= 저장에 대한 이해

- 사실 박스안에 값을 저장한다라고 했지만, 엄밀하게 말해서는 해당하는 객체의 주소가 담기게 되는 것.

immutable의 대표인 숫자!

In [ ]:
```
a = 210125
```

In [ ]:
```
b = a
```

In [ ]:
```
print(a, b)
print(id(a), id(b)) # id가 동일하다.
```

- 기존의 숫자 객체를 바꿀 수 있느냐... 없다.. 그래서 할 수 없다.

In [ ]:
```
a[0] = 100
```

- 아래의 코드는 아예 다른 객체를 **재할당** 한 것임.

In [ ]:
```
# 실제 해당하는 주소에 있는 210125를 바꿀 수 있는 방법은 전혀 없음.
# 그러므로, 이 것은 새로운 숫자의 새로운 주소를 b에 할당한 것!
b = 1004
```

In [ ]:
```
print(a, b)
print(id(a), id(b)) # 아예 다른 것으로 할당되었으니 주소도 다름
```

mutable의 대표인 리스트!

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

In [ ]:
```
b = a
```

In [ ]:
```
print(a, b)
print(id(a), id(b)) # id가 동일하다.
```

- 여기서 기존의 리스트를 변경할 수 있다. mutable하니까!

In [ ]:
```
a[0] = 'A'
print(a, b)
print(id(a), id(b)) # id가 동일하다.
```

- 리스트도 당연히 재할당하면 id가 달라짐

In [ ]:
```
b = [3, 2, 1]  # 새로운 리스트를 할당함.
print(a, b)
print(id(a), id(b)) # 당연히 id가 다름
```

정리

- 두 `b = a` 의 코드에서, immutable / mutable 모두 주소를 저장을 했지만,
- mutable 한 경우에는 변경이 가능하므로, 해당 객체(숫자 혹은 리스트)를 일부 바꿔서(인덱스 0의 값 변경) a와 b 모두 변경되는 것 처럼 보임.
- immutable 한 경우에는 변경이 불가능하므로, 해당 객체(숫자 혹은 리스트)를 **바꿀 방법이 없는 것**
- **두 상황 모두 재할당하면 주소가 바뀌는 것은 맞음.**

shallow copy vs deep copy

- 지금까지 `b = a` 는 단순 복사!

- shallo copy와 deep copy의 차이점은 지금 단일의 리스트만 복사할 것이냐, 그 리스트의 내부의 모든 것을 복사할 것인가의 차이!

In [ ]:
```
# shallow copy
# 리스트 내부의 리스트는 주소로 여전히 존재
a = [[1, 2, 3], 2, 3]
b = list(a)
print(a, b)
b[0][0] = 100
print(a, b)
```

In [ ]:
```
# deep copy
# 내부의 모든 것을 값으로 복사
import copy
a = [[1, 2, 3], 2, 3]
b = copy.deepcopy(a)
print(a, b)
b[0][0] = 100
print(a, b)
```