TITLE: 24 Hour Clock
LEVEL: Easy

The Problem:
Given the time on a 24-hour clock, convert it to the time on a 12 hour clock (AM/PM).

The Input:
Each Input will contain the time on a 24 hour clock, in the format HH:MM, where 00<=HH<=23 and 00<=MM<=59.

The Output:
The output corresponds to the input time. Each line will contain the time in a 12-hour clock, in the format HH:MM XM, where 0<=HH<=12, 00<=MM<=59, and X in [‘A’,’P’].

Example:

Sample Input	**Sample Output**
09:30	9:30 AM
11:08	11:08 AM
17:12	5:12 PM

TITLE: Harshad Number
LEVEL: Easy

The Problem:
A Harshad number is a positive integer that is divisible by the sum of its digits. They are also called Niven numbers.
For example, 720 is a Harshad number because 7 + 2 + 0 = 9, which divides evenly into 720.

The Input:
Each Input is a number not more than 1000000

The Input:
The Output will determine if the given number is a Harshad Number.

Example:

Sample Input	**Sample Output**
14763	Harshad Number
91534	Not a Harshad Number
512	Harshad Number

TITLE: **Credit Card Check Digit**
LEVEL: **Average**

The Problem:

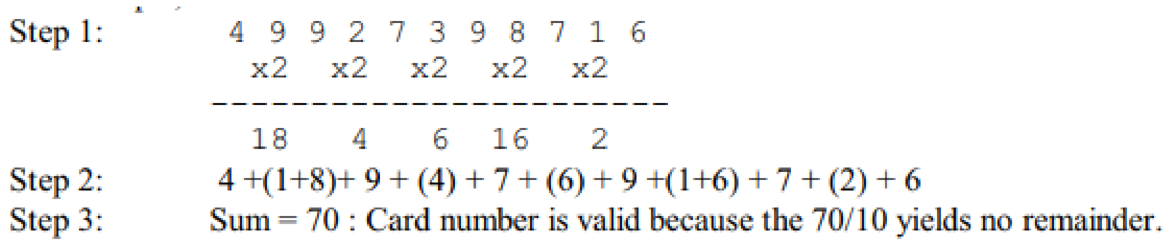
Most credit card numbers are encoded with a "Check Digit". A check digit is a digit added to a number (either at the end or the beginning) that validates the authenticity of the number. A simple algorithm is applied to the other digits of the number which yields the check digit. By running the algorithm, and comparing the check digit you get from the algorithm with the check digit encoded with the credit card number, you can verify that they make a valid combination. The LUHN Formula (Mod 10) for validating credit card numbers has the following steps:

Step 1: Double the value of alternate digits of the credit card number beginning with the second digit from the right (the rightmost digit is the check digit.)

Step 2: Add the individual digits comprising the products obtained in Step 1 to each of the unaffected digits in the original number.

Step 3: The total obtained in Step 2 must be a number ending in zero (30, 40, 50, etc.) for the account number to be validated. The total mod 10 = 0.

For example, to validate the credit card account number 49927398716:



```
Step 1:      4 9 9 2 7 3 9 8 7 1 6
               x2  x2  x2  x2  x2
             -----------------------
               18   4   6  16   2
Step 2:      4 +(1+8)+ 9 + (4) + 7 + (6) + 9 +(1+6) + 7 + (2) + 6
Step 3:      Sum = 70 : Card number is valid because the 70/10 yields no remainder.
```

Using the LUHN Formula (Mod 10) for validating credit card numbers, write a program to determine if credit card numbers are valid or invalid.

The Input:

Each input contains a credit card number which should be less than 20 digits

The Output:

If the credit card number is valid, Output the word "VALID", other wise output the word "INVALID"

Example:

Sample Input	**Sample Output**
49927398716	VALID
513467882134	INVALID
432876126	VALID

TITLE: **Adding Hex Numbers**
LEVEL: **Average**

The Problem:
Unlike the decimal system (base 10) with its 10 digits, the hexadecimal system (base 16) with its 16 digits allows for more compact representation of numbers. One of the many nice things that can be done with hexadecimal numbers is adding them.
e.g.

A + 5	= F
AA + 11	= BB
AA + FF	= 1A9

When adding the multiple-digit hexadecimal numbers it seems that the number of carry-operations is closely linked to complexity of performing the task. Your task is to measure the number of carry-operations required in adding two hexadecimal numbers.

The Input:
Each input will contain two hexadecimal numbers. The hexadecimal numbers will be positive, of arbitrary length, be in upper case, and will have at the most 256 digits.

The Output:
The output is an integer indicating the number of carry-operations for the corresponding addition question.

Example:

Sample Input:	Sample Output:
8 3 59 A7 4FE 20	0 2 1

TITLE: Fraction to Decimal
LEVEL: Difficult

The Problem:
In math class, writing fractions in their decimal form was a common task. However, the teachers would usually have you round off the number to a certain number of decimal places. But all fractions have a nice property: in decimal form, they all either terminate or repeat in a pattern. For example, the fraction 1/10 terminates, as 1/10 = 0.1. However, the fraction 1/3 repeats, as 1/3 = 0.333... = 0.(3) (note that the repeated part is parenthesized). Given a fraction, output its decimal form, parenthesizing the cyclic part of the fraction, if such exists.

The Input:
Each input will contain pair of integers $1 <= N < D <= 400$, separated by a single space.

The Output:
The output is the decimal forms of N/D, truncating and parenthesizing the repeating digits, if such is the case.

Note: the answers must match exactly and use shortest repeating pattern. Since D is always greater than N, all of the answers are less than 1.0

Example:

Sample Input	**Sample Output**
1 10	0.1
1 3	0.(3)
2 7	0.(285714)
2 3	0.(6)
6 10	0.6

TITLE: Seating Arrangement
LEVEL: Difficult

The Problem:

It's interesting to see where people sit when entering a room of other people that they do not know. One of the strategies is that when a person enters a room, they will sit at a location that maximizes the distance between themselves and the closest other person. If multiple seats are equally far, then the person will sit closer to the entrance of the room. Here we will only be dealing with linear rooms. The seats are numbered from 1 to N, where 1 is the closest to the entrance and N is the furthest.

Note: as the first person enters an empty room, every seat is equally far away from any other person, so they pick seat 1. For a room with 5 seats, with the entrance on the left, the seats will be filled up in the following order:

```
H:*****
H:1****
H:1***2
H:1*3*2
H:143*2
H:14352
```

The Input:

Each input data will be a pair of integers 1<=N<=1000 and 1<=P<=N, separated by a single space. N is the number of seats in the room; P is the number of people entering this room.

The Output:

The output is an integer indicating the seat in which Pth person is seating.

Example:

Sample Input	Sample Output
5 2	5
5 4	2

TITLE: Vowel Count
LEVEL: Easy
TIME ALLOTED: 30 minutes

The Problem:
Given a name, determine whether or not it has more vowels than consonants.

The Input:
The input is a name starts in column 1 and consists of 1-20 lowercase letters (assume the name will not contain any other characters).

The Output:
Output each name on a separate line as it appears in the input followed by a 1 (one) or 0 (zero) indicating whether or not it has more vowels than consonants. Follow the format illustrated in Sample Output.

Example:

Sample Input:	Sample Output:
ali arup travis orooji	ali 1 arup 0 travis 0 orooji 1

TITLE: **Adding Hex Numbers**
LEVEL: **Average**
TIME ALLOTED: 45 minutes

The Problem:
Unlke the decimal system(base 10) with its 10 digits, the hexadecimal system (base 16) with its 16 digits allows for more compact representation of numbers. One of the many nice things that can be done with hexadecimal numbers is adding them.
e.g.

A + 5	= F
AA + 11	= BB
AA + FF	= 1A9

When adding the multiple-digit hexadecimal numbers it seems that the number of carry-operations is closely linked to complexity of performing the task. Your task is to measure the number of carry-operations required in adding two hexadecimal numbers.

The Input:
Each input will contain two hexadecimal numbers. The hexadecimal numbers will be positive, of arbitrary length, be in upper case, and will have at the most 256 digits.

The Output:
The output is an integer indicating the number of carry-operations for the corresponding addition question.

Example:

Sample Input:	**Sample Output:**
8 3 59 A7 4FE 20	0 2 1

TITLE: **Palindromes**
LEVEL: **Difficult**
TIME ALLOTED: 60 minutes

The Problem:
A palindrome is a symmetric character sequence that looks the same when read backwards, right to left. Write a program that finds all palindromes in the input lines.

The Input:
Each line contains fewer than 3000 letters (plus an unbounded amount of other stuff).

The Output:
For each input line, output a line consisting of all the character sequences (letters only) of more than two characters that are palindromes in the input line. Ignore any characters in the input except letters—punctuation characters, digits, spaces, and tabs should all be ignored. Ignore case: treat uppercase and lowercase versions of a letter as identical. Whether the character sequences represent proper English words or sentences does not matter. Report palindromes in all uppercase characters, sorted lexicographically (in dictionary order) and separated by space characters when there are two or more for a given line. In any given line, report each palindrome only once, no matter how often it recurs in the line. When a palindrome overlaps with another, even when one is completely included in the other, report both, unless the shorter palindrome is exactly in the middle of the longer one, in which case, don't report the shorter one, even if it appears elsewhere in the line. For example, for an input line of "AAAAAA", report "AAAAAA" and "AAAAA", but not "AAAA" or "AAA". If an input line does not contain any suitable palindromes, output an empty line.

Example:

Input	Output
`As the first man said to the`	`TOT`
`first woman:`	
`"Madam, I'm Adam."`	`MADAM MADAMIMADAM`
`She responded:`	`DED ERE`
`"Eve."`	`EVE`

In the example above, the second line in the output is empty, corresponding to the second input line containing no palindromes. Some of the palindromes in the third input line, namely "ADA", "MIM", "AMIMA", "DAMIMAD", and "ADAMIMADA", are not reported because they appear at the center of reported ones. "MADAM" is reported, as it does not appear in the center, but only once, disregarding its second occurrence.

EASY ROUND - 10 pts each

1. Write a function that takes in a sorted array of integers as well as a target integer. The function should use the **Binary search** algorithm to determine if the target integer is contained in the array and should return its index if it is, otherwise -1.

Note: Usage of built-in search functions is not allowed.

Sample input:

```
array = [0,1,21,33,45,45,61,71,72,73]

    target = 33
```

Sample output:

```
3
```

2. Create a program that accepts a **POSITIVE numerical** input **only**. The output is to count and display the number of sequences needed to reach 1.

If it's Odd the next number should grow: (N*3)+1 and if it's even, the next number should decrease: (N/2). In collatz sequence, at any given number, it will eventually reach the number 1.

Sample input:

```
97
```

Sample output:

```
118
```

AVERAGE ROUND - 20 pts each

3. You're given an unordered list of unique integers nums in the range [1,n], where n represents the length of nums + 2. This means that two numbers in this range are missing from the list.

Write a function takes in the list and returns a new list with the two missing numbers sorted numerically

Sample input:

```
num = [1,4,3]
```

Sample output:

num = [2,5] // *meaning completed list should be [1,2,3,4,5]*

4. Several men were planning to set out and fish . However, the boat has a certain capacity limit which means one must be left behind. They must come up with a fair way of choosing who will be left behind.

Your task is to determine that for a given number of fishermen **N** and a constant step **K**, the position of a person who remains the last - i.e. the left-behind position.

For example if there are 10 people and each third will be allowed to ride the boat,
N=10
K=3

The left-behind is the one who initially stands at position 4.

Create an application that accepts a numeric input that represents the total number of fishermen and another numeric input for the constant step. The output is to display the only number of the position left.

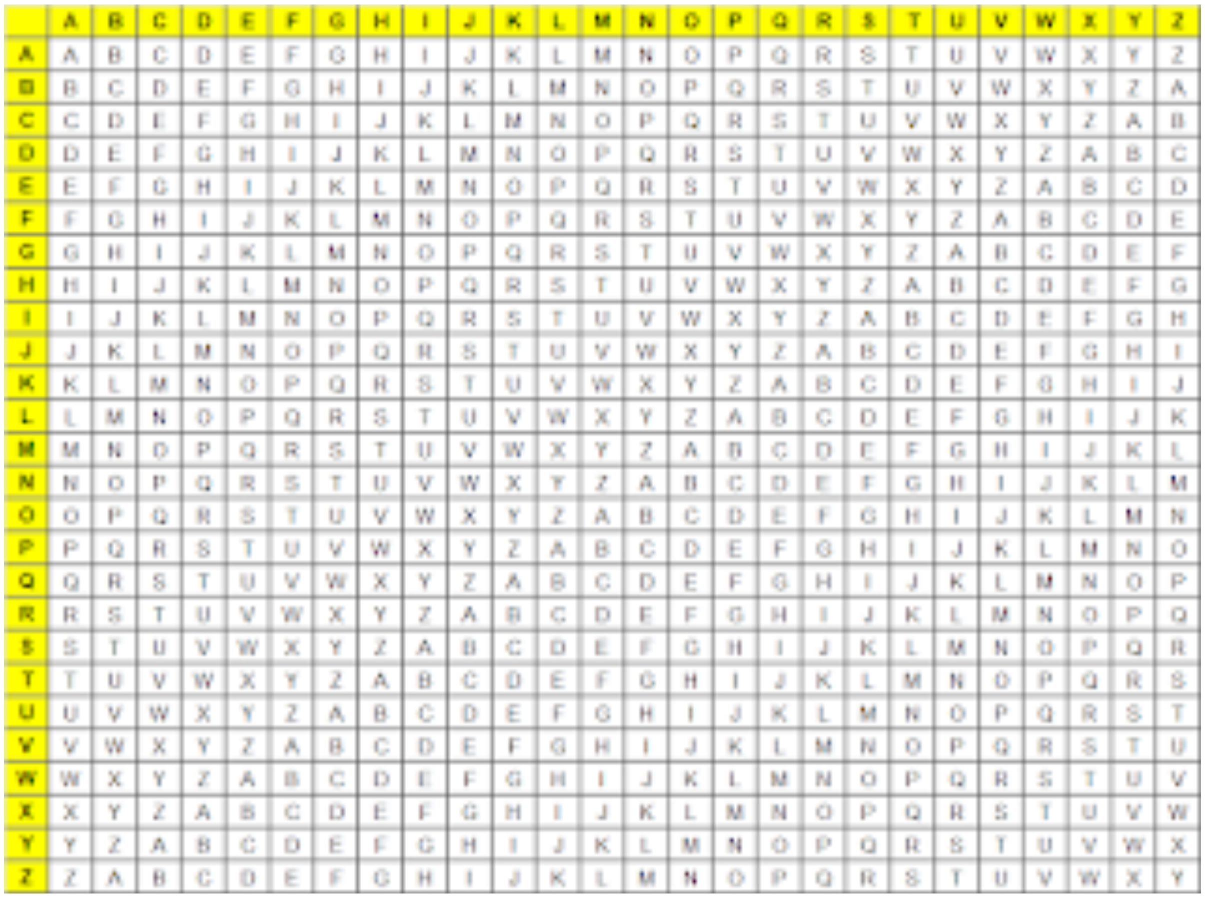
Sample Input/Output:
Input:
N: 10
K: 3

Output:
4

DIFFICULT ROUND - 30 pts each

5. The Vigenère cipher is a polyalphabetic cipher because it uses two or more cipher alphabets to encrypt the data. In other words, the letters in the Vigenère cipher are shifted by different amounts, normally done using a word or phrase as the encryption key .

Unlike the monoalphabetic ciphers, polyalphabetic ciphers are not susceptible to frequency analysis, as more than one letter in the plaintext can be represented by a single letter in the encryption.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

JAVA EXAM - IT Olypmics 2...

a key to encipher a message.

So, if we were to encode a message using the key **COUNTON**, we write it as many times as necessary above our message. To find the encryption, we take the letter from the intersection of the **Key** letter row, and the **Plaintext** letter column.

Key	C	O	U	N	T	O	N	C	O	U	N	T	O	N
Plaintext	V	I	G	E	N	E	R	E	C	I	P	H	E	R
Encryption	X	W	A	R	G	S	E	G	Q	C	C	A	S	E

Sample input/output:

Input:

Key: TEST
Text: THISISASAMPLEMESSAGE

Output:

```
MLALBWSLTQHEXQWLLEYX
```

6. You are to print numbers from **01** to **NN**. but instead of numbers which are multiples of 3 the word Fizz should be printed - and instead of multiples of 5 - the word Buzz. If the value is the multiple of both - print FizzBuzz instead.

The program should accept **Positive Integer Only**

Sample Input/Output:
Input: 16

Output:

```
01 02 Fizz 04 Buzz Fizz 07 08 Fizz Buzz 11 Fizz 13 14 FizzBuzz 16
```