Important Contest Instructions!!

Please *read the following instructions carefully. They contain important information on how to run your programs and submit your solutions to the judges. If you have any questions regarding these instructions, please ask a volunteer before the start of the competition.*

Program Input
Most programs will require input. You have two options:

1) Your program may read the input from a file. The input data will be in the local directory in the file **probXX.txt**, where 'XX' is the problem number.

2) Your program may read the input from the keyboard (standard in). You may type everything on the keyboard, or you may copy the data from **probXX.txt** into the standard in. ***Tip:*** *Type 'Ctrl-Z <return>' to signal the end of keyboard input.*

   Note: An easy way to enter keyboard data is by redirecting the contents of a file to your program. For example, if you are executing prob01, the input file **prob01.txt** can be redirected to the standard in of your program using syntax like this (examples are shown for each of the allowed languages):

```
%> java prob01 < prob01.txt
%> java –jar js.jar prob01.js < prob01.txt
%> python prob01.py3 < prob01.txt
%> prob01.exe < prob01.txt
```

   Your program will behave exactly as if you were typing the input at the keyboard.

Program Output
All programs must send their output to the screen (standard out, the default for any print statement).

Submitting your Programs

Interpreted Programs (Java, JavaScript, Python.) Your program must be named probXX.java / probXX.js / probXX.py2 / probXX.py3, where 'XX' corresponds to the problem number. For Python, use the extension that matches the Python version you are using. Please submit only the source (.java, .js, .py2 or .py3). For java, the main class must be named probXX. Note there is no capitalization. All main and supporting classes should be in the default (or anonymous) package.

Native Programs (C, C++, etc.) Your program should be named probXX.exe, where 'XX' corresponds to the problem number.

You are <u>strongly</u> encouraged to submit solutions for Problems #0 and #1 (see next pages) <u>prior</u> to the start of the competition to ensure that your build environment is compatible with the judges' and that you understand the Input and Output methods required.

Problem 0 Welcome! [1 point]

NOTE – this is the 1st of two problems that can be solved and submitted before the start of the CodeWars competition. Teams are **strongly** encouraged to submit this problem **prior** to the start of the competition – hey, it's basically a free point!

Summary

The sole purpose of this problem is to allow each team to submit a test program to ensure the programs generated by their computer can be judged by our judging system. Your task for this program is a variation on the classic "Hello World!" program by saying hello to our two newest CodeWars sites, Barcelona and Newcastle. All you have to do is print "Welcome, Barcelona and Newcastle!" to the screen.

Output

```
Welcome, Barcelona and Newcastle!
```

Problem 1 Greetings [1 point]

NOTE – this is the 2nd of two problems that can be solved and submitted before the start of the CodeWars competition. Teams are **strongly** encouraged to submit this problem **prior** to the start of the competition – hey, it's basically a free point!

Summary

You'll have no chance to win at HP CodeWars (or life) if you don't know how to do Input and Output properly. You also won't do well at CodeWars if you are rude to your judges.

Write a program to greet your esteemed judges appropriately. Read in the name of a judge and output your greeting in the appropriate format.

If you're confused at this point, go back and re-read your contest instructions.

Input

The input will be your judge's first name, a single word with no spaces:

```
Simon
```

Output

Welcome your judge with a friendly, creative greeting of some sort that includes the judge's name (does not have to match the below example):

```
Greetings, Simon the Great!  I genuflect in your general direction!
```

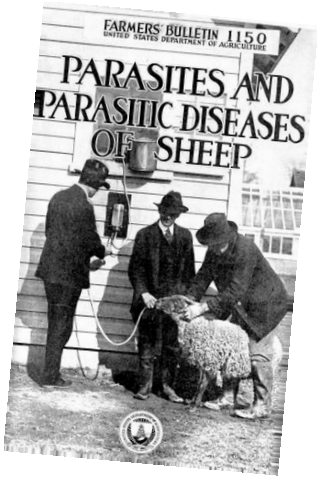
Problem 2 Parasitoid / Host Interaction Models [2 points]

Summary

W. R. Thompson (1924) was interested in what happened to parasitoid/host populations in areas where parasites were released into an area with a large host density. In the model initially he assumed that the parasitoid would only lay one egg in each host that was found. Thus:

```
(No. eggs laid) = (Mean Female Egg Compliment)(No. Females Searching)
            Pe = C X P
```

However, this model did not work too well because many parasites are unable to distinguish between parasitized and unparasitized hosts. Thus Thompson's model predicted a higher rate of parasitism than would actually occur. In reality, a host can end up with more than one parasitoid egg and is then "superparasitized".



Thompson's model may not have been very accurate, but it makes a great CodeWars program. Write a program that uses Thompson's model to predict the rate of parasitism for a pair of input values.

Input

The first line of input is the value of C, the Mean Female Egg Compliment, an integer between 1 and 10,000. The second line is the value of P, the Number of Females Searching, an integer between 1 and 100,000.

```
1300
97450
```

Output

The program must print the value of Pe, the Number of Eggs Laid. The answer must match the judge's expected value precisely.

```
126685000
```

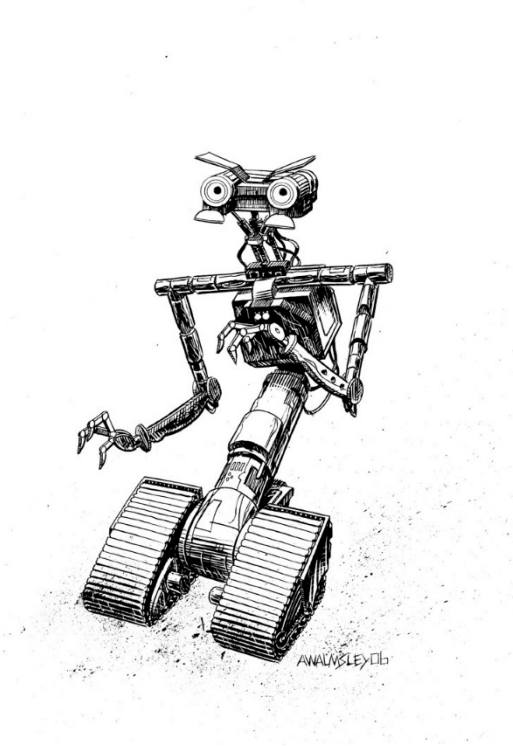
Problem 3 No Disassemble! [3 points]

Summary

There's a short circuit in the management fabric induced by a power surge from a lightning strike. The engineering team needs you to take an unmarked white van and investigate the numbered input nodes in the fabric.

You must inspect each node by disassembling it. If it is working, you'll send a message back to the laboratory at corporate headquarters and reassemble the node. With the fabric not working correctly you'll be using a low-frequency channel that can only send very short messages. In fact, you can only send a single number.

The management at corporate headquarters would like to see a human-friendly message instead of a single number. So before leaving, you must write a program that reads a single number and writes a friendly message on the screen.



Input

The input is a single integer between one and ten, inclusive.

Example 1:
```
5
```

Example 2:
```
10
```

Output

The program must print a sentence in the format shown in the examples below, using the input number written as a word.

Example 1:
```
Number five is alive!
```

Example 2:
```
Number ten is alive!
```

Problem 4 NetOS App [3 points]

Summary

The IQ Computer Corporation* has recently acquired Hand Computing* and plans to release a new line of products based on Hand's innovative technology. IQ has rebranded HandOS with the name netOS and is preparing for product launch of its newest devices.

There's just one catch: netOS doesn't have nearly as many apps as those other guys.

Never ones to worry, the IQ marketing team has decided to offer free netOS devices to the first 1,000 software developers who submit apps for the new platform. Being a savvy software developer, you have decided to write a simple app that calculates the area of a room. The user just enters two integers for the length and width of a room and the app will display the area.

If your conscience starts to bother you because you're writing such a simplistic app, don't fret. As it turns out, IQ Computer Corporation has no intention of actually publishing any of the apps submitted, so writing a useful app would just be a waste of everyone's time. This is how multinational corporations roll.

Besides, it's not like 1,000 more apps would make a difference anyway.

Input

Each line of the input is two integer values. The last line of input is two zeros.

```
15 9
11 8
0 0
```

Output

For each line of input, the program must multiply the two input numbers and print the result.

```
135
88
```

* NOTE -- The corporations appearing in this work are fictitious. Any resemblance to real corporations, profitable or bankrupt, is purely coincidental.

Problem 5 Distributive Law [3 points]

Summary

In mathematics, the Distributive Law says

$$A \times (B + C) = A \times B + A \times C$$

Write a program to demonstrate the Distributive Law.

Input

There will be three lines of input. The first line has the value for A, the second for B, and the third for C. All values will be positive integers.

```
11
5
4
```

Output

The program must print three lines demonstrating the Distributive Law with the input values. The first line must substitute the values into the equation. The second line must show the result of the first level of evaluation. The third line must show the result of the final evaluation. Follow the pattern shown below.

```
11 x (5 + 4) = 11 x 5 + 11 x 4
11 x 9 = 55 + 44
99 = 99
```

Problem 6 Dirt Yard [3 points]

Summary

In the United States, landscaping companies sell various types of soil and gravel by the "yard". This is an abbreviation for a cubic yard, i.e., a cube that is one yard long, one yard wide, and one yard deep.

Now this bit of trivia is important to a suburban gardener who wishes to grow a grape vine in their (ahem) back yard. Grape vines require soil with good drainage, so if the gardener's soil does not drain well they might decide to dig a hole and fill it with a mix of sand and topsoil. Plant spacing is typically published in feet, so the gardener is likely to plan the size of the hole in feet, but needs to order the dirt in “yards.”

Write a program to compute the number of (cubic) yards of soil required to fill a hole that is measured in feet. There are three feet in a yard, which equates to twenty-seven cubic feet per cubic yard.

> *To forget how to dig the earth and to tend the soil is to forget ourselves.*
>
> *– Mahatma Gandhi*

Input

The input consists of three lines, each with a single integer. These are the length, width, and depth of the hole, measured in feet.

```
4
6
3
```

Output

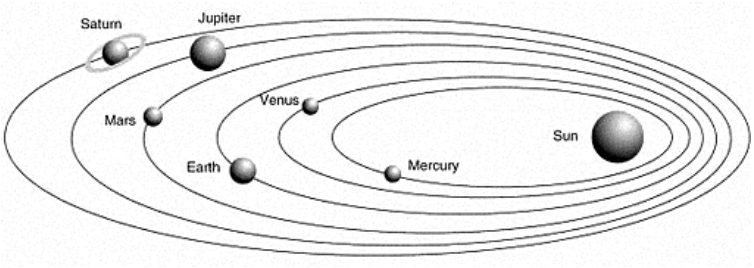
The program must print the minimum number of (cubic) yards of soil required to fill the hole. Landscaping companies sell soil in whole yards, not in fractional parts, so the answer must be the smallest integer number of cubic yards that will fill the hole.

```
3
```

Problem 7 Kepler's Third Law [4 points]

Summary

Johannes Kepler derived three mathematical laws for planetary motion in the early 1600's. This work was based on measurements made before the advent of the telescope using instruments called quadrants and sextants. Oh, and before the advent of the computer as well.



Kepler's third law of Planetary Motion captures the relationship between the distance of planets from the Sun and their orbital periods. It states that the square of the orbital period of a planet is proportional to the cube of the semi-major axis of its orbit. If we assume (for convenience) that the period P is measured in Earth-years and the orbital radius R is measured in Astronomical Units (AU), then the equation is simply this:

Mathematically:

$$P^2 = R^3$$

where P is the period measured in Earth-years, and R is the semi-major axis (orbital radius) in Astronomical Units (AU).

Write a program to calculate the orbital radius of a planet given its orbital period. To solve this problem you may need to know that you can express square roots and cube roots as *fractional exponents*. For example:

$$x^{1/2} = \sqrt[2]{x}$$

$$x^{1/3} = \sqrt[3]{x}$$

Input

Each line of input is the orbital period (P), in years, of a body orbiting the sun. The input ends with a value of zero.

```
1.8808
4.60
0.615198
30.07
0
```

Output

For each period, the program must print the semi-major axis (R) in AU. The answer must be accurate to within +/- 0.01 AU.

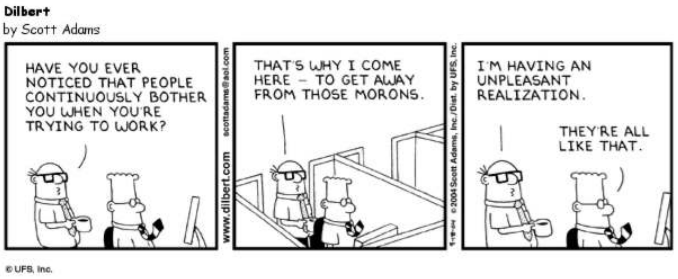
```
1.523679
2.7668
0.723327
9.6699
```

Problem 8 Counting Cubes [5 points]

Summary

We've just opened up a new facility that has 300 new spaces available for our employees. We're now in the process of identifying the employees that are moving into this new space. We have a cube assignment list and want to ensure that we're using the space effectively. In order to know that, we need to find out if there are any empty cubes, any duplicate cube assignments, and any unassigned cubes. Write a program that can analyze the current list and provide answers to those questions.

The first line of input will contain the number of employee to cube mappings that follow. Each employee to cube mapping will consist of the employee's first name followed by a single space and a cube location (0 to 300). An employee with cube location of 0 does not have an assigned cube. If a cube is empty the employee name will be "NA". The employee names are unique and we have ensured that each employee is listed only once.



The output will consist of the number of empty cubes, the number of duplicate cube assignments (*two or more* employees assigned to the same cube), and the number of unassigned cubes.

Sample Input

```
16
Joe 11
Bob 123
NA 101
Katy 125
Sam 47
Mike 59
NA 23
Vivek 62
Fred 0
Lars 74
Oscar 86
Caroline 11
NA 90
Erin 11
Rachel 111
Nate 125
```

Sample Output

```
Empty Cubes: 3
Duplicate Cube Assignments: 2
Employees without Cube: 1
```