Mutability and Immutability :

PART: Mutability and Immutability of Atomic Data Types:

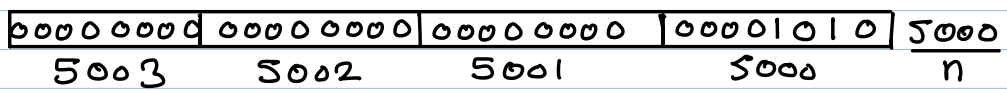
Mutation: To change from within.

Data type is mutable if value component of its objects can be changed after their creation.

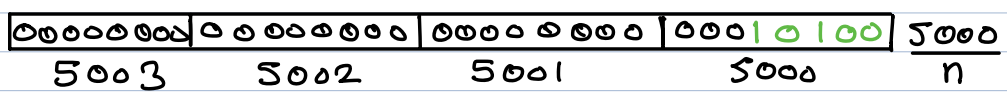
Data type is immutable if value component of its objects can't be changed after their creation.

Mutability of int in C programming language.

int n = 10; 1010 → binary



n = 20 10100 → binary



This behaviour is possible because int is a mutable type in C.

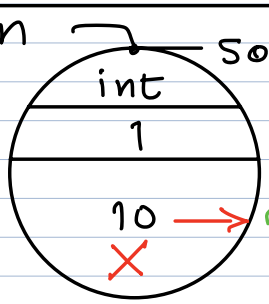
---

n = 10

n = 20

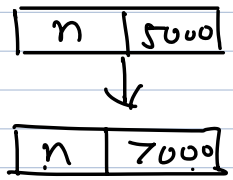
print (n)

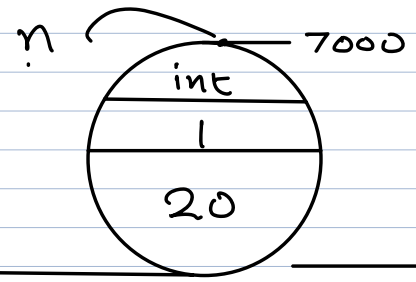
20



Not possible why?

because int is immutable in Python.





All atomic data types in Python are immutable.

n = 10 n = 20	n = 1.1 n = 2.2	n = False n = True	n = 'A' n = 'B'

Existing variable name

↓

left hand side of assignment

↓

existing object → UNBINDING.

new object → BINDING.

n = 10 id(n)

n = 100-90 id(n)

PART B: Mutability and Immutability of containers.

① Add new element to a container object.

② Replace an existing object in a container by other object.

③ Remove an existing object from a container.

Value component of container object

≡

Collection of id's (addresses) of the contained objects.

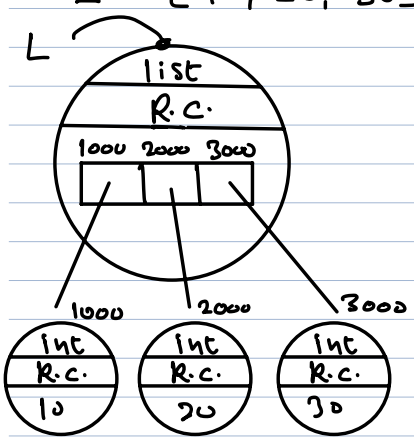
Mutable container class

- Such a container class whose object's value component (= collection of contained object's addresses) can be changed.

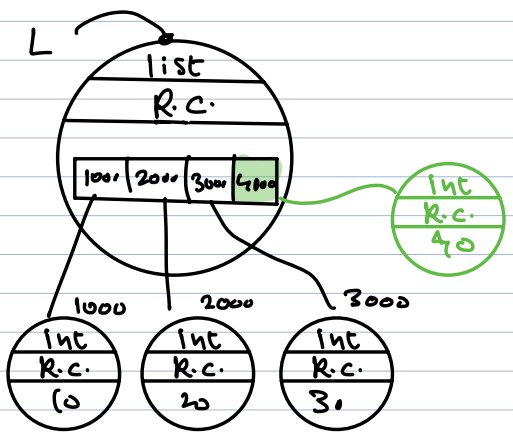
In case of list

L.append()	L[i] = rhs	del L[i]	L.remove()
L.extend()	L[i:j] = rhs	del L[i:j]	L.pop()
L.insert()	L[i:j:k] = rhs	del L[i:j:k]	L.clear()

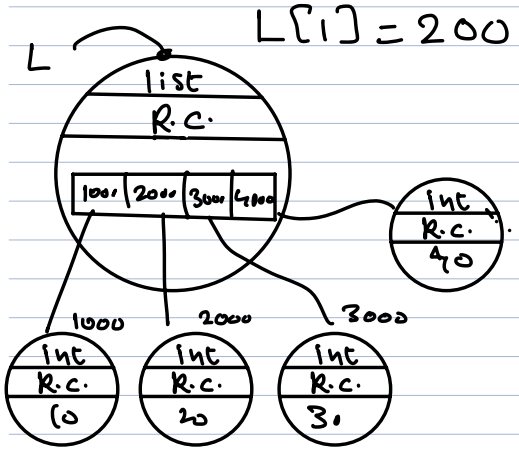
L = [10, 20, 30]



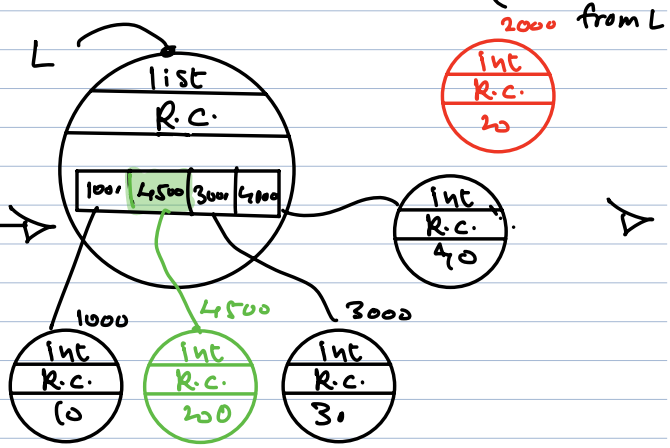
L.append(40)



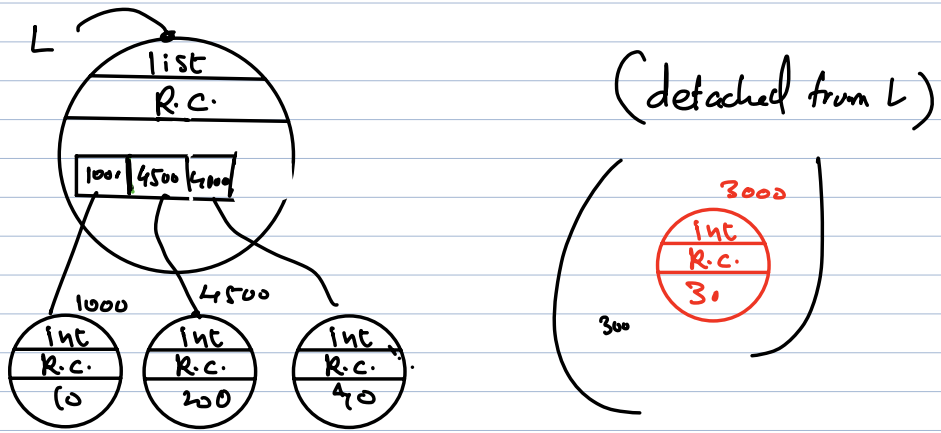
L[1] = 200



(detached from L)



L.remove(30)



dict / list → full mutability
add / replace / remove

Set

Set → size mutable
add / remove

5000: Existing Content → 5000: New_Content

Memory block → Memory block
size

IN-PLACE CHANGE.

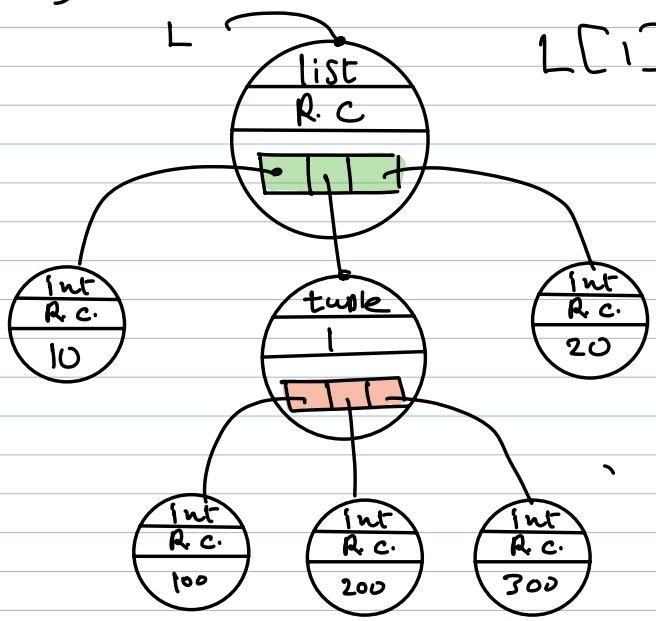
Python data value → NO IN-PLACE CHANGE

only for addresses → IN-PLACE CHANGE

PRINCIPLE : Once Mutable, Anywhere/Alway mutable

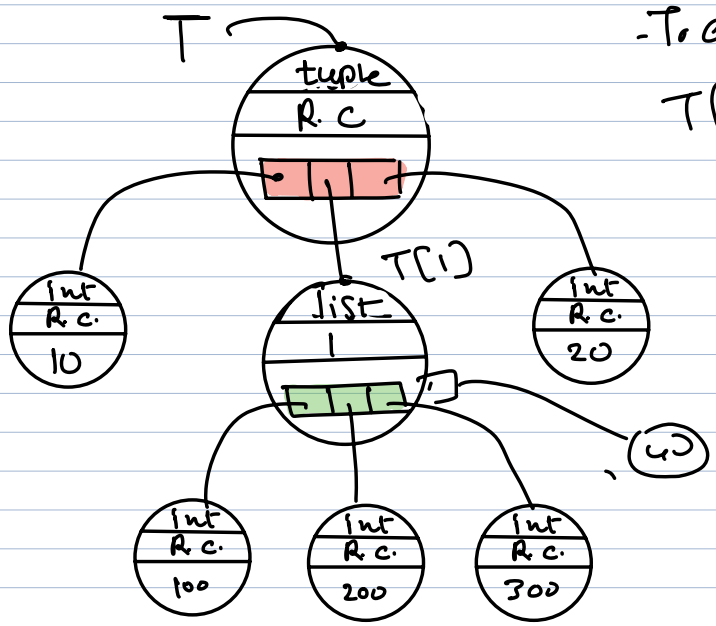
Once immutable, Anywhere / Always immutable.

L = [10, (100, 200, 300), 20]



L[1] = 500

T = (10, [100, 200, 300], 20)



- T. append (40)

T[1]. append(40)

```
T = (10, [100, 200, 300], 20)
print (T)
```

```
(10, [100, 200, 300], 20)
T[1].append(400)
print(T)
(10, [100, 200, 300, 400], 20)
```

Test (Session-1)

Shallow copy