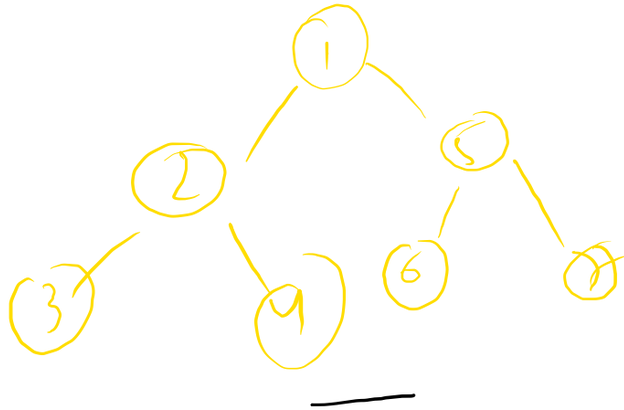
Tree → Binary Tree → Node → 0/1/2 ↦ childs

3 types of traversal



(1) Preorder [Root – LST – RST]

→ 1 → 2 → 3 → 4 → 5 → 6 → 7

(2) inorder [LST → Root → RST]

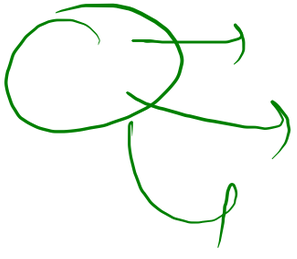
3 → 2 → 4 → 1 → 6 → 5 → 7

(3) postorder [LST → RST → Root]

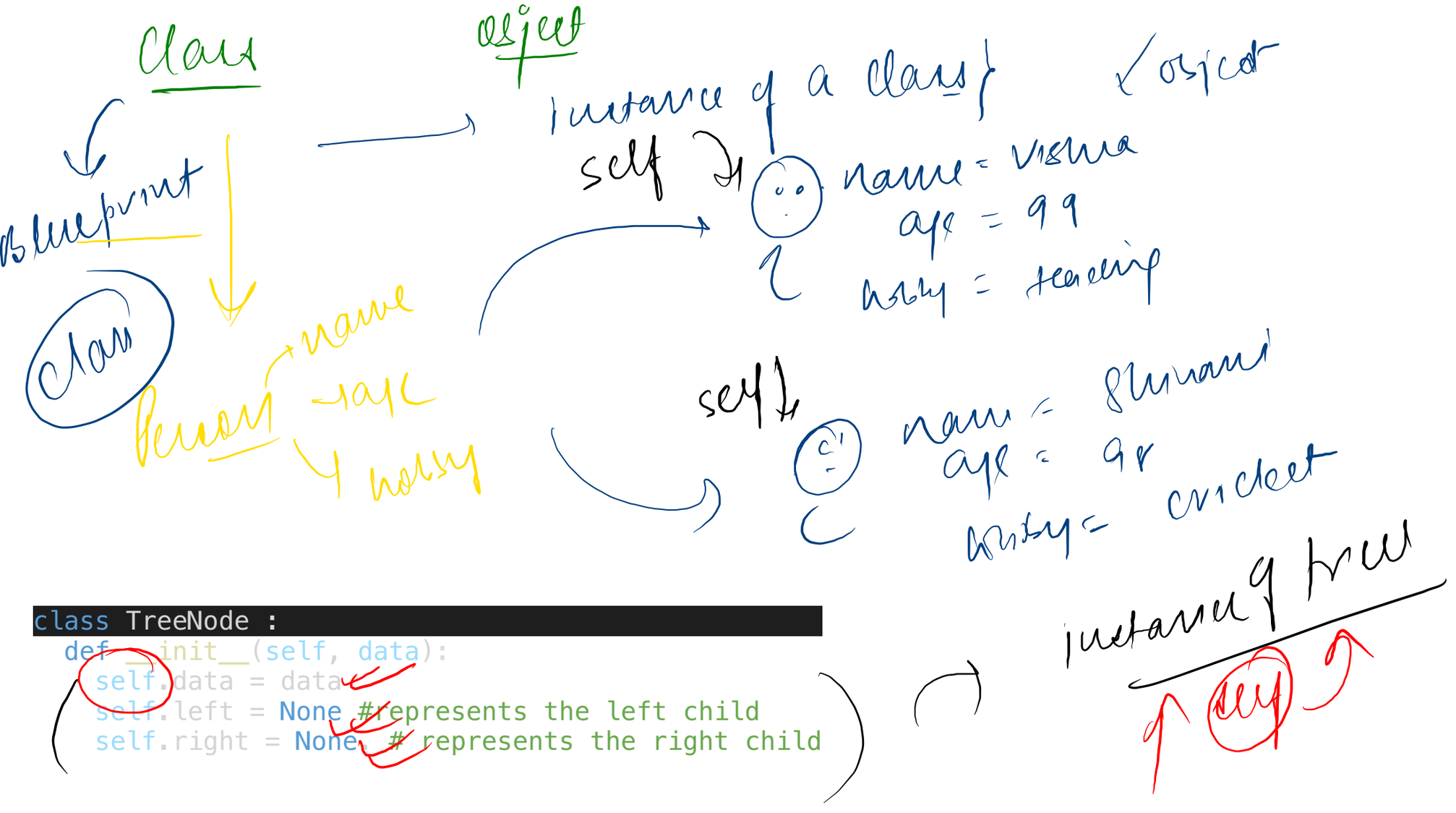
3 → 4 → 2 → 6 → 7 → 5 → 1

Python → TreeNode

code for the traversal

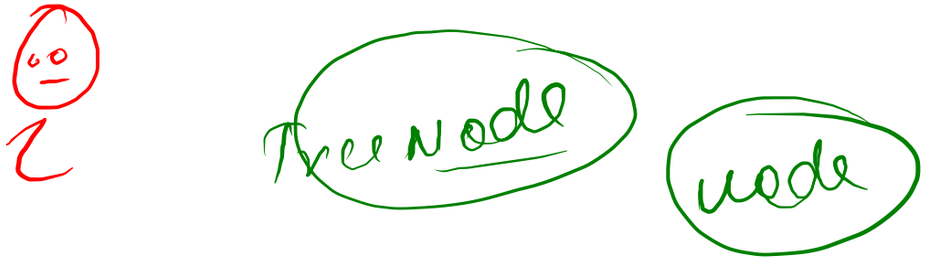


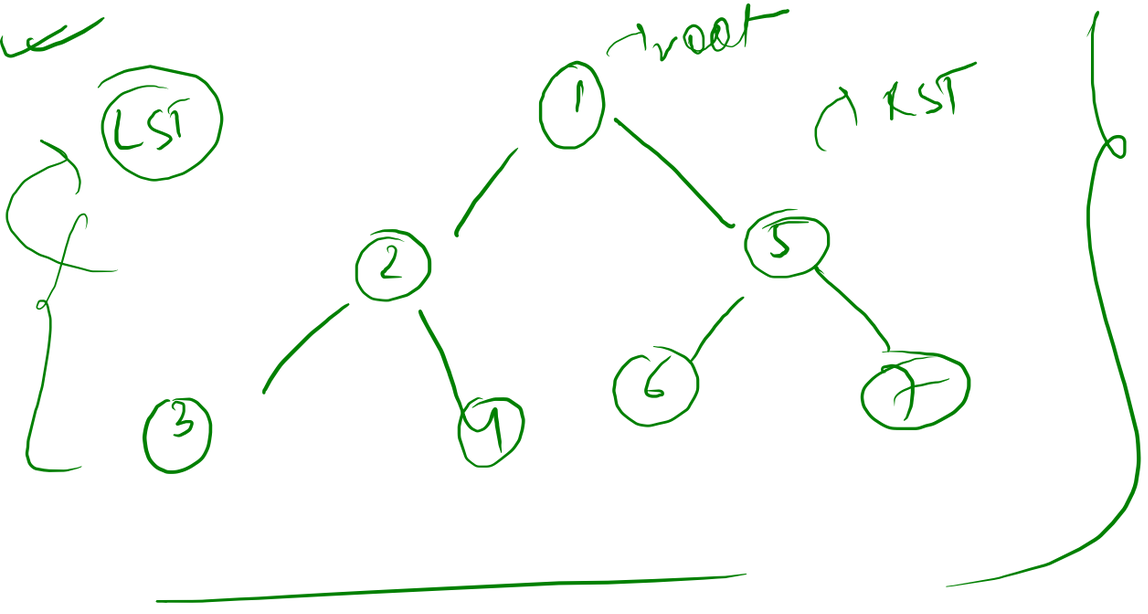
data
left child
right child




```
class TreeNode :
  def __init__(self, data):
    self.data = data
    self.left = None #represents the left child
    self.right = None  # represents the right child
```

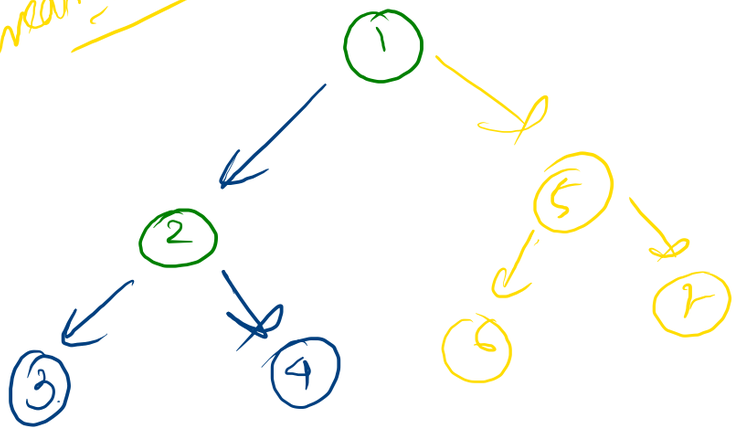
```
class TreeNode :
  def __init__(self, data):
    self.data = data
    self.left = None #represents the left child
    self.right = None # represents the right child


  def setData(self, data):
    self.data = data

  def getData(self):
    return self.getData()

  def setLeft(self, node):
    self.left = node
  def getLeft(self):
    return self.left
  def setRight(self,node):
    self.right = node
  def getRight(self):
    return self.right
```


TreeNode
node



free

```
# Constructing the tree
root = TreeNode(1)

#Left SubTree
leftR = TreeNode(2)
leftR.setLeft(TreeNode(3))
leftR.setRight(TreeNode(4))
root.setLeft(leftR)

#Right SubTree
rightR = TreeNode(5)
rightR.setLeft(TreeNode(6))
rightR.setRight(TreeNode(7))
root.setRight(rightR)
```



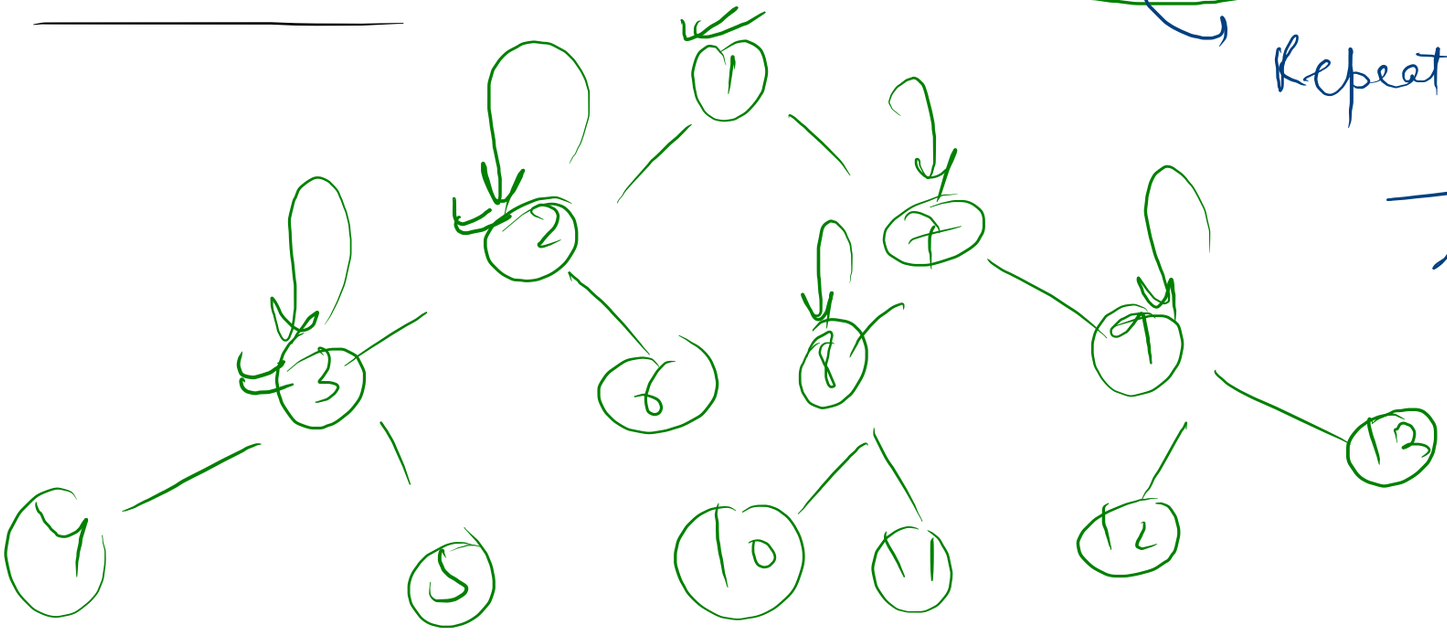
insert a node
delete a node

PreOrder traversal

R – LST – RST

Repeated multiply true

same → Recursion

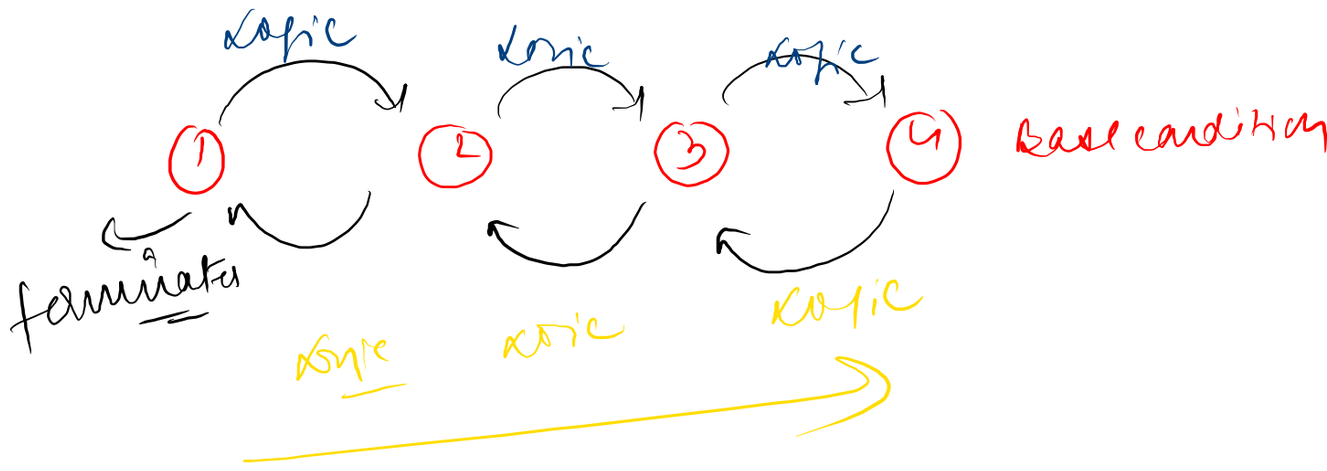


Recursion

1. Base condition
2. Logic
3. Recursive call

(2) Recursive call

(3) Logic



Recursion → define the function very well

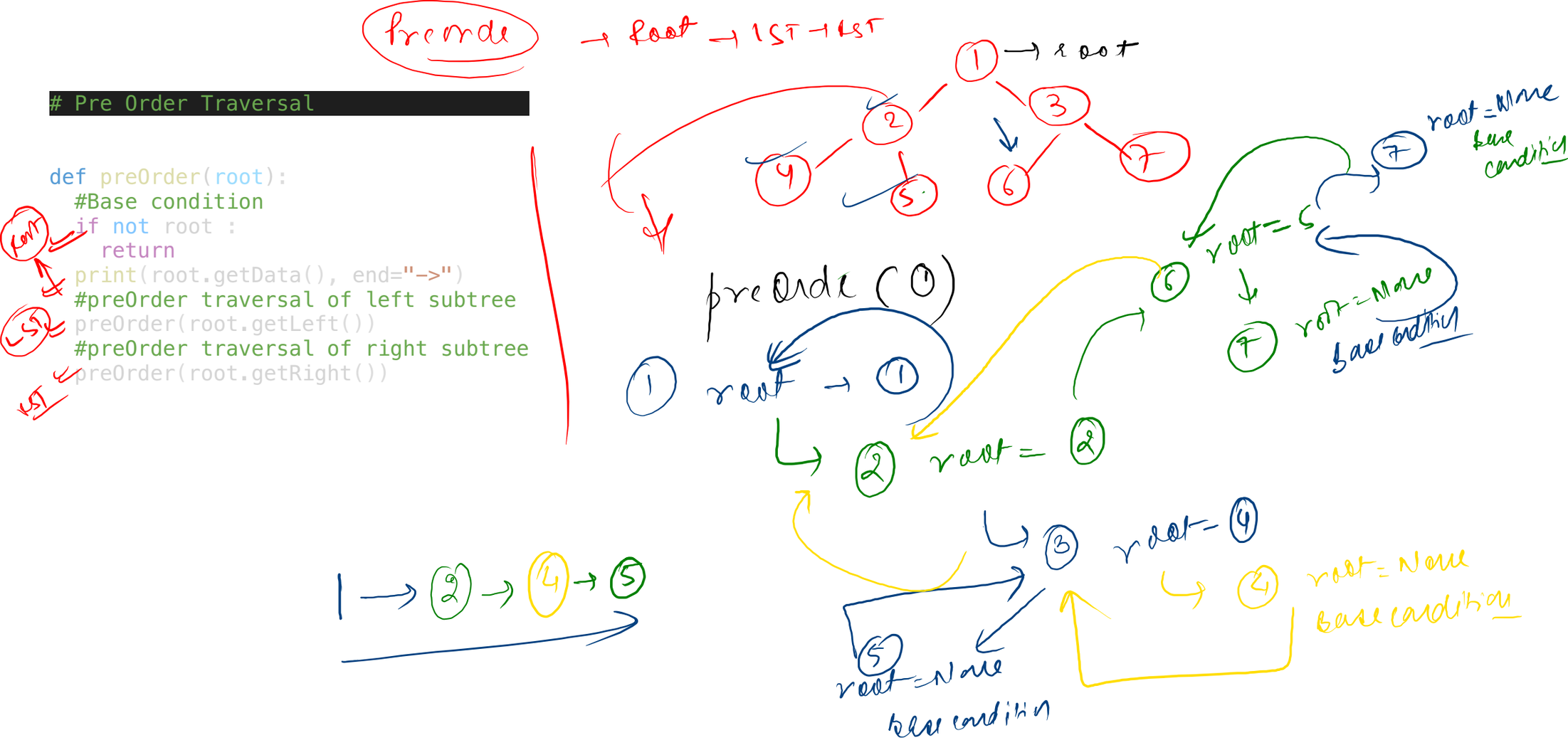
```
def preOrder(root):
```

→ Mean

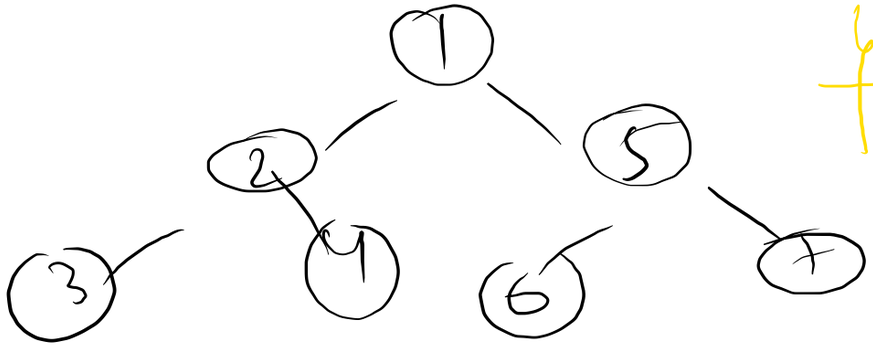
fn → if we provid root

↳ traverse the tree under root in preorder way

```
# Pre Order Traversal


def preOrder(root):
  #Base condition
  if not root :
     return
  print(root.getData(), end="->")
  #preOrder traversal of left subtree
  preOrder(root.getLeft())
  #preOrder traversal of right subtree
  preOrder(root.getRight())
```

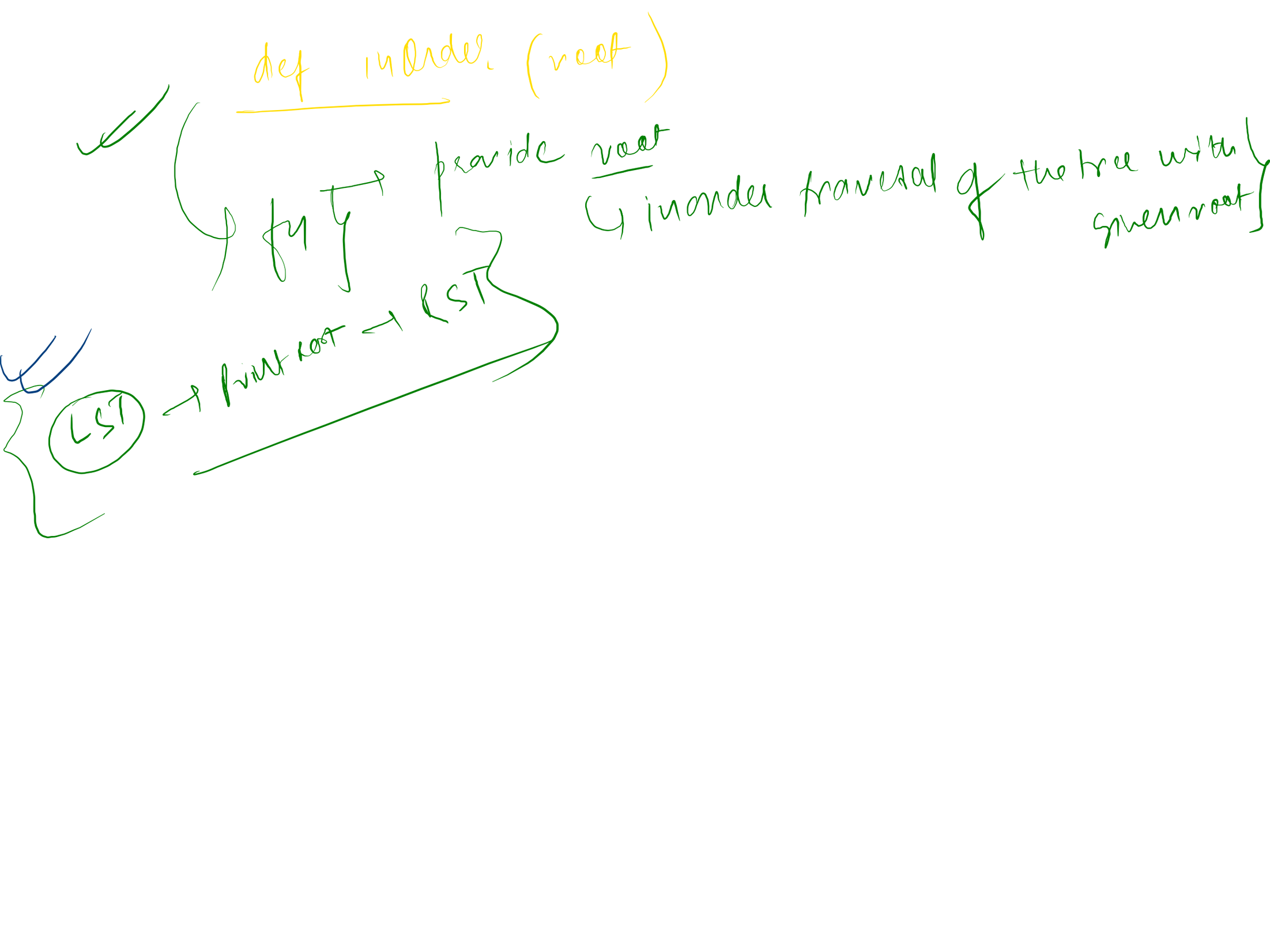


Problem

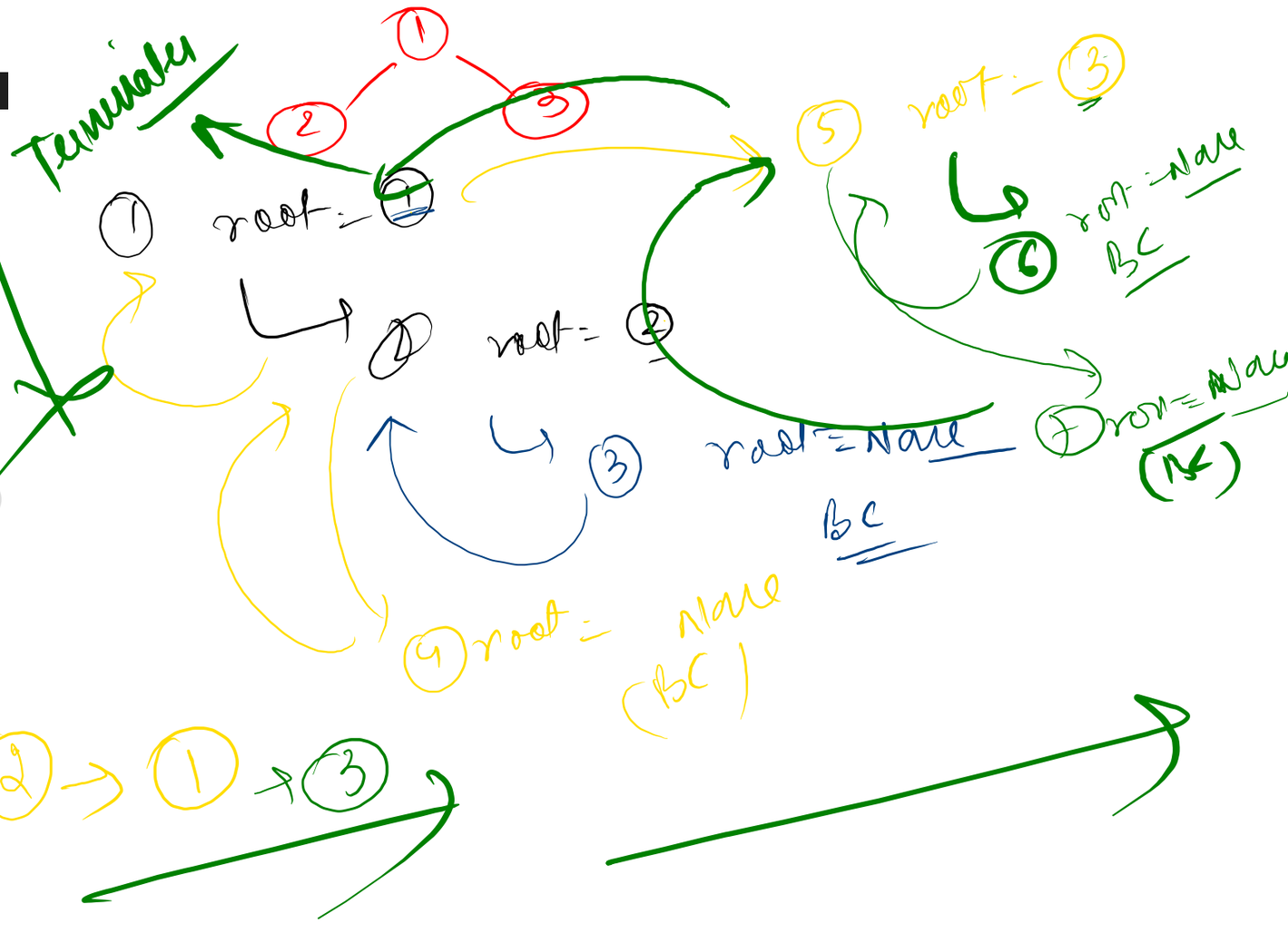


↳ code problem

1 → 2 → 3 → 4 → 5 → 6 → 7


def inOrder(root)
provide root
inorder traversal of the tree with given root
LST
print root
RST

```python
def inOrder(root):
    #Base condition
    if not root :
        return

    #traverse the left subtree
    inOrder(root.getLeft())

    #Print the root
    print(root.getData(), end="->")

    #Traverse the right subtree
    inOrder(root.getRight())
```



10:40 AM → Break time

↓

on huge demand

↳ 5 min extra

↓

starts at 10:45 AM

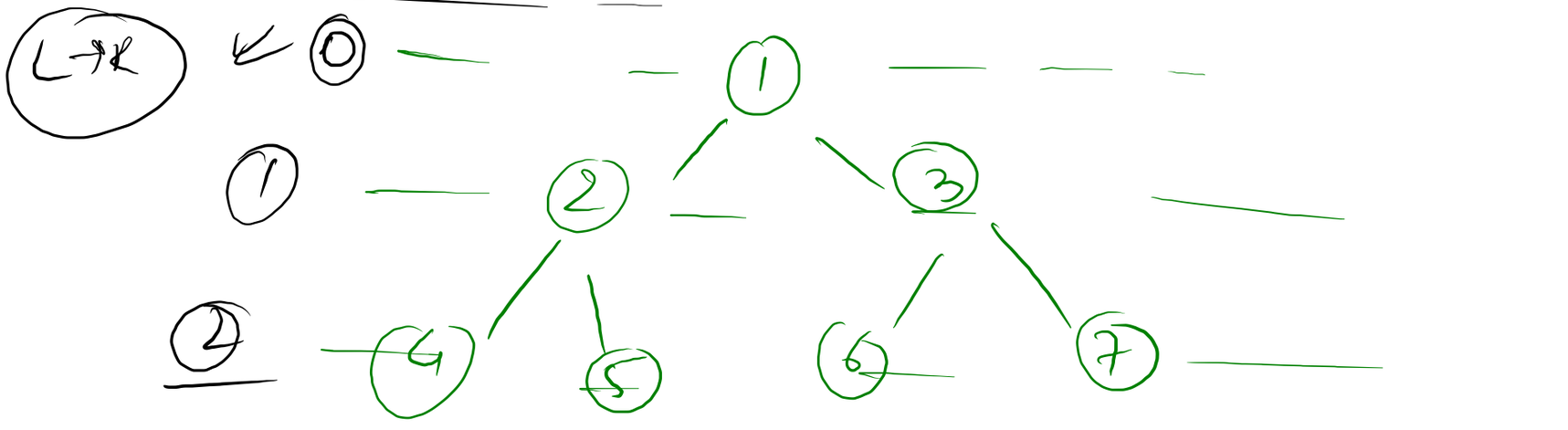
:)

interesting problems of DS ✓

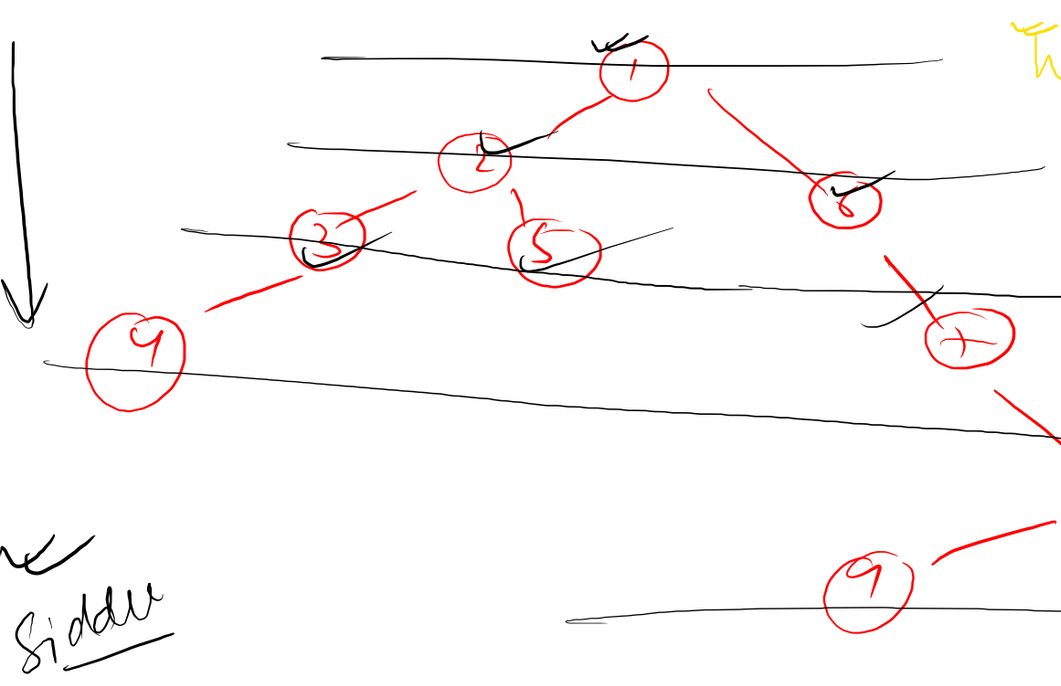
↑ Free

pre order / in order / post order }

Level order traversal :- (BFS)



1 → 2 → 3 → 4 → 5 → 6 → 7



Write the code

① No repeatition of logic → Queue

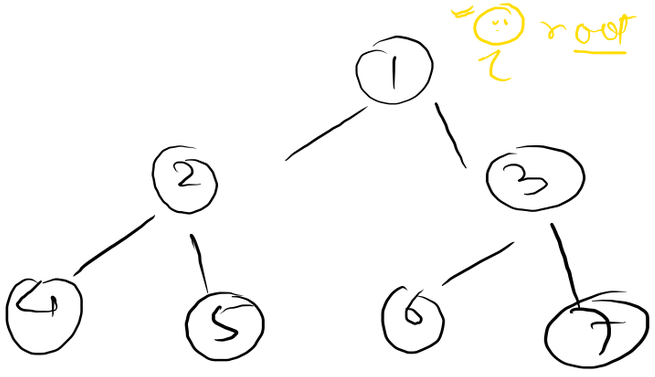
② Node which comes first is printed first → FIFO

Recursion → No repeatition

↳ Stack → FIFO / LIFO

Siddu

1 → 2 → 6 → 3 → 5 → 7 → 4 → 8 → 9 → 10



Level order traversal

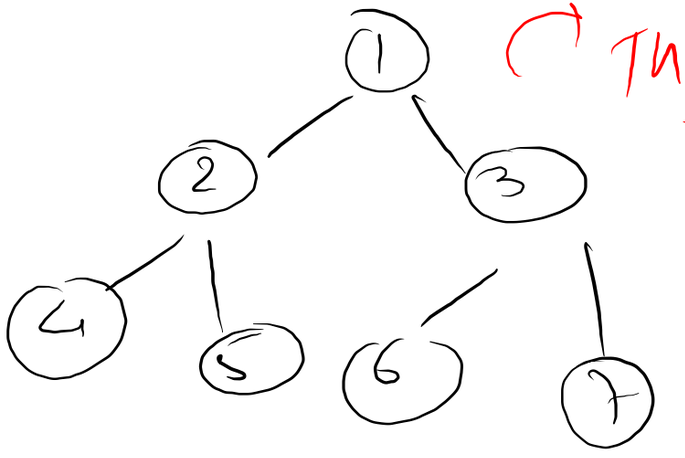
↳ use queue

Think for 5 minutes] 10:55 - 11 AM {5 minutes]

1 → 2 → 3 → 4 → 5 → 6 → 7

Pre order

1 → 2 → 4 → 5 → 3 → 6 → 7



Theory

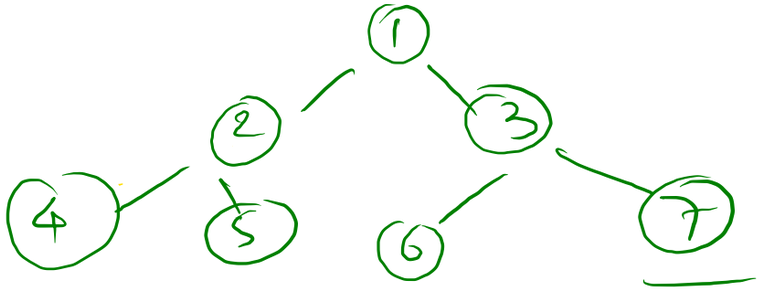
Queue

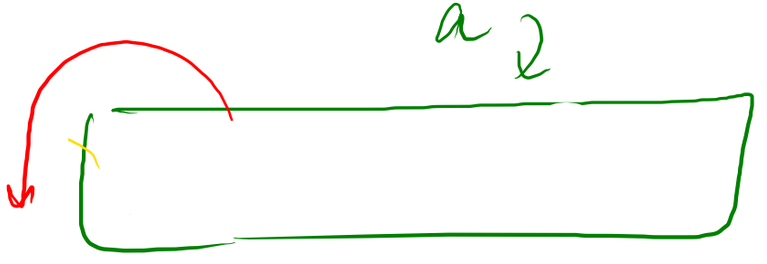
(Loop)

Level order traversal

1 → 2 → 3 → 4 → 5 → 6 → 7

```
def levelOrder(root):
  if not root :
    return
  q = QueueLL()
  q.enQue(root)
  while not q.isEmpty():
    #Deque the element from the queue
    temp = q.deQueue()
    print(temp.getData(), end="->")


    if temp.getLeft():
      q.enQue(temp.getLeft())
    if temp.getRight():
      q.enQue(temp.getRight())
```





temp = 7

1 → 2 → 3 → 4 → 5 → 6 → 7

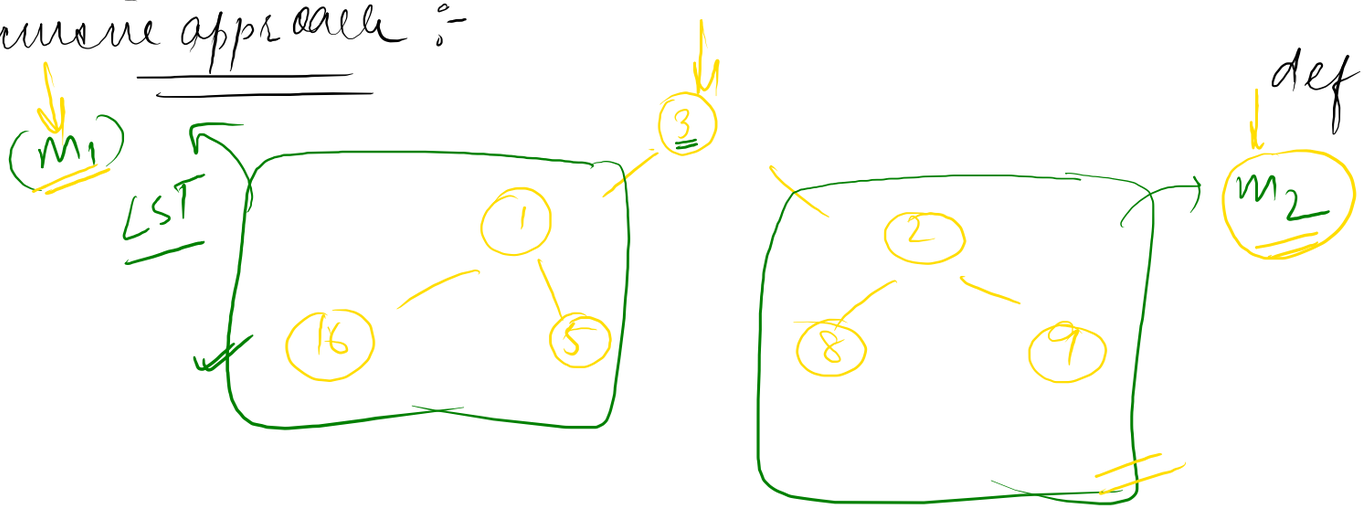
```
def max(root):
```

find the maximum element of tree

Traversal

→ Finish 11:29 AM

Recursive approach :-

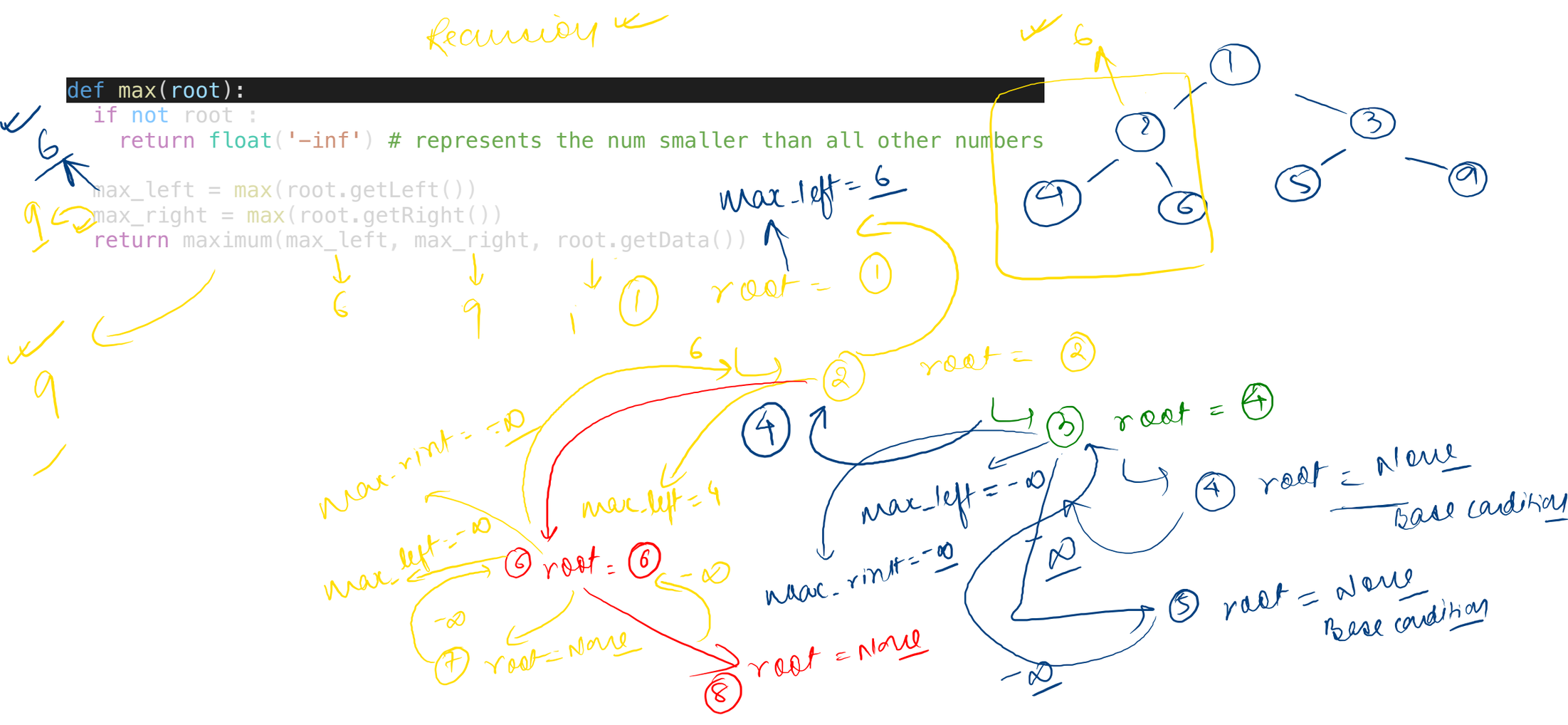


Max (m1, 3, m2)

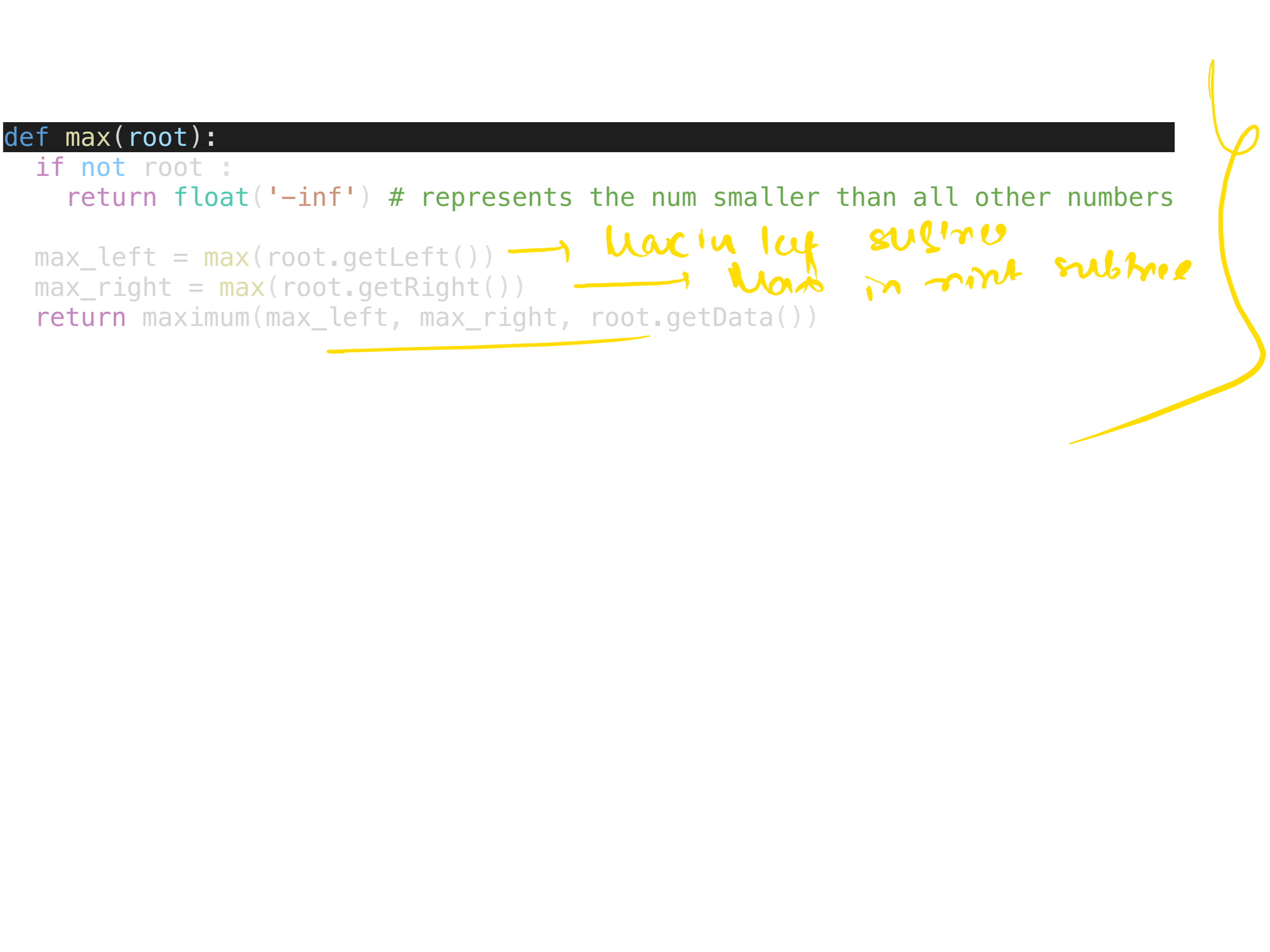
```
def max(root):
  if not root :
    return float('-inf') # represents the num smaller than all other numbers
  max_left = max(root.getLeft())
  max_right = max(root.getRight())
  return maximum(max_left, max_right, root.getData())
```

Recursion

```
def max(root):
  if not root :
    return float('-inf') # represents the num smaller than all other numbers

  max_left = max(root.getLeft())
  max_right = max(root.getRight())
  return maximum(max_left, max_right, root.getData())
```



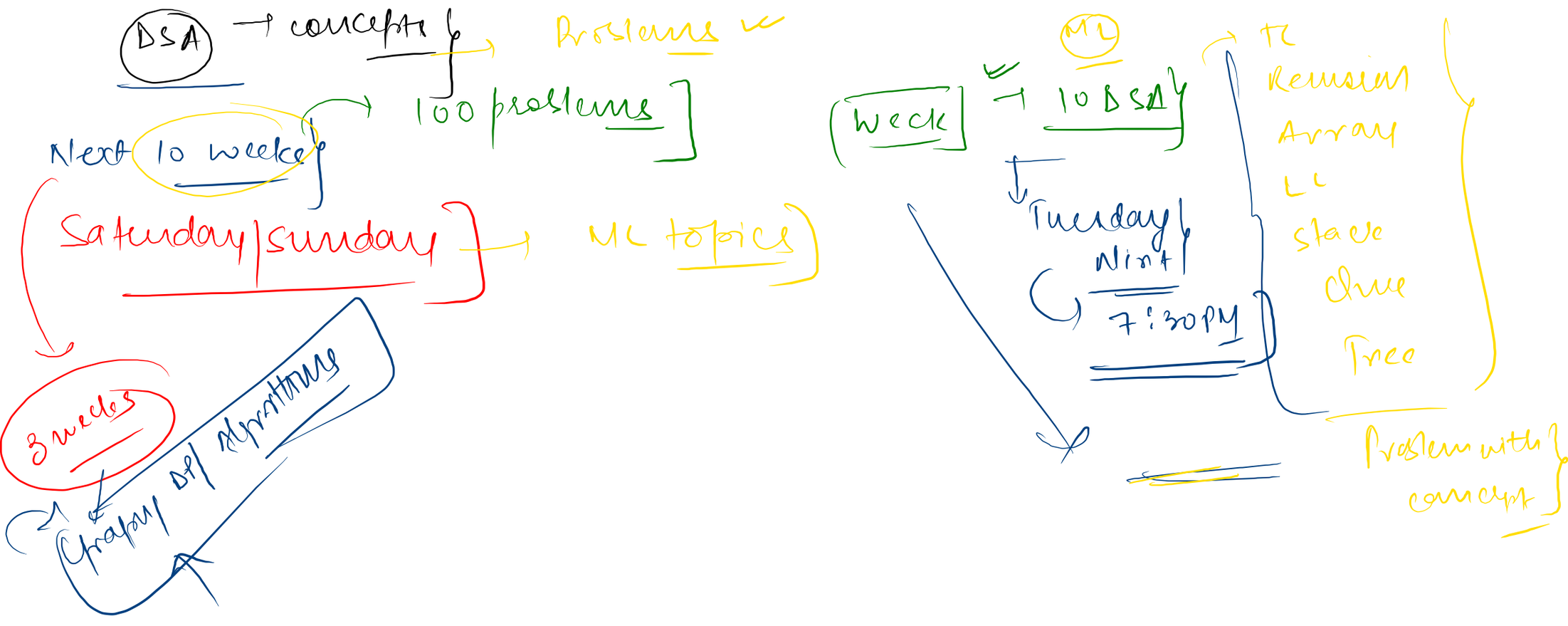
```
def max(root):
  if not root :
    return float('-inf') # represents the num smaller than all other numbers

  max_left = max(root.getLeft())
  max_right = max(root.getRight())
  return maximum(max_left, max_right, root.getData())
```



Minimum element in the tree

```
def min(root):
```

so questions


DSA
concepts
Problems
100 problems
Next 10 weeks
Saturday/Sunday
ML topics
3 weeks
Graph/DP/Algorithms
ML
Week
10 DSA
Tuesday
Night
7:30PM
TC
Recursion
Array
LL
Stack
Queue
Tree
Problem with concept

Next week → Questions → Next to Next week

Sat/Sunday → Next ML topics

M-Tue

Assignments

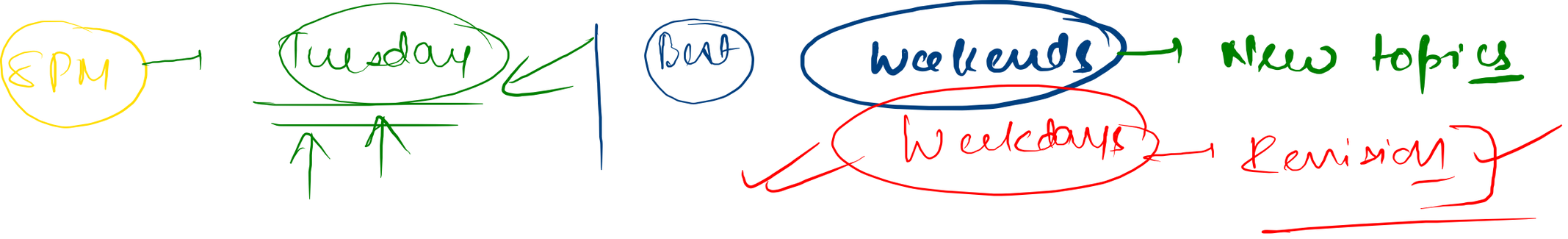
10 problems → week vs not well

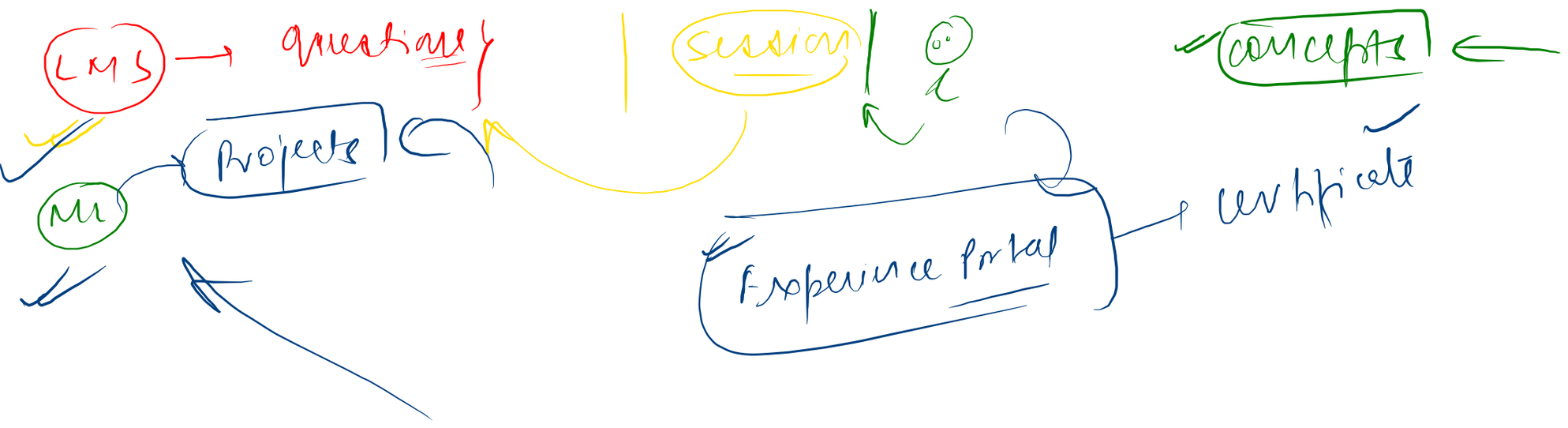
Weekdays → Revision → Phone

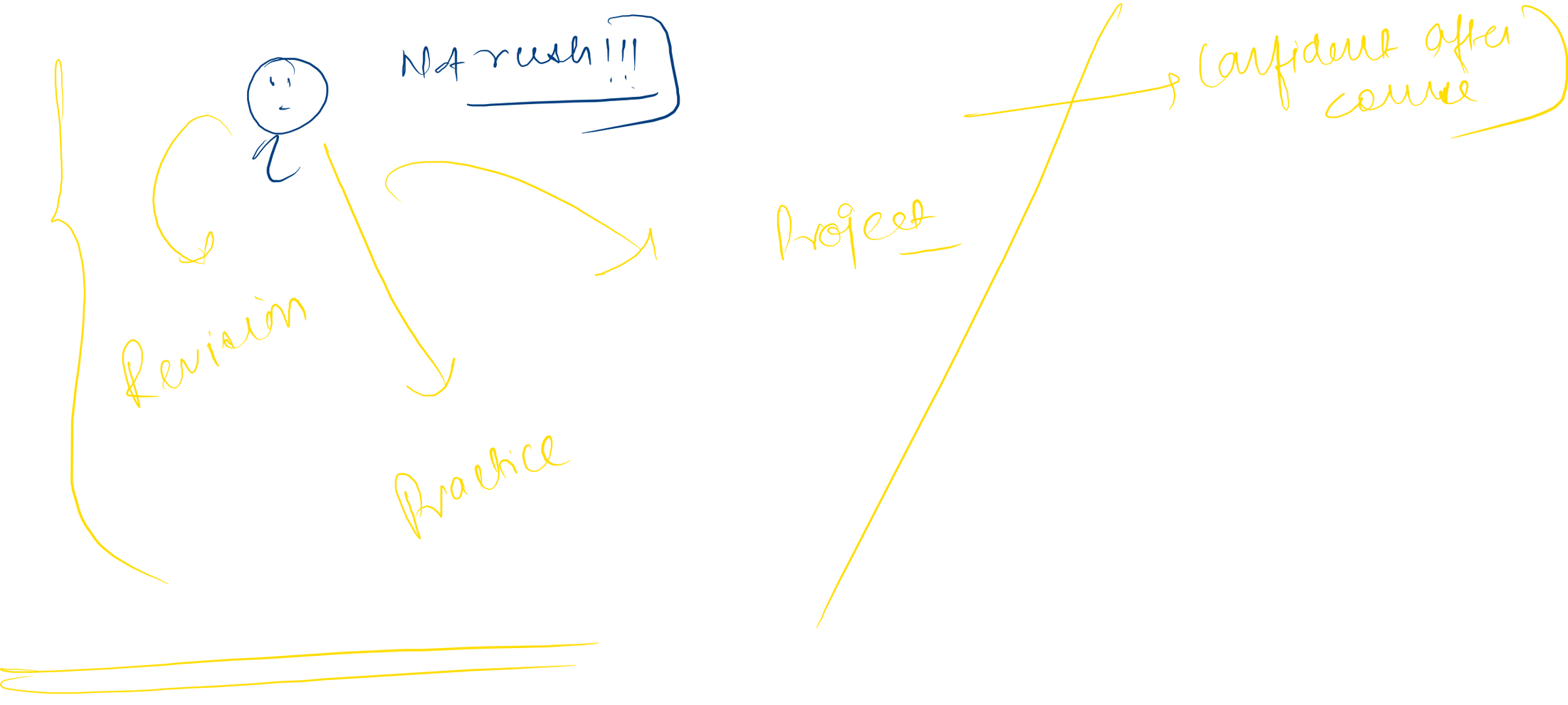
DSA problems

Interview ready ↓ lowers

Pr DSA → 10 questions

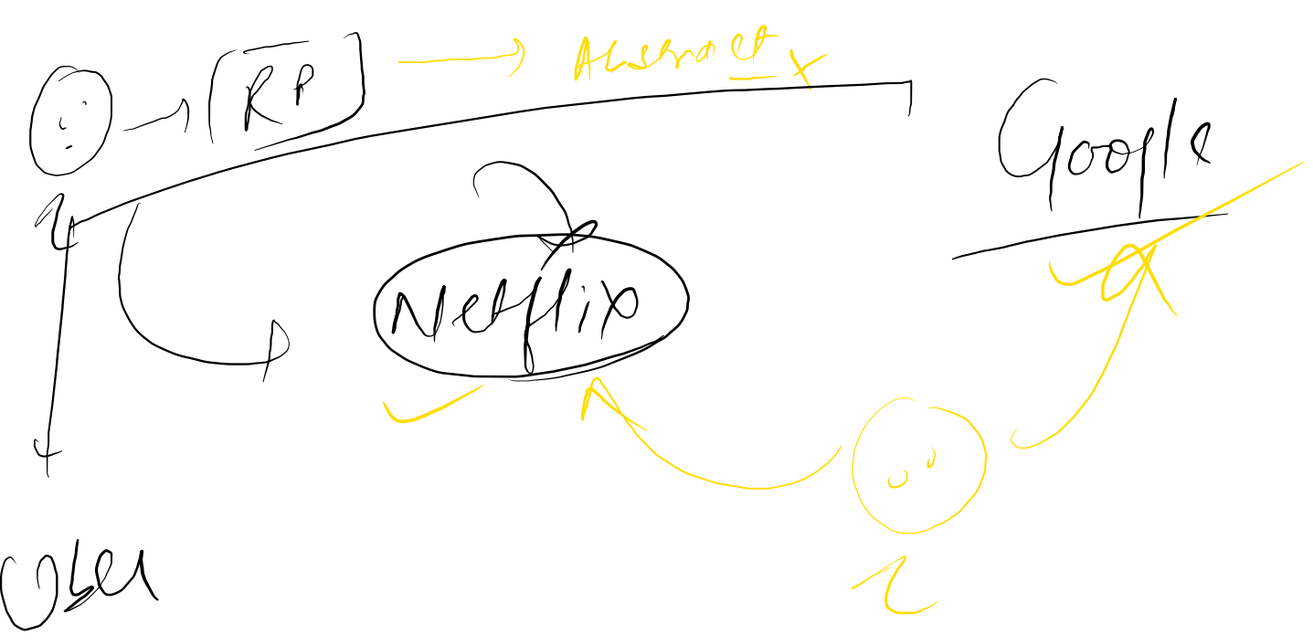

EPM
Tuesday
But
Weekends
New topics
Weekdays
Revision


LMS
Questions
Session
Concepts
Projects
ML
Experience Portal
certificate


No rush !!!
Revision
Practice
Project
Confidence after course


Day/Time
Survey
Telegram
email

Doubts :- 12:22 PM


RP
Google
Netflix


Problem
Run

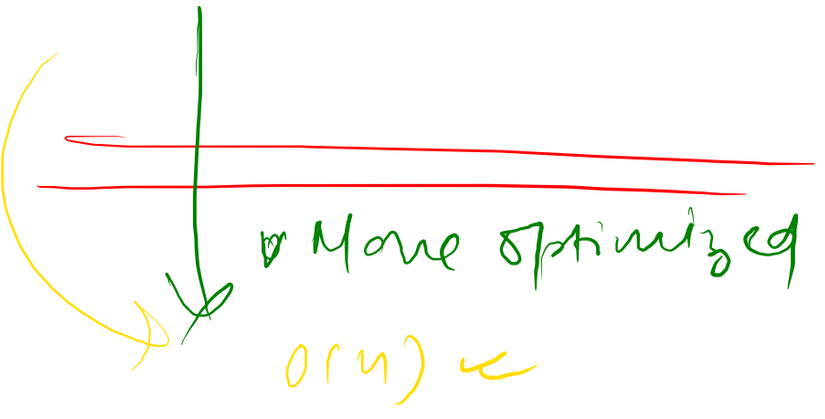
1st approach ①

Solve

② Boundary conditions

Dry run

$O(n^2)$



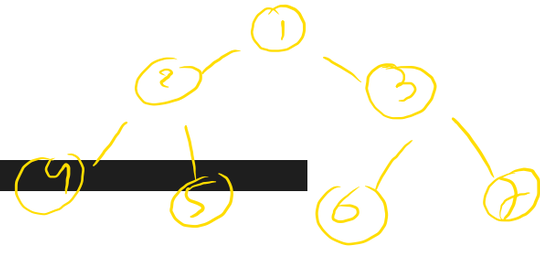
More optimized

$O(n)$

Recursion

max (1)

Queue 2




```
def max(root):
  if not root :
    return float('-inf') # represents the num smaller than all other numbers

  max_left = max(root.getLeft())
  max_right = max(root.getRight())
  return maximum(max_left, max_right, root.getData())
```

BC

```
def levelOrder(root):
  if not root :
    return
  q = QueueLL()
  q.enQue(root)
  while not q.isEmpty():
    #Deque the element from the queue
    temp = q.deQueue()
    print(temp.getData(), end="->")


    if temp.getLeft():
      q.enQue(temp.getLeft())
    if temp.getRight():
      q.enQue(temp.getRight())
```

argument check

levelorder (1)

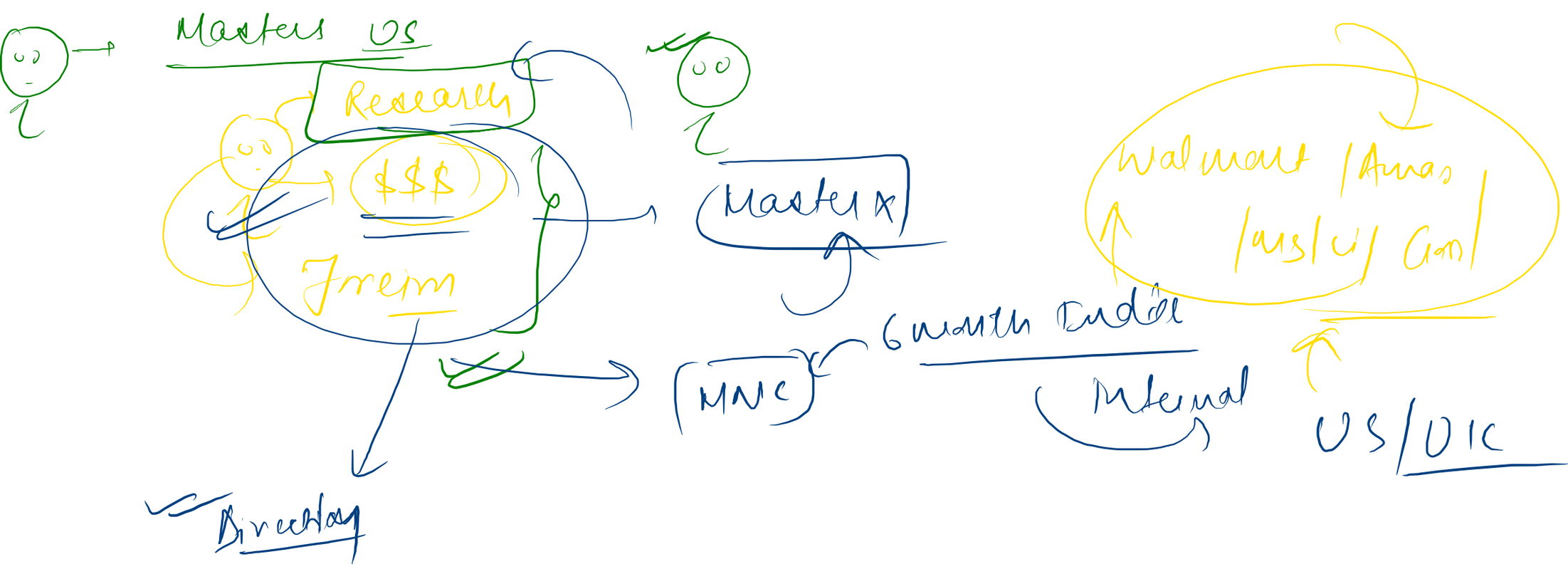
Masters → [GRE ← Preparation
TOEFL]

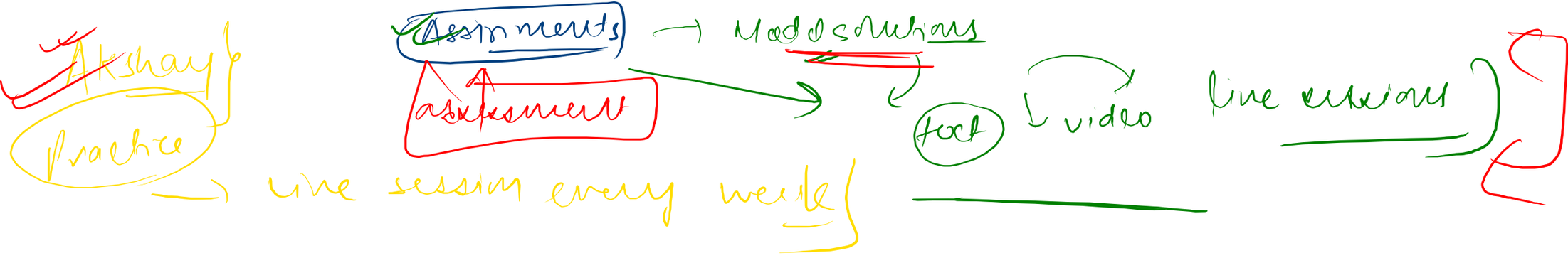
→ Universities
1. score
2. OG
3. companies
4. Projects

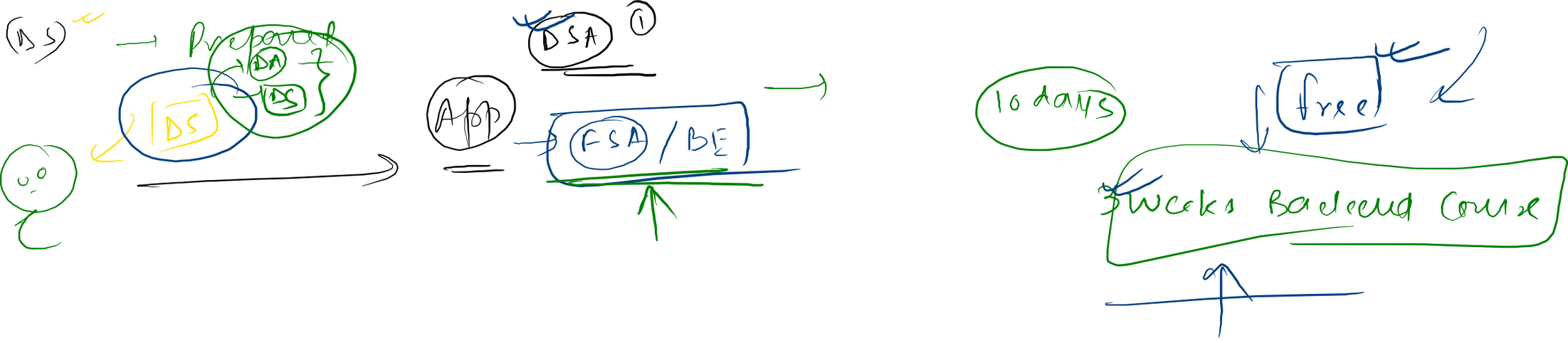
interview]

half hearted efforts
positive

full smart effort


Markets US
Research
$$$
Team
Market X
MNC
Growth Engine
Internal
Walmart / Amazon / US / Gen
US / UK
Directory


Akshay
Practice
Assignments
assessment
Model solutions
text
video
live sessions
live session every week


DS
Prepare
DA
DS
DS
DSA
1
App
FSA / BE
10 days
free
3 Weeks Backend Course

```
def minimum(a,b,c):
  if a<=b:
    if a<=c:
      return a
    else:
      return c
  else:
    if b<=c:
      return b
    else:
      return c
```

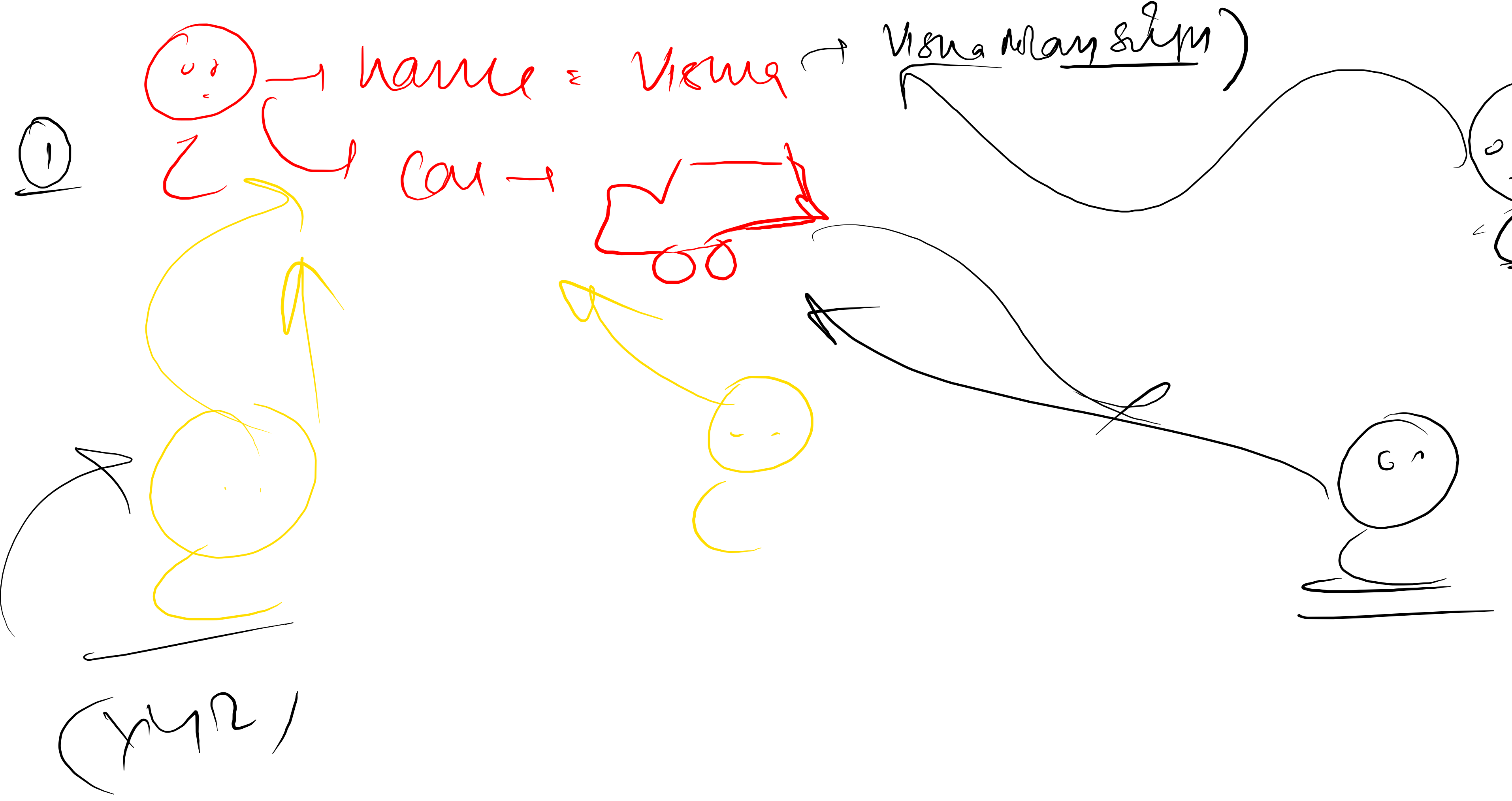
```
def min(root):
  if not root:
    return

  min_left = min(root.getLeft())
  min_right = min(root.getRight())
  return minimum(min_left, min_right, root.getData)
```

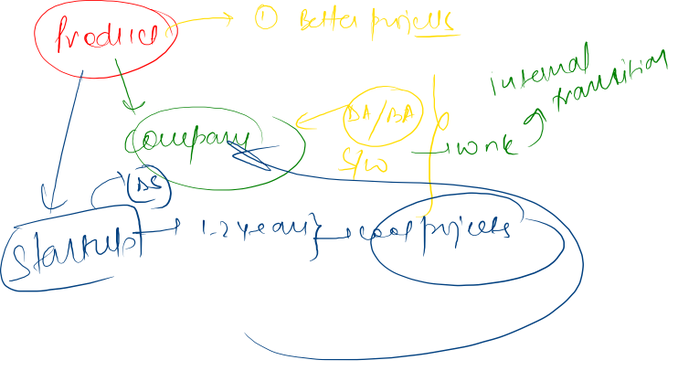
root.getData()

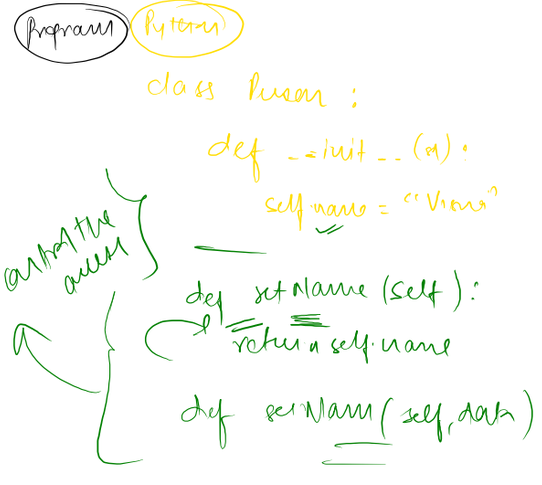
Er

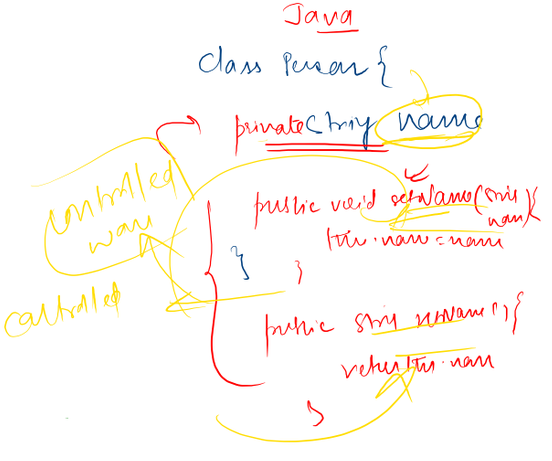
'Matrix' → your fields are private

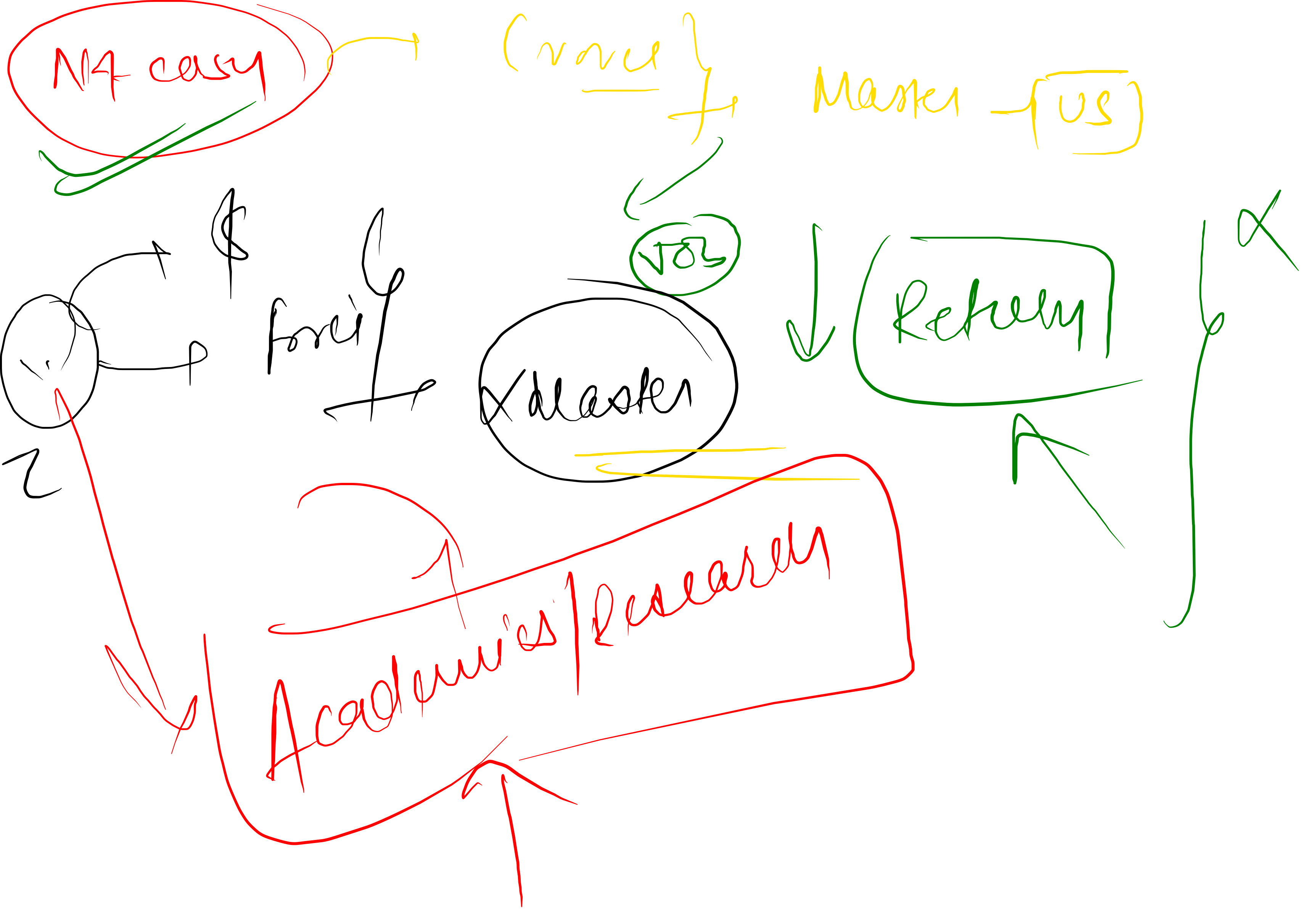
↑ Name should change it directly


name =
car
(XYZ)

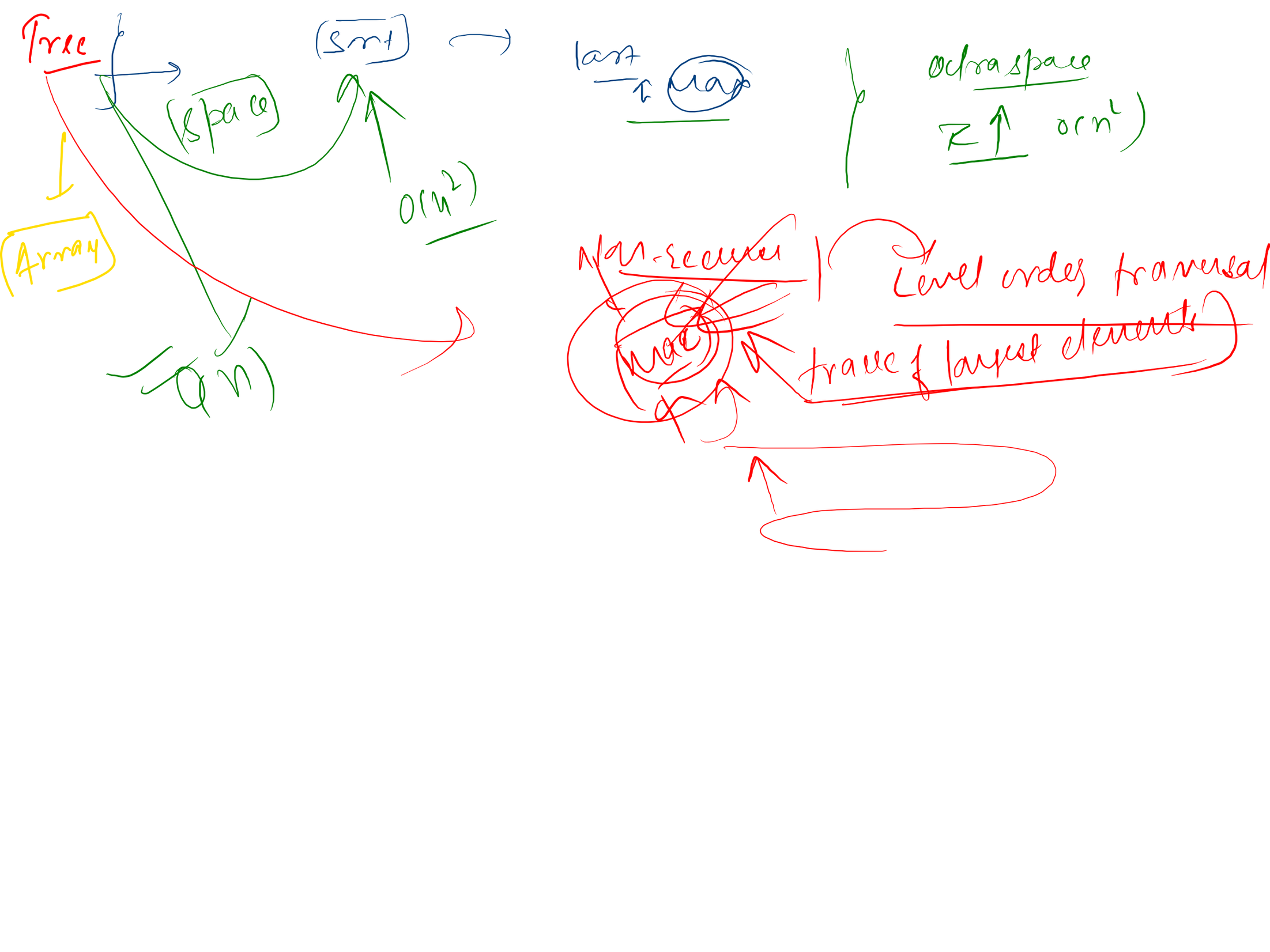

Product
1) Better projects
DA/BA
Company
work
internal transition
DS
Startup
1-2 years
projects

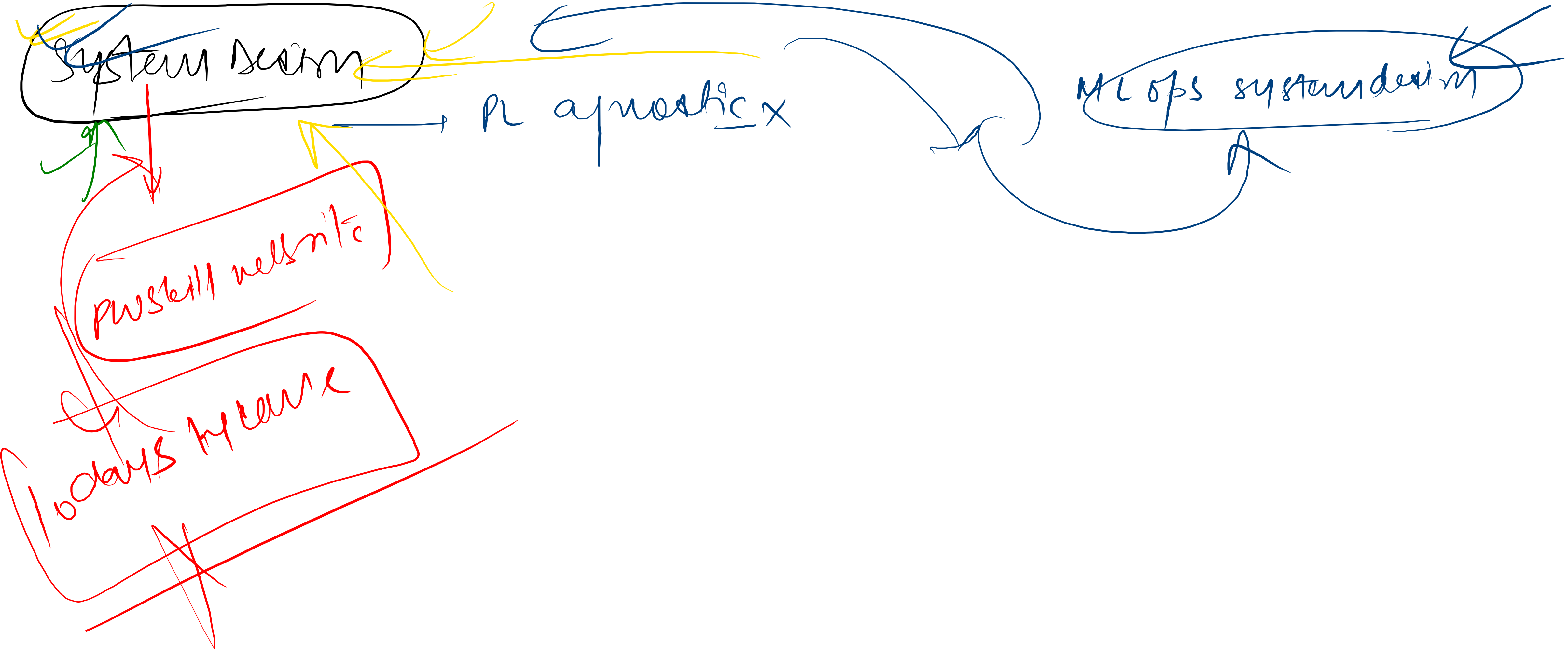

Program
Python
class Person :
def __init__(self):
self.name = "Vinay"
attribute access
def setName(self):
return self.name
def setName(self, data)


Java
class Person {
private String name
Constructor
name
public void setName(String name){
this.name = name
}
}
Getter
public String getName(){
return this.name
}

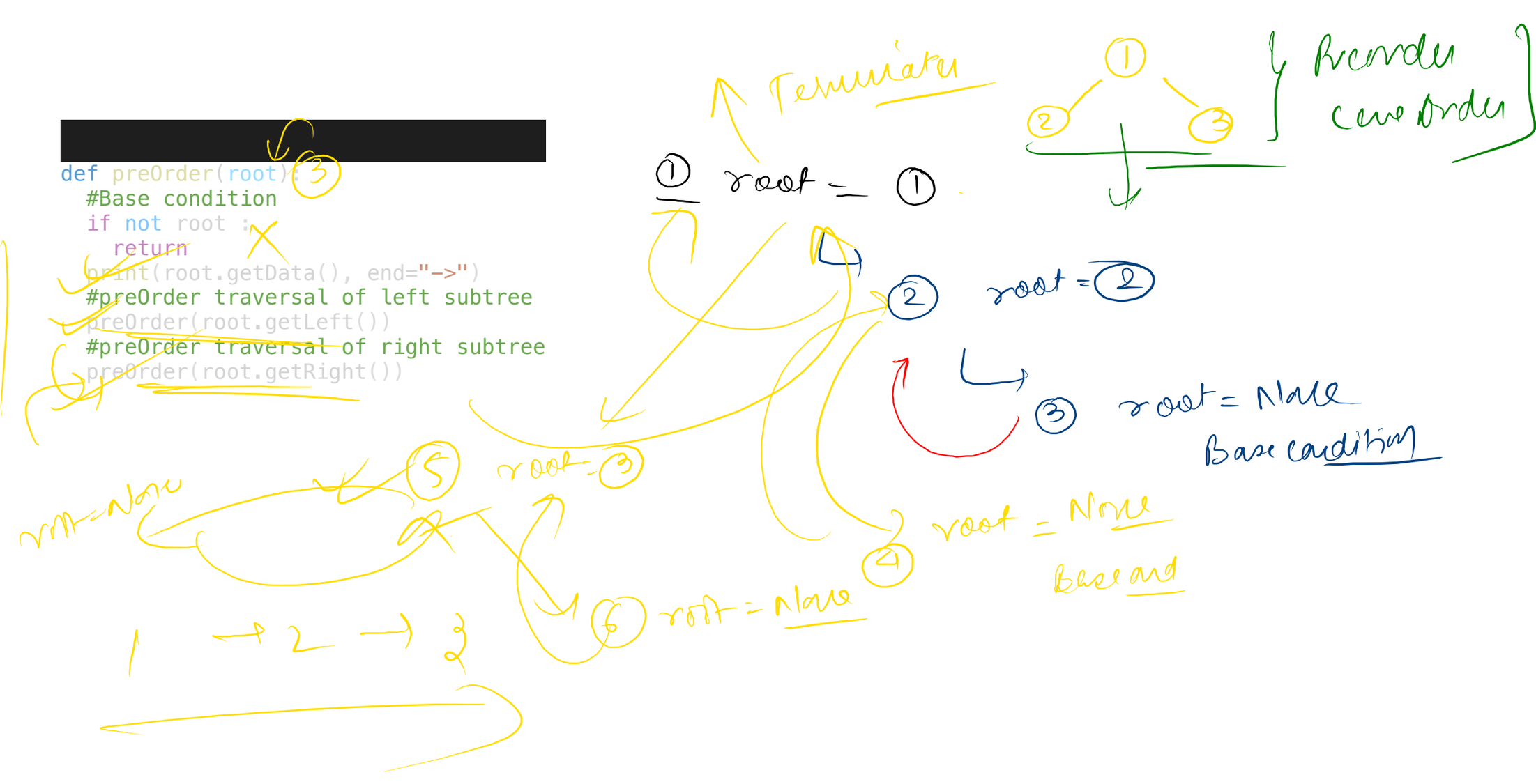


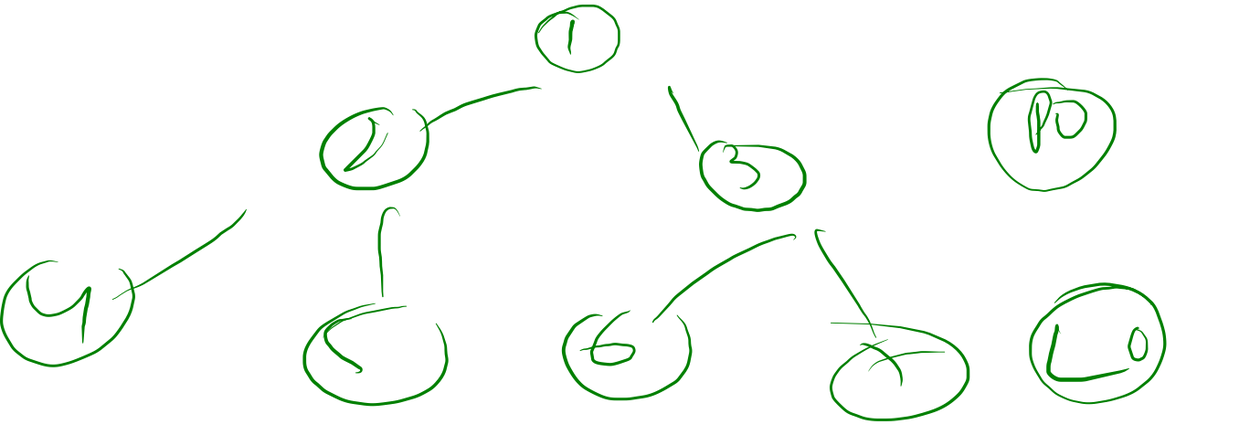
Master
US
VOL
Return
force
$
Master


Tree
(sort)
(space)
O(n²)
Array
O(n)
last
map
extra space
O(n²)
Max-heap
Level order traversal
trace of largest element


System Design
PL agnostic
ML ops system design

```
def preOrder(root):
  #Base condition
  if not root :
    return
  print(root.getData(), end="->")
  #preOrder traversal of left subtree
  preOrder(root.getLeft())
  #preOrder traversal of right subtree
  preOrder(root.getRight())
```

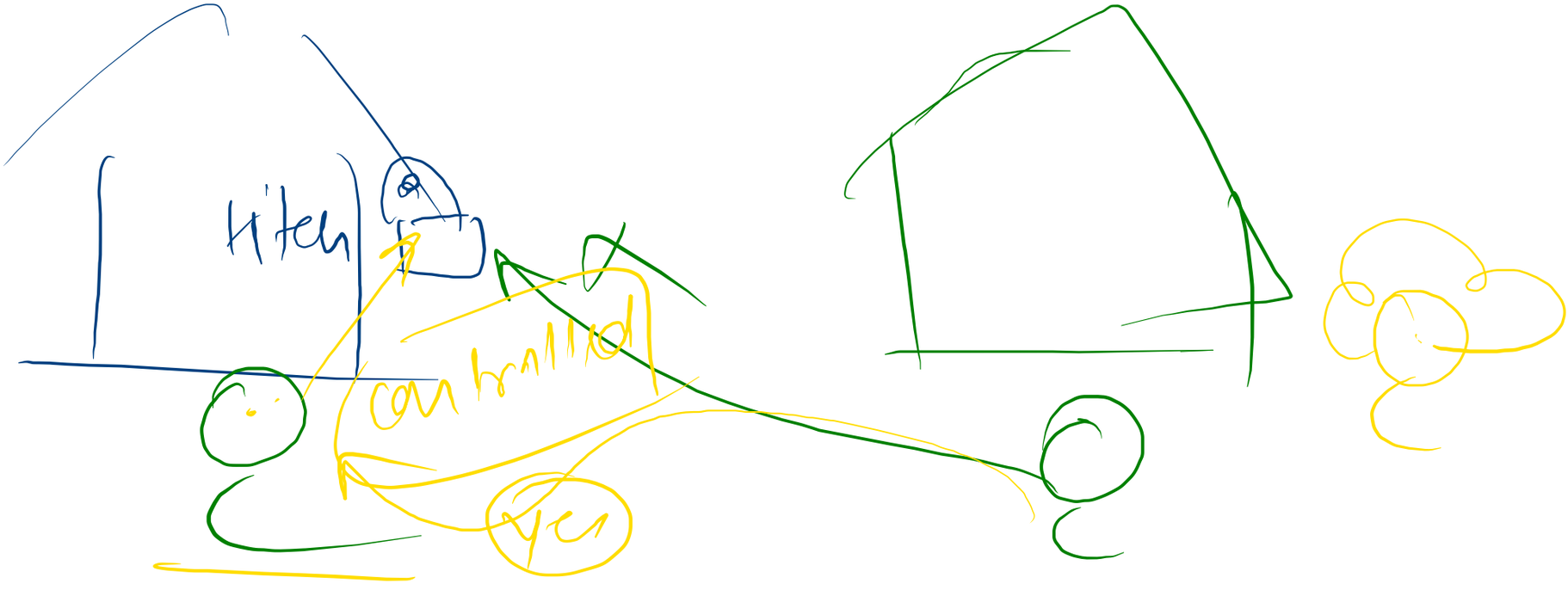




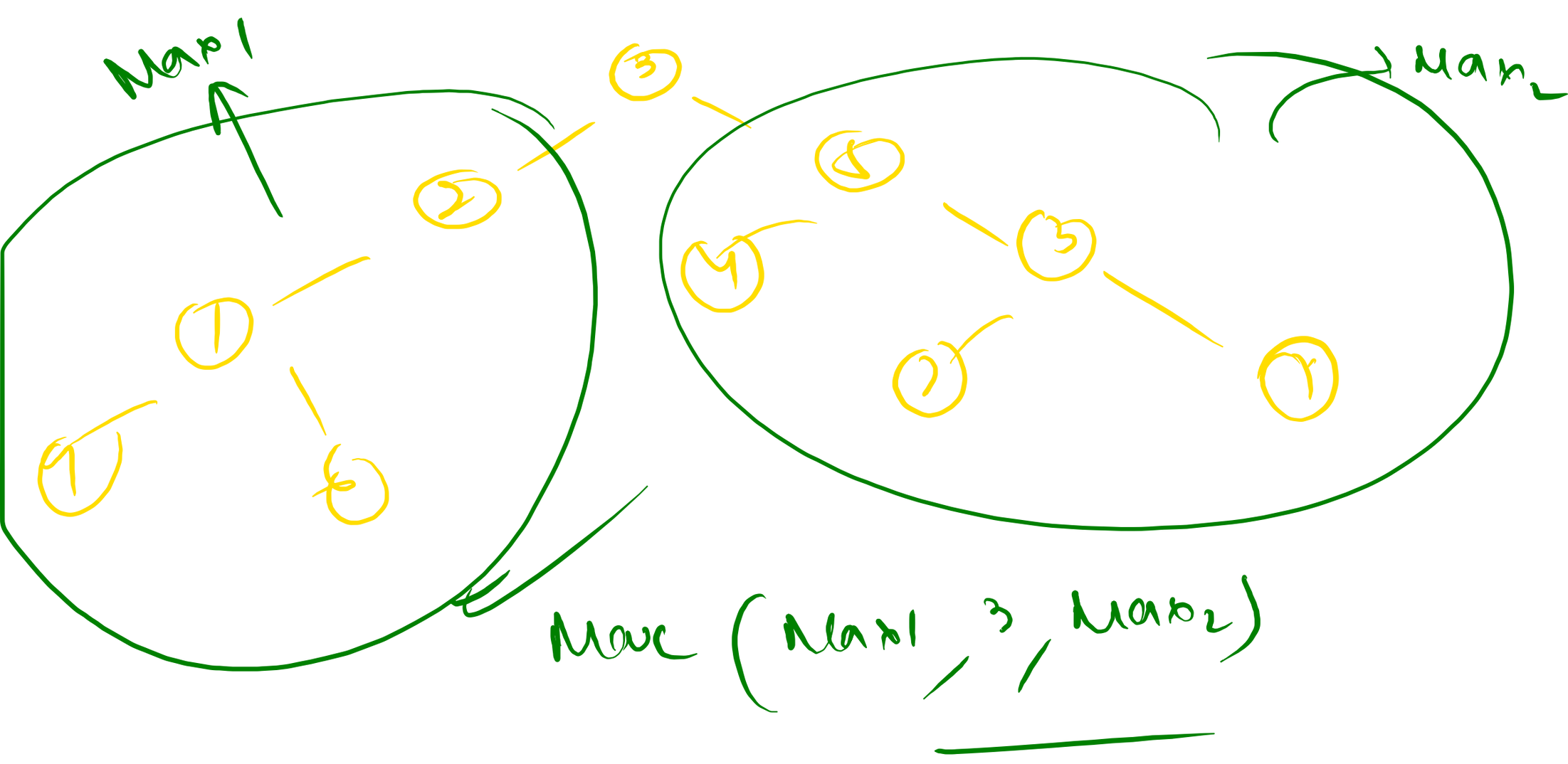
Preorder: 1, 2, 4, 5, 3, 6, 7

Postorder: 4, 5, 2, 6, 7, 3, 1


super


Hitch
car hauled

Python Libraries → will be started



Max (Max1, 3, Max2)