**Day 0 Hello, World**

**Objective**   
In this challenge, we review some basic concepts that will get you started with this series.

**Task**   
To complete this challenge, you must save a line of input from stdin to a variable, print Hello, World. on a single line, and finally print the value of your variable on a second line.

You’ve got this!

**Note:** The instructions are Java-based, but we support submissions in many popular languages. You can switch languages using the drop-down menu above your editor, and the ***inputString*** variable may be written differently depending on the best-practice conventions of your submission language.

**Input Format**

A single line of text denoting ***inputString*** (the variable whose contents must be printed).

**Output Format**

Print Hello, World. on the first line, and the contents of ***inputString*** on the second line.

**Sample Input**

Welcome to 30 Days of Code!

**Sample Output**

Hello, World.

Welcome to 30 Days of Code!

**Solution:**

input\_string.

input\_string = input()

# Print a string literal saying "Hello, World." to stdout.

print('Hello, World.')

print(input\_string)

**Day 1 : Data Types**

**Objective**   
Today, we’re discussing data types.

**Task**   
Complete the code in the editor below. The variables **i**, **d**and **s** are already declared and initialized for you. You must:

1. Declare **3** variables: one of type *int*, one of type *double*, and one of type *String*.
2. Read **3** lines of input from stdin (according to the sequence given in the *Input Format* section below) and initialize your **3**variables.
3. Use the **+** operator to perform the following operations:   
   1. Print the sum of **i** plus your int variable on a new line.
   2. Print the sum of **d** plus your double variable to a scale of one decimal place on a new line.
   3. Concatenate **s** with the string you read as input and print the result on a new line.

**Note:** If you are using a language that doesn’t support using + for string concatenation (e.g.: C), you can just print one variable immediately following the other on the same line. The string provided in your editor *must* be printed first, immediately followed by the string you read as input.

**Input Format**

The first line contains an integer that you must sum with **i**.   
The second line contains a double that you must sum with **d**.   
The third line contains a string that you must concatenate with **s**.

**Output Format**

Print the sum of both integers on the first line, the sum of both doubles (scaled to **1** decimal place) on the second line, and then the two concatenated strings on the third line.

**Sample Input**

12

4.0

is the best place to learn and practice coding!

**Sample Output**

16

8.0

This is the best place to learn and practice coding!

**Solution:**

i = 4

d = 4.0

s = 'HackerRank '

i2 = int(input())

d2 = float(input())

s2 = input()

print(i + i2)

print(d + d2)

print(s + s2)

**Day 2 : Operators**

**Objective**   
In this challenge, you’ll work with arithmetic operators.

**Task**   
Given the *meal price* (base cost of a meal), *tip percent* (the percentage of the *meal price* being added as tip), and *tax percent*(the percentage of the *meal price* being added as tax) for a meal, find and print the meal’s *total cost*.

**Note:** Be sure to use precise values for your calculations, or you may end up with an incorrectly rounded result!

**Input Format**

There are **3** lines of numeric input:   
The first line has a double, **mealCost** (the cost of the meal before tax and tip).   
The second line has an integer, **tipPercent** (the percentage of **mealCost** being added as tip).   
The third line has an integer, **taxPercent** (the percentage of **mealCost** being added as tax).

**Output Format**

Print the total meal cost, where **totalCost** is the rounded integer result of the entire bill (**mealCost** with added tax and tip).

**Sample Input**

12.00

20

8

**Sample Output**

15

**Solution:**

mealCost = float(input())

PercentTip = int(input())

PercentTax = int(input())

tip\_cost = (PercentTip/100.)\* mealCost

tax\_cost = (PercentTax/100.)\* mealCost

totalCost = int(round(mealCost + tip\_cost + tax\_cost))

print(totalCost)

**Day 3: Intro to Conditional Statements**

**Objective**   
In this challenge, we’re getting started with conditional statements.

**Task**   
Given an integer, **n**, perform the following conditional actions:

* If **n** is odd, print Weird
* If **n** is even and in the inclusive range of **2** to **5**, print Not Weird
* If **n** is even and in the inclusive range of **6** to **20**, print Weird
* If **n** is even and greater than **20**, print Not Weird

Complete the stub code provided in your editor to print whether or not **n** is weird.

**Input Format**

A single line containing a positive integer, **n**.

**Constraints**

* **1≤ n ≤ 100**

**Output Format**

Print Weird if the number is weird; otherwise, print Not Weird.

**Sample Input 0**

3

**Sample Output 0**

Weird

**Sample Input 1**

24

**Sample Output 1**

Not Weird

**Solution:**

import math

import os

import random

import re

import sys

N=int(input())

if(N % 2 == 1)

    print(“Weird”)

else if (N % 2 == 0)

if (N >= 2 && N <= 5)

    print(“Not Weird”)

else if (N >= 6 && N <= 20)

    print(“Weird”)

else if (N > 20)

    print(“Not Weird”)

**Day 4 : Class vs Instance**

**Objective**   
In this challenge, we’re going to learn about the difference between a *class* and an *instance*; because this is an *Object Oriented*concept, it’s only enabled in certain languages.

**Task**   
Write a *Person* class with an instance variable, **age**, and a constructor that takes an integer, **initialAge**, as a parameter. The constructor must assign **initialAge** to **age** after confirming the argument passed as **initialAge** is not negative; if a negative argument is passed as **initialAge**, the constructor should set **age** to **0** and print Age is not valid, setting age to 0.. In addition, you must write the following instance methods:

1. *yearPasses()* should increase the **age**  instance variable by .
2. *amIOld()* should perform the following conditional actions:
   * If , **age** < **13**print You are young..
   * If **age ≥  13** and **age < 18**, print You are a teenager..
   * Otherwise, print You are old..

To help you learn by example and complete this challenge, much of the code is provided for you, but you’ll be writing everything in the future. The code that creates each instance of your *Person* class is in the *main* method. Don’t worry if you don’t understand it all quite yet!

**Note:** Do not remove or alter the stub code in the editor.

**Input Format**

Input is handled for you by the stub code in the editor.

The first line contains an integer, **t** (the number of test cases), and the **t** subsequent lines each contain an integer denoting the **age** of a Person instance.

**Constraints**

* **1 ≤ T ≤ 4**
* **-5 ≤ age ≤ 30**

**Output Format**

Complete the method definitions provided in the editor so they meet the specifications outlined above; the code to test your work is already in the editor. If your methods are implemented correctly, each test case will print **2** or **3** lines (depending on whether or not a valid **initialAge** was passed to the constructor).

**Sample Input**

4

-1

10

16

18

**Sample Output**

Age is not valid, setting age to 0.

You are young.

You are young.

You are young.

You are a teenager.

You are a teenager.

You are old.

You are old.

You are old.

**Solution:**

class Person:

    def \_\_init\_\_(self, initialAge):

        self.age = initialAge if initialAge >= 0 else 0

        if initialAge < 0:

            print("Age is not valid, setting age to 0.")

    def amIOld(self):

        if self.age < 13:

            print("You are young.")

        elif self.age < 18:

            print("You are a teenager.")

        else:

            print("You are old.")

    def yearPasses(self):

        self.age += 1

        return self.age

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("")

**Day 5 : Loops**

**Objective**   
In this challenge, we’re going to use loops to help us do some simple math.

**Task**   
Given an integer, **n**, print its first **10** multiples. Each multiple **n x i** (where **1 ≤ i ≤ 10**) should be printed on a new line in the form: n x i = result.

**Input Format**

A single integer, **n**.

**Constraints**

* **2 ≤ n ≤ 20**

**Output Format**

Print **10** lines of output; each line **i** (where **1 ≤ i ≤ 10**) contains the **result** of **n x i** in the form:   
n x i = result.

**Sample Input**

2

**Sample Output**

2 x 1 = 2

2 x 2 = 4

2 x 3 = 6

2 x 4 = 8

2 x 5 = 10

2 x 6 = 12

2 x 7 = 14

2 x 8 = 16

2 x 9 = 18

2 x 10 = 20

**Solution:**

import math

import os

import random

import re

import sys

n = int(input().strip())

for i in range(1,11):

    print(n,'x',i,'=',n\*i)

**Day 6 : Lets Review**

**Objective**   
Today we’re expanding our knowledge of Strings and combining it with what we’ve already learned about loops.

**Task**   
Given a string, **S**, of length **N** that is indexed from **0** to **N – 1**, print its *even-indexed* and *odd-indexed* characters as **2** space-separated strings on a single line (see the *Sample* below for more detail).

**Note:** **0** is considered to be an *even* index.

**Input Format**

The first line contains an integer, **T** (the number of test cases).   
Each line **i** of the **T** subsequent lines contain a String, **S**.

**Constraints**

* **1 ≤ T ≤ 10**
* **2 ≤ length of S ≤ 10000**

**Output Format**

For each String **Sj** (where **0 ≤ j ≤ T – 1**), print **Sj** ‘s *even-indexed* characters, followed by a space, followed by **Sj** ‘s *odd-indexed* characters.

**Sample Input**

2

Hacker

Rank

**Sample Output**

Hce akr

Rn ak

**Solution:**

import java.io.\*;

import java.util.\*;

import java.text.\*;

import java.math.\*;

import java.util.regex.\*;

public class Solution {

    public static void main(String[] args)

    {

        Scanner in = new Scanner(System.in);

        int N = in.nextInt();

        in.nextLine();

        for (int i = 0; i < N; i++)

        {

            String string = in.nextLine();

            char[] charArray = string.toCharArray();

            for (int j = 0; j < charArray.length; j++)

            {

                if (j % 2 == 0)

                {

                    System.out.print(charArray[j]);

                }

            }

            System.out.print(" ");

            for (int j = 0; j < charArray.length; j++)

            {

                if (j % 2 != 0)

                {

                    System.out.print(charArray[j]);

                }

            }

            System.out.println();

        }

        in.close();

    }

}

**Day 7 : Arrays**

**Objective**   
Today, we’re learning about the *Array* data structure.

**Task**   
Given an array, **A**, of **N** integers, print **A**‘s elements in *reverse* order as a single line of space-separated numbers.

**Input Format**

The first line contains an integer,  **N**(the size of our array).   
The second line contains **N** space-separated integers describing array **A**‘s elements.

**Constraints**

* **1 ≤ N ≤ 1000**
* **1 ≤ Ai ≤ 10000**, where **Ai** is the **ith** integer in the array.

**Output Format**

Print the elements of array **A** in reverse order as a single line of space-separated numbers.

**Sample Input**

4

1 4 3 2

**Sample Output**

2 3 4 1

**Solution:**

N = int(input())

def factorial(x):

    if x == 0:

        return 1

    return x \* factorial(x-1)

print(factorial(N))

**Day 8 : Dictionaries and Maps**

**Objective**   
Today, we’re learning about Key-Value pair mappings using a *Map* or *Dictionary* data structure.

**Task**   
Given **n** names and phone numbers, assemble a phone book that maps friends’ names to their respective phone numbers. You will then be given an unknown number of names to query your phone book for. For each **name** queried, print the associated entry from your phone book on a new line in the form name=phoneNumber; if an entry for **name** is not found, print Not foundinstead.

**Note:** Your phone book should be a Dictionary/Map/HashMap data structure.

**Input Format**

The first line contains an integer, **n**, denoting the number of entries in the phone book.   
Each of the **n** subsequent lines describes an entry in the form **2**of  space-separated values on a single line. The first value is a friend’s name, and the second value is an **8**-digit phone number.

After the **n** lines of phone book entries, there are *an unknown number of lines of queries*. Each line (query) contains a **name** to look up, and you must continue reading lines until there is no more input.

**Note:** Names consist of lowercase English alphabetic letters and are *first names* only.

**Constraints**

* **1 ≤ n ≤ 105**
* **1 ≤ queries ≤ 105**

**Output Format**

On a new line for each query, print Not found if the name has no corresponding entry in the phone book; otherwise, print the full **name** and **phoneNumber** in the format name=phoneNumber.

**Sample Input**

3

sam 99912222

tom 11122222

harry 12299933

sam

edward

harry

**Sample Output**

sam=99912222

Not found

harry=12299933

**Solution:**

n = int(input().strip())

phonebook = {}

for i in range(n):

    line = input()

    k, v = line.split()

    phonebook[k] = v

while True:

    try:

        number = input()

    except EOFError:

        break

    if number in phonebook:

        print("{}={}".format(number, phonebook[number]))

    else:

        print("Not found")

**Day 9 : Recursion**

**Objective**   
Today, we’re learning and practicing an algorithmic concept called *Recursion*.

**Recursive Method for Calculating Factorial**

**Task**   
Write a *factorial* function that takes a positive integer, **N** as a parameter and prints the result of **N!** (**N** factorial).

**Note:** If you fail to use recursion or fail to name your recursive function *factorial* or *Factorial*, you will get a score of **0**.

**Input Format**

A single integer, **N** (the argument to pass to *factorial*).

**Constraints**

* **2 ≤ N ≤ 12**
* Your submission must contain a recursive function named *factorial*.

**Output Format**

Print a single integer denoting **N!**.

**Sample Input**

3

**Sample Output**

6

**Solution:**

N = int(input())

def factorial(x):

    if x == 0:

        return 1

    return x \* factorial(x-1)

print(factorial(N))

sc.close();

}

}

**Day 10 : Binary Number**

**Objective**   
Today, we’re working with binary numbers.

**Task**   
Given a base-**10** integer, **n**, convert it to binary (base-**2**). Then find and print the base-**10** integer denoting the maximum number of consecutive **1**‘s in **n‘s** binary representation.

**Input Format**

A single integer, **n**.

**Constraints**

* **1 ≤ n ≤ 106**

**Output Format**

Print a single base-**10** integer denoting the maximum number of consecutive **1**‘s in the binary representation of **n**.

**Sample Input 1**

5

**Sample Output 1**

1

**Sample Input 2**

13

**Sample Output 2**

2

**Solution:**

import sys

n = int(input().strip())

binary = list(bin(n)[2:])

count = 0

max\_count = 0

for i in binary:

    if (i == '1'):

        count += 1

    else:

        if count > max\_count:

            max\_count = count

        count = 0

if count > max\_count:

    max\_count = count

print (max\_count)**Day 12 : 2D Arrays**

**DAY 11**

**Objective**   
  
Today, we’re building on our knowledge of *Arrays* by adding another dimension.

**Context**   
Given a **6 x 6** *2D Array*, **A**:

1 1 1 0 0 0

0 1 0 0 0 0

1 1 1 0 0 0

0 0 0 0 0 0

0 0 0 0 0 0

0 0 0 0 0 0

We define an hourglass in **A** to be a subset of values with indices falling in this pattern in **A**‘s graphical representation:

a b c

d

e f g

There are **16** hourglasses in **A**, and an *hourglass sum* is the sum of an hourglass’ values.

**Task**   
Calculate the hourglass sum for every hourglass in **A**, then print the *maximum* hourglass sum.

**Input Format**

There are **6** lines of input, where each line contains **6** space-separated integers describing *2D Array* **A**; every value in **A** will be in the inclusive range of **-9** to **9**.

**Constraints**

* **-9 ≤ A[i][j] ≤ 9**
* **0≤ i,j ≤ 5**

**Output Format**

Print the largest (maximum) hourglass sum found in **A**.

**Sample Input**

1 1 1 0 0 0

0 1 0 0 0 0

1 1 1 0 0 0

0 0 2 4 4 0

0 0 0 2 0 0

0 0 1 2 4 0

**Sample Output**

19

**Solution:**

import sys

a = []

for arr\_i in range(6):

   arr\_t = [int(arr\_temp) for arr\_temp in input().strip().split(' ')]

   a.append(arr\_t)

slist = []

def calsum(i,j):

    return(a[i][j] + a[i][j+1] + a[i][j+2] + a[i+1][j+1] + a[i+2][j] + a[i+2][j+1] + a[i+2][j+2])

for j in range(0,4):

    for i in range(0,4):

        sum = calsum(i,j)

        slist.append(sum)

print(max(slist))

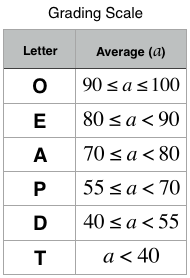
**Day 12 : Inheritance**

**Objective**   
Today, we’re delving into Inheritance.

**Task**   
You are given two classes, *Person* and *Student*, where *Person* is the base class and *Student* is the derived class. Completed code for *Person* and a declaration for *Student* are provided for you in the editor. Observe that *Student* inherits all the properties of *Person*.

Complete the *Student* class by writing the following:

* A *Student* class constructor, which has **4** parameters:
  1. A string, **firstName**.
  2. A string, **lastName**.
  3. An integer, **id**.
  4. An integer array (or vector) of test scores, **scores**.
* A *char calculate()* method that calculates a Student object’s average and returns the grade character representative of their calculated average:



**Input Format**

The locked stub code in your editor calls your *Student* class constructor and passes it the necessary arguments. It also calls the *calculate* method (which takes no arguments).

*You are not responsible for reading the following input from stdin:*   
The first line contains , **firstName**, **lastName**and **id**, respectively. The second line contains the number of test scores. The third line of space-separated integers describes **scores**.

**Constraints**

* **1 ≤ |firstName|, |lastName| ≤ 10**
* **|id| = 7**
* **0 ≤ score, average ≤ 100**

**Output Format**

*This is handled by the locked stub code in your editor.* Your output will be correct if your *Student* class constructor and *calculate()* method are properly implemented.

**Sample Input**

Heraldo Memelli 8135627

2

100 80

**Sample Output**

Name: Memelli, Heraldo

ID: 8135627

Grade: O

**Solution:**

class Person:

    def \_\_init\_\_(self, firstName, lastName, idNumber):

        self.firstName = firstName

        self.lastName = lastName

        self.idNumber = idNumber

    def printPerson(self):

        print("Name:", self.lastName + ",", self.firstName)

        print("ID:", self.idNumber)

class Student(Person):

    #   Class Constructor

    #

    #   Parameters:

    #   firstName - A string denoting the Person's first name.

    #   lastName - A string denoting the Person's last name.

    #   id - An integer denoting the Person's ID number.

    #   scores - An array of integers denoting the Person's test scores.

    #

    # Write your constructor here

    #   Function Name: calculate

    #   Return: A character denoting the grade.

    #

    # Write your function here

line = input().split()

firstName = line[0]

lastName = line[1]

idNum = line[2]

numScores = int(input()) # not needed for Python

scores = list( map(int, input().split()) )

s = Student(firstName, lastName, idNum, scores)

s.printPerson()

print("Grade:", s.calculate())

**Day 13 : Abstract Classes**

**Objective**   
Today, we’re taking what we learned yesterday about [*Inheritance*](https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html) and extending it to [*Abstract Classes*](https://docs.oracle.com/javase/tutorial/java/IandI/abstract.html). Because this is a very specific Object-Oriented concept, submissions are limited to the few languages that use this construct.

**Task**   
Given a *Book* class and a *Solution* class, write a *MyBook* class that does the following:

* Inherits from *Book*
* Has a parameterized constructor taking these **3** parameters:
  1. string **title**
  2. string **author**
  3. int **price**
* Implements the *Book* class’ abstract *display()* method so it prints these **3** lines:
  1. , a space, and then the current instance’s **title**.
  2. , a space, and then the current instance’s **author**.
  3. , a space, and then the current instance’s **price**.

**Note:** Because these classes are being written in the same file, you must not use an access modifier (e.g.: public) when declaring *MyBook* or your code will not execute.

**Input Format**

You are not responsible for reading any input from stdin. The *Solution* class creates a *Book* object and calls the *MyBook* class constructor (passing it the necessary arguments). It then calls the *display* method on the *Book* object.

**Output Format**

The **void display()** method should print and label the respective **title**, **author**, and **price** of the *MyBook* object’s instance (with each value on its own line) like so:

Title: $title

Author: $author

Price: $price

**Note:** The **$** is prepended to variable names to indicate they are placeholders for variables.

**Sample Input**

The following input from stdin is handled by the locked stub code in your editor:

The Alchemist

Paulo Coelho

248

**Sample Output**

The following output is printed by your *display()* method:

Title: The Alchemist

Author: Paulo Coelho

Price: 248

**Solution:**

import java.util.\*;

abstract class Book {

    String title;

    String author;

    Book(String title, String author) {

        this.title = title;

        this.author = author;

    }

    abstract void display();

}

    class MyBook extends Book

    {

    int  price;

   public MyBook(String title, String author,int price)

    {

       super(title,author);

        this.price= price;

    }

    public void display()

    {

        System.out.println("Title: " +title);

        System.out.println("Author: " +author);

        System.out.println("Price: " +price);

    }

    }

public class Solution {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        String title = scanner.nextLine();

        String author = scanner.nextLine();

        int price = scanner.nextInt();

        scanner.close();

        Book book = new MyBook(title, author, price);

        book.display();

    }

}

**Day 14: Scope**

**Objective**   
Today we’re discussing *scope*.

The *absolute difference* between two integers, **a** and **b**, is written as **|a – b|.** The *maximum absolute difference* between two integers in a set of positive integers,**elements** , is the largest absolute difference between any two integers in **elements** .

The *Difference* class is started for you in the editor. It has a private integer array (**elements** ) for storing **N** non-negative integers, and a public integer (**maximumDifference**) for storing the maximum absolute difference.

**Task**   
Complete the *Difference* class by writing the following:

* A class constructor that takes an array of integers as a parameter and saves it to the **elements**  instance variable.
* A *computeDifference* method that finds the maximum absolute difference between any **2** numbers in **N** and stores it in the **maximumDifference** instance variable.

**Input Format**

You are not responsible for reading any input from stdin. The locked *Solution* class in your editor reads in **2** lines of input; the first line contains **N**, and the second line describes the **elements**  array.

**Constraints**

* **1 ≤ N ≤ 10**
* **1 ≤ elements[i] ≤ 100**, where **0 ≤ i ≤ N – 1**

**Output Format**

You are not responsible for printing any output; the *Solution* class will print the value of the **maximumDifference** instance variable.

**Sample Input**

3

1 2 5

**Sample Output**

4

**Solution:**

class Difference:

    def \_\_init\_\_(self, a):

        self.\_\_elements = a

    def computeDifference(self):

        self.maximumDifference = max(self.\_\_elements) - min(self.\_\_elements)

# End of Difference class

\_ = input()

a = [int(e) for e in input().split(' ')]

d = Difference(a)

d.computeDifference()

print(d.maximumDifference)

**Day 15 : Linked List**

**Objective**   
Today we’re working with *Linked Lists*.

A *Node* class is provided for you in the editor. A *Node* object has an integer data field, **data**, and a *Node* instance pointer, **next**, pointing to another node (i.e.: the next node in a list).

A *Node insert* function is also declared in your editor. It has two parameters: a pointer, **head**, pointing to the first node of a linked list, and an integer **data** value that must be added to the end of the list as a new *Node* object.

**Task**   
Complete the *insert* function in your editor so that it creates a new *Node* (pass **data** as the *Node* constructor argument) and inserts it at the tail of the linked list referenced by the **head** parameter. Once the new node is added, return the reference to the **head** node.

**Note:** If the **head** argument passed to the *insert* function is *null*, then the initial list is empty.

**Input Format**

The *insert* function has **2** parameters: a pointer to a *Node* named **head**, and an integer value, **data**.   
The constructor for *Node* has **1** parameter: an integer value for the **data** field.

You *do not* need to read anything from stdin.

**Output Format**

Your *insert* function should return a reference to the **head** node of the linked list.

**Sample Input**

The following input is handled for you by the locked code in the editor:   
The first line contains *T*, the number of test cases.   
The  subsequent lines of test cases each contain an integer to be inserted at the list’s tail.

4

2

3

4

1

**Sample Output**

2 3 4 1

**Solution:**

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):

    #Complete this method

        node = Node(data)

        current = head

        if current:

            while current.next:

                current = current.next

            current.next = node

        else:

            head = node

        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);

**Day 16 : Exception – String to Integer**

**Objective**   
Today, we’re getting started with *Exceptions* by learning how to parse an integer from a string and print a custom error message.

**Task**   
Read a string, **S**, and print its integer value; if **S** cannot be converted to an integer, print Bad String.

**Note:** You *must* use the String-to-Integer and exception handling constructs built into your submission language. If you attempt to use loops/conditional statements, you will get a **0** score.

**Input Format**

A single string, **S**.

**Constraints**

* **1 ≤ |S| ≤ 6**, where **|S|** is the length of string **S**.
* **S** is composed of *either* lowercase letters (a – z) *or* decimal digits (0 – 9).

**Output Format**

Print the parsed integer value of **S**, or Bad String if **S** cannot be converted to an integer.

**Sample Input 0**

3

**Sample Output 0**

3

**Sample Input 1**

za

**Sample Output 1**

Bad String

**Solution:**

import sys

s = input().strip()

try:

    n = int(s)

    print(n)

except:

    print('Bad String')

**Day 17 : More Exceptions**

**Objective**   
Yesterday’s challenge taught you to manage exceptional situations by using *try* and *catch* blocks. In today’s challenge, you’re going to practice throwing and propagating an exception.

**Task**   
Write a *Calculator* class with a single method: *int power(int,int)*. The *power* method takes two integers, **n** and **p**, as parameters and returns the integer result of **np**. If either **n** or **p** is negative, then the method must throw an exception with the message: n and p should be non-negative.

**Note:** Do not use an access modifier (e.g.: public) in the declaration for your *Calculator* class.

**Input Format**

Input from stdin is handled for you by the locked stub code in your editor. The first line contains an integer, **T**, the number of test cases. Each of the **T** subsequent lines describes a test case in **2** space-separated integers denoting **n** and **p**, respectively.

**Constraints**

* No Test Case will result in overflow for correctly written code.

**Output Format**

Output to stdout is handled for you by the locked stub code in your editor. There are **T** lines of output, where each line contains the result of **np** as calculated by your *Calculator* class’ *power* method.

**Sample Input**

4

3 5

2 4

-1 -2

-1 3

**Sample Output**

243

16

n and p should be non-negative

n and p should be non-negative

**Solution:**

import java.util.\*;

import java.io.\*;

class Calculator

{

    public int power(int n,int p) throws Exception

    {

        if(n>=0 && p >=0)

        {

            return (int)Math.pow(n,p);

        }

        else

        {

            throw new Exception("n and p should be non-negative");

        }

    }

}

class Solution{

    public static void main(String[] args) {

        Scanner in = new Scanner(System.in);

        int t = in.nextInt();

        while (t-- > 0) {

            int n = in.nextInt();

            int p = in.nextInt();

            Calculator myCalculator = new Calculator();

            try {

                int ans = myCalculator.power(n, p);

                System.out.println(ans);

            }

            catch (Exception e) {

                System.out.println(e.getMessage());

            }

        }

        in.close();

    }

}

}

}

**Day 18 : Queues and Stacks**

Welcome to Day 18! Today we’re learning about Stacks and Queues.

A *palindrome* is a word, phrase, number, or other sequence of characters which reads the same backwards and forwards. Can you determine if a given string, **s**, is a palindrome?

To solve this challenge, we must first take each character in **s**, *enqueue* it in a *queue*, and also *push* that same character onto a *stack*. Once that’s done, we must *dequeue* the first character from the *queue* and *pop* the top character off the *stack*, then compare the two characters to see if they are the same; as long as the characters match, we continue dequeueing, popping, and comparing each character until our containers are empty (a non-match means **s** isn’t a palindrome).

Write the following declarations and implementations:

1. Two instance variables: one for your **stack**, and one for your **queue**.
2. A *void pushCharacter(char ch)* method that pushes a character onto a stack.
3. A *void enqueueCharacter(char ch)* method that enqueues a character in the **queue** instance variable.
4. A *char popCharacter()* method that pops and returns the character at the top of the **stack** instance variable.
5. A *char dequeueCharacter()* method that dequeues and returns the first character in the **queue** instance variable.

**Input Format**

You *do not* need to read anything from stdin. The locked stub code in your editor reads a single line containing string **s**. It then calls the methods specified above to pass each character to your instance variables.

**Constraints**

* **s** is composed of lowercase English letters.

**Output Format**

You are *not* responsible for printing any output to stdout.   
If your code is correctly written and **s** is a palindrome, the locked stub code will print **The word, s, is a palindrome.;**otherwise, it will print **The word, s, is not a palindrome.**

**Sample Input**

racecar

**Sample Output**

The word, racecar, is a palindrome.

**Solution:**

public class Day19QueuesAndStacks {

import java.io.\*;

import java.util.\*;

public class Solution {

    Queue <Character> queue = new LinkedList<>();

    Deque <Character> stack = new ArrayDeque<>();

    public void pushCharacter(char ch){

        stack.push(ch);

    }

    public void enqueueCharacter(char ch){

        queue.add(ch);

    }

    public char popCharacter(){

        return stack.pop();

    }

    public char dequeueCharacter(){

        return queue.remove();

    }

    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);

        String input = scan.nextLine();

        scan.close();

        // Convert input String to an array of characters:

        char[] s = input.toCharArray();

        // Create a Solution object:

        Solution p = new Solution();

        // Enqueue/Push all chars to their respective data structures:

        for (char c : s) {

            p.pushCharacter(c);

            p.enqueueCharacter(c);

        }

        // Pop/Dequeue the chars at the head of both data structures and compare them:

        boolean isPalindrome = true;

        for (int i = 0; i < s.length/2; i++) {

            if (p.popCharacter() != p.dequeueCharacter()) {

                isPalindrome = false;

                break;

            }

        }

        //Finally, print whether string s is palindrome or not.

        System.out.println( "The word, " + input + ", is "

                           + ( (!isPalindrome) ? "not a palindrome." : "a palindrome." ) );

    }

}

**Day 19 : Interfaces**

**Objective**   
Today, we’re learning about Interfaces.

**Task**   
The AdvancedArithmetic interface and the method declaration for the abstract divisorSum(n) method are provided for you in the editor below.

Complete the implementation of Calculator class, which implements the AdvancedArithmetic interface. The implementation for the divisorSum(n) method must return the sum of all divisors of **n**.

**Input Format**

A single line containing an integer, **n**.

**Constraints**

* **1 ≤ n ≤ 1000**

**Output Format**

You are not responsible for printing anything to stdout. The locked template code in the editor below will call your code and print the necessary output.

**Sample Input**

6

**Sample Output**

I implemented: AdvancedArithmetic

12

**Solution:**

import java.io.\*;

import java.util.\*;

interface AdvancedArithmetic{

   int divisorSum(int n);

}

class Calculator implements AdvancedArithmetic

{

@Override

public int divisorSum(int n)

{

int sum = n;

for(int x=1;x<=n/2;x++)

{

if(n%x==0)

{

sum = sum + x;

}

}

return sum;

}

}

class Solution {

    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);

        int n = scan.nextInt();

        scan.close();

        AdvancedArithmetic myCalculator = new Calculator();

        int sum = myCalculator.divisorSum(n);

        System.out.println("I implemented: " + myCalculator.getClass().getInterfaces()[0].getName() );

        System.out.println(sum);

    }

}

**Day 20 : Sorting**

**Objective**   
Today, we’re discussing a simple sorting algorithm called *Bubble Sort*.

**Task**   
Given an array, **a**, of size **n** distinct elements, sort the array in *ascending* order using the *Bubble Sort* algorithm above. Once sorted, print the following **3** lines:

1. Array is sorted in numSwaps swaps.   
   where **numSwaps** is the number of swaps that took place.
2. First Element: firstElement   
   where **firstElement** is the *first* element in the sorted array.
3. Last Element: lastElement   
   where **lastElement** is the *last* element in the sorted array.

**Hint:** To complete this challenge, you will need to add a variable that keeps a running tally of *all* swaps that occur during execution.

**Input Format**

The first line contains an integer, **n**, denoting the number of elements in array **a**.   
The second line contains **n** space-separated integers describing the respective values of **a0,a1,…,an-1**.

**Constraints**

* **2 ≤ n ≤ 600**
* **1 ≤ ai≤ 2 x 106**, where 0**≤ i ≤ n**.

**Output Format**

Print the following three lines of output:

1. Array is sorted in numSwaps swaps.   
   where **numSwaps** is the number of swaps that took place.
2. First Element: firstElement   
   where **firstElement**is the *first* element in the sorted array.
3. Last Element: lastElement   
   where **lastElement**is the *last* element in the sorted array.

**Sample Input 0**

3

1 2 3

**Sample Output 0**

Array is sorted in 0 swaps.

First Element: 1

Last Element: 3

**Solution:**

import sys

def swap(a, p, q):

    temp = a[p]

    a[p] = a[q]

    a[q] = temp

def bubblesort(a):

    n = len(a)

    TotalSwapCount = 0

    for i in range(0, n):

        swap\_Count = 0

        for j in range(0, n - 1):

            if a[j] > a[j + 1]:

                swap(a, j, j + 1)

                swap\_Count += 1

        if swap\_Count == 0:

            break

        else:

            TotalSwapCount += swap\_Count

    return (a[0], a[n - 1], TotalSwapCount)

n = int(input().strip())

a = [int(a\_temp) for a\_temp in input().strip().split(' ')]

head, tail, swaps = bubblesort(a)

print("Array is sorted in", swaps, "swaps.")

print("First Element:", head)

print("Last Element:", tail)

**Day 21 : Generics**

**Objective**   
Today we’re discussing Generics; be aware that *not all languages support this construct*, so fewer languages are enabled for this challenge.

**Task**   
Write a single generic function named *printArray*; this function must take an array of generic elements as a parameter (the exception to this is C++, which takes a *vector*). The locked *Solution* class in your editor tests your function.

**Note:** You must use generics to solve this challenge. *Do not* write overloaded functions.

**Input Format**

The locked *Solution* class in your editor will pass different types of arrays to your *printArray* function.

**Constraints**

* You must have exactly **1** function named *printArray*.

**Output Format**

Your *printArray* function should print each element of its generic array parameter on a new line.

**Solution:**

import java.util.\*;

class Printer <T> {

    /\*\*

    \*    Method Name: printArray

    \*    Print each element of the generic array on a new line. Do not return anything.

    \*    @param A generic array

    \*\*/

    // Write your code here

    static <E>  void printArray( E[] inputArray)

{

for( E e : inputArray)

    {

    System.out.println(""+e);

}

}

}

public class Generics {

    public static void main(String args[]){

        Scanner scanner = new Scanner(System.in);

        int n = scanner.nextInt();

        Integer[] intArray = new Integer[n];

        for (int i = 0; i < n; i++) {

            intArray[i] = scanner.nextInt();

        }

        n = scanner.nextInt();

        String[] stringArray = new String[n];

        for (int i = 0; i < n; i++) {

            stringArray[i] = scanner.next();

        }

        Printer<Integer> intPrinter = new Printer<Integer>();

        Printer<String> stringPrinter = new Printer<String>();

        intPrinter.printArray( intArray  );

        stringPrinter.printArray( stringArray );

        if(Printer.class.getDeclaredMethods().length > 1){

            System.out.println("The Printer class should only have 1 method named printArray.");

        }

    }

}

**Day 22 : Binary Search Trees**

**Objective**   
Today, we’re working with Binary Search Trees (BSTs).

**Task**   
The height of a binary search tree is the number of edges between the tree’s root and its furthest leaf. You are given a pointer, **root**, pointing to the root of a binary search tree. Complete the *getHeight* function provided in your editor so that it returns the height of the binary search tree.

**Input Format**

The locked stub code in your editor reads the following inputs and assembles them into a binary search tree:   
The first line contains an integer, **n**, denoting the number of nodes in the tree.   
Each of the **n** subsequent lines contains an integer, **data**, denoting the value of an element that must be added to the BST.

**Output Format**

The locked stub code in your editor will print the integer returned by your *getHeight* function denoting the height of the BST.

**Sample Input**

7

3

5

2

1

4

6

7

**Sample Output**

3

**Solution:**

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:

            left = 0

            right = 0

            if root.left:

                left = 1 + self.getHeight(root.left)

            if root.right:

                right = 1 + self.getHeight(root.right)

            return max(left, right)

        else:

            return 0

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)

**Day 23 : BST Level – Order Traversal**

**Objective**   
Today, we’re going further with Binary Search Trees.

**Task**   
A level-order traversal, also known as a breadth-first search, visits each level of a tree’s nodes from left to right, top to bottom. You are given a pointer, **root**, pointing to the root of a binary search tree. Complete the *levelOrder* function provided in your editor so that it prints the level-order traversal of the binary search tree.

**Hint:** You’ll find a queue helpful in completing this challenge.

**Input Format**

The locked stub code in your editor reads the following inputs and assembles them into a BST:   
The first line contains an integer,**T**  (the number of test cases).   
The **T** subsequent lines each contain an integer, **data**, denoting the value of an element that must be added to the BST.

**Output Format**

Print the **data** value of each node in the tree’s level-order traversal as a single line of **N** space-separated integers.

**Sample Input**

6

3

5

4

7

2

1

**Sample Output**

3 2 5 1 4 7

**Solution:**

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]

        while len(queue) is not 0:

            curr = queue[0]

            queue = queue[1:]

            print(str(curr.data) + " ", end="")

            if curr.left is not None:

                queue.append(curr.left)

            if curr.right is not None:

                queue.append(curr.right)

T=int(input())

myTree=Solution()

root=None

for i in range(T):

    data=int(input())

    root=myTree.insert(root,data)

myTree.levelOrder(root)

**Day 24: More Linked List**

**Task**   
A *Node* class is provided for you in the editor. A *Node* object has an integer data field, **data**, and a Node instance pointer, **next**, pointing to another node (i.e.: the next node in a list).

A *removeDuplicates* function is declared in your editor, which takes a pointer to the **head** node of a linked list as a parameter. Complete *removeDuplicates* so that it deletes any duplicate nodes from the list and returns the head of the updated list.

**Note:** The **head** pointer may be null, indicating that the list is empty. Be sure to reset your **next** pointer when performing deletions to avoid breaking the list.

**Input Format**

You do not need to read any input from stdin. The following input is handled by the locked stub code and passed to the *removeDuplicates* function:   
The first line contains an integer, **N**, the number of nodes to be inserted.   
The **N** subsequent lines each contain an integer describing the **data** value of a node being inserted at the list’s tail.

**Constraints**

* The data elements of the linked list argument *will always be* in non-decreasing order.

**Output Format**

Your *removeDuplicates* function should return the head of the updated linked list. The locked stub code in your editor will print the returned list to stdout.

**Sample Input**

6

1

2

2

3

3

4

**Sample Output**

1 2 3 4

**Solution:**

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 remove\_duplicates(self, head):

        # Write your code here

        previous = head

        s = set()

        s.add(previous.data)

        current = previous.next

        while current:

            if current.data in s:

                previous.next = current.next

            else:

                s.add(current.data)

                previous = current

            current = current.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);

**Day 25 : Running Time and Complexity**

**Objective**   
Today we’re learning about running time!

**Task**   
A *prime* is a natural number greater than **1** that has no positive divisors other than **1** and itself. Given a number, **n**, determine and print whether it’s **Prime** or **Not prime**.

**Note:** If possible, try to come up with a **O(√n)** primality algorithm, or see what sort of optimizations you come up with for an **O(n)** algorithm. Be sure to check out the *Editorial* after submitting your code!

**Input Format**

The first line contains an integer, **T**, the number of test cases.   
Each of the **T** subsequent lines contains an integer, **n**, to be tested for primality.

**Constraints**

* **1 ≤ T ≤ 30**
* **1 ≤ n ≤ 2 x 109**

**Output Format**

For each test case, print whether **n** is **Prime** or **Not prime** on a new line.

**Sample Input**

3

12

5

7

**Sample Output**

Not prime

Prime

Prime

**Solution:**

def prime\_no(n):

    if n == 1:

        return False

    else:

        sq\_root = int(n\*\*0.5)

        for i in range(2, sq\_root + 1):

            if ((n % i) == 0) and (i != n):

                return False

        return True

t = int(input())

for \_ in range(t):

    n = int(input())

    if prime\_no(n):

        print("Prime")

    else:

        print("Not prime")

**Day 26 : Nested Logic**

**Objective**   
Today’s challenge puts your understanding of nested conditional statements to the test. You already have the knowledge to complete this challenge

**Task**   
Your local library needs your help! Given the expected and actual return dates for a library book, create a program that calculates the fine (if any). The fee structure is as follows:

1. If the book is returned on or before the expected return date, no fine will be charged (i.e.: **fine = 0**).
2. If the book is returned after the expected return *day* but still within the same calendar month and year as the expected return date, **fine = 15 Hackos x (the number of days late)**.
3. If the book is returned after the expected return *month* but still within the same calendar year as the expected return date, the **fine = 500 Hackos x (the number of months late)**
4. If the book is returned after the calendar *year* in which it was expected, there is a fixed fine of **10000 Hackos**.

**Input Format**

The first line contains **3** space-separated integers denoting the respective **day**, **month**, and **year** on which the book was *actually* returned.   
The second line contains **3** space-separated integers denoting the respective **day**, **month**, and **year** on which the book was *expected* to be returned (due date).

**Constraints**

* **1≤ D ≤ 31**
* **1≤ M ≤ 12**
* **1≤ Y ≤ 3000**
* **It is guaranteed that the dates will be valid Georgian calendar date.**

**Output Format**

Print a single integer denoting the library fine for the book received as input.

**Sample Input**

9 6 2015

6 6 2015

**Sample Output**

45

**Solution:**

cal\_actual = str(input()).split(" ")

date = int(cal\_actual[0])

month = int(cal\_actual[1])

year = int(cal\_actual[2])

cal\_expect = str(input()).split(" ")

date\_e = int(cal\_expect[0])

month\_e = int(cal\_expect[1])

year\_e = int(cal\_expect[2])

fine = 0

if year > year\_e:

    fine = 10000

elif year == year\_e:

    if month > month\_e:

        fine = (month - month\_e) \* 500

    elif month == month\_e and date > date\_e:

        fine = (date - date\_e) \* 15

print(fine)

**Day 27 : Testing**

This problem is all about unit testing.

Your company needs a function that meets the following requirements:

* For a given array of **n** integers, the function returns the index of the element with the minimum value in the array. If there is more than one element with the minimum value, the returned index should be the smallest one.
* If an empty array is passed to the function, it should raise an Exception.

Note: The arrays are indexed from **0**.

get\_array() method in class TestDataEmptyArray has to return an empty array.

get\_array() method in class TestDataUniqueValues has to return an array of size at least 2 with all unique elements, while method get\_expected\_result() of this class has to return the expected minimum value index for this array.

get\_array() method in class TestDataExactlyTwoDifferentMinimums has to return an array where there are exactly two different minimum values, while method get\_expected\_result() of this class has to return the expected minimum value index for this array.

Take a look at the code template to see the exact implementation of functions that your colleagues already implemented.

**Solution:**

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):

    data = []

    for i in range(5):

        data.append(i)

    data[::-1]

    @staticmethod

    def get\_array():

        return TestDataUniqueValues.data

    @staticmethod

    def get\_expected\_result():

        data = TestDataUniqueValues.get\_array()

        return data.index(min(data))

        # complete this function

class TestDataExactlyTwoDifferentMinimums(object):

    data = []

    for i in range(5):

        data.append(i)

    data[::-1]

    data.insert(0,0)

    @staticmethod

    def get\_array():

        return TestDataExactlyTwoDifferentMinimums.data

        # complete this function

    @staticmethod

    def get\_expected\_result():

        data = TestDataExactlyTwoDifferentMinimums.get\_array()

        return data.index(min(data))

        # complete this function

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")

**Day 28 : RegEx, Patterns and Intro to Database**

**Objective**   
Today, we’re working with regular expressions.

**Task**   
Consider a database table, *Emails*, which has the attributes *First Name* and *Email ID*. Given **N** rows of data simulating the *Emails* table, print an alphabetically-ordered list of people whose email address ends in **@gmail.com**.

**Input Format**

The first line contains an integer,**N**  , total number of rows in the table.   
Each of the **N**  subsequent lines contains **2** space-separated strings denoting a person’s first name and email ID, respectively.

**Constraints**

* **2 ≤ N ≤ 30**
* Each of the first names consists of lower case letters **[a – z]** only.
* Each of the email IDs consists of lower case letters **[a – z]**, **@** and **.** only.
* The length of the first name is no longer than 20.
* The length of the email ID is no longer than 50.

**Output Format**

Print an alphabetically-ordered list of first names for every user with a gmail account. Each name must be printed on a new line.

**Sample Input**

6

riya riya@gmail.com

julia julia@julia.me

julia sjulia@gmail.com

julia julia@gmail.com

samantha samantha@gmail.com

tanya tanya@gmail.com

**Sample Output**

julia

julia

riya

samantha

tanya

**Solution:**

import re

array = []

num = int(input())

for i in range(num):

    data = str(input()).split(" ")

    name = data[0]

    email = data[1]

    if re.search(".+@gmail\.com$", email):

        array.append(name)

array.sort()

for name in array:

    print(name)

**Day 29: Bitwise AND**

**Objective**   
Welcome to the last day! Today, we’re discussing bitwise operations.

**Task**   
Given set **S = {1,2,3,…,N}**. Find two integers, **A** and **B** (where **A < B**), from set **S** such that the value of **A&B** is the maximum possible *and also less than a given integer,****K***. In this case, **&** represents the *bitwise AND* operator.

**Input Format**

The first line contains an integer, **T**, the number of test cases.   
Each of the **T** subsequent lines defines a test case as **2** space-separated integers, **N** and **K**, respectively.

**Constraints**

* **1 ≤ T ≤ 103**
* **2 ≤ N ≤ 103**
* **2 ≤ K ≤ N**

**Output Format**

For each test case, print the maximum possible value of **A&B** on a new line.

**Sample Input**

3

5 2

8 5

2 2

**Sample Output**

1

4

0

**Solution:**

T = int(input().strip())

for \_ in range(T):

    n , p = map(int , input().split())

    print(p-1 if ((p-1) | p) <= n else p-2)