Given the names and grades for each student in a class of  students, store them in a nested list and print the name(s) of any student(s) having the second lowest grade.

**Note:** If there are multiple students with the second lowest grade, order their names alphabetically and print each name on a new line.

**Example**

The ordered list of scores is , so the second lowest score is . There are two students with that score: . Ordered alphabetically, the names are printed as:

alpha

beta

**Input Format**

The first line contains an integer, , the number of students.  
The  subsequent lines describe each student over  lines.  
- The first line contains a student's name.  
- The second line contains their grade.

**Constraints**

* There will always be one or more students having the second lowest grade.

**Output Format**

Print the name(s) of any student(s) having the second lowest grade in. If there are multiple students, order their names alphabetically and print each one on a new line.

**Sample Input 0**

5

Harry

37.21

Berry

37.21

Tina

37.2

Akriti

41

Harsh

39

**Sample Output 0**

Berry

Harry

**Explanation 0**

There are  students in this class whose names and grades are assembled to build the following list:

python students = [['Harry', 37.21], ['Berry', 37.21], ['Tina', 37.2], ['Akriti', 41], ['Harsh', 39]]

The lowest grade of  belongs to *Tina*. The second lowest grade of  belongs to both *Harry* and *Berry*, so we order their names alphabetically and print each name on a new line.

**if** **\_\_name\_\_** == '\_\_main\_\_':

    students = []

**for** **\_** **in** **range**(**int**(**input**())):

        name = **input**()

        score = **float**(**input**())

        students.append([name,score])

    scores = **sorted**(**set**([score **for** name, score **in** students]))

    second\_low = scores[1]

    names = [name **for** name, score **in** students **if** score == second\_low]

    names.sort()

**for** name **in** names:

**print**(name)

2. **if** **\_\_name\_\_** == '\_\_main\_\_':

    n = **int**(**input**())

    student\_marks = {}

**for** **\_** **in** **range**(n):

        name, \*line = **input**().split()

        scores = **list**(**map**(**float**, line))

        student\_marks[name] = scores

    query\_name = **input**()

    marks = student\_marks[query\_name]

    average = **sum**(marks)/**len**(marks)

**print**(f"{average:.2f}")

3. Consider a list (list = []). You can perform the following commands:

1. insert i e: Insert integer  at position .
2. print: Print the list.
3. remove e: Delete the first occurrence of integer .
4. append e: Insert integer  at the end of the list.
5. sort: Sort the list.
6. pop: Pop the last element from the list.
7. reverse: Reverse the list.

Initialize your list and read in the value of  followed by  lines of commands where each command will be of the  types listed above. Iterate through each command in order and perform the corresponding operation on your list.

**Example**

* : Append  to the list, .
* : Append  to the list, .
* : Insert  at index , .
* : Print the array.  
  Output:

[1, 3, 2]

**Input Format**

The first line contains an integer, , denoting the number of commands.  
Each line  of the  subsequent lines contains one of the commands described above.

**Constraints**

* The elements added to the list must be *integers*.

**Output Format**

For each command of type print, print the list on a new line.

**Sample Input 0**

12

insert 0 5

insert 1 10

insert 0 6

print

remove 6

append 9

append 1

sort

print

pop

reverse

print

**Sample Output 0**

[6, 5, 10]

[1, 5, 9, 10]

[9, 5, 1]

**if** **\_\_name\_\_** == '\_\_main\_\_':

    N = **int**(**input**())

    my\_list = [ ]

**for** **\_** **in** **range**(N):

        command = **input**().split()

**if** command[0] == "insert":

            my\_list.insert(**int**(command[1]),**int**(command[2]))

**elif** command[0] == "print":

**print**(my\_list)

**elif** command[0] == "remove":

            my\_list.remove(**int**(command[1]))

**elif** command[0] == "append":

            my\_list.append(**int**(command[1]))

**elif** command[0] == "sort":

            my\_list.sort()

**elif** command[0] == "pop":

            my\_list.pop()

**elif** command[0] == "reverse":

            my\_list.reverse()

Object orientation

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. Check out the [Tutorial](https://www.hackerrank.com/challenges/30-class-vs-instance/tutorial) tab for learning materials and an instructional video!

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

1. *yearPasses()* should increase the  instance variable by .
2. *amIOld()* should perform the following conditional actions:
   * If , print You are young..
   * If  and , 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.

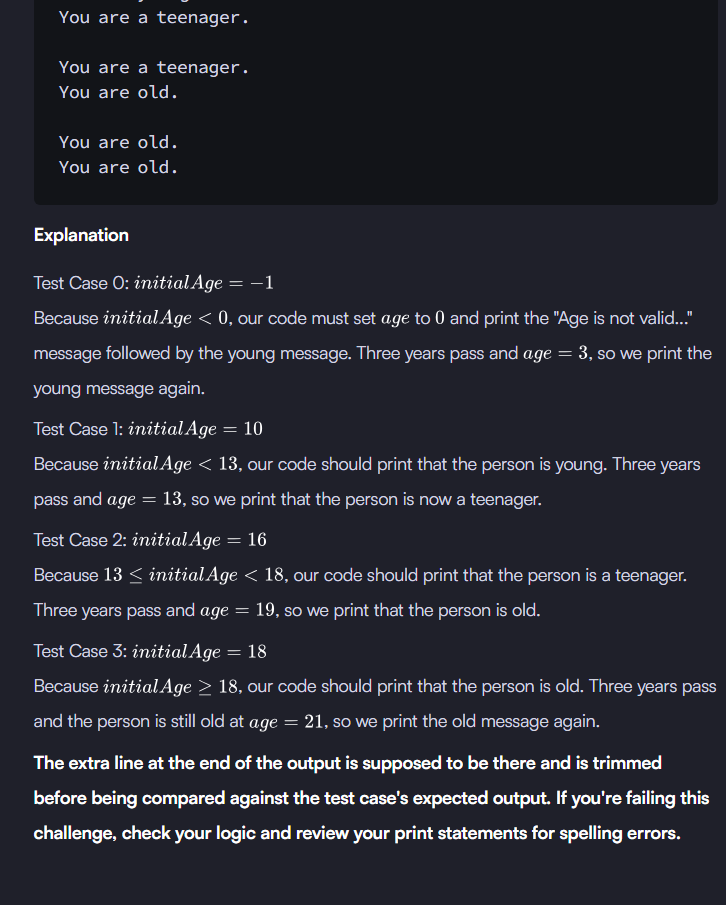
The first line contains an integer,  (the number of test cases), and the  subsequent lines each contain an integer denoting the  of a Person instance.

**Constraints**

**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  or  lines (depending on whether or not a valid  was passed to the constructor).

**Sample Input**



class Person:

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

        if (initialAge < 0):

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

            self.age = 0

        else:

            self.age = initialAge

    def amIOld(self):

        if (self.age < 13):

            print("You are young.")

        elif(13 <= self.age and self.age < 18):

            print("You are a teenager.")

        else:

            print("You are old.")

    def yearPasses(self):

        self.age += 1

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("")