Name:

Date:

Final Exam

ACES Python I Summer 2019

In this course we learned much about Python programming. See if you can use what you learned to solve the following puzzles!

Section 1

Problem 1

The volume of a sphere is given by $(4/3)\pi * r^3$ where π is a mathematical constant representing the ratio of every circle's circumference to it diameter and *r* is the radius of the circle. Let *r* equal to 73. ***Write Python code*** to calculate and print the volume of such a sphere, using 3.14 as an approximation for π. An answer in regular math terms (not Python) will not be accepted.

Your solution here:

Problem 2

Given a string of length 6, return the second half. So the string "WooHoo" yields "Hoo."

For example:

second_half('WooHoo') → 'Hoo'

second_half('abcdef') → 'def'

```
def second_half(str):
```

Problem 3

Given 2 int values greater than 0, return whichever value is nearest to 73 without going over. Return 0 if they both go over.

For example:

seventy_three(68, 74) → 68

seventy_three(72, 73) → 73

seventy_three(56, 56) → 56

```
def seventy_three(a, b):
```

Problem 4

Using a for loop, write code which will print the first 1000 integers. However, if a number is divisible by both 5 and 3, then you should print the word ‘lasangia’ instead of the number.

Your code goes here:

Section 2

Problem 1a

A year is considered a leap year if it meets the following criteria:

1. The year can evenly be divided by 4.
2. However, if the year can be evenly divided by 100, then it is NOT a leap year; UNLESS…
3. The year can be divided by 400, in which cas it IS a leap year.

For example, the year 120 is a leap year since it is divisible by four but not 100. However, the year 300 is not a leap year since it is divisible by 100, but not 400. In contrast, year 800 is a leap year since it is divisible 400. Of course, the year 431 is not a leap year since it is not divisible by four.

Write a program which will take a single integer as input from the user and output whether or not that year is a leap year. For example, an input of 120 would result in the program printing **True** while an input of 431 would result in the program printing **False.**

Problem 1b

Now, write a program which will take a single integer *n* as input from the user and print the number of leap years in the range (1, *n*), including *n.* For instance, an input of 20 would result in an output of 5, while an input of 100 would result in an output of 24 (since the year 100 is not a leap year).

Problem 2a

There exists a group of special Martians who only have three fingers. Naturally, they prefer to count only in the *trinary* numbering system. This numbering system looks a lot like the base-10 system which we're used to, except you can only have the digits 0, 1, or 2. Given a string from the user, use a "for" loop to confirm if that string is a number in trinary system. That is, *your program should print "Yes!" if the string only contains the digits 0, 1, and 2*, and "No!", otherwise. For instance,

`110022` results in `Yes!`

`1220` results in `Yes!`

`11f0011` results in `No!`

`11330` results in `No!`

Your code goes here:

Problem 2b

The way to convert from this Martian trinary system to our base-10 is actually rather simple. In fact, they have left a little set of instructions for you to follow.

1. Let str be the string from the user.
2. Let sum be zero.
3. Multiply sum by three.
4. Let digit be the last character in str.
5. Convert digit to an integer and add it to sum.
6. Remove the last character from str.
7. If the length of str is greater than zero, go to instruction 3.
8. Otherwise, you are done. Print sum.

See if you can use these instructions to write code to convert a string from the

user from Martian trinary to our base-10 system. Hint: You should use a while loop for this.

You code goes here:

Problem 3

The GCD or Greatest Common Divisors of two numbers *A* and *B* is the largest integer which divides both *A* and *B*. For instance the GCD of 5 and 10 is 5, while the GCD of 12 and 16 is 4. Note that the GCD of any number *n* and 0 is *n* since any integer divides 0.

The ancient Greeks developed a very fast algorithm for finding the GCD of two integers A and B. The way it works is as follows:

```
def GCD (A, B):

      If A equals 0, return B.

      If B equals 0, return A.

      Let X be the greater of A and B.

      Let Y be the smaller of A and B.

      Let R be the remainder when X is divided by Y.

      Return GCD(Y, R). # Here the function calls itself!
```

Unfortunately, the ancient Greeks did not know a lot of Python so the algorithm above is written mostly in proto-code. Write this proto-code as a working Python function:

```
def GCD (A, B):
```

Challenge Problem 1

Given a list of lists, count the number of even numbers in each column. Hint: Visualize each row one on top of the other, so you can see each column clearly. Then, think of a way to use two "for" loops to access each element in each column.

For instance,

count_even ([[1, 2, 1], [2, 4, 1], [3, 1, 2]]) → [1, 2, 1]

count_even ([[1, 2, 1, 4], [2, 4, 1, 2], [3, 1, 2, 0]]) → [1, 2, 1, 3]

count_even ([[1, 2], [2, 4], [3, 1], [1, 2]]) → [1, 3]

```
def count_even(list_of_lists):
```

Challenge Problem 2

Write code which will take a base-10 number from the user, convert it to binary, and print the result.

For example,

```
8 → 1000

15 → 1111

5 → 101
```