print(id(original_list))
```

In [ ]:
```
# 뮤터블 변수는 주소 복사, 이뮤터블 변수(스티링, 튜플, 레인지)는 값 복사
```

데이터의 분류 (복습)

> `mutable vs. immutable`

데이터는 크게 변경 가능한 것(`mutable`)들과 변경 불가능한 것(`immutable`)으로 나뉘며, python은 각각을 다르게 다룹니다.

변경 불가능한(`immutable`) 데이터

- 리터럴(literal)
  - 숫자(Number)
  - 글자(String)
  - 참/거짓(Bool)
- range()
- tuple()
- frozenset()

In [ ]:
```
# immutable 데이터의 복사는 어떻게 이루어질까?
a = 20
b = a
b = 10

print(a)
print(b)

# a는 안바뀜
```

In [ ]:
```
%%html
<iframe width="800" height="500" frameborder="0" src="http://pythontutor.com/iframe-embed.ht
```

변경 가능한(`mutable`) 데이터

- `list`
- `dict`
- `set`

In [ ]:
```
# mutable 데이터의 복사는 어떻게 이루어질까?
a = [1, 2, 3, 4]
b = a
b[0] = 100

print(a)
print(b)

# a가 바뀜
```

In [ ]:
```
%%html
<iframe width="800" height="500" frameborder="0" src="http://pythontutor.com/iframe-embed.ht
```

리스트 복사 방법

- 이렇게 하면 값 달라짐

slice 연산자 사용 [:]

In [ ]:
```
# 리스트를 복사해봅시다. # 슬라이싱
```

In [ ]:
```
a = [1, 2, 3, 4]
b = a[:]
b[0] =100

print(a)
print(b)
```

In [ ]:
```
# 다른 방법으로 복사해봅시다.
```

list() 활용

In [ ]:
```
a = [1, 2, 3, 4]
b = list(a)
b[0] =100

print(a)
print(b)
```

- 하지만, 이렇게 하는 것도 일부 상황에만 서로 다른 얕은 복사(shallow copy) 이다.

In [ ]:
```
# 2차원 배열을 복사해봅시다.
```

In [ ]:
```
a = [[1,2,3],2,3]
b = list(a)  # 원래 이거 하면 위처럼 값이 달라지는데, 2차원 배열이면
print(a,b)
b[0][0] =100  # 리스트안에 리스트를 수정하면 값이 같음
print(a,b)
b[1] = '원소'  # 리스트 안에꺼를 바꾸면 값이 다름
print(a,b)
```

```
# 그러니까 겉리스트는 서로 다른데 안에 리스트는 같은 거를 바라보고 있는 것 !
```

- 만일 중첩된 상황에서 복사를 하고 싶다면, `깊은 복사(deep copy)` 를 해야한다.
- 즉, 내부에 있는 모든 객체까지 새롭게 값이 변경된다.

In [ ]:
```
# 깊은 복사를 사용해봅시다.
# 따라서 deepcopy를 하면 위와 같은 문제 상황을 해결할 수 있다.
```

In [ ]:
```
import copy
a = [[1,2,3],2,3]
b = copy.deepcopy(a)
print(a,b)
b[0][0] =100
print(a,b)
# id값을 완전히 다르게!
```

List Comprehension

List Comprehension은 표현식과 제어문을 통해 리스트를 생성합니다.

여러 줄의 코드를 한 줄로 줄일 수 있습니다.

활용법

```
[expression for 변수 in iterable]
```

```
list(expression for 변수 in iterable)
```

세제곱 리스트

> 다음의 리스트를 작성하세요.
>
> - 1~10까지의 숫자로 만든 세제곱 담긴 리스트 `cubic_list`

In [ ]:
```
numbers = range(1, 11)
```

In [ ]:
```
# 반복문을 활용하여 작성하세요.
```

In [ ]:
```
cubic_list = []
for number in numbers:
    cubic_list.append(number ** 3)
```

In [ ]:
```
print(cubic_list)
```

In [ ]:
```
# List comprehension을 활용하여 작성하세요.
```

In [ ]:
```
[number ** 3 for number in numbers]
```

In [ ]:
```
print(cubic_list)
```

List Comprehension + 조건문

조건문에 참인 식으로 리스트를 생성합니다.

활용법

```
[expression for 변수 in iterable if 조건식]
```

```
[expression if 조건식 else 식 for 변수 in iterable]
```

[연습] 짝수리스트

> 다음의 리스트를 작성하세요.

- 1~10까지의 숫자중 짝수만 담긴 리스트 `even_list`
- 여러개의 `for` 혹은 `if` 문을 중첩적으로 사용 가능합니다.

In [ ]:
```
# 반복문을 활용하여 작성하세요.
```

In [ ]:
```
even_list = []
for i in range(1, 11):
    if i % 2 == 0:
        even_list.append(i)
```

In [ ]:
```
print(even_list)
```

In [ ]:
```
# List comprehension을 활용하여 작성하세요.
```

In [ ]:
```
[i for i in range(1,11) if i % 2 == 0 ]   # 조건식을  뒤에 활용
```

In [ ]:
```
print(even_list)
```

In [ ]:
```
# 홀소는 음수, 짝수는 양수
n = 3
'홀수' if n%2 == 1 else '짝수'   # 조건표현식
```

In [ ]:
```
[-i if i % 2 ==1 else i for i in range(1,11)]
```

[실습] 곱집합

> 주어진 두 list의 가능한 모든 조합을 담은 `pair` 리스트를 작성하세요.

> 1. 반복문 활용
> 2. list comprehension 활용

[입력 예시]

```
girls = ['jane', 'ashley', 'mary']
boys = ['justin', 'eric', 'david']
```

[출력 예시]

```
[('justin', 'jane'), ('justin', 'ashley'), ('justin', 'mary'),
('eric', 'jane'), ('eric', 'ashley'), ('eric', 'mary'), ('david',
'jane'), ('david', 'ashley'), ('david', 'mary')]
```

In [ ]:
```
girls = ['jane', 'ashley', 'mary']
boys = ['justin', 'eric', 'david']
```

In [ ]:
```
# 반복문을 활용하여 작성하세요.
```

In [ ]:
```
pair = []
for boy in boys:
    for girl in girls:
        pair.append((boy, girl))  # 괄호를 하나 더 넣어서 한다.
```

In [ ]:
```
print(pair)
```

In [ ]:
```
# List comprehension을 활용하여 작성하세요.
```

In [ ]:
```
pair = [(boy, girl) for boy in boys for girl in girls]
```

In [ ]:
```
print(pair)
```

[응용] 피타고라스 정리

> 주어진 조건(x < y < z < 50) 내에서 피타고라스 방정식의 해를 찾으세요.
>
> 1. 반복문 활용
> 2. list comprehension 활용

[출력 예시]

```
[(3, 4, 5), (5, 12, 13), (6, 8, 10), (7, 24, 25), (8, 15, 17), (9, 12,
15), (9, 40, 41), (10, 24, 26), (12, 16, 20), (12, 35, 37), (15, 20,
25), (15, 36, 39), (16, 30, 34), (18, 24, 30), (20, 21, 29), (21, 28,
35), (24, 32, 40), (27, 36, 45)]
```

In [ ]:
```
# 반복문을 활용하여 작성하세요.
```

In [ ]:
```
result = []
for x in range(1, 50):
    for y in range(x, 50):
        for z in range(y, 50):
            if x**2 + y**2 == z**2:
                result.append((x, y, z))   # 리스트에 튜플 형태로 추가할 때는 이렇게!
```

In [ ]:
```
print(result)
```

In [ ]:
```
# List comprehension을 활용하여 작성하세요.
```

In [ ]:
```
result = [(x, y, z) for x in range(1, 50) for y in range(x, 50) for z in range(y, 50) if x**
```

In [ ]:
```
print(result)
```

[응용] 모음 제거하기

> 다음의 문장에서 모음(a, e, i, o, u)를 모두 제거하세요.

[입력 예시]

```
words = 'Life is too short, you need python!'
```

[출력 예시]

```
Lf s t shrt, y nd pythn!
```

In [ ]:
```
vowels = 'aeiou'
words = 'Life is too short, you need python!'
```

In [ ]:
```
# 반복문을 활용하여 작성하세요.
```

In [ ]:
```
result = []
for x in words:
    if x not in vowels:
        result.append(x)
```

In [ ]:
```
print(''.join(result))
```

In [ ]:
```
# List comprehension을 활용하여 작성하세요.
```

In [ ]:
```
result = [x for x in words if x not in vowels]
```

In [ ]:
```
print(''.join(result))
```

데이터 구조에 적용가능한 Built-in Function

순회 가능한(iterable) 데이터 구조에 적용가능한 Built-in Function

> iterable 타입 - `list` , `dict` , `set` , `str` , `bytes` , `tuple` , `range`

- `map()`
- `filter()`
- `zip()`
- ~~`reduce()`~~ (참고)

`map(function, iterable)`

- 순회가능한 데이터 구조(iterable)의 모든 요소에 function을 적용한 후 그 결과를 돌려준다.
- return은 `map_object` 형태이다.

In [ ]:
```
numbers = [1, 2, 3]

# 위의 코드를 문자열 '123'으로 만드세요
```

In [ ]:
```
# List comprehension 활용
```

In [ ]:
```
[str(i) for i in numbers]
```

In [ ]:
```
''.join([str(i) for i in numbers])
```

In [ ]:
```
# map() 활용
```

In [ ]:
```
map(str,numbers)
```

In [ ]:
```
list(map(str,numbers))
```

In [ ]:
```
# 반복을 하면서 특정 함수를 적용한 결과!
[str(i) for i in numbers]
list(map(str,numbers))
```

`map()` 함수는 입력값을 처리할 때 자주 활용됩니다.

In [ ]:
```
numbers = ['1', '2', '3']

# 위의 코드를 숫자 '123'으로 만드세요
```

In [ ]:
```
# List comprehension 활용
```

In [ ]:
```

```

In [ ]:
```
print(new_numbers)
```

In [ ]:
```
# map() 활용
```

In [ ]:
```

```

In [ ]:
```
print(new_numbers)
```

첫번째 인자 function은 사용자 정의 함수도 가능합니다.

In [ ]:
```
# 세제곱의 결과를 나타내는 함수가 있습니다.
def cube(n):
    return n ** 3
```

In [ ]:
```
# 세제곱 함수를 각각의 요소에 적용한 결과값을 구해봅시다.
numbers = [1, 2, 3]
```

In [ ]:
```
new_numbers = list(map(cube, numbers))   # function이니까 cube가능한것!!
```

In [ ]:
```
print(new_numbers)
```

In [ ]:
```

```

In [ ]:
```
numbers = ['1','2','3']
result = 0
for i in list(map(int, numbers)):
    result = 10 * result + i
print(result)
```

[연습] 코딩 테스트의 기본

> 두 정수를 입력 받아 더한 값을 출력하시오.

[입력 예시]

3 5

[출력 예시]

8

In [ ]:
```
# 아래에 코드를 작성하시오.
a = map(int,input().split())
print(sum(a))
```

아래 세가지 예시 비교

In [ ]:
```
a = ['1','2','3']
```

In [ ]:
```
# 단순 반복
b = []
for i in a:
    b.append(int(i))
print(b)
```

In [ ]:
```
# 리스트 컴프리핸션
c = [int(i) for i in a]
print(c)
```

In [ ]:
```
# map
d = list(map(int,a))
print(d)
```

filter(function, iterable)

- iterable에서 function의 반환된 결과가 `True` 인 것들만 구성하여 반환한다.
- `filter object` 를 반환한다.

In [ ]:
```
# 특정 list에서 홀수만을 걸러내는 코드를 작성해봅시다.
```

In [ ]:
```
# 홀수를 판별하는 함수가 있습니다.
def odd(n):
    return n % 2
```

In [ ]:
```
# 홀수인 요소만 뽑아 new_numbers에 저장합니다.
numbers = [1, 2, 3]
```

In [ ]:
```
new_numbers = list(filter(odd, numbers))      # function의 반환된 결과가 True인 것들만 구성하
```

In [ ]:
```
print(new_numbers)
```

In [ ]:
```
# 다음의 list comprehension과 동일합니다.
```

In [ ]:
```

```

zip(*iterables)

- 복수의 iterable 객체를 모아(`zip()`)준다.

- 결과는 튜플의 모음으로 구성된 `zip object` 를 반환한다.

In [ ]:
```
girls = ['jane', 'ashley', 'mary']
boys = ['justin', 'eric', 'david']
```

In [ ]:
```
# zip() 활용하여 짝을 맞추어 본다.
```

In [ ]:
```
pair = list(zip(girls, boys))
```

In [ ]:
```
print(pair)
```