Written problems:

1. This problem is meant to give you a taste of some experimental number theory, and to practice using some code you've written to do experiments.

   (a) Given a positive integer B, suppose that you choose two numbers a, b at random among all integers exactly B bits long (that is, $2^{B-1} \leq a, b < 2^B$). Then one could find the probability that this pair of numbers satisfies $\gcd(a, b) = 1$ (the two numbers are said to be "coprime" if so). Do you think this probability gets larger or smaller as B grows, and why? There is no right or wrong answer here; just think about it and write what comes to mind. Good-faith effort will receive full points.

   (b) (Solve programming problem 1 before doing this part) using your code for `numCoprimePairs`, compute this probability, as a decimal to at least four places, for each value of B from 4 to 15 (you can get this probability by dividing the number of coprime pairs you find by the total number of pairs). Note that this will involve calling your code on input larger than the test bank, so you should expect it to take more than a second, but it shouldn't take more than a few minutes. Describe the trend that you see.

2. Textbook exercise 1.10, parts (a) and (b) (use a calculator/computer for the arithmetic, but show the steps). (Extended Euclidean algorithm examples)

3. Textbook exercise 1.15 (congruence is "compatible with" addition and multiplication).

4. Prove the following basic facts about congruence, asserted in class.

   (a) For any integer $a \in \mathbb{Z}$ and positive integer m, $a \equiv (a\%m) \pmod{m}$.

   (b) With a, m as above, the number $a\%m$ is the *unique* element of $\{0, 1, \cdots, m-1\}$ that is congruent to a modulo m (that is, no other element of this set is congruent to a modulo m).

   (c) For any two integers $a, b \in \mathbb{Z}$ and any positive integer m, $a \equiv b \pmod{m}$ *if and only if* $a\%m = b\%m$.

5. Textbook exercise 1.16, parts (a), (b), and (c). (Multiplication tables in modular arithmetic; we did one of these in class but it is a useful exercise to write it out again)

Programming problems:

1. Write a function `numCoprimePairs(A,B)` that receives two positive integers A, B with $A \leq B$, and returns the number of pairs of integers (m, n) such that both m and n are chosen from the set $\{A, A+1, \cdots, B\}$ and $\gcd(m, n) = 1$. All test cases will satisfy $B \leq 2^{10}$. (This function will be used in one of the written problems above.)

   Note Here and in future assignments, feel free to use Python's built-in `gcd` function. To do so, add the line `from math import gcd` to the top of your source file.

2. Write a function `bezout(a,b)` that takes two positive integers a, b and returns three integers g, u, v, where $g = \gcd(a, b)$ and $au + bv = g$. The numbers in the test bank will range up to 256 bits in size, but there will also be smaller case that can be solved by a naive approach. I recommend that you implement the extended Euclidean algorithm (either the way we discussed in class, or following one of the methods in the text), but other methods may also work.

3. Write a function `disclog(g,h,p)` that solves the following problem: given a prime p and two elements $g, h \in \mathbb{Z}/p\mathbb{Z}$, find an integer x such that $g^x \equiv h \pmod{p}$. Multiple answers will be possible; your code may return any of them. You may assume in all test cases that a solution exists. Your code will earn full points if it is able to solve this problem for primes p up to 20 bits in size in 1 second or less. However, to allow you to see exactly where the naive approach becomes too slow (or, if you're up for it, to allow you to try to implement better methods), the test bank will include cases where p ranges up to 40 bits, but **you will receive full credit as long as your code solves the test cases up to 20-bit primes**. A trial-and-error approach will suffice to solve cases up to 20 bits, but you will need to find an approach that keeps arithmetic to a minimum.

   Note Later, we'll discuss an algorithm that can solve the entire test bank. This problem is called the *discrete logarithm problem,* and there is substantial research aiming to solve it as efficiently as possible. It is believed to be impossible to solve it pratically for cryptographically large integers.