Ruchika Shashidhara
NU 002245068

PSA - Class 1

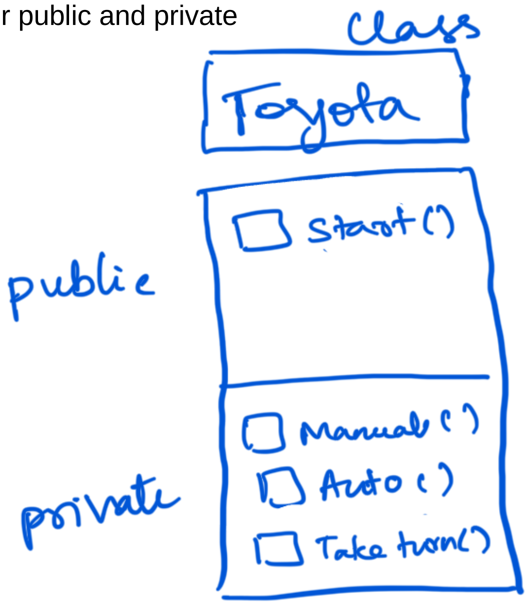
Introduction

- We need to create a class so that we can instantiate an object to use it.

- Toyota a;
  Toyota b;
  Toyota → class
  b → object

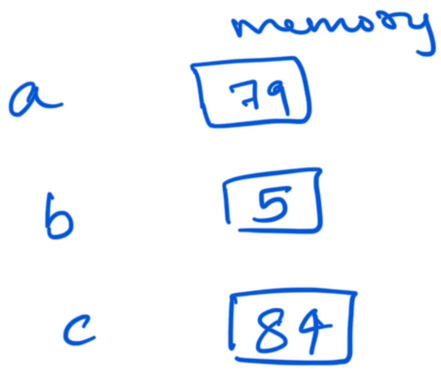
- We can use Toyota class to create multiple objects → a, b
  a.start();
  - only on object a
  - does not affect object b

Need for public and private



- exposed to the user
- should be intuitive

- not exposed to the user
- implementation is internal
- only known by dev



typed lang

ex: C++
Java

class → int
object → a

int a = 79;

int b = 5;

int c = a+b;

int → DS

+ → Algo

typeless lang

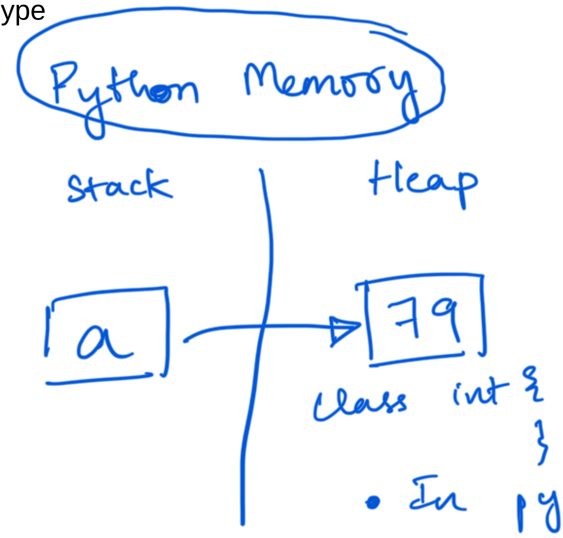
ex: Python

- In python we don't

need to assign a type

~~Int~~ a= 79

Basic Data Type



- Python assigns a type by looking at its RHS

79 → int small

'e' → char

79.79 → float

- In python, first 79 object is created and then a is pointed to 79 object

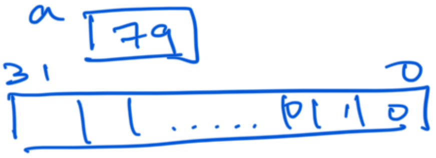
· uses 2 spaces

- In other lang, C++,Java, a space for object 79 is created called as 'a' with space allocated 4bytes for int

· Uses 1 space

C++ memory

max = $2^{32}-1$, for int

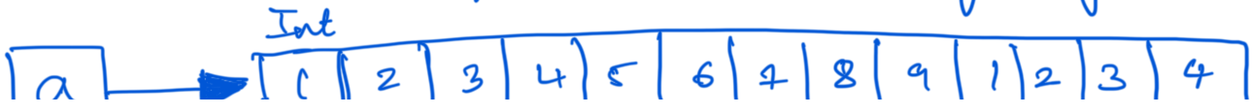


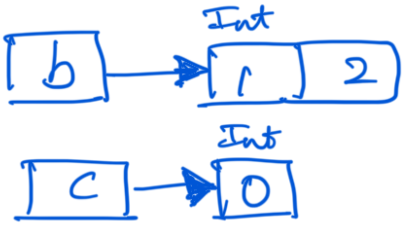
(Python support any size as 'a' is pointer class can be int, double, etc)

Int Class

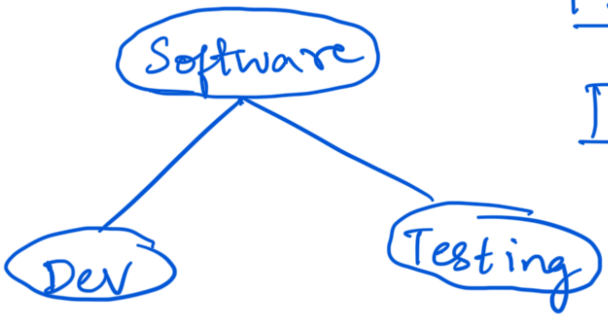
- Write a custom class to represent Big 'Int' so that large digits of a number can be represented & used using int DS & operated overloading Algo

ex:





- size a,b,c is the same
- Int on heap size can be small / big



Int.py

- make it work
- use OOP
- used by user

IntTest.py

- test all scenario
- doesn't need to ~~know~~ how it works, but needs to check that it works

Classes & Objects

In Python, main() is the start function, to perform OOP, we can try to control main()

ex: def main():

t = Inttest ()

t → object

Inttest () → class constructor

__ init__()

__ → reserved by Python lang

Other lang :

Inttest t = Inttest();

Inttest → class

t → object

Inttest(); → class constructor same as name of the class Inttest()

def __init__(self): → similar to 'this' in Ja in Java

- Python does not have concept of private, we can use single underscore _ for OOP
- __ double underscore is reserved by Python

```
class Inttest

    def __init__(self):
        self._why()


    def _why(self):
```

→ defined in same class Inttest that's why we need self

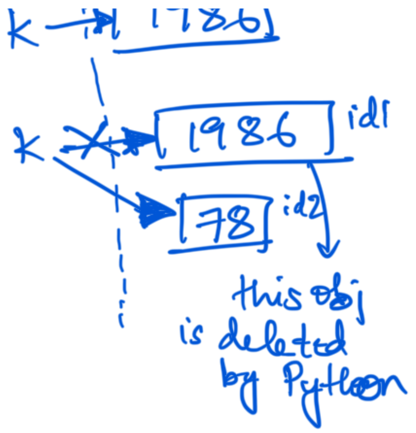
```
k = 1986
print (type(k))
 # output :   <class 'int'>
                  value of k = 1986
                        id = 18664578.....
```

Python has 2 spaces
- pointer k
- address that holds value of k
- lets you create value of any size

C++ / Java will have only 1 space address with value, no pointer

k = 1986

k = 78



Need for operator overoading

- When we create a custom class in Python we can override functions:

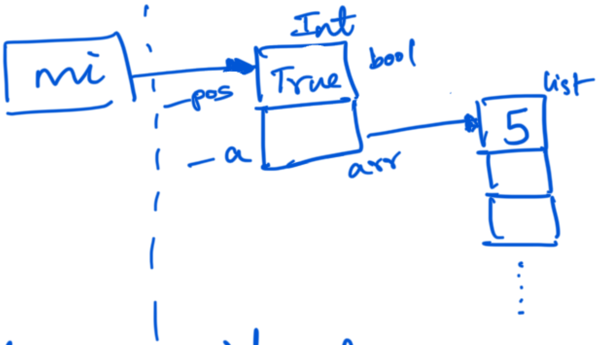
  +, −, *, /, //, >, <, ==, etc.

Data Structure of Int

i = 5

mi = Int (i)

- calls __init__(self, n) constructor of class Int



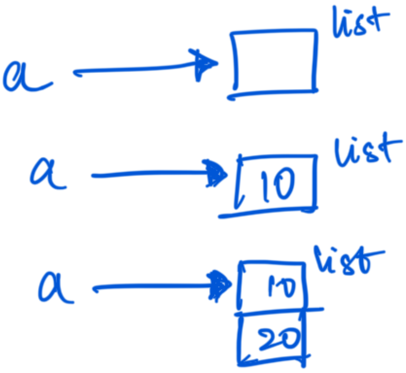
- In Python, 0 (zero) is also considered as a +ve num, −0 will throw an error, −0 cannot be represented. 0 is represented as +0

- Python list

  a = [ ]

  a.append(10)

  a.append(20)



Alg 1 : Find the position digit value of a number

ex: int i = 1986

l[2] = ? # won't compile

- First divide every time until we get 0
- reverse the list (use swapping left & right)

```
 0   1   2   3
| 6 | 8 | 9 | 1 |

 0   1   2   3
| 1 | 9 | 8 | 6 |
  ↑   ↑   ↑   ↑
  1   2   2   1
```

Convert int to Python list

```
def build (self, n: "Python int")  → required since python should know what kind of class it is of any size
        → "list of int")

if  n < 0
    n = -n   # if num is neg, make it pos
             (false is already set)
l = [ ]
if  n < 10:
    l.append (n)
else
    while n != 0:
        l.append (n % 10)
        n = n // 10
self._reverse (l)
return l
```

if
6
① [6]
reverse
[6]

else
1986
① 6
② 8
③ 9
④ 1

```
⇒  0   1   2   3
  | 6 | 8 | 9 | 1 |
```

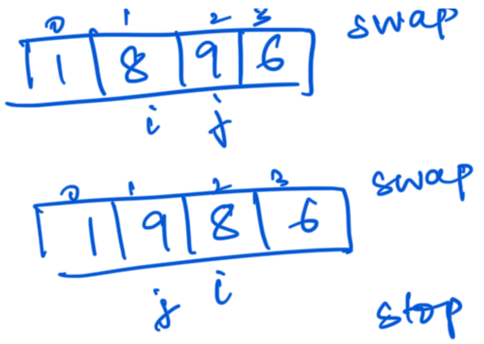
reverse

```
 0   1   2   3
| 1 | 9 | 8 | 6 |
```

How to reverse in-place Python List

```
def _reverse (self, l: "list of int")
        → "None" :

    i = 0
    j = len(l - 1)
    while  i < j : # checks until ptrs don't collide
        t = l[i]        }
        l[i] = l[j]     } # swaps val of pos of i & j
```

```
 0   1   2   3
| 6 | 8 | 9 | 1 |
  i           j
```

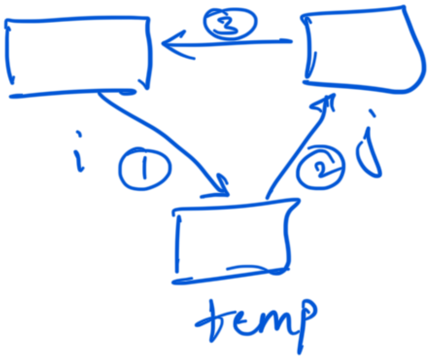


l[j] = t

i = i+1 # moves i right

j = j-1 # move j left

Swap :



Convert Python list to int

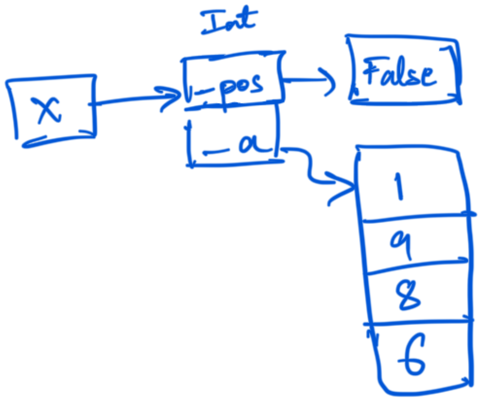
```
def int(self) -> "Python int":
    v = 0
    for e in self._a:
        v = 10 * v + e
    if v == 0 or self._positive:
        return v
    return -v
```

- 1 9 8 6

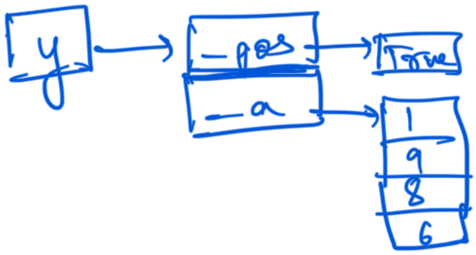
v = 1
v = 19
v = 198
v = 1986

return -v

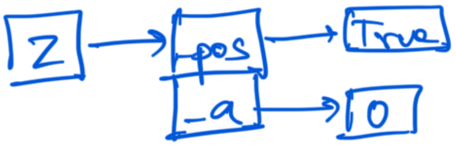
x = Int(-1986)



y = Int(1986)



z = x + y # +0



Need for operator overloading

Int is a custom class, we cannot directly use builtiin +, -, <, ==, etc. - we require operator overloading

overload :

d = Dog(23)

__init__(self, n)

d. print()

__str__()

```
def __str__(self):
    return ("-" if not self._positive else "") + "".join([str(digit) for digit in self._a])
```

overload

x = Int(1986)

__init__(self, n)

+1986   x = Int(1986)   __init__(self, n)

```
def __len__(self) -> "Python int":
        return len(self._a)
```

print(len(x)) # 4   __len__(self)

```
def __getitem__(self, pos: "Python int") -> "Python int":
n = len(self)
        assert pos >= 0 and pos < n
        return self._a[pos]
```

print(x[2]) # 8   __getitem__(self, pos)

print (x[100]) # should throw error → handle it

+ 1906   x[2] = 0   __setitem__(self, pos, v)

```
def __setitem__(self, pos: "Python int", val: "Python int") -> "None":
        n = len(self)
        assert pos >= 0 and pos < n
        assert val >= 0 and val < 10
        self._a[pos] = val
```

y = Int(2)

z = x+y   __add__(self, b: "Int") → "Int"

```
def __add__(self, other: "Int") -> "Int":
        python_int_operation = self.int() + other.int()
        return Int(python_int_operation)
```

z2 = x - y   __sub__(self, b: "Int") → "Int"

```
def __sub__(self, other: "Int") -> "Int":
        python_int_operation = self.int() - other.int()
        return Int(python_int_operation)
```

z3 = x * y   __mul__(self, b: "Int") → "Int"

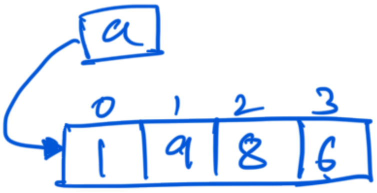
```
def __mul__(self, other: "Int") -> "Int":
        python_int_operation = self.int() * other.int()
        return Int(python_int_operation)
```

z4 = x // y   __div__(self, b: "Int") → "Int"

- Implementing __add__ :
  - Ripple carry adder:

```
   1 1 0 [0]  carry
+  1 9 8 6
+      4 2
-----------
   2 0 2 8    sum
```

- Implementing python small int of Int :



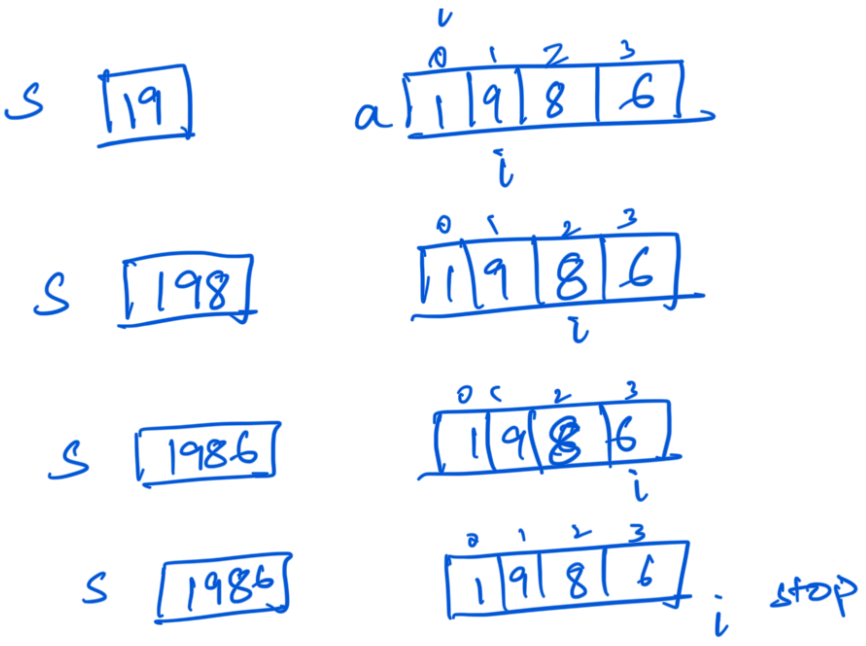
s = 0 , i = 0

s = s*10 + a[i]

i = i+1 (while i<n)

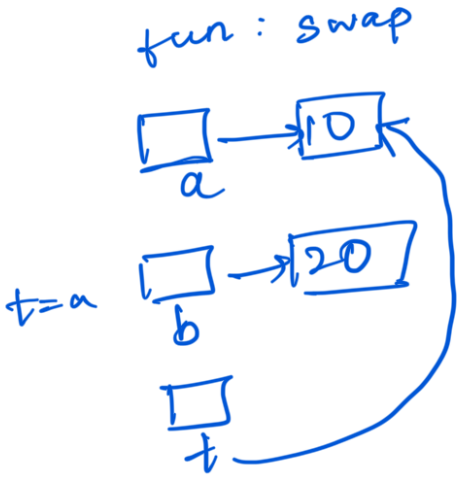
s $\boxed{1}$

a	0	1	2	3
	1	9	8	6



Pass By Value

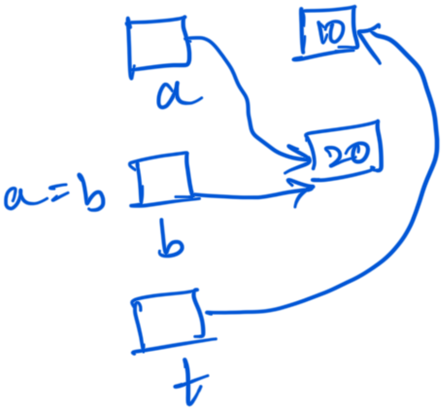
- Python uses pass by value
  Java



if we do :

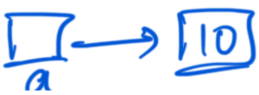
```
swap(a, b):
    t = a
    a = b
    b = t
```

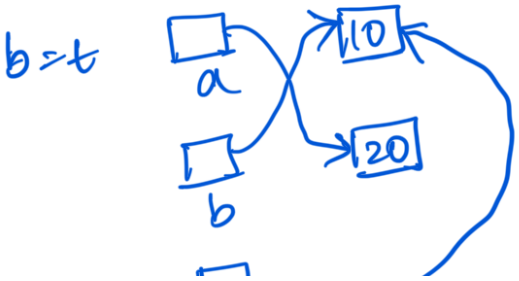
```
a = 10
b = 20
swap(a, b)
# a = 10
# b = 20
```



But, when we return:

main



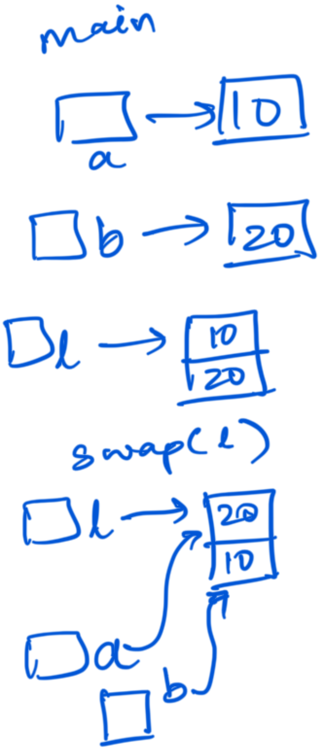


It is not swapped as a & b are pointers (will not work in Python & Java)

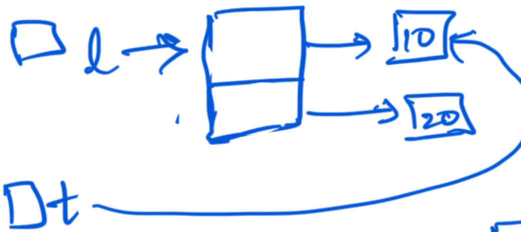
Swapping two objects in constant time

implement swap by value not reference:

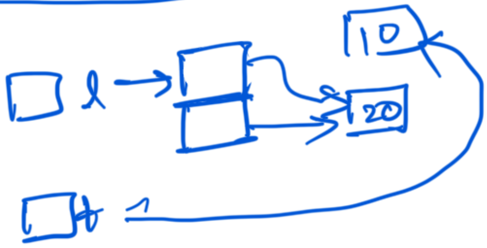
```
swap( l: " list of size" ):
    t = l[0]
    l[0] = l[1]
    l[1] = t

a = 10
b = 20
l = [10, 20]
swap( l)
a = l [0]     # a = 20
b = l [1]     # b = 10
```



fun: swap

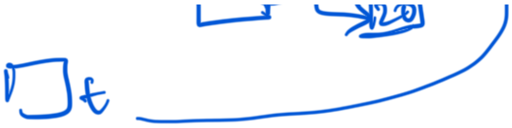


t = l[0]



l[0] = l[1]

l[1] = t



- In C++, C

```
swap( int &a, int &b)  → reference
        t = a
        a = b
        b = t
```

Pass by reference swap works!
in C, C++