# MATH 157: Mathematics in the world
## Homework 4 (Due March 5th, 2019 at 1:00PM)

**Please solve the first 4 problems completely. You can choose to solve 1 of the last 2 problems. The higher one will be scored if you decide to do both.**

If you run into any issues, please do not hesitate to consult the Python documentation and post in the "Python questions" discussion on Canvas.

### Problem 1

Write a paragraph explaining each of the following techniques in your own words:

1. proof by contradiction,
2. proof by induction.

### Problem 2

1. If $p$ and $q$ are prime numbers greater than 3, show that

$$p^2 - q^2$$

   is divisible by 24. [1]

2. Prove that if $ac = bc \bmod p$ and $(c, p) = 1$, then $a = b \bmod p$.

### Problem 3

1. Find all primes of the form $n^3 - 1$ where $n$ is a positive integer.
2. Find all primes of the form $n^3 + 1$ where $n$ is a positive integer.

In both parts, make sure to provide an argument showing your list is complete.

[1] Given an integer $n$, study the possible remainders of $n^2$ modulo 3 and 8.

## Problem 4

In class, we discussed the RSA encryption algorithm. To setup your own RSA encryption, one of the steps is to find $rs = 1 \bmod (p-1)(q-1)$.

In this question, you are guided to use Euclid's algorithm to find the integer $s$ given $r$ and $m = (p-1)(q-1)$.

The following identity (see `https://en.wikipedia.org/wiki/Bzout%27s_identity`) will be very handy:

**Theorem 1 (Bézout's identity)** *Let $a$ and $b$ be integers with greatest common divisor $(a, b) = d$. Then, there exist integers $x$ and $y$ such that $ax + by = d$. More generally, the integers of the form $ax + by$ are exactly the multiples of $d$.*

We will now use the following recursion to compute the coefficients $x, y$ in the Bézout's identity. Let $a, b$ be two integers, which are not both 0.

1. If $b = 0$, what are the values of $x, y, d$?
2. If $a = n * b + r$, and $bx' + ry' = d'$, what are the values of $x, y, d$ in terms of $x', y', d', a, b$?
3. Using the above recursive formula, write a python function `Egcd` which takes two integers $a, b$ as input, and produces $x, y$ and the GCD $(a, b)$ as output. [2]
4. If $(r, m) = 1$, then we can find the integer $s$ such that $rs = 1 \bmod m$ using the previous function `Egcd` by setting $a = r, b = m$. Write a function `Inverse` which takes two integers $r, m$ as input, and produces $s$ such that $rs = 1 \bmod m$ as output. [3]
5. Find the value $s$ for $r = 11189203$ and $m = 20192020$.

## Problem 5

This exercise will help us count in how many ways we can arrange 8 peaceful queens on a chess board. Since the possible moves of a queen subsume those of a rook, it follows that an arrangement of peaceful queens is necessarily an arrangement of peaceful rooks as well. We already know that the latter correspond to the permutations of 8 elements, so it suffices to iterate through these and keep track of the arrangements which are also valid in the case of queens.

The second purpose of the problem is to introduce a number of Python constructs which make programming quicker, more natural, and closer to the way we think mathematically. Even though the code can be surprisingly short, I would encourage you to take sufficient time and understand its meaning. Read the footnotes for suggestions and hints.

[2] You may use $a//b$ to find the largest $n$ so that $b * n \leq a$. Your code can be as short as 6 lines.

[3] You may call the function `Egcd`. Your code can be as short as 3 lines.

1. Use `itertools.petmutations` to write a program which prints all permutations of 0, 1, and 2. [4]

2. Write a function `distinct_elements` which verifies whether a list/tuple contains distinct elements. [5]

3. Write a function `is_peaceful` which verifies whether a permutation of size $n$ corresponds to a peaceful arrangement of $n$ queens on an $n \times n$ board. [6]

4. Write a function which counts the number of $n$ peaceful queen arrangements on a board of size $n$.

5. For each $n = 1, \ldots, 10$, compute the number of $n$ peaceful queen arrangements on a board of size $n \times n$.

**Remark.** The algorithm we described in this exercise is not optimal. The main shortcoming rests on the following observation. When enumerating the permutations of $\{1, \ldots, n\}$ in lexicographic order, the first $(n-2)!$ entries start with $1, 2$. All of these correspond to queen arrangements which exhibit a clash between the figures in the first two columns. Our current implementation does not recognize this optimization, and hence goes through all possible permutations.
A better solution uses backtracking to construct admissible arrangements step by step. See `http://en.wikipedia.org/wiki/Backtracking`.

## Problem 6

Logo is an educational programming language designed to introduce computers to children by using graphics. The premise is that you control a turtle which walks in the plane. As the turtle moves, we see the path it traces. For example, the following commands trace out a square.

```
forward 100
right 90
forward 100
right 90
forward 100
```

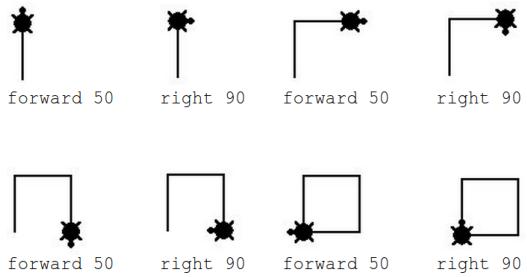
[4] See `https://docs.python.org/2/library/itertools.html#itertools.permutations`. The idea here is to iterate through `itertools.permutations(range(3))`.

[5] Given an input $X$, the simplest solution is to convert it to a set and check that its size is the same as that of $X$. See `https://docs.python.org/2/library/stdtypes.html#set`.

[6] Given a permutation, the arrangement it corresponds to already contains only one figure in every row and column. It suffices to check that each forward and backward diagonal contains a single figure each.

Hint: If an arrangement is given as a permutation sequence $s_i$ for $i = 0, \ldots, n-1$, what do the sequences $s_i + i$ and $s_i - i$ tell us about the diagonals? Don't forget to use comprehension to construct these. See `https://docs.python.org/2/tutorial/datastructures.html#list-comprehensions`.

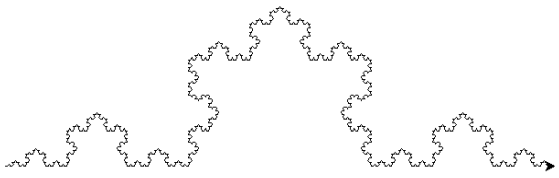
```
right 90
forward 100
right 90
```



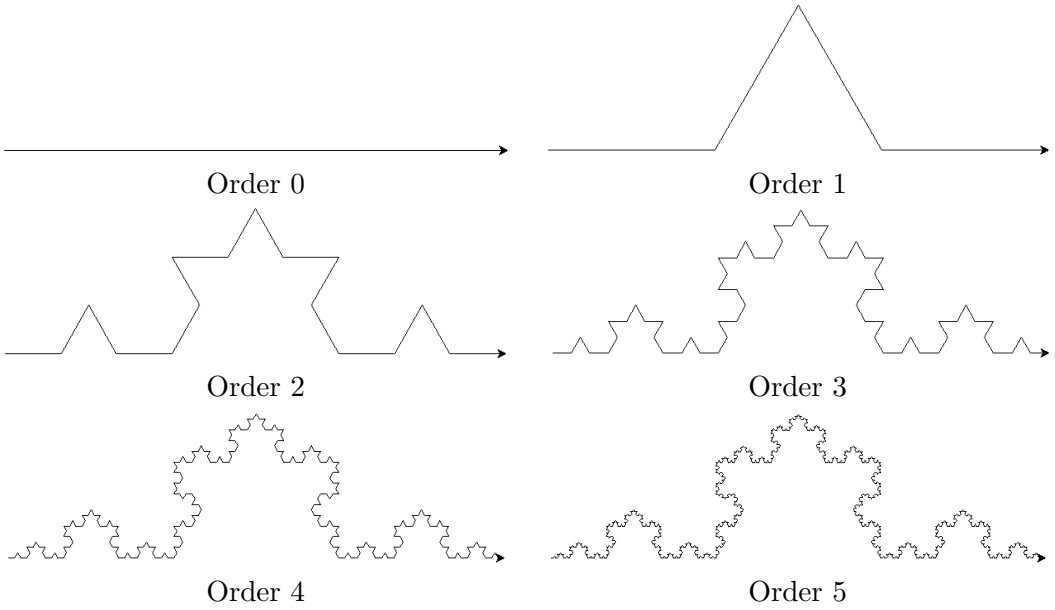

While Python is not the same as Logo, there is a standard library which has very similar functionality. The following program draws a square in Python.

```
import turtle

t = turtle.Turtle()

## Use this if the drawing speed becomes too slow
t.speed(0)

for i in range(4):
    t.forward(50)
    t.right(90)

## Use the following command to save the image in a EPS file. If you
## are having trouble converting the EPS to PNG, it may be easier to
## take a screenshot.
# turtle.getcanvas().postscript(file = "square.eps")

turtle.Screen().exitonclick()
```

Fractal curves are customarily used to teach recursion. In this exercise, we will learn how to draw the Koch curve shown below.



It looks complicated but turtle graphics make this quite easy. We will call the straight line, a Koch curve of order 0. To get to order 1, we subdivide the order 0 segment in three and replace the middle part with a triangle up. Note that the order 1 curve consists of four segments. We will use this shape as a template. In general, to construct an order $n$ Koch curve, we start with the template. Instead of drawing four segments, we place four curves of order $n-1$ in their place. The following table illustrates the first few Koch curves.

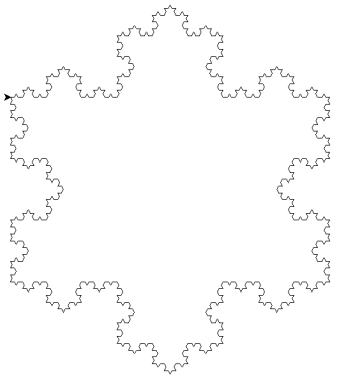



1. Write a program which draws an equilateral triangle with side length 300.

2. Write a function `koch` which takes two arguments, order and width, and draws a Koch curve with the appropriate specifications. [7] Use your function to draw a Koch curve of order 4 and length 300. Include the graphics in your write-up.

3. Use your answers to the previous two parts to write a function `snowflake` which draws a Koch snowflake of a given order and length. Draw a snowflake of order 4 and length 300, and include the graphics in your write-up.

[7] Hint: This sounds harder than it really is. If the order is 0, all we need to do is move the turtle forward the specified length. Otherwise, if the order is positive, we first draw a curve of one lower order and length a third of the specified one, rotate appropriately, draw another Koch curve of lower order, etc. In total, if the order is positive, you should make four calls to `koch` with order one less.



Koch snowflake

4. Let $P(n, \ell)$ be the Perimeter of a Koch curve of order $n$ and length $\ell$. What is $P(0, \ell)$? Find a recursive expression for $P(n, \ell)$ in terms of $P(n-1, \ell)$.

5. Find a closed formula for $P(n, \ell)$ and use it to find the perimeter of the snowflake you drew.

6. (Extra credit) Draw up to 5 other curves. If you are looking for inspiration, you can try the Hilbert curve, the Peano curve, Sierpiński arrowhead curve, or one of the variants of the Koch curve. Feel free to use colors and lines of different thickness. **Be creative!**