ML Assignment 1

By Harshitha MU

Roll No.: J076

In [59]:

```
#Q0
inputString = input()

print('Hello, World.')
print(inputString)
```

```
heyoo~
Hello, World.
heyoo~
```

In [9]:

```
#Q1
j=int(input())
e=float(input())
t=input()
i=4
d=4.0
s='Hacker rank'
print(i+j)
print(d+e)
print(s+t)
```

```
2
3
4
6
7.0
Hacker rank4
```

In [1]:

```
#Q2
mealCost = float(input())
tip = int(input())
tax = int(input())
tip=tip*mealCost/100;
tax=tax*mealCost/100;
totalcost=mealCost+tip+tax;

print ("The total meal cost is %s dollars." %str(int(round(totalcost, 0))))
```

```
20
3
2
The total meal cost is 21 dollars.
```

In [2]:

```
#Q3

import sys


n = int(input().strip())

if n%2==1:
    ans = "Weird"

elif n>20:
    ans = "Not Weird"

elif n>=6:
    ans = "Weird"

else:
    ans = "Not Weird"

print(ans)
```

```
3
Weird
```

In [11]:

```
#Q4

class Person:
    def __init__(self,initialAge):

        if(initialAge>0):
            self.age=initialAge
        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("")
```

```
3
24
You are old.
You are old.

4
You are young.
You are young.

80
You are old.
You are old.
```

In [5]:

```
#Q5

import sys


N = int(input().strip())
```

```
for i in range(1, 11):
    print(str(N) +" x " + str(i) + " = " + str(N*i))
```

```
7
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70
```

In [6]:

```
#Q6
import sys

def printEvenIndexChar(s):
    l = len(s)
    output = ""
    for i in range(0,l,2):
        output += s[i]
    return output

def printOddIndexChar(s):
    l = len(s)
    output = ""
    for i in range(1,l,2):
        output += s[i]
    return output

t = int(input())
for a0 in range(0,t):
    s = input()
    print(printEvenIndexChar(s) + " " + printOddIndexChar(s))
```

```
3
4
4
5
5
6
6
```

In [7]:

```
#Q7
import sys


n = int(input().strip())
arr = list(map(int,input().strip().split(' ')))
ans = ""
for i in range(len(arr)-1 , -1, -1):
    ans += str(arr[i]) + " "

print(ans)
```

```
30
20
20
```

In [ ]:

```
#Q8
import sys
inputList=[]

for line in sys.stdin:
    inputList.append(line)

n = int(inputList[0])
entries = inputList[1:n+1]
queries = inputList[n+1:]

phoneBook = {}

for entry in entries:
    name, id = entry.split()
    phoneBook[name] = id

for query in queries:
    stripQuery = query.rstrip()
    if stripQuery in phoneBook:
        print(stripQuery + "=" + str(phoneBook[stripQuery]))
    else:
        print("Not found")
```

In [12]:

```
#Q9
def fact(n):
    if n<=1:
        return 1
    else:
        return n*fact(n-1)
n=int(input())
print(fact(n))
```

```
6
720
```

In [13]:

```
#Q10
n=int(input())
count=0
while n:
    n=n&(n<<1)
    count+=1
print(count)
```

```
60
4
```

In [ ]:

```
#Q11
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)
```

In [18]:

```
#Q12
```

```
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'
```

In [ ]:

```
#Q13
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()
```

In [ ]:

```
#Q14
class Difference:
    def __init__(self,a):
        self.a=a

    def computeDifference(self):
        return max(self.a)-min(self.a)
```

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

In [22]:

```
#Q15
class 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):
        if head is None:
            head=Node(data)
        elif head.next is None:
            head.next=Node(data)
        else:
            self.insert(head.next,data)
        return head

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

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

In [23]:

```
#Q16
import sys
S=input().strip()
try:
    r=int(S)
    print(r)
except ValueError:
    print("Bad String")
```

```
ml
Bad String
```

In [33]:

```
#Q17
class Calculator(Exception):
    def power(self,n,p):
        if (n<0 or p<0):
            raise Calculator("n and p should be non-negative")
        else:
            return pow(n,p)

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)
```

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

In [35]:

```
#Q18
import sys
from collections import deque

class Solution:
    def __init__(self):
        self.stack=deque()
        self.queue=deque()

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

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

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

    def dequeueCharacter(self):
        return self.queue.popleft();

s=input()
obj=Solution()
l=len(s)

for i in range(l):
    obj.pushCharacter(s[i])
    obj.enqueueCharacter(s[i])

isPalindrome=True

for i in range(l//2):
    if obj.popCharacter()!=obj.dequeueCharacter():
        isPalindrome=False
        break

if isPalindrome:
    print("The word, "+s+", is a palindrome.")
else:
    print("The word, "+s+", is not a palindrome.")
```

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

In [36]:

```
#Q19
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)
```

```
45
I implemented: AdvancedArithmetic
46
```

In [38]:

```
#Q20
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]))
```

```
9
1 2 3 4 43 3 4 87 99
Array is sorted in 3 swaps.
First Element: 1
Last Element: 99
```

In [39]:

```
#Q21
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)

#######

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*yforx,yinv)
```

In [ ]:

```
#Q22
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):

        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)
```

In [47]:

```
#Q23
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)
```

```
9
2
3
4
2
2
3
6
1
1
2 2 3 2 3 4 1 6 1
```

In [48]:

```
#Q24
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
6
6
2
8
8
6 2 8
```

In [50]:

```
#Q25
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
3
Prime
2
Prime
6
Not prime
13
Prime
12
Not prime
```

In [52]:

```
#Q26
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 3 3000
2 3 2000
10000
```

In [ ]:

```
#Q27
def minimum_index(seq):
    if len(seq)==0:
        raiseValueError("Cannot get the minimum value index from an empty seq")
    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:
        raiseValueError("Cannot get the minimum value index from an empty seq")
    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()
    assertlen(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")
```

In [57]:

```
#Q28
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
Harshitha harshitha@gmail.com
MU muharshitha@gmail.com
Harshitha
MU
```

In [58]:

```
#Q29
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
5 6
4
6 7
4
```