In [ ]:

```
from IPython.display import IFrame
```

문자열 메소드 활용하기

변형

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

`.capitalize()` : 앞글자를 대문자로 만들어 반환합니다.

`.title()` : 어포스트로피나 공백을 이후를 대문자로 만들어 반환합니다.

`.upper()` : 모두 대문자로 만들어 반환합니다.

In [ ]:

```
a = "hI! Everyone, I'm kim"
```

In [ ]:

```

```

In [ ]:

```

```

`lower()`, `swapcase()`

`lower()` : 모두 소문자로 만들어 반환합니다.

`swapcase()` : 대<->소문자로 변경하여 반환합니다.

In [ ]:

```

```

In [ ]:

```

```

`.join(iterable)`

특정한 문자열로 만들어 반환합니다.

In [ ]:

```

```

In [ ]:

```

```

`.replace(old, new[, count])`

바꿀 대상 글자를 새로운 글자로 바꿔서 반환합니다.

count를 지정하면 해당 갯수만큼만 시행합니다.

In [ ]:

```
```

In [ ]:

```
```

글씨 제거 (`strip([chars])`)

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

지정하지 않으면 공백을 제거합니다.

In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
```

탐색 및 검증

`.find(x)` : x의 첫 번째 위치를 반환합니다. 없으면, -1을 반환합니다.

In [ ]:

```
```

In [ ]:

```
```

`.index(x)` : x의 첫번째 위치를 반환합니다. 없으면, 오류가 뜹니다.

In [ ]:

```
```

In [ ]:

```
```

다양한 확인 메소드 : 참/거짓 반환

`.isalpha()` : 문자열이 오직 알파벳으로만 구성되어있으면 True (공백 포함 False)

- 문자열의 모든 문자가 10진 숫자문자일 때 True
  - `.isdecimal()` : (Only Decimal Numbers)
  - `.isdigit()` : (Decimals, Subscripts, Superscripts)
  - `.isnumeric()` : (Digits, Vulgar Fractions, Subscripts, Superscripts, Roman Numerals, Currency Numerators)

`.isspace()` : 모든 공백문자이면 True

`.isspace()` : 모두 공백문자이면 True

`.isupper()` : 모두 대문자이면 True

`.istitle()` : 모든 문자열이 title 스타일 (단어마나 첫글자가 대문자) 이면 True

`.islower()` : 모두 소문자이면 True

`split()`

문자열을 특정한 단위로 나누어 리스트로 반환합니다.

In [ ]:

```

```

In [ ]:

```

```

문자열 뒤집기

문자열을 하나씩 반대로 잘라서 다시 입력시킨 수 출력

In [ ]:

```

```

Reverse() 함수 사용

In [ ]:

```

```

In [ ]:

```

```

Pythonic

In [ ]:

```

```

리스트 메소드 활용하기

값 추가 및 삭제

`.append(x)`

리스트에 값을 추가할 수 있습니다.

In [ ]:

```
# 리스트 하나를 만들어봅시다.
```

In [ ]:

```
# 값을 추가해봅시다.
```

In [ ]:

```
# 어렵게 넣어보도록 해봅시다.
```

`.extend(iterable)`

리스트에 iterable(list, range, tuple, string *유의*) 값을 붙일 수가 있습니다.

In [ ]:

```
# 앞서 만든 리스트에 추가해봅시다.
```

In [ ]:

```
# 앞서 배운 list concatenate와 동일합니다.
```

In [ ]:

```
# append와 비교해봅시다.
```

In [ ]:

```

```

`insert(i, x)`

정해진 위치 `i`에 값을 추가합니다.

In [ ]:

```
# 앞서 만든 리스트의 가장 앞에 'hi'를 넣어주세요.
```

In [ ]:

```
# 앞서 만든 리스트의 가장 뒤에 'bye'를 넣어주세요.
```

In [ ]:

```
# 길이를 넘어서는 인덱스는 무조건 마지막에 하나만 붙습니다.
```

`remove(x)`

리스트에서 값이 x인 것을 삭제합니다.

In [ ]:

```
numbers = [1, 2, 3, 1, 2]
print(numbers)
```

In [ ]:

```
# 중복된 값 1을 삭제 해봅시다.
```

In [ ]:

```
# 한번 더 삭제해봅시다.
```

In [ ]:

```

```

```
# remove는 값이 없으면 오류가 발생합니다. 확인해봅시다.
```

`.pop(i)`

정해진 위치 `i`에 있는 값을 삭제하며, 그 항목을 반환합니다.

`i`가 지정되지 않으면 마지막 항목을 삭제하고 되돌려줍니다.

In [ ]:

```
a = [1, 2, 3, 4, 5, 6]
```

In [ ]:

```
# 가장 앞에 있는 것을 삭제해봅시다. return도 확인해보세요.
```

In [ ]:

```

```

In [ ]:

```
# 값이 return이 된다는 것은 별도의 변수에 저장할 수 있다는 것입니다.
```

탐색 및 정렬

`.index(x)`

원하는 값을 찾아 index 값을 반환합니다.

In [ ]:

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

In [ ]:

```
# index는 없을 시 오류가 발생합니다. 확인해봅시다.
# 앞서 remove 역시도 같은 에러가 발생하였습니다. (ValueError)
```

`.count(x)`

원하는 값의 갯수를 확인할 수 있습니다.

In [ ]:

```
a = [1, 2, 5, 1, 5, 1]
```

In [ ]:

```
# 따라서 원하는 값을 모두 삭제하려면 다음과 같이 할 수 있습니다.
```

In [ ]:

```
# 모두 삭제되었는지 검증해봅시다.
```

`.sort()`

정렬을 합니다.

sorted()와는 다르게 원본 list를 변형시키고, None을 리턴합니다.

sorted()와는 다르게 원본 list를 변형시키고, None을 리턴합니다.

In [ ]:

```
```

In [ ]:

```
```

`reverse()`

반대로 뒤집습니다. (정렬 아님)

In [ ]:

```
```

복사

In [ ]:

```
# 리스트 복사를 해봅시다.
```

In [ ]:

```
print(original_list)
# b의 값을 바꾸고 a를 출력해봅시다.
```

In [ ]:

```
```

In [ ]:

```
# 숫자를 확인해봅시다.
```

In [ ]:

```
# 딕셔너리도 확인해봅시다.
```

In [ ]:

```
IFrame('https://goo.gl/vx1yGx', width='100%', height='300px')
```

In [ ]:

```
IFrame('https://goo.gl/N43pw6', width='100%', height='300px')
```

- 파이썬에서 모든 변수는 객체의 주소를 가지고 있을 뿐입니다.

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

- 위와 같이 변수를 생성하면 hong이라는 객체를 생성하고, 변수에는 객체의 주소가 저장됩니다.
- 변경가능한(mutable) 자료형과 변경불가능한(immutable) 자료형은 서로 다르게 동작합니다.

따라서, 복사를 하고 싶을 때에는 다음과 같이 해야한다.

In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
IFrame('https://goo.gl/ZH6yZd', width='100%', height='300px')
```

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

In [ ]:

```
```

In [ ]:

```
IFrame('https://goo.gl/FZcYbJ', width='100%', height='300px')
```

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

In [ ]:

```
```

In [ ]:

```
IFrame('https://goo.gl/dtnCzY', width='100%', height='300px')
```

삭제

리스트의 모든 항목을 삭제합니다.

In [ ]:

```
```

List Comprehension

List를 만들 수 있는 간단한 방법이 있습니다.

`[ 식 for 변수 in 리스트 ]`

`list(식 for 변수 in 리스트)` => # not good

- 리스트 컴프리헨션의 반복은 인터프리터 내부에서 C언어의 속도로 실행되기 때문에, 일반 for 반복문보다 최대 2배 빠르게 실행되기도 한다.

In [ ]:

```
```

사전 준비

> 다음의 리스트를 만들어보세요.

1. 1~10까지의 숫자 중 짝수만 담긴 리스트 `even_list`
2. 1~10까지의 숫자로 만든 세제곱 담긴 리스트 `cubic_list`

In [ ]:

```
```

In [ ]:

```
```

성 추출하기

> 리스트 컴프리헨션을 사용하여 `names` 리스트 요소 이름의 성만을 모아 `first_name` 로 만들어주세요.(소문자로)

```
names = ["Rick Sanchez", "Morty Smith", "Summer Smith", "Jerry Smith", "Beth Smith"]

print(first_names)
```

In [ ]:

```
# 아래에 반복문을 활용하여 만들어주세요.
```

In [ ]:

```
# 아래에 List comprehension을 활용하여 만들어주세요.
```

활용법

여러개의 `for` 혹은 `if` 문을 중첩적으로 사용 가능합니다.

리스트 표현식에 for가 여러 개일 때 처리 순서는 뒤에서 앞으로 순서이다.

`[식 for 변수1 in 리스트1 if 조건식1 for 변수2 in 리스트2 if 조건식2]`

`[식 for 변수 in 리스트 if 조건식]`

`[식 if 조건식 else 조건식 for 변수 in 리스트 ]` **주의! else 가 있다면 순서가 바뀐다.**

예시) 구구단 리스트 생성

```
gugudan = [a * b for a in range(2, 10)
                 for b in range(1, 10)]
>>> [2, 4, 6, 8, 10, 12, 14, 16, 18, 3, 6, 9, 12, 15, 18, 21, 24, 27, 4, 8, 12, 16, 20, 24, 28
, 32, 36, 5, 10, 15, 20, 25, 30, 35, 40, 45, 6, 12, 18, 24, 30, 36, 42, 48, 54, 7, 14, 21, 28
, 35, 42, 49, 56, 63, 8, 16, 24, 32, 40, 48, 56, 64, 72, 9, 18, 27, 36, 45, 54, 63, 72, 81]
```

예시) 1~10 숫자 중 2의 배수인 숫자(짝수)로 리스트 생성

```
even_list = [i for i in range(1, 11) if i % 2 == 0]
>>> [2, 4, 6, 8, 10]
```

연습 문제

짝짓기 - 곱집합

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

1. 반복문 활용
2. list comprehension 활용

```
girls = ['jane', 'iu', 'mary']
```

```
boys = ['justin', 'david', 'kim']

예시 출력)

[('justin', 'jane'), ('justin', 'iu'), ('justin', 'mary'), ('david', 'jane'), ('david', 'iu'
), ('david', 'mary'), ('kim', 'jane'), ('kim', 'iu'), ('kim', 'mary')]
```

In [ ]:

```
# 아래에 반복문을 활용하여 만들어주세요.
```

In [ ]:

```
# 아래에 List comprehension을 활용하여 만들어주세요.
```

모음 제거하기

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

1. list comprehension만 사용해보세요.

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

예시출력)
Lf s t shrt, y nd pythn!
```

In [ ]:

```
# 아래에 List comprehension을 활용하여 만들어주세요.
```

피타고라스 정리

> 주어진 조건(x < y < z < 50) 내에서 피타고라스 방정식의 해를 찾아보세요.

1. 반복문 활용
2. list comprehension 활용

```
예시 출력)
[(3, 4, 5), (5, 12, 13), (6, 8, 10), (7, 24, 25), (8, 15, 17), (9, 12, 15), (9, 40, 41), (1
0, 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 [ ]:

```
# 아래에 반복문을 활용하여 만들어주세요.
```

In [ ]:

```
# 아래에 List comprehension을 활용하여 만들어주세요.
```

Slice

부분 집합에 대한 참조

- `시작:끝:간격` 구조 / `start:end:step`
- 시작 인덱스 부터 끝 인덱스까지, 설정한 간격만큼 건너뛴 원소를 선택한다. (끝 인덱스는 포함 x)
- `시작` 은 끝이 있는 경우 생략할 수 있고 이때는 0 이 된다.
- `끝` 은 생략할 수 있으며, 생략하는 경우 리스트의 길이이다. (마지막 인덱스).
- `간격` 은 생략할 수 있으며, 생략하는 경우 1 이다.
- 간격이 음수 이면 만들어지는 결과는 뒤에서부터 추출하여 역순이 된다.

In [ ]:

```
slice_list = list(range(1, 11))
print(slice_list)
```

In [ ]:

```
# 처음부터 3번 인덱스까지
```

In [ ]:

```
# 처음부터 끝까지 두 칸 간격으로
```

In [ ]:

```
# 시작값을 생략해버리면 리스트의 처음부터라는 의미
```

In [ ]:

```
# 끝값을 생략해버리면 리스트의 끝까지라는 의미
```

In [ ]:

```
slice_list[0:5:2] == slice_list[:5:2]
```

In [ ]:

```
# 간격이 명시되면 start, end 값은 동시에 생략 가능
```

In [ ]:

```
# 리스트 전체에 대한 사본
```

In [ ]:

```
slice_list[9:3:-1]
```

In [ ]:

```
# 리스트 전체를 뒤집은 사본
```

인덱스의 이해

정확히하면 인덱스는 특정한 원소가 아닌, `그 원소의 바로 앞쪽` 을 가리키는 숫자이다.

- `list[0]` 은 실질적으로 “인덱스 0으로부터 1칸의 값”을 의미한다.
- 그냥 어렵게 생각하지말고 **슬라이스는 아래 그림들로 이해하면 끝난다.**

In [ ]:

```
# 이해가 끝났다면 slice_list[3:3] 은 무엇인가.
```