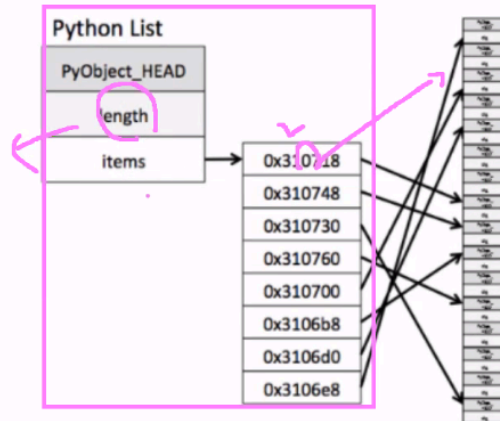


## Day 4 - Python Collection Type (List)

리스트:

여러 값들을 하나의 변수에 저장하고 관리할 수 있게 해주는 순차적인 자료구조입니다.



**List contains:**

1. item's type
2. items
3. length
4. values

**Array**

1. 데이터를 저장하기 전 크기를 미리 지정을 해야한다.
2. 데이터를 저장하다 칸이 모자라면 배열을 늘리지 못한다.
  - a. 더 큰 배열을 만들고 복붙을 한다. (비효율적)
3. 같은 타입만 넣을 수 있다.

따라서, 배열을 더 편리하게 만든것이 리스트이다.

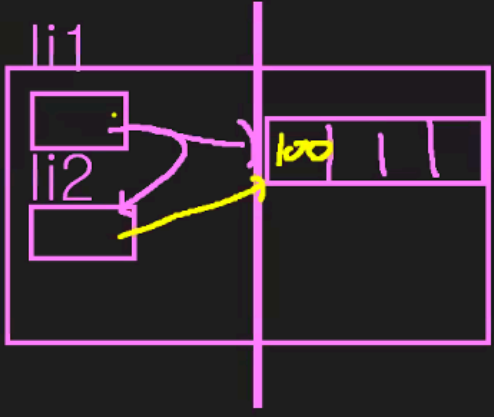
**List**

1. Mutable 하다 (크기 조절가능)
2. 훨씬 더 현대적으로 바뀐 자료구조 형태이다.
3. 다른 타입도 넣을 수 있다.

## 리스트 슬라이싱:

```
li1 = [10, 20, 30, 40, 50]
print(li1)
print(li1[0:3])

li2 = li1
print(li2)
li2[0] = 100
print(li2)
print(li1)
```



```
[10, 20, 30, 40, 50]
[10, 20, 30]
[10, 20, 30, 40, 50]
[100, 20, 30, 40, 50]
```

li1 = [10, 20, 30, 40, 50]

- li1은 리스트를 가르키고 있다.

li2 = li1

- li1, 2는 같은 리스트를 가르키고 있다.

li2[0] = 100

- index 0를 100으로 바꿨을때 li1, li2 둘다 같은 메모리 리스트를 가리키고 있으므로 둘다 변형된 리스트를 출력한다: [100, 20, 30, 40, 50]

## 리스트 변경:

### ▼ 4. 변경 가능

리스트의 항목들은 변경할 수 있습니다. 즉, 리스트의 항목들을 수정, 추가, 삭제할 수 있습니다.

```
[11] li4 = [10, 20, 30, 40, 50]
# 슬라이싱을 이용하여 요소를 추가한 경우 리스트에 데이터만 포함
li4[1:2] = ['🍕', '🍕', '🍕', '🍕']
print(li4)
```

```
🔄 [10, '🍕', '🍕', '🍕', '🍕', 30, 40, 50]
```

```
▶ li4 = [10, 20, 30, 40, 50]
# 인덱싱을 이용하여 요소를 추가한 경우 리스트 안에 리스트를 만들고 포함
li4[1] = ['🍕', '🍕', '🍕', '🍕']
print(li4)
```

```
🔄 [10, ['🍕', '🍕', '🍕', '🍕'], 30, 40, 50]
```

### 슬라이싱을 이용

- 차원이 유지된다.
- 같은 차원에서 리스트 안에 삽입된다.

### 인덱싱을 이용

- 뽑아낼때는 차원을 벗어나고, 삽입할때는 차원을 입힌다([ ]).
- 스칼라 값이 나온다 (단일값)

리스트에 새 요소 삽입:

### 1. 하나의 요소

- `append(100)`

```
[33] # append(): 리스트 요소의 끝에 새로운 요소를 추가
      li6 = [10, 20, 30]
      print(li6)

      li6.append(100)
      print(li6)
      # li6.append(200, 300)
      li6.append([200, 300])
      print(li6)
```

```
☞ [10, 20, 30]
   [10, 20, 30, 100]
   [10, 20, 30, 100, [200, 300]]
```

### 2. 하나 이상의 요소

- `.extend([200, 300, 400])`

```
] # extend(): 리스트 요소의 끝에 새로운 여러 요소를 추가
   li6 = [10, 20, 30]
   print(li6)

   # li6.extend(100)
   li6.extend([100])
   print(li6)
   li6.extend([200, 300, 400])
   print(li6)
```

```
✓ [10, 20, 30]
   [10, 20, 30, 100]
   [10, 20, 30, 100, 200, 300, 400]
```

### .pop()

```
# pop(): 리스트 마지막 요소를 삭제하고 삭제된 요소를 반환
li6 = [10, 20, 30]
print(li6)

# print(li6.pop())
temp = li6.pop()
print(li6)
print(temp)
```

[10, 20, 30]  
[10, 20]  
30

- 특이하게 삭제된 요소도 반환시킬 수 있다.

### .index()

```
# index(): 리스트에서 특정 요소의 값 인덱스를 반환
li6 = [10, 20, 30]
print(li6)

print(li6.index(30))
# print(li6.index(100)) # 값이 없으면 에러
```

[10, 20, 30]  
2

- index(30)
- 30의 값을 넣으면 리스트 안에서 30을 찾은뒤 index값을 반환시켜준다.
- 없는경우 에러를 발생시킨다.

### .reverse()

```
# reverse(): 리스트의 요소들의 순서를 반대로 설정
li7 = [100, 50, 70, 60, 20]
print(li7)

li7.reverse()
print(li7)
```

[100, 50, 70, 60, 20]  
[20, 60, 70, 50, 100]

[ ] 코딩을 시작하거나 시로 코드를 선택하세요.

### .reverse 없이 값을 뒤집는법:

```
li7 = [100, 50, 70, 60, 20]
print(li7[::-1])          # step이 -1일 경우 start의 기본값은 -1, stop의 기본값은 0으로 설정
print(li7[-1::-1])
print(li7[-1:-6:-1])
```

```
[20, 60, 70, 50, 100]
[20, 60, 70, 50, 100]
[20, 60, 70, 50, 100]
```

- li[::-1] 로 해결가능

### .sort() (method)

```
6] # sort(): 리스트의 요소를 오름차순으로 정렬
li7 = [100, 50, 70, 60, 20]
print(li7)

li7.sort()
print(li7)

# sort(reverse=True): 리스트의 요소를 내림차순으로 정렬
li7.sort(reverse=True)
print(li7)

li8 = ['Apple', 'apple', 'orange', 'banana', 'melon']
li8.sort()
print(li8)

li9 = ['김사과', '오렌지', '반하나', '이메론', '배애플']
li9.sort()
print(li9)
```

```
[100, 50, 70, 60, 20]
[20, 50, 60, 70, 100]
[100, 70, 60, 50, 20]
['Apple', 'apple', 'banana', 'melon', 'orange']
['김사과', '반하나', '배애플', '오렌지', '이메론']
```

- 오름차순이 default
- 리스트 전용함수
- inplace method
  - 자기가 자기한테 덮어씌우는 함수

## sorted()

```
# sorted(): 모든 요소를 정렬한 후 반환해주는 함수
li7 = [100, 50, 70, 60, 20]
print(li7)

result = sorted(li7)
print(li7)
print(result)
```

- 파이썬에서 기본적으로 제공
- 튜플, 딕셔너리 등 다른 자료구조형에서도 사용할 수 있다.
- 변수로 저장이 가능하며, 반환해준다.
- 덮어쓰워 주는게 아니라 반환해준다.

```
# sorted(): 모든 요소를 정렬한 후 반환해주는 함수
li7 = [100, 50, 70, 60, 20]
print(li7)
temp = sorted(li7)
print(temp)
print(li7)
```

```
[100, 50, 70, 60, 20]
[20, 50, 60, 70, 100]
[100, 50, 70, 60, 20]
```

## .count()

```
[107] # count(): 리스트에서 특정 요소의 갯수를 반환
li9 = [10, 20, 30, 50, 20, 40, 30, 20]
print(li9)
print(li9.count(20))
print(li9.count(100))
```

```
[10, 20, 30, 50, 20, 40, 30, 20]
3
0
```

```
li10 = ['a', 'aa', 'aaa', 'b', 'bb', 'A', 'AA']
print(li10.count('aa'))
```

```
1
```

- .count(x)
- 특정 요소의 갯수를 반환시켜주며, 문자열, integer 둘다 가능하다.