**Q.1.** **The provided code stub reads an integer, n , from STDIN. For all non-negative integers , print .**

**Ans:**

    n = int(input()) #input the value

    for i in range(n): #for loop to define range

        if i<n: #condition to apply

            print(i\*\*2) #print if condition is satisfied

**Q.2.** **An extra day is added to the calendar almost every four years as February 29, and the day is called a *leap day*. It corrects the calendar for the fact that our planet takes approximately 365.25 days to orbit the sun. A leap year contains a leap day.**

**In the Gregorian calendar, three conditions are used to identify leap years:**

* **The year can be evenly divided by 4, is a leap year, unless:**
  + **The year can be evenly divided by 100, it is NOT a leap year, unless:**
    - **The year is also evenly divisible by 400. Then it is a leap year.**

**This means that in the Gregorian calendar, the years 2000 and 2400 are leap years, while 1800, 1900, 2100, 2200, 2300 and 2500 are NOT leap years.**

**Given a year, determine whether it is a leap year. If it is a leap year, return the Boolean True, otherwise return False.**

**Note that the code stub provided reads from STDIN and passes arguments to the is\_leap function. It is only necessary to complete the is\_leap function.**

**Ans:**

def is\_leap(year):

    leap = False

    if year%4 == 0:            #Here we are checking if the year is normal year

        leap = True

        if year%100 == 0:        #Here we are checking if the year is a century year

            if year%400 == 0:

                leap = True

            else:

                leap = False

      return leap

year = int(input())

print(is\_leap(year))

**Q.3. So far, we have only heard of Python's powers. Now, we will witness them!**

**Powers or exponents in Python can be calculated using the built-in power function. Call the power function  as shown below:**

**>>> pow(a,b)**

**or**

**>>> a\*\*b**

**It's also possible to calculate .**

**>>> pow(a,b,m)**

**This is very helpful in computations where you have to print the resultant % mod.**

**Note: Here,  a and  b can be floats or negatives, but, if a third argument is present,  cannot be negative.**

**Note: Python has a math module that has its own *pow()*. It takes two arguments and returns a float. It is uncommon to use *math.pow()*.**

**Task  
You are given three integers: a, b, and m. Print two lines.  
On the first line, print the result of *pow(a,b)*. On the second line, print the result of *pow(a,b,m)*.**

**Ans:**

a=int(input())

b=int(input())

m=int(input())

print(pow(a,b))

print(pow(a,b,m))

**Q.4. Read four numbers a,b,c and d , and print the result of a^b+c^d.**

**Ans:**

a = int(input())

b = int(input())

c = int(input())

d = int(input())

print(pow(a,b)+pow(c,d))

**Q.5.** **You are given a positive integer N . Print a numerical triangle of height N-1  like the one below:**

**1**

**22**

**333**

**4444**

**55555**

**......**

**Can you do it using only arithmetic operations, a single *for* loop and print statement?**

**Use no more than two lines. The first line (the *for* statement) is already written for you. You have to complete the print statement.**

**Ans:**

for i in range(1, int(input())):

    print((10\*\*(i)//9)\*i)

# (10\*\*(i)): Raises 10 to the power of i

# //9: Performs integer division of 10\*\*(i) by 9. It generates a sequence of repeated digits. For ex:-

# When i = 1, 10\*\*(1)//9 results in 1.

# When i = 2, 10\*\*(2)//9 results in 11.

# When i = 3, 10\*\*(3)//9 results in 111 and so on

# \*i = Multiplies the result of (10\*\*(i)//9) by i.