(H) A (C) K (E) R

Day-5

```
for i in range(0,T):
    S = input()
    even, odd = S[::2], S[1::2]
    Print (even + " " + odd)
```

[::2] will bring out even numbered indexes

↳ 0, 2, 4 → H C E

it means start from zero and give every second value till the end of iteration

[1::2]

it means start from one and give every second value till the end of iteration.

1, 3, 5 → A K R

```
T = int(input())
for i in range(0, T):
    S = input()
    even = S[::2]
    odd = S[1::2]
    print(even + " " + odd)
```

indent

Question:
Is it not necessary to define odd and even first?
→ No, as we keep adding 2nd string.

if we used
even += S[i]
format, then we would have to.

Hepaximin

Pandemic.

Defen

Online rounses

---

example-

h 5 = N

1,2 3 4 5 = A

#we need to print

— 5, 4 3 2 1

Son in Range (0,N):

nev = ' '

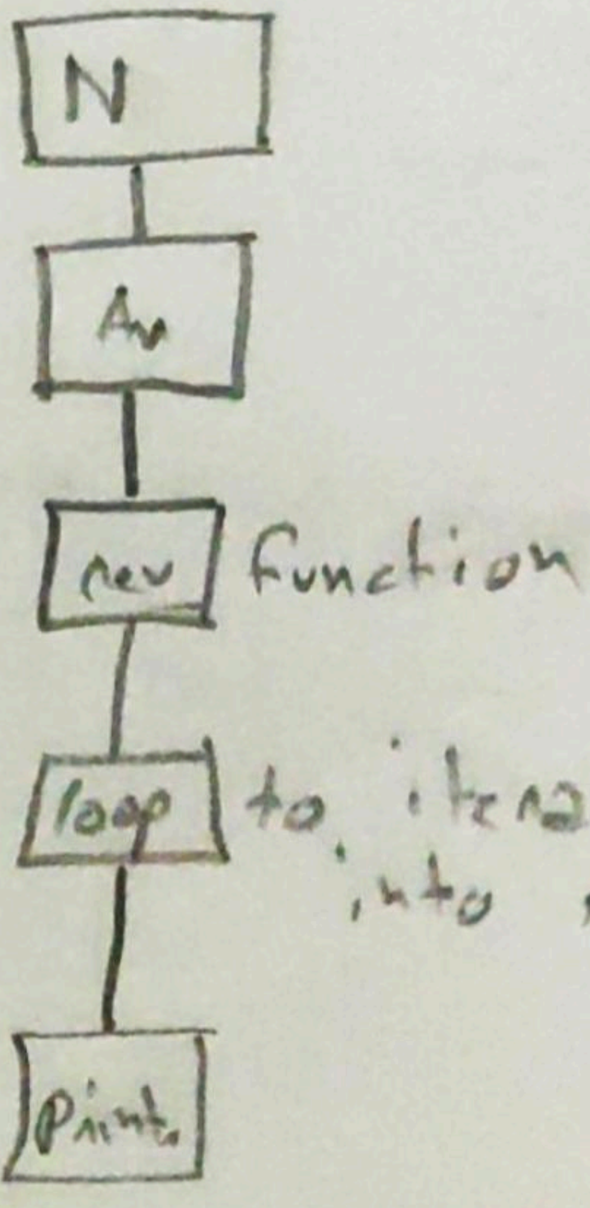
Print(reversed (An))

Array(A) of N numbers

the numbers seperated by spaces

Solved code-

```
if__name__=='__main__':
    n = int(input())
    am = list(map(int,input().rstrip()
                  .split()))

    nev = list(reversed(am))
    Print(*nev)
# print As numbers in reversed
order.
```



Print(reversed(An)) needs to be added

*(*nev) prints a list as space seperated integers.

* We could have also used one liner-

Print(*am[::-1])

Grading Students

Input the number of student grades - n

Input the each grade which is denoted by grades=[ ]

Rules to round-up - 1) If grade[i] is >38:

no rounding off multiple the closest

2) If the difference between and grade is <3'.

round up to next multiple

else: stay as it is.

Sample input	output
4	
73	75
67	67
38	40
33	33

∋ round(g[i]) - g[i] < 2:

~~Print~~ Print round (g[i])

else:

Print g[i]

28 - 22 = -2

✗ -2 < -2 ✓

20

Hepaximin

#save a dictionary which has names and their respective numbers.

How to create a dictionary

```
>>> d = { }
>>> d["George"] = 24
>>> d["Tom"] = 16
Print(d)
>>> { George = 24, Tom = 16 }
```

d[10] = 100

Given the keys and values →

Not given the keys and values →

```
n = int(input("Enter a n value:"))
d = { }
for i in range(n):
    keys = input()
    values = int(input())
    d[keys] = values
Print(d)
```

Dictionaries and Mapping

Algorithm - # input n, the number of queries in the dictionary

create an empty dictionary, d

iterate over n to input keys and values to dictionary.

Print out the key-value pair if matched

Else print ("Not found")

```
n = int (input ("Enter the n value: ")
d = { }
for i in rang (n):
    key = input()
    values = int (input())
    d[keys] = values.
    print (d[key] "=" d[value])"
else:
    print ("Not found")
```

P.T.O

Hepaximin

Day-8 - Dictionaries and maps

```
n = int (input ())
d = { }

for x in range (n):
    key, num = input (). split ()
    d[key] = num

while True:
    try:
        search = input ()
        if search in d[i]
            print (search + ' = ' + d [search])
        else:
            print ('Not found')
    except:
        break
```

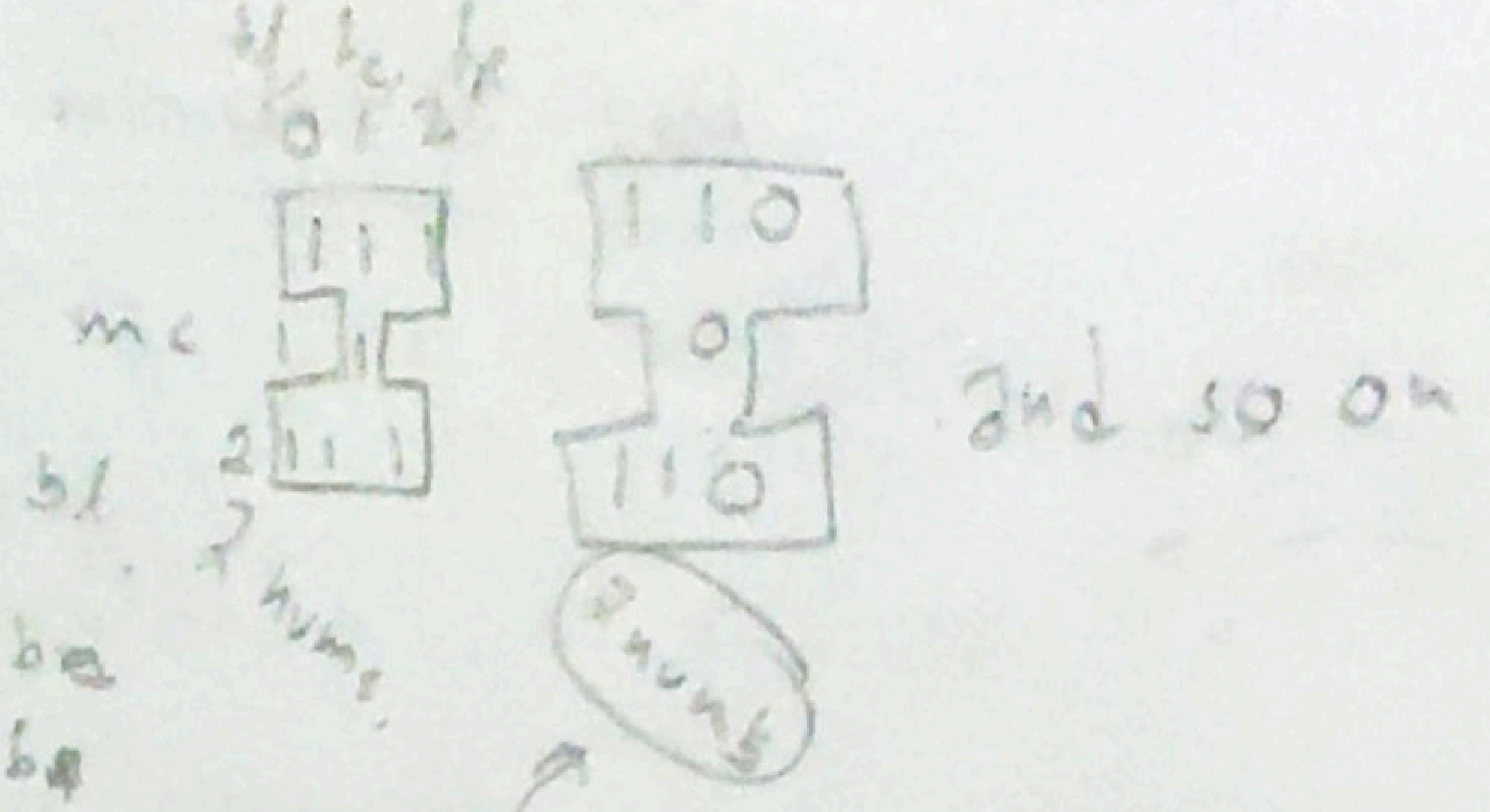
This line splits the inputs and makes them as keys and value-variables.

Try, except helps to find a problem without creating errors.

Columns

	0	1	2	3	4	5
0	1	1	1	0	0	0
1	0	1	0	0	0	0
2	1	1	1	0	0	0
3	0	0	2	4	4	0
4	0	0	0	2	0	0
5	0	0	1	2	4	0

— Rows



$-9 \times 7 = -63$

This is the minimum number we can get

lower value of Hourglass sum.

Column, 0,12, 123, 234, 345

Row — for centre only & rows are being used.

✓

11 - 2D - Arrays

Code - Day - 11

range (4), as a max four Hourglasses are possible in for column of 6

```
if name == '__main__':
    arr = [ ]

for _ in range (6):
    arr. append (list (map(int, input(). rstrip. split
                                                  ())))
highest = -9 *7
                6  - 2  = 4
for row in range (len (arr) - 2):
    for col in rang (len (arr[row] - 2)):
                        4           - 2   = 2
    tl = arr[ row] [col]
                                          0,1,2
    tc = arr[ row] [col+1]
    tr = arr[ row ] [ col +2]
    c = arr[ row+1] [col+1]

    bl = arr [row +2] [col]
    bc = arr [row+2] [col+1]
    br = arr [row+z] [col +2]

    total = tl + tc + tr + c + bl + bc + br

    highest = max( total, highest)

    pring = (highest)  => print the max of 2
```

Hepaximin

Day - 12 Inheritance

Takes inputs of Person's (class 1) first name, last name and id and Student's (class 2) scores.

calculate the average of their scores. (sum/n)

Output Name, ID, and grade.

We just need to create an inherited class.

- Inherited class inherits the instances of the base class.

Code-

Class Person: → Base

```
def __init__ (self, firstName, lastName, idNumber):
    self.firstName = firstName
    self.lastName = lastName
    self.idNumber = idNumber
```

def Print

```
def printPerson (self):
    Print ("Name:" self.lastName. + ", " self.firstName)
    Print ("ID:" self.idNumber)
```

class Student(Person): → inheited.

```
def __init__ (self, firstName, lastName, idNumber, Scores)
    self.scones = scones
```

Person.__init__ (self, firstName, lastName, idNumber) → invoke, else won't get the functions and instances.

```
def calculate (self):
    scone = sum(scones) / numscones
    if sone < 40:
        return "f"
    if scone < 55:
        return "D"
```

```
elif score <=70:
    return "P"

elif score <=80:
    return "A"

elif score <90:
    return "E"

elif score <= 100:
    return "O"


line = input().split()
firstName = line[0]
lastName = line[1]
idNum = line[2]
numScores = int(input())
scores = list(map(int, input().split()))
s = student(firstName, lastName, idNum, scores)
s.printperson()
Print("Grade:", s.calculate())
```

ABstract classes – Day-12

Given a Book class and a solution class, Create a MyBook class that ~~does~~
- inherits from Book

# has 3 ~~constant~~ parameters – str.(title)
str.(author)
int (Price)

invoke the derived class to the base class.

Code – importing ABstract Base clars method.

```
from abc import ABCMeta, abstractmethod
class Book (object, metaclass = ABCMeta):
    def __init__ (self, title, author):
        self.title = title
        self.author = author
    @abstractmethod
    def display (): pass
class MyBook (Book):
    def __init__ (self, title, author, price):
        self.price = price
        Book.__init__ (self, author, title)
def display (self):
    Print ("Title:", title)
    Print ("Author:", Author)
    Print ("Price:" Price)
```

code used to create an ABC

used to mark this method as abstract

invoking the parameters

implementation of instances

```
title = input ()
author = input ()
price = input ()
new_ novel = MyBook (title, author, price)
new_ novel. display()
```

A class is called an Abstract class if it contains one of more abstract methods.

An Abstract method is a method that is declared, but contains no implementation. (In this case, display().)

Always use code to invoke subclasses to the base class.

@abstract method decoration is used to mark abstract methods of the base class.

-meta class = ABCMeta - defines the class as an abstract base class.

We instantanized the display function in the subclass.

Note- use (self) default functionality to subclasses.

Why use ABC : ABstract classes allow you to provide

Day-14 Scope

Given a class difference, we have to

- Create a constructor that takes an array of integers and saves it to elements.
- A computeDifference function that finds the maximum absolute difference between any two numbers in the array and stores it in maximumDifference

Example-

1, 2, 3

|1-2| = 1

|1-3| = 2 → max

|2-3| = 1

min

5 1, 2, 3, 4, 5

~~+3 +2~~ |1-2| = 1

|1-3| = 2

|1-4| = 3

|2-3| = 1 → min

|2-4| = 2

|3-4| = 1

2 = 1, 2, 3

|max(a) - min(a)| → use self, as no attribute attached to class.

```
Class Difference:
    def __init__(self, a):
        self.__elements = a

    def computeDifference(self):
        self.maximumDifference = abs(max(a) - min(a))
```

Hepaximin

Day-15 - Linked list

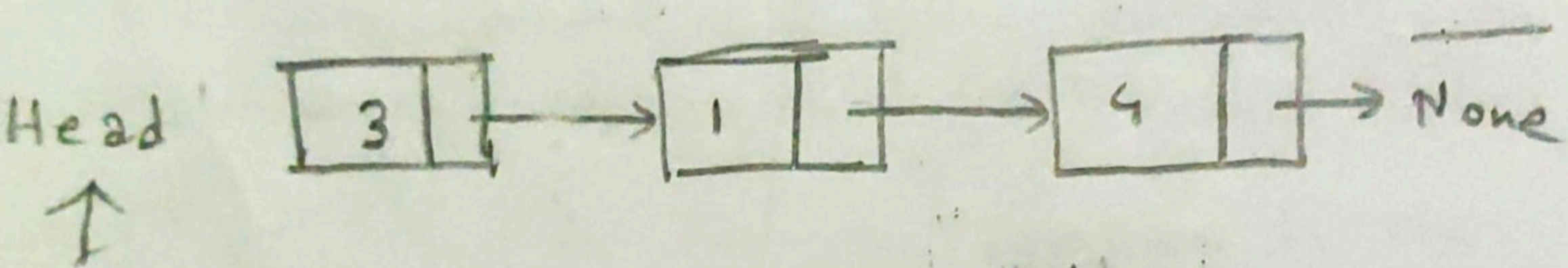
Linked list - It is a collection of nodes.

Head - The first node in the collection.

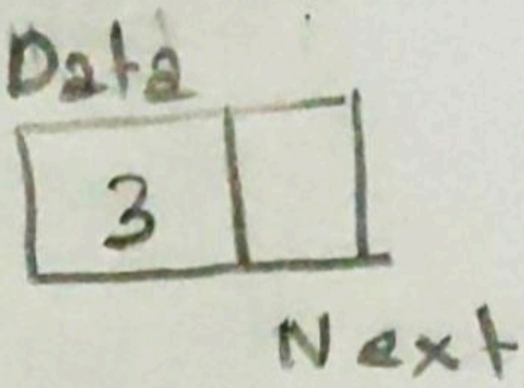
~~No~~

The last node must have its next reference pointing to None to determine the end of the list. Here's how it looks.

* singly Linked list.



Example of a node -



Each node has two fields as shown above -

1) Data contains the value to be stored in the node.
2) Next contains a reference to the next node on the list.

address / memory.

How to create a linked list with code-

Note- we basically create a Node, and the keep attaching nodes to it and the rename the last one to be ~~as~~ refferring to. None.

The Head node is never going to have a data. It just helps us to know that it is the starting of the list.

- Algorithm to create a linked List-

1) create a class node
2) create a class Linked list which is a subclass of node.
3) create a function to iterate over the values to be added to ~~creat~~ link them to the nodes.
4) Name the last Node as None.

Class node:

```
def __init__ (self, data = None):
    self.data = data.
    self. next = None

class Linked list:
    def __init__ (self):
        self.head = node()

    def append (self, data):
        new_node = node (data)
        cur = self.head
```

while cun.nextt != None:

    cun = cun.next

cun.next = new_node.

return head → Because we need to give out the starting point as reference.

Day-15 - code- ~~day~~ ~~def insent~~

```
def insent (self, head, data):
    if (head == = None):
        head = Node (data)
    elif (head.next = = None):
        head. next = Node (data)
    else:
        self. insent (head. next, data)

    return head.
```

head.next refers to the next element after head.

Explanation -

The function gets called with the whole linked list

If the list is empty,
    insent the element as head.

Otherwise if there is no element after head,
    put the data after the head.

otherwise,
    call the function with all of the elements except ^ the current head.

Day-16 – Exception – Try: Except

We use Try: Except to handle errors. We can also choose to print something if we run into an error.

Example –

```
try:
    number = int(raw_input("Enter a number between 1-10"))
except ValueError:
    Print e("Err numbers only")
    Sys.exit()

Print("you entered number", number)
```

Day - 16

We have to read a string
convert it into an integer.

If we can't convert it to an integer, print
"Bad string".

Things to use - str to Int converter and exception handling.

```
s = input().strip()

Try:
```

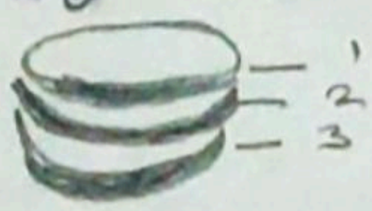
Day-18 - Queues and stacks

*Both use Lists.

Stacks - A data structure which uses the principle (LiFO) - Last-in - Finst-Out.

stacks should be able to perform atleast three operations.
- Peek - Return the object at the top. (without removing it)
- Push - Add an object as an argument to the top of stack.
- Pop - Remove the object at top and return it.

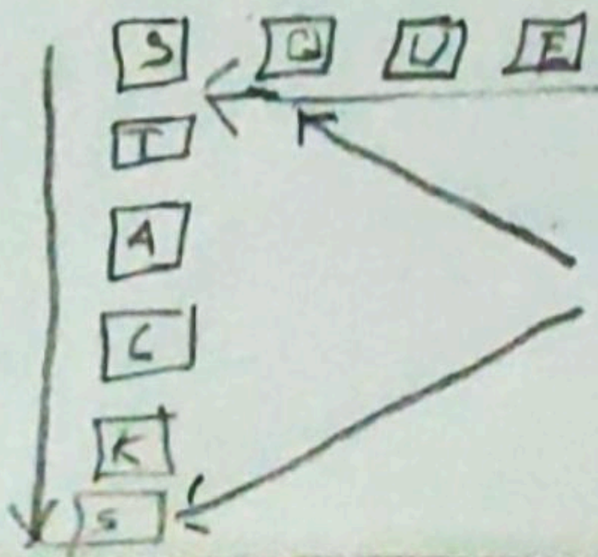
For - visual Reference, Imagine a stack of plates.



Queues - A Queues is a datap structure which uses a principle, (FiFO) - First in First Out.

Queues should be able to perform at least two operations.
- Enqueue - Add an object to the back of the line.
- Dequeue - Remove an object at the head of the line and return it; the element that was previously second is now the first item.

For visual Reference, Imagine a Queue to get in a bus.



N A N
0 1 2

Stack -

Pop() - 2

Push(A) - NANA

Peek() -

Queues -

Enqueue(self, item):

self.que.append(A) OR

return self.~~stack~~ Q == NANA

self.queue.insert(0, A)
return self.queue
>>> NANA.

Dequeue(self):

if len(self.queue) < 1:

return None

return self.queue.pop(0),

>>> ANA

Difference Between Append() and Insert()

- Append() adds a new entry at the end of the list
- Insert(position, new_entry) can create a new entry at exactly in the position you want.

Day - 18 - Code -

code -

```
class Solution:
    def __init__(self):
        self.stack = []
        self.queue = []

    def pushCharacter(self, char):
        self.stack.append(char)

    def enqueueCharacter(self, char):
        self.queue.insert(0, char)

    def popCharacter(self):
        return self.stack.pop()

    def dequeueCharacter(self):
        return self.queue.pop()
```

trick, instead of using append.

Hepaximin

Day - 10 - Interfaces

Interfaces are platform that is a class and only can have abstract methods.

Note- Abstract methods can have static, class and instance methods.

It is not compulsory to inherit abstract class but if it is compulsory to inherit interface.

Abstract class has parameters which need to inherited unless it is redefined.

Interface

but methods do not need to be defined.

Vidy - ~~12~~ 20 Generics

confusions — Generics

— If --name__ == '__main__'

— — name__

- duck typing.
- Recursion

If module which refers to --name--
was indeed set to "__main__" and it calls two functions, printing the two strings of the function. of --name--

- suppose we are writing a program/ module and name it calculator. It has a function that will be executed only if it is the main program. and not imported. Example-

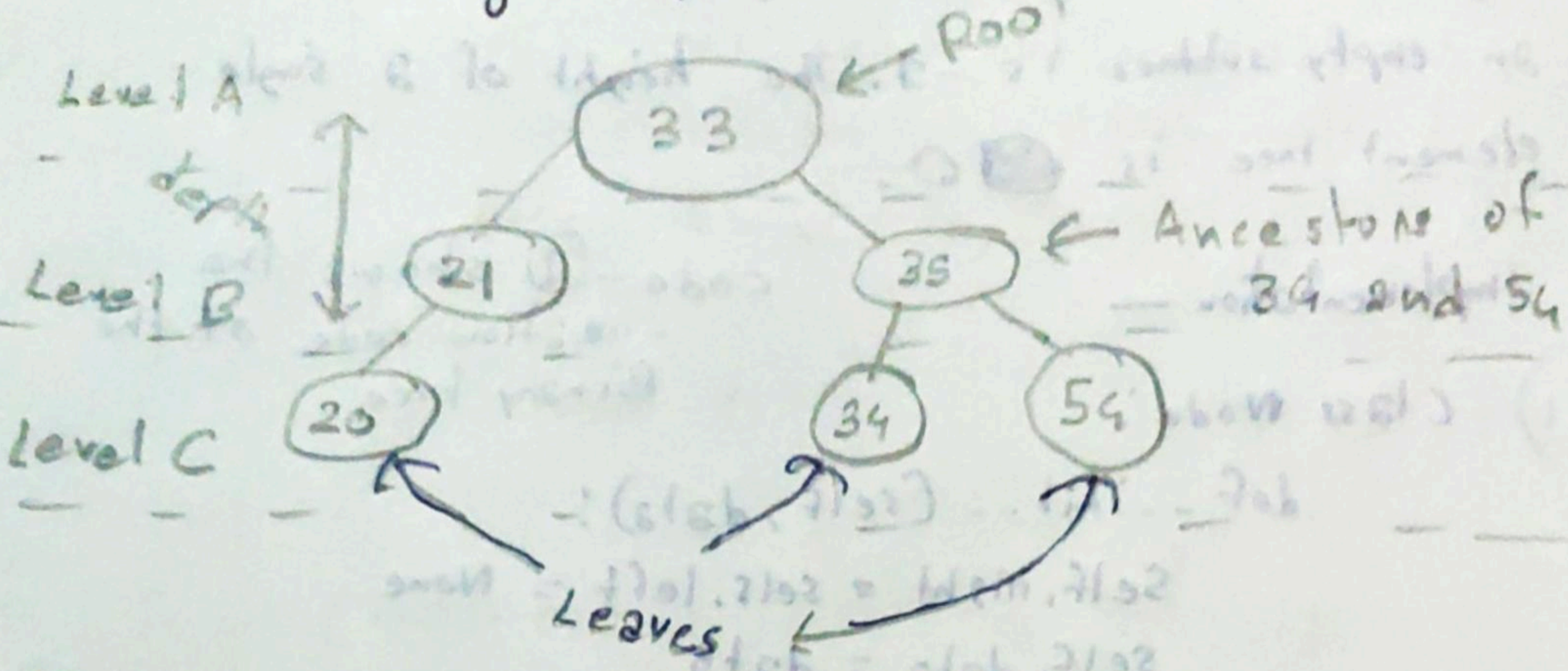
```
# Calculator.py

def myFunction():
    Print ("1+2") (The
    Print ('The value of __name__ is' + __name)

def main():
    myFunction()

if __name__ == '__main__': main()
```

3 2) 1)

Hepaximin

Day-22 - Binary Search Trees

Is a form of Linked list, but is more organised in an ascending order.



Things to know-

Root - The first node which is the topmost node and is the parent to all other nodes.

Ancestor - the parent node of the nodes connected to it. For example, 35 is ancestor of 34 and 54, and 33 is ancestor of all values.

Leaf - A non-root node with no children is called a leaf. Example from the diagram above - 20, 34 and 54.

Depth - The depth of some node is the distance from the tree's root node.

Height - The height of a tree is the distance (Level gap) between the root node and its furthest leaf. Any node has a height of 1, and the height of an empty subtree is -1. The height of a single element tree is 0.

Implementation —

code - (1) shows the creation code of the Binary tree.

(1)

```
Class Node:
    def __init__ (self, data):
        Self.right = self.left = None
        self.data = data

Class Solution:
    def insert (self, root, data):
        if root == None:
            ~~return Node (data)~~
            return Node (data)

        else:
            if data <= root.data:
                cur = self.insert (root.left, data)
                root.left = cur

            else:
                cur = self.insert (root.right, data)
                root.right = cur

        return root.
```

def get

(2) shows the creation and function to find the height of the tree.

(2)
```
def getHeight(self, root):
    if root:
        left = self.getHeight(root.left) + 1
        right = self.getHeight(root.right) + 1
        return max(left, right)
    else:
        return -1
```

Recursion

(3)
```
T= int (input())
myTree = Solution()
root = None
for i in range(T):
    data = int(input())
    root = myTree.insert(root, data)
height = myTree.getHeight(root)
Print(height)
```

(3) is the call-up and inputs of the tree values.

Day - 26

Inputs - 1) Due_date (d_1, m_1, y_1)
2) Return_date. (d_2, m_2, y_2)

What my program will do -

1) If book is returned before return date $(d_1 > d_2, m_1 > m_2, y_1 \geq y_2)$
no fine. (Print)

2) If book returned after the return date but $(d_1 < d_2, m_1 \geq m_2, y_1 \geq y_2)$
Fine = 15 Hackos × (number of days) $d_2 - d_1$

3) If the book returned after the return date $\begin{pmatrix} d_1 > d_2 \\ m_1 < m_2 \\ y_1 \geq y_2 \end{pmatrix}$

4) If the book is returned after Calender year in which it was expected $(d_1 \geq d_2, m_1 \geq m_2, y_1 < y_2)$
Fixed fine of 1000 Hackos.

[d, m, y]

Day-27 - testing

Unit testing is the testing of all the small components of a function or program.

Today, we have to test if our function replies with appropriate answers when inputs are given.

Input-1) Empty array should return an empty array followed by a exception.

Input 2) Array with Unigues values which outputs the lowest input.

Input 3) Array with two same minimum values but has to output the lower index.

Code - Class TestData Empty Array (object):

```
@staticmethod
def get_array():
    return []
```

class TestData with Unigue Values (object):

```
@staticmethod
def getarray():
    return [3,1,2]

def get_expected_result():
    return 4
```

Hepaximin

```
class TestData Exactly Two Different Minimums (object):
    @staticmethod
    def get_array():
        return [3, 1, 1]
    @staticmethod
    def get_expected_result():
        return 1
```

Day - 28 - RegEx, - Databases

A Regular Expression is a sequence of characters that defines a search pattern.

For example - ^a ... s$

This pattern is: Any five letters string starting with [a] and ending with [s]

Meta characters, like [^] and [$], are used to specify regular expressions.

Meta characters - [ ], [.], [^], [$], [+], [?], [{}], [()], [\], [/]

[ ] specifies a set of characters you wish to match.

- [a-c] string - adb — 2 matches.

[.] A period matches any single character.

[^] caret symbol is used to check if a string starts with a certain character.

— ^ab — string - abc — 1 match

Hepaximin

[$] Dollar sign is ud to check if a string ends with a certain character.

[*] star symbol matches zero or more occurances of the pattern to it.

[ma*n] - string - man — 1 match

- string mn - 1 match.
- string - main - no match (a isnt followed by n)

Code - 1

```
if __name__ == '__main__':
    N= int(input())

    Pattern = r"@gmail\.com$"
    regex = re.compile(pattern)
    finstNames list = []

for N_itn in range(N):
    finstName EmailID = input().split()
    finstName = finstName EmailID [0]
    email ID = finstName EmailID [1]

    if regex.seanch(email ID):
        finstNameslist.append(finstNane)

    finstNanesList.sont()

    for name in finstNanesList:
        print(name).
```

Hepaximin