

K-NET DEVELOP #3

Hyungyu Kim

K-NET

2024-04-30

지난 시간 이야기 솔직히 기억 안 나죠?

- 변수, 배열, 리스트, 딕셔너리, 조건문, 반복문, 함수, ...
- 이번 회차에서 다룰 주제는 변수와 리스트입니다.
- 이번 회차에서는 조금 디테일하게 살펴보겠습니다.



```
even_nums = [x for x in range(0, 101) if x % 2 == 0]
print(even_nums)
# Output: [0, 2, 4, 6, 8, 10, ..., 94, 96, 98, 100]
```

List = [0, 1, 2, 3, 4, 5]

0	1	2	3	4	5
---	---	---	---	---	---

List[0] = 0

List[0:] = [0,1,2,3,4,5]

List[1] = 1

List[:] = [0,1,2,3,4,5]

List[2] = 2

List[2:4] = [2, 3]

List[3] = 3

List[1:3] = [1, 2]

List[4] = 4

List[:4] = [0, 1, 2, 3]

List[5] = 5

컴퓨터에게 정보를 기억시키는 방법

- 변수란 값을 저장하는 공간을 말합니다.
- 파이썬의 변수는 객체를 나타냅니다.
- 객체라는 것은 숫자(정수형, 실수형), 문자열, 리스트, 딕셔너리 등을 말하는 것입니다.
- 파이썬의 모든 것은 객체입니다! 객체라는 용어와 친해지는 것이 좋겠습니다.



```
# 파이썬의 변수 사용 예시
```

```
greeting = "여러분, 벌써 3주차네요. 반갑습니다!"  
print(greeting)
```

변수 선언에서 “=”의 의미는...

- 한 가지 충격적인 사실: 파이썬에서 “ $n=1$ ”의 의미는 “ n 은 1과 같다”가 아닙니다!
- 이는 변수 n 에 1을 대입한다는 뜻입니다.
- “ $x = x+1$ ”은 변수 x 에 $x+1$ 에 해당하는 값을 대입(할당)한다는 뜻입니다. (1 증가)
- 위의 식은 하도 많이 써서 $x += 1$ 이라고 줄여서 씁니다.



변수 선언에서 “=”의 의미는...

- “n은 1과 같다”는 것은 “`n==1`”으로 나타낼 수 있습니다.
- 위의 식은 True나 False의 값을 가집니다.
- 참고: 파이썬은 증감 연산자인 `n++` 혹은 `++n`의 표현이 없습니다.



```
x = 3
```

```
x = x + 1  
print(x)
```

```
x += 1  
print(x)
```

파이썬 변수의 장점

- 파이썬은 변수 선언 시 자료형(데이터 타입)은 신경 쓰지 않아도 됩니다!
- 코드 중간에 뜬금없이 내놓고 선언해도 되고, 변수의 타입이 바뀌어도 됩니다.



```
e = 100
print(e)
print(type(e)) # type( ) : 변수 자료형

e = 2.718
print(e)
print(type(e))

e = "자연로그의 밑"
print(e)
print(type(e))
```

파이썬 변수의 장점

- C언어 특: 이런 거 쉽게 못 함 ㅋㅋ



```
1  e = 100
2  print(e)
3  print(type(e)) # type() : 변수 자료형
4
5  e = 2.718
6  print(e)
7  print(type(e))
8
9  e = "자연로그의 밑"
10 print(e)
11 print(type(e))
```

```
100
<class 'int'>
2.718
<class 'float'>
자연로그의 밑
<class 'str'>
```

자료형과 관련하여

- C언어에서는 변수 선언 시에 자료형을 적어줘야만 합니다.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     int arr[5];
7     printf("%d\n", arr[5]);
8 }
```

- 참고: 자료형이 뭐냐고 질문 들어오면 “변수 선언할 때 쓰는 것”이라 답하지 마시고 “값을 저장하기 위한 공간(변수)의 형태/종류”라고 답하면 됩니다.

자료형 체크하기

- 파이썬에서도 자료형이 중요할 때가 있습니다.
- 만약 변수의 자료형을 알아내야 한다면 `type()`을 쓸 수 있습니다.
- 실수를 정수로 바꾸거나, 문자열로 바꾸고 싶다면 `int()`, `str()` 등을 사용할 수 있습니다.

```
[ ] 1 print(type(10), type(3.14), type(True), type("abc"), type(print))
```

```
<class 'int'> <class 'float'> <class 'bool'> <class 'str'> <class 'builtin_function_or_method'>
```

```
[15] 1 print(int(4.30))  
      2 print(float(4))  
      3 print(str(4.30))
```

```
4  
4.0  
4.3
```

자료형의 종류

- 참고: 위키백과

파이썬은 다음과 같은 자료형들을 갖고 있다.

- 기본 자료형:
 - 정수형
 - 긴 정수형(long integer) - 메모리가 허락하는 한 무제한의 자릿수로 정수를 계산할 수 있다. 파이썬 3 버전에서는 사라지고, 대신 정수형의 범위가 무제한으로 늘어났다.
 - 부동소수점형
 - 복소수형
 - 문자형
 - 유니코드 문자형
 - 함수형
 - 논리형(boolean)
- 집합형 자료형:
 - 리스트형 - 내부의 값을 나중에 바꿀 수 있다.
 - 튜플(tuple)형 - 한 번 값을 정하면 내부의 값을 바꿀 수 없다.
 - 사전형 - 내부의 값을 나중에 바꿀 수 있다.
 - 집합형 - 중복을 허락하지 않는다. 변경 가능하게도, 변경 불가능하게도 만들 수 있다.

또 많은 객체 지향 언어와 같이, 사용자가 새롭게 자신의 형을 정의할 수도 있다.

파이썬은 동적 타이핑의 일종인 덕 타이핑을 사용하는 언어이기 때문에, 변수가 아닌 값이 타입을 가지고 있고, 변수는 모두 값의 참조(C++의 참조)이다.

파이썬의 궁극기

- 파이썬에서는 변수를 한꺼번에 선언하고 할당할 수 있습니다.



```
a, b = 1, 3 # 한 줄에 두 값을 할당  
c = d = e = 5 # 모두 같은 값을 할당  
f, g = [8, 9] # 리스트 각 값을 할당
```

```
a, b = b, a # 값 바꾸기!
```

```
print(a, b)  
print(c, d, e)  
print(f, g)
```

변수끼리 계산 및 비교하기

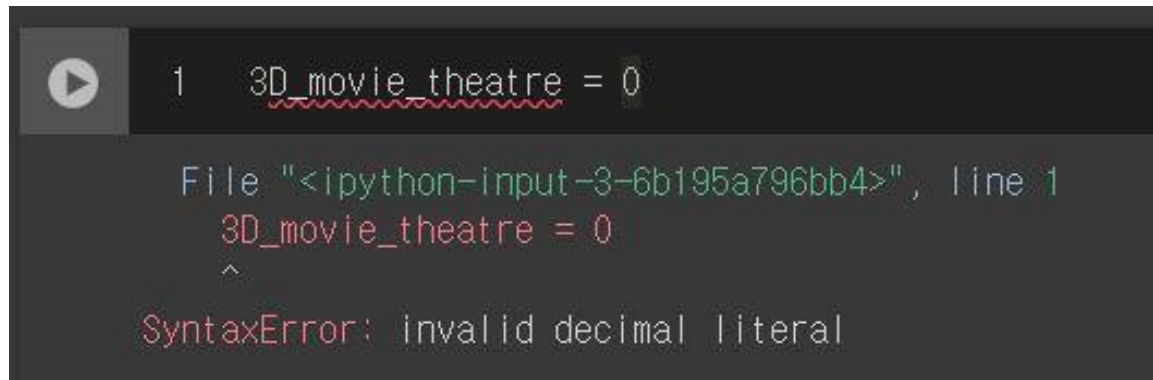
- 수학 기호랑 약간 비슷한 면이 있습니다.
- 사칙 연산이 모두 제공됩니다: $a+b$, $a-b$, $a*b$, a/b
- a 를 b 로 나눈 몫과 나머지: $a//b$, $a\%b$
- a 의 b 제곱: $a**b$
- 괄호 사용: $(a+b)*(c+d)$
- 부등호(크다, 작다): $a>b$, $a<b$
- 크거나 같다, 작거나 같다: $a\geq b$, $a\leq b$
- 같다: $a==b$

변수끼리 계산 및 비교하기

- 파이썬에서는 정수와 실수를 더하면 실수가 됩니다: $3.14 + 100 = ?$
- 파이썬에서는 정수끼리 나뉘어도 실수가 됩니다: $15/4 = ?$
- 참고: C언어에서는 정수끼리 나누면 무조건 정수가 됩니다: $1/2 = ?$
- 이것들은 전부 직접 실행해보면 경험적으로 습득할 수 있습니다.

변수명 짓기?

- 변수명 짓는 센스보다는, 쓸 수 없는 변수명을 소개하겠습니다.
- 1. 숫자로 시작하는 변수명
ex) 2D_girlfriend, 3D_movie_theatre, 4D_movie_theatre, ...
- 2. 예약어 (파이썬의 문법적 의미를 갖는 키워드)
ex) and, as, assert, break, class, continue, def, del, elif, else, except, finally, for, from, global, if, import, in, is, lambda, nonlocal, not, or, pass, raise, return, try, while, with, yield, False, None, True
 - 이는 VS code에 입력하면 색깔이 바뀝니다!



The screenshot shows a Jupyter Notebook interface. At the top, there is a play button icon. Below it, a code cell contains the line `1 3D_movie_theatre = 0`. The variable name `3D_movie_theatre` is underlined with a red wavy line, indicating an error. Below the code cell, the output area shows the error message: `File "<ipython-input-3-6b195a796bb4>", line 1`
`3D_movie_theatre = 0`
`^`
`SyntaxError: invalid decimal literal`

변수명 짓기?

- 이 단어로 변수명을 지으면 안 된다는 걸 미리 알아야겠죠?

```
[ ] 1  try = 0 # 시도 횟수
      2
      3  try += 1
      4  print(try)
```

```
File "<ipython-input-2-33aa33248d34>", line 1
    try = 0 # 시도 횟수
      ^
SyntaxError: expected ':'
```

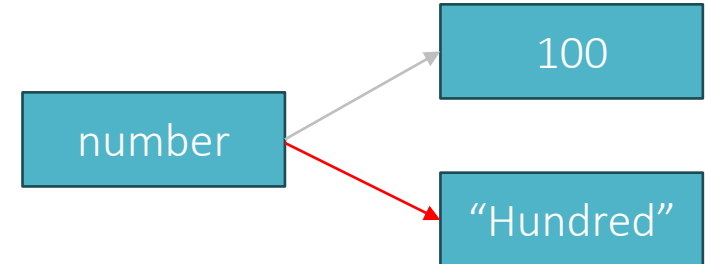
```
[ ] 1  and, as, assert, break, class, continue, def, del, elif, else, except,
      2  finally, for, from, global, if, import, in, is, lambda, nonlocal, not,
      3  or, pass, raise, return, try, while, with, yield, False, None, True
```

잠깐! 포인터 험오를 멈춰주세요...

- 파이썬의 변수는 C언어의 변수보다는 복잡하게 작동합니다.
- 변수는 실제로는 객체의 주소 값을 저장하며, 값을 불러올 때 주소 값을 통해 참조합니다. 그런 이유에서 파이썬은 변수 선언 시 타입 지정이 필요없는 것입니다.
- 만약 이런 코드가 있다면 중간에 '100' 객체는 필요 없어져서 자동으로 삭제됩니다.

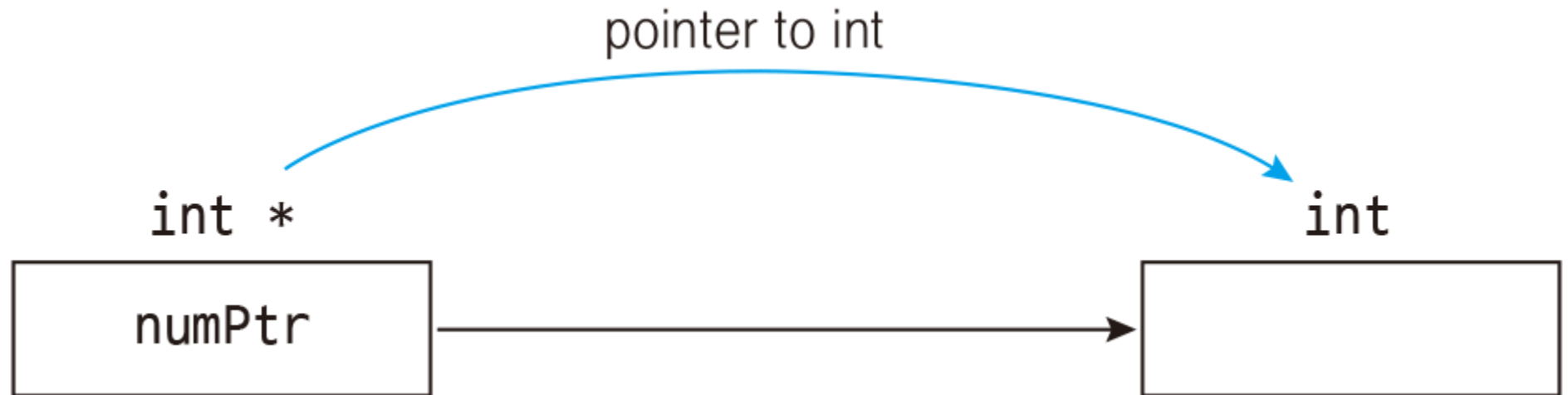


```
number = 100 # 정수 100 대입  
number = "Hundred" # 문자열 "Hundred" 대입
```



잠깐! 포인터 험오를 멈춰주세요...

- 이렇듯 파이썬 변수는 객체를 직접 저장하지 않고, 그저 객체를 가리키는 방식을 사용합니다.
- 파이썬 개발 시에 아주 많이 신경써야 하는 부분은 아닙니다.
- 하지만 **프로그래밍을 공부하는 사람이라면 이런 것들을 두려워하지 않으셨으면 좋겠습니다!**



리스트는 변수를 여러 개 묶은 것

- 리스트는 값을 여러 개 묶어서 저장하는 역할을 합니다.
- 리스트는 수학의 집합이나 수열과 비슷한 역할을 합니다.
- 리스트는 변수가 많을 때 이용하면 편리합니다.



```
x = [0, 1, 1, 2, 3, 5, 8, 13, 21, ...]
```



```
x_0 = 0  
x_1 = 1  
x_2 = 1  
x_3 = 2  
x_4 = 3  
x_5 = 5  
x_6 = 8  
x_7 = 13  
...
```

리스트는 변수를 여러 개 묶은 것

- 리스트 각각의 값을 리스트의 원소(element, entry)라고 부르고, 그 값이 몇 번째 위치에 있는지를 인덱스라고 부릅니다.
- **중요:** 파이썬 리스트의 인덱스는 **1이 아니라 0부터 시작**합니다!
- 예시: 리스트 x의 3번째 인덱스에 있는 원소는 2이다.
(x의 3번째 원소는 2이다.)
(x의 3번째 인덱스에는 2가 있다.)



```
x = [0, 1, 1, 2, 3, 5, 8, 13, 21, ...]
```

파이썬 리스트의 최대 장점

- 파이썬 리스트 원소는 타입이 상관 없습니다.
- 파이썬 리스트는 길이를 우리 맘대로 변경할 수 있습니다! (동적 배열)
- 원소의 타입이 모두 같아야 한다거나, 리스트의 길이를 미리 정할 필요가 없습니다.
- 이런 괴물도 허용됩니다!



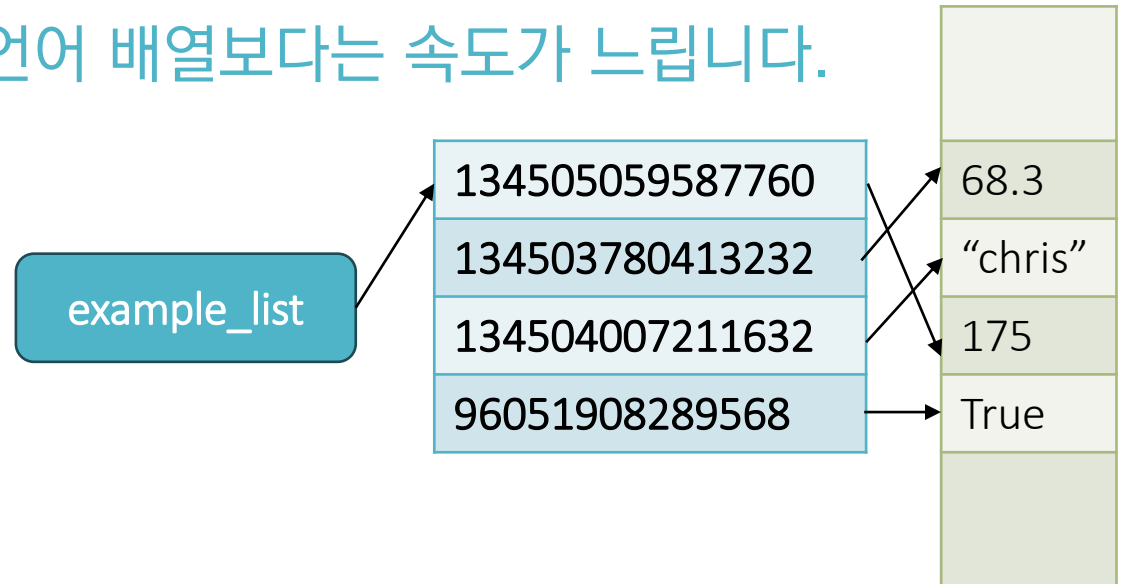
```
example_list = [175, 68.3, "chris", True]
```

리스트나 이중 포인터나 거기서 거기

- 이런 것이 가능한 이유는 파이썬 리스트가 각 값의 주소만을 저장하기 때문입니다.
(변수랑 똑같죠?)
- 그래서 `example_list[0]`와 같이 접근하면 그 주소로 접근해서 값을 참조하는 것입니다.
- 이는 사실 이중 포인터가 사용된 것인데, C언어 배열보다는 속도가 느립니다.

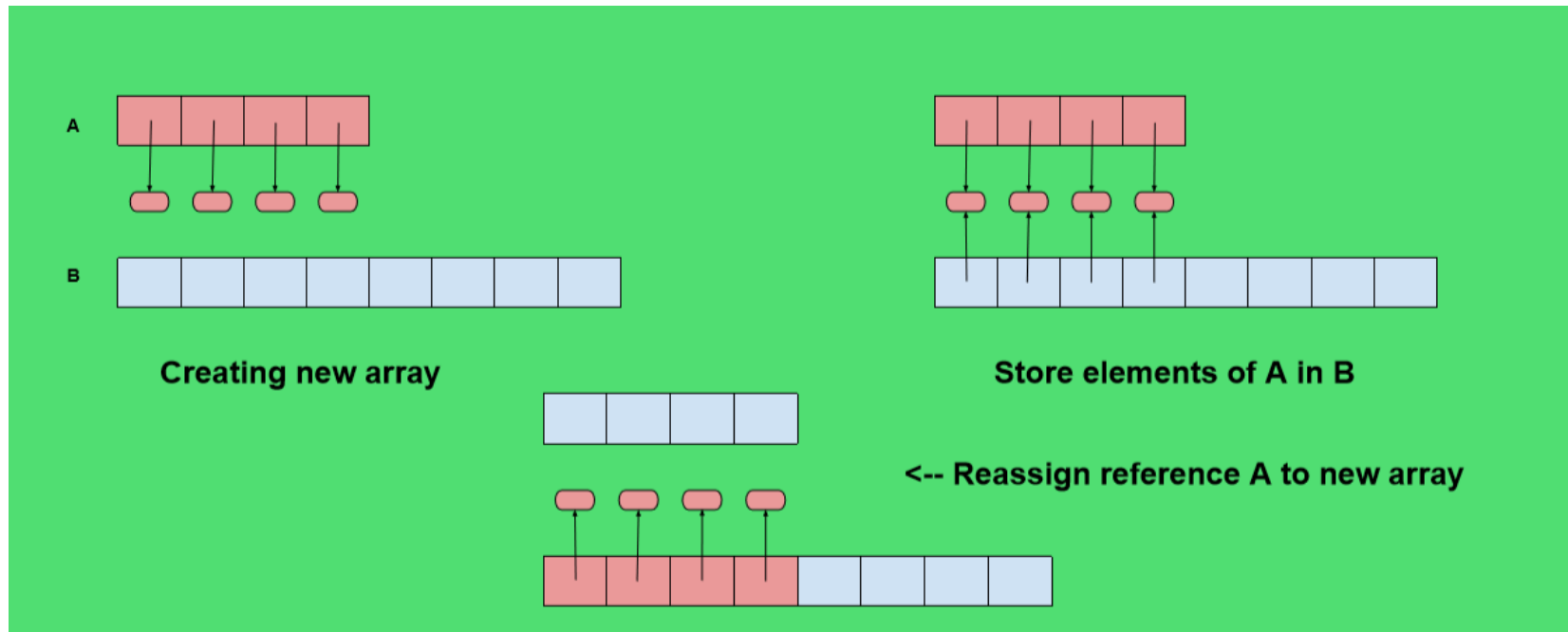


```
example_list = [175, 68.3, "chris", True]
```



원피스 루피 팔도 아닌 것이...

- 또한 파이썬 리스트는 내부적으로 동적 할당이 자동적으로 이루어집니다.
- 더 많이 저장하기 위해 확장하거나, 메모리 절약을 위해 축소가 이루어집니다.
- 파이썬 리스트는 속도가 느린 대신 개발자가 편리합니다.



실제적인 리스트 조작에 대하여...

- 파이썬 리스트와 관련하여 정말 기능이 많지만, 자주 쓰는 것만 빠르게 훑고 넘어가겠습니다.
- 지금 다 외우려면 체해요! 천천히 친해져도 안 늦어요.
- 이 파트는 시간 상 빠르게 넘어가겠습니다.
- 처음에는 `len()`, `append()`, `max()`, `min()`, `sum()`, `sorted()`만 알면 될 것 같아요.

실제적인 리스트 조작에 대하여... (1)

- 빈 리스트 생성과 리스트의 길이에 대한 것입니다. `len()`은 2주차에도 등장했습니다.

```
[ ] 1 # 빈 리스트를 생성해서 원하는 것들을 하나씩 저장하거나 삭제할 수 있습니다.  
    2  
    3 a = []  
    4  
    5 print(a)
```

```
[ ] []
```

```
[ ] 1 # len()은 리스트의 길이를 알려줍니다.  
    2  
    3 a = ["철수", "영희", "맹구", "훈이", "유리", "짱구"]  
    4 print(len(a)) # 인원 수
```


실제적인 리스트 조작에 대하여... (2)

- `append`(자주 씬)와 `extend`로 리스트에 원소를 추가합니다.

```
[ ] 1 # append로 리스트에 추가할 수 있습니다.  
2  
3 a = ["철수", "영희", "맹구", "훈이", "유리", "짱구"]  
4 a.append("흰둥") # 새로운 인원 추가  
5  
6 print(a)  
7 print(len(a))
```

```
['철수', '영희', '맹구', '훈이', '유리', '짱구', '흰둥']  
7
```

```
[ ] 1 # extend로 리스트를 연결할 수 있습니다.  
2  
3 a = [1, 2, 3, 4, 5]  
4 b = [6, 7, 8, 9, 10]  
5  
6 a.extend(b)  
7 print(a)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

실제적인 리스트 조작에 대하여... (3)

- 리스트의 **덧셈**과 **곱셈**은 concatenation의 의미입니다.

```
[ ] 1  # 파이썬 리스트의 덧셈은 extend와 비슷합니다.  
2  
3  a = [1, 2, 3]  
4  b = [5, 6, 7]  
5  c = a + b  
6  print(c)  
7  
8  # 곱셈은 덧셈을 반복하는 것을 뜻합니다.  
9  c = a * 5  
10 print(c)
```

```
[1, 2, 3, 5, 6, 7]
```

```
[1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]
```

실제적인 리스트 조작에 대하여... (4)

- 리스트의 `count()`와 `index()`로 개수, 위치를 찾을 수 있습니다.

```
[ ] 1 # count로 개수를 셀 수 있습니다.  
2  
3 a = ["찬성", "찬성", "반대", "찬성", "반대", "찬성",  
4      "반대", "찬성", "찬성", "반대", "반대", "찬성"]  
5 print(a.count("찬성"))  
6 print(a.count("반대"))
```

```
7  
5
```

```
[ ] 1 # index로 인덱스를 찾을 수 있습니다. (0번째부터!) 없으면 -1을 반환합니다.  
2  
3 a = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31]  
4 a.index(17)
```

```
6
```

실제적인 리스트 조작에 대하여... (5)

- 리스트의 `max()`, `min()`, `sum()`을 이용하면 굉장히 편리합니다.

```
[ ] 1  # max, min, sum을 이용해서 다 더하거나 최대값, 최소값을 찾을 수 있습니다.  
    2  
    3  a = [260, 255, 270, 280, 275, 255, 260, 255, 270, 270, 265]  
    4  print(max(a), min(a), sum(a))
```

```
280 255 2915
```

실제적인 리스트 조작에 대하여... (6)

- 리스트의 **인덱스 슬라이싱**입니다. 음수 인덱싱은 **르ㅇ** 혜자 기능입니다.

```
1  # 리스트의 인덱스 슬라이싱 -> 지금 당장 다 알 필요는 없습니다. 저도 간혹 실수합니다.
2
3  a = [4, 5, 6, 7, 8]
4
5  print(a[0]) # 0번째 인덱스 잘라오기
6  print(a[1:3]) # 1번째부터 3 "미만"의 인덱스까지 잘라오기
7  print(a[1:]) # 1번째부터 마지막 끝까지 잘라오기
8  print(a[-1]) # -1로 인덱싱하면 리스트의 마지막 원소에 접근할 수 있습니다.
9  print(a[:-1]) # 0번째부터 마지막 인덱스 미만까지 잘라오기
10 print(a[1:-1]) # 1번째부터 마지막 인덱스 미만까지 잘라오기
11 print(a[::-2]) # 0번째부터 두칸씩 띄엄띄엄 출력
12 print(a[::-1]) # 거꾸로 출력
```

```
4
[5, 6]
[5, 6, 7, 8]
8
[4, 5, 6, 7]
[5, 6, 7]
[4, 6, 8]
[8, 7, 6, 5, 4]
```

실제적인 리스트 조작에 대하여... (7)

- 리스트의 인덱싱을 활용한 예시입니다.

```
1  # 리스트 인덱싱으로 리스트를 수정할 수 있습니다.  
2  
3  a = [4, 5, 6, 7, 8]  
4  a[0] = 3  
5  a[2] -= 1 # a[2]의 값을 1 빼기  
6  a[4] += 1 # a[4]의 값을 1 더하기  
7  print(a)
```

```
[3, 5, 5, 7, 9]
```

```
[ ] 1  # 인덱싱 활용 예시  
2  
3  a = [4, 5, 6, 7, 8]  
4  print(a[2] + a[4])
```

실제적인 리스트 조작에 대하여... (8)

- 코딩하다가 이런 에러가 나면 인덱스를 벗어난 것입니다.

```
[6] 1  # 인덱스를 벗어나면 에러가 떠요!  
    2  
    3  a = [4, 5, 6, 7, 8]  
    4  print(a[100])
```

```
-----  
IndexError                                Traceback (most recent call last)  
<ipython-input-6-d5db0f20b85e> in <cell line: 4>()  
      2  
      3  a = [4, 5, 6, 7, 8]  
----> 4  print(a[100])  
  
IndexError: list index out of range
```

실제적인 리스트 조작에 대하여... (9)

- `del, remove()`을 이용해 리스트에서 원소를 삭제할 수도 있습니다.

```
1 # 삭제
2
3 a = [5, 6, 7, 8, 9, 10]
4 del a[3]
5 print(a)
```

[5, 6, 7, 9, 10]

```
[4] 1 # remove를 이용한 삭제
2
3 a = [5, 6, 7, 8, 9, 10]
4 a.remove(8)
5
6 print(a)
```

[5, 6, 7, 9, 10]

실제적인 리스트 조작에 대하여... (10)

- `sort()`, `sorted()`를 이용해 리스트를 정렬할 수도 있습니다.

```
▶ 1 # 리스트 정렬하기
   2
   3 a = ['arsenal', 'chelsea', 'man city', 'man united', 'liverpool']
   4 a.sort()
   5 print(a)
```

```
↳ ['arsenal', 'chelsea', 'liverpool', 'man city', 'man united']
```

```
[ ] 1 # 리스트 정렬하기 - 계속
     2
     3 a = ['arsenal', 'chelsea', 'man city', 'man united', 'liverpool']
     4 print(sorted(a)) # a.sort()는 a가 참조하는 객체를 바꾸지만 sorted(a)는 적용한 이후에도 a의 원본을 바꾸지 않습니다.
```

```
['arsenal', 'chelsea', 'liverpool', 'man city', 'man united']
```

실제적인 리스트 조작에 대하여... (11)

- `sort()`, `sorted()`의 `key` 속성을 이용해 기준에 맞춰 리스트를 정렬할 수도 있습니다.

```
1  # 참고: 리스트 정렬하기 - 승점에 따라 정렬
2
3  a = [['arsenal', 56], ['chelsea', 66], ['man city', 81], ['man united', 66], ['liverpool', 99]]
4  a.sort(key = lambda x: (x[1], x[0]))
5  print(a)
```

```
➞ [['arsenal', 56], ['chelsea', 66], ['man united', 66], ['man city', 81], ['liverpool', 99]]
```

인덱스 슬라이싱 주의 사항

- 파이썬 인덱스 슬라이싱에서 `list[a:b]`라고 하면 그 결과는 `[list[a], list[a+1], ..., list[b-1]]`라고 쓸 수 있습니다.
- 위의 결과에 `list[b]`는 포함되지 않습니다.

화석 김현규의 끈대질 시간...

- 급체 주의! 아까 나열한 거 지금 다 외우려고 하면 체합니다.
- 혹시 다 삼킨 분? 빨리 토하세요..ㅋㅋㅋ
- 저런 것들은 써보면서 자연스레 익숙해지는 것이 제일 편해요
- 잘 외우는 사람보다는 구글링을 잘 하는 사람이 되면 개꿀이에요
- 이번 한 시간으로 끝내지 말고 이것저것 구현해봐요! 백준 쉬운 문제도 좋고...

특히 조건문에서 요긴하게 쓰이는 기능

- `in` 키워드를 이용해 포함 관계를 살펴볼 수 있습니다.
- 출력 값은 True 아니면 False입니다. (boolean 형)



```
good_men_list = ["Steven Gerrard", "John Terry", "Kun Aguerro", "Lionel Messi"]  
  
print("Ronaldo" in good_men_list)
```

특히 반복문에서 요긴하게 쓰이는 기능

- 파이썬 반복문에서 `range()`를 잘 쓰면 아주 좋습니다. 곧 다시 보게 될 것입니다.

ex) for i in `range(10)`: print(i) → i = 0, 1, 2, ... , 9까지 총 10번 반복

ex) for j in `range(1, 11)`: print(j) → j = 1, 2, 3, ... , 10까지 총 10번 반복

ex) for k in `range(1, 11, 2)`: print(k) → k = 1, 3, 5, 7, 9까지 반복

ex) for l in `range(10, 0, -1)`: print(l) → l = 10, 9, 8, 7, 6, 5, 4, 3, 2, 1까지 반복

특히 반복문에서 요긴하게 쓰이는 기능

- 눈 여겨볼 점은 range() 사용 자체만으로도 귀찮음을 줄일 수 있다는 것입니다.
- [2, 4, 6, 8, 10, 12, ..., 100]을 만들어야 할 때 일일이 입력하거나 반복문을 쓰지 않고도 range(2, 102, 2)를 쓰는 것으로 충분하기 때문입니다.
- range(1, 10) 자체만으로 리스트는 아니고, list(range(1, 10))처럼 list()로 감싸주어야만 리스트가 됩니다.
- 주의: range(1, 10)에서는 1, 2, ..., 9까지만 포함되고 10은 포함되지 않습니다.
(리스트의 인덱스 슬라이싱과 비슷한 느낌)

간단히만 살펴보는 이차원 리스트

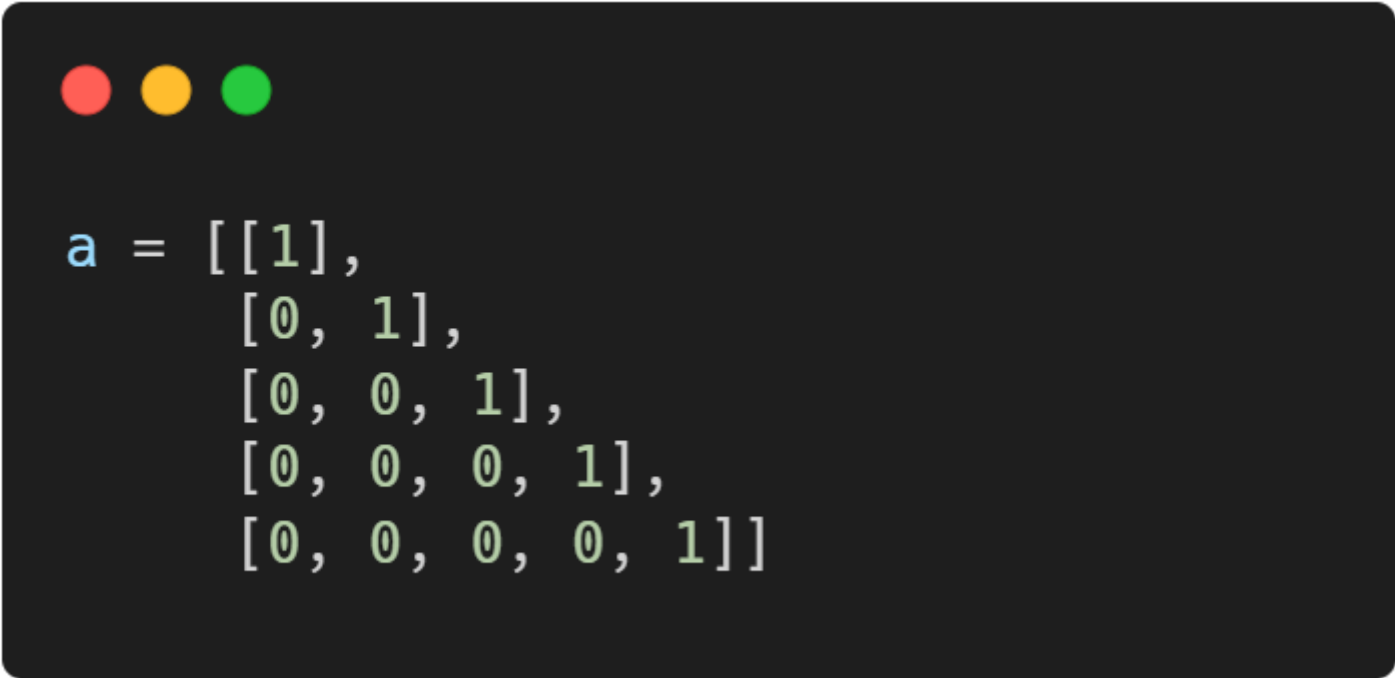
- 리스트의 원소에 리스트가 들어간 형태입니다.
- 행렬처럼 다룰 수 있습니다. (엑셀 데이터, 이미지 처리 시에 활용)
- 가로와 세로를 알아보기 쉽도록 아래처럼 두 가지 방법으로 쓰기도 합니다.

```
a = [[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 0, 1]]
```

```
a = [[1, 0, 0, 0],  
      [0, 1, 0, 0],  
      [0, 0, 0, 1]]
```


간단히만 살펴보는 이차원 리스트

- 참고: 꼭 직사각형 모양이 아니라 요란한 모양으로 만들어도 됩니다.



```
a = [[1],  
      [0, 1],  
      [0, 0, 1],  
      [0, 0, 0, 1],  
      [0, 0, 0, 0, 1]]
```

간단히만 살펴보는 이차원 리스트

- 이차원 리스트도 일차원 리스트와 똑같은 연산을 쓸 수 있습니다.

```
[ ] 1  a = [[1, 0, 0, 0],  
2      [0, 1, 0, 0],  
3      [0, 0, 0, 1]]  
4  
5  a[2][2] += 3  
6  a.append([1, 2, 3, 4])  
7  
8  print(a)
```

```
[[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 3, 1], [1, 2, 3, 4]]
```

리스트 컴프리헨션

- 마지막입니다!
- 리스트의 element에 여러 개를 규칙적으로 넣어서 생성하고 싶은데, range()만 가지고는 구현할 수 없을 때 쓸 수 있습니다.
- 리스트 안에 for 키워드가 들어가 있는데, 잘 쓰면 실무에서도 좋지만 코테에서도 아주 좋습니다.

```
1 str_list = [i**2 for i in range(11)] # 제곱수들의 리스트는 range()만으로 만들기 힘들겠죠?  
2 str_list
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

리스트 컴프리헨션

- 리스트 컴프리헨션은 반복문보다 조금 더 빠릅니다.
- 다음은 구구단표를 리스트 컴프리헨션으로 작성한 것입니다.

```
[ ] 1  # 중첩 반복문을 배우시면 이해가 더 빠르시겠지만, 이런 것도 만들 수 있다는 걸 참고해봐요!  
    2  
    3  nine_by_nine = [[x*y for x in range(1, 10)] for y in range(2, 10)]  
    4  nine_by_nine
```

```
[[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]]
```

다음 시간에 잠시 발표할 사람?

- 두 변수를 바꾸는 간단한 프로그램 또는 함수를 작성하고 설명해보세요.
- 단, 아래 방식은 반칙!
 $x, y = y, x$

우리 스터디에서 다룰 주요 주제

- 파이썬 조감도
- 변수와 리스트
- 문자열, 튜플, 딕셔너리
- 조건문과 반복문
- 함수
- 클래스와 상속
- 파이썬을 이용한 문제 해결 (간단한 알고리즘 문제들) – 미정
- 파이썬 라이브러리로 할 수 있는 일 (이미지 처리, 엑셀 데이터 처리, 그래프 그리기, ...)

다음 시간에 다룰 주제

- 문자열, 튜플, 딕셔너리

