Name: Shubham Kale

Roll No: J026

Day 1 : Hello World

In [1]:

```
input_string=input()
print("Hello World")
print(input_string)
```

```
Hey
Hello World
Hey
```

Day 2 : Operators

In [2]:

```
mealprice=float(input("Enter Meal price: "))
tip=int(input("Enter tip: "))
tax=int(input("Enter tax: "))
tip=tip*mealprice/100;
tax=tax*mealprice/100;

totalmealprice=mealprice+tip+tax;

print("Total Meal Cost is:",round(totalmealprice))
```

```
Enter Meal price: 69
Enter tip: 21
Enter tax: 42
Total Meal Cost is: 112
```

Day 3: Intro to Conditional Statements

In [3]:

```
n=int(input("Enter Integer: "))

if n%2==1:
    print("Weird")
elif 2<=n<=20:
    print("Not weird")
elif 6<=n<=20:
    print("Weird")
else:
    print("Not weird")
```

```
Enter Integer: 69
Weird
```

Day 4: Class Vs Instance

In [5]:

```
class Person():
```

```
    def __init__(self,intialAge):

        if intialAge>0:
            self.age= intialAge
        else:
            self.age=0
            print('Age is not valid, setting age to 0.')

    def yearPasses(self):
         self.age= self.age +1

    def amIOld(self):

        if self.age< 13:
            print('You are young.')
        elif 13<=self.age<18:
            print('You are a teenager.')
        else:
            print('You are old.')

t = int(input())
for i in range(0, t):
    age = int(input())
    p = Person(age)
    p.amIOld()
    for j in range(0, 3):
        p.yearPasses()
    p.amIOld()
    print("")
```

```
2
21
You are old.
You are old.

2
You are young.
You are young.
```

Day 5: Loops

In [6]:

```
n=int(input('Enter Integer: '))
for i in range(1,11):
    print(str(n)+"x"+str(i)+"="+str((n*i)))
```

```
Enter Integer: 18
18x1=18
18x2=36
18x3=54
18x4=72
18x5=90
18x6=108
18x7=126
18x8=144
18x9=162
18x10=180
```

Day 6:Lets Reviews

In [7]:

```
T=int(input())

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

```
3
Shubham
Suhmhba
Kale
Klae
Football
Foblotal
```

Day 7: Arrays

In [8]:

```
if __name__ == '__main__':
    n = int(input().strip())

    arr = list(map(int, input().rstrip().split()))
    m=map(str,arr[::-1])
    print(" ".join(m))
```

```
3
3
3
```

Day 8:Dictionaries and Maps

In [10]:

```
# n, Enter number of record you need to insert in dict
n = int(input())
d = dict()

# enter name and number by separate space
for i in range(0, n):
    name, number = input().split()
    d[name] = number
# print(d)      #print dict, if needed

# enter name in order to get phone number
for i in range(0, n):
    try:
        name = input()
        if name in d:
            print(f"{name}={d[name]}")
        else:
            print("Not found")
    except:
        break
```

```
3
Shubham 8888888888
Shantanu 9999999999
Abhishek 7777777777
Shubham
Shubham=8888888888
Abhishek
Abhishek=7777777777
Swapnil
Not found
```

Day 9:Recursion

In [11]:

```
def fact(n):
    if n<=1:
        return 1
    else:
        return n*fact(n-1)
```

```
n=int(input())
print(fact(n))
```

```
6
720
```

Day 10: Binary Numbers

In [12]:

```
n=int(input())
count=0
while n:
    n=n&(n<<1)
    count+=1
print(count)
```

```
69
1
```

Day 11: 2D array

In [13]:

```
arr = []
for arr_i in range(6):
    arr_temp = list(map(int,input().strip().split(' ')))
    arr.append(arr_temp)
max = 0

for i in range(0,4):
    for j in range(0,4):
        sum = 0
        sum= arr[i][j]+arr[i][j+1]+arr[i][j+2]+arr[i+1][j+1]+arr[i+2][j]+arr[i+2][j+1]+
arr[i+2][j+2]
        if i==0 and j==0:
            max = sum
        if sum > max:
            max =sum

print(max)
```

```
1 2 3 4 5 6
6 5 4 3 2 1
2 3 5 6 4 1
9 8 5 6 1 2
9 7 8 4 3 1
6 3 7 4 8 2
45
```

Day 12:Inheritance

In [16]:

```
class Person:
    def __init__ (self,first_name,last_name,id_number):
        self.first_name=first_name
        self.last_name=last_name
        self.id_number=id_number
    def printperson(self):
        print("Name: ",self.first_name+","+self.last_name)
        print("ID:",self.id_number)

class student(Person):
    def __init__(self,first_name,last_name,id_number,scores):
        self.first_name=first_name
        self.last_name=last_name
        self.id_number=id_number
```

```
        self.scores=scores
        Person(self.first_name,self.last_name,self.id_number)

    def Calculate(self):
        g= sum(scores)/len(scores)

        if 90<g<=100:
            return 'O'
        elif 80<=g<=90:
            return 'E'
        elif 70<=g<=80:
            return 'A'
        elif 55<=g<=70:
            return 'P'
        elif 40<=g<=55:
            return 'D'
        elif p<40:
            return 'T'
```

Day 13: Abstract Classes

In [17]:

```
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):
    price = 0
    def __init__(self, title, author, price):
        super(Book, self).__init__()
        self.price = price

    def display(self):
        print("Title: "+ title)
        print("Author: "+ author)
        print("Price: "+ str(price))

title=input()
author=input()
price=int(input())
new_novel=MyBook(title,author,price)
new_novel.display()
```

```
Basics 101
Kobe
500
Title: Basics 101
Author: Kobe
Price: 500
```

Day 14: Scope

In [40]:

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

    def computeDifference(self):
        x=self.a
        max=0
        min=0
        for i in x:
            if i>max:
```

```
                max=i
            if i<min:
                min=i
        return max-min

diff=Difference([3,4,7,3,2,8,9,1,7,8,1,0])
diff.computeDifference()
```

Out[40]:

```
9
```

Day 15: Linked list

In [31]:

```
3class Node:
    def __init__(self,data):
        self.data = data
        self.next = None
class Solution:
    def display(self,head):
        current = head
        while current:
            print(current.data,end=' ')
            current = current.next

    def insert(self, head, data):
        new = Node(data)
        if head:
            tail = head
            while tail.next:
                tail = tail.next
            tail.next = new
            return head
        else:
            return new

mylist= Solution()
T=int(input())
head=None
for i in range(T):
    data=int(input())
    head=mylist.insert(head,data)
mylist.display(head);
```

```
5
2
4
1
6
3
2 4 1 6 3
```

Day 16: strings to integer

In [41]:

```
S = input().strip()
try:
    print(int(S))
except ValueError:
    print('Bad String')
```

```
Shubham
Bad String
```

Day 17: More Exceptions

In [33]:

```
class Calculator(object):
    def __init__(self):
                self.object = object
    def power(self, a,b):
                self.a = a
                self.b = b
                if self.a>=0 and self.b >=0:
                   return (a**b)
                else:
                   e = "n and p should be non-negative"
                   return e


myCalculator=Calculator()
T=int(input())
for i in range(T):
    n,p = map(int, input().split())
    try:
        ans=myCalculator.power(n,p)
        print(ans)
    except Exception as e:
        print(e)
```

```
4
3 5
243
2 4
16
-1 2
n and p should be non-negative
-1 -2
n and p should be non-negative
```

Day 18: Queues and Stacks

In [43]:

```
class Solution:
    def __init__(self):
        self.mystack = list()
        self.myqueue = list()
        return(None)

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

    def popCharacter(self):
          return(self.mystack.pop(-1))

    def enqueueCharacter(self, char):
          self.myqueue.append(char)

    def dequeueCharacter(self):
          return(self.myqueue.pop(0))
# read the string s
s=input()
#Create the Solution class object
obj=Solution()

l=len(s)
# push/enqueue all the characters of string s to stack
for i in range(l):
    obj.pushCharacter(s[i])
    obj.enqueueCharacter(s[i])

isPalindrome=True
'''
pop the top character from stack
```

```
dequeue the first character from queue
compare both the characters
'''
for i in range(l // 2):
    if obj.popCharacter()!=obj.dequeueCharacter():
        isPalindrome=False
        break
#finally print whether string s is palindrome or not.
if isPalindrome:
    print("The word, "+s+", is a palindrome.")
else:
    print("The word, "+s+", is not a palindrome.")
```

```
mom
The word, mom, is a palindrome.
```

Day 19: Interfaces

In [36]:

```
class AdvancedArithmetic(object):
    def divisorSum(n):
        raise NotImplementedError

class Calculator(AdvancedArithmetic):

    def divisorSum(self, n):
            if n == 1:
                return 1
            else:
                factor_sum = 1 + n
                for i in range(2, n//2 + 1):
                    if n % i == 0:
                        factor_sum += i
                return factor_sum

n = int(input())
my_calculator = Calculator()
s = my_calculator.divisorSum(n)
print("I implemented: " + type(my_calculator).__bases__[0].__name__)
print(s)
```

```
65
I implemented: AdvancedArithmetic
84
```

Day 20: Sorting

In [37]:

```
import sys

n = int(input().strip())
a = list(map(int, input().strip().split(' ')))

swaps = 0
is_sorted = False

while not is_sorted:
    is_sorted = True
    i = 0
    for i in range(0, len(a)):
        if i < len(a) - 1:
            if a[i] > a[i+1]:
                a[i], a[i+1] = a[i+1], a[i]
                is_sorted = False
                swaps += 1

print('Array is sorted in {} swaps.'.format(swaps))
```

```
print('First Element: {}'.format(a[0]))
print('Last Element: {}'.format(a[len(a)-1]))
```

```
10
0 9 8 7 6 5 4 3 2 1
Array is sorted in 36 swaps.
First Element: 0
Last Element: 9
```

Day 21: Generics

In [44]:

```
#1
from typing import TypeVar, Generic
from logging import Logger

T = TypeVar('T')

class LoggedVar(Generic[T]):
    def __init__(self, value: T, name: str, logger: Logger) -> None:
        self.name = name
        self.logger = logger
        self.value = value

    def set(self, new: T) -> None:
        self.log('Set ' + repr(self.value))
        self.value = new

    def get(self) -> T:
        self.log('Get ' + repr(self.value))
        return self.value

    def log(self, message: str) -> None:
        self.logger.info('%s: %s', self.name, message)
```

In [45]:

```
#2
from typing import TypeVar, Iterable, Tuple, Union
S = TypeVar('S')
Response = Union[Iterable[S], int]

# Return type here is same as Union[Iterable[str], int]
def response(query: str) -> Response[str]:
    ...

T = TypeVar('T', int, float, complex)
Vec = Iterable[Tuple[T, T]]

def inproduct(v: Vec[T]) -> T: # Same as Iterable[Tuple[T, T]]
    return sum(x*y for x, y in v)
```

Day 22: Binary serach Tree

In [5]:

```
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)
        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 getHeight(self,root):
        #Write your code here
        if root == None or root.left == root.right == None:
            return 0
        else:
            return 1+ max(self.getHeight(root.left),
                          self.getHeight(root.right))

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)
```

```
5
1
2
6
3
7
3
```

Day 23: BST level Order Traversal

In [44]:

```
import sys

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)
        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 levelOrder(self,root):
        queue = [root] if root else []

        while queue:
            node = queue.pop()
            print(node.data, end=" ")

            if node.left: queue.insert(0,node.left)
            if node.right: queue.insert(0,node.right)

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

```
8
2
4
2
1
5
3
5
6
2 2 4 1 3 5 5 6
```

Day 24: More Linked lists

In [45]:

```
class Node:
    def __init__(self,data):
        self.data = data
        self.next = None
class Solution:
    def insert(self,head,data):
            p = Node(data)
            if head==None:
                head=p
            elif head.next==None:
                head.next=p
            else:
                start=head
                while(start.next!=None):
                    start=start.next
                start.next=p
            return head
    def display(self,head):
        current = head
        while current:
            print(current.data,end=' ')
            current = current.next

    def removeDuplicates(self,head):
        if head == None:
            return head
        fptr = head.next
        sptr = head
        ha = {}
        while fptr != None:
            if sptr.data not in ha:
                ha[sptr.data] = True
            if fptr.data in ha:
                sptr.next = fptr.next
                fptr = fptr.next
                continue
            sptr = fptr
            fptr = fptr.next

        return head

mylist= Solution()
T=int(input())
head=None
for i in range(T):
    data=int(input())
    head=mylist.insert(head,data)
head=mylist.removeDuplicates(head)
mylist.display(head);
```

```
5
5
3
4
5
2
```

```
5 3 4 2
```

Day 25: Running Time and Complexity

In [46]:

```
for _ in range(int(input())):
    num = int(input())
    if(num == 1):
        print("Not prime")
    else:
        if(num % 2 == 0 and num > 2):
            print("Not prime")
        else:
            for i in range(3, int(num**(1/2))+1, 2):
                if num % i == 0:
                    print("Not prime")
                    break
            else:
                print("Prime")
```

```
5
2
Prime
6
Not prime
1
Not prime
9
Not prime
4
Not prime
```

Day 26: Nested Logic

In [47]:

```
rd, rm, ry = [int(x) for x in input().split(' ')]
ed, em, ey = [int(x) for x in input().split(' ')]

if (ry, rm, rd) <= (ey, em, ed):
    print(0)
elif (ry, rm) == (ey, em):
    print(15 * (rd - ed))
elif ry == ey:
    print(500 * (rm - em))
else:
    print(10000)
```

```
1 2 2000
4 3 1000
10000
```

Day 27: Testing

In [48]:

```
def minimum_index(seq):
    if len(seq) == 0:
        raise ValueError("Cannot get the minimum value index from an empty sequence")
    min_idx = 0
    for i in range(1, len(seq)):
        if seq[i] < seq[min_idx]:
            min_idx = i
    return min_idx

def minimum_index(seq):
    if len(seq) == 0:
```

```
        raise ValueError("Cannot get the minimum value index from an empty sequence")
    min_idx = 0
    for i in range(1, len(seq)):
        if seq[i] < seq[min_idx]:
            min_idx = i
    return min_idx

class TestDataEmptyArray(object):

    @staticmethod
    def get_array():
        return []

class TestDataUniqueValues(object):

    @staticmethod
    def get_array():
        return [7, 4, 3, 8, 14]

    @staticmethod
    def get_expected_result():
        return 2

class TestDataExactlyTwoDifferentMinimums(object):

    @staticmethod
    def get_array():
        return [7, 4, 3, 8, 3, 14]

    @staticmethod
    def get_expected_result():
        return 2

def TestWithEmptyArray():
    try:
        seq = TestDataEmptyArray.get_array()
        result = minimum_index(seq)
    except ValueError as e:
        pass
    else:
        assert False


def TestWithUniqueValues():
    seq = TestDataUniqueValues.get_array()
    assert len(seq) >= 2

    assert len(list(set(seq))) == len(seq)

    expected_result = TestDataUniqueValues.get_expected_result()
    result = minimum_index(seq)
    assert result == expected_result


def TestiWithExactyTwoDifferentMinimums():
    seq = TestDataExactlyTwoDifferentMinimums.get_array()
    assert len(seq) >= 2
    tmp = sorted(seq)
    assert tmp[0] == tmp[1] and (len(tmp) == 2 or tmp[1] < tmp[2])

    expected_result = TestDataExactlyTwoDifferentMinimums.get_expected_result()
    result = minimum_index(seq)
    assert result == expected_result

TestWithEmptyArray()
TestWithUniqueValues()
TestiWithExactyTwoDifferentMinimums()
print("OK")
```

```
OK
```

Day 28: Regex,patterns and Intro to Databases

In [48]:

```
import sys
import re


N = int(input().strip())
list =[]
for a0 in range(N):
    firstName,emailID = input().strip().split(' ')
    firstName,emailID = [str(firstName),str(emailID)]
    if re.search("@gmail.com",emailID):
        list.append(firstName)

list2 = (sorted(list))
for elem in list2:
    print (elem)
```

```
2
shubham@gmail.com kale@gmail.com
ok@gmail.com hi@gmail.com
ok@gmail.com
shubham@gmail.com
```

Day 29: Bitwise And

In [54]:

```
import math
import os
import random
import re
import sys

def FindMaxAB(n, k):
    max_ab = 0
    for i in range(k - 2, n):
        for j in range(i + 1, n + 1):
            ab = i & j
            if ab == k - 1:
                return ab
            if max_ab < ab < k:
                max_ab = ab
    return max_ab


for i in range(int(input().strip())):
    n, k = map(int, input().split())
    print(FindMaxAB(n, k))
```

```
3
4 5
0
4 6
0
5 2
1
```

In [ ]:

```

```