Written problems:

1. (a) Find the inverse of 7 modulo 1009. Use a calculator or computer for the arithmetic, but show the steps you use.

   (b) Solve the linear congruence $7x \equiv 3 \pmod{1009}$. Express your answer in the form of a congruence, i.e. $x \equiv \cdots \pmod{1009}$.

2. Let m be a positive number.

   (a) Prove that if $a, b \in (\mathbb{Z}/m\mathbb{Z})^\times$, then $ab\%m \in (\mathbb{Z}/m\mathbb{Z})^\times$. (The unit group is "closed under multiplication.")

   (b) Prove, by induction on n, that if $a \in (\mathbb{Z}/m\mathbb{Z})^\times$, then $a^n\%m \in (\mathbb{Z}/m\mathbb{Z})^\times$ for all nonnegative integers n.

   (c) Prove, using (b), that if $a \in (\mathbb{Z}/m\mathbb{Z})^\times$, then $a^n\%m \in (\mathbb{Z}/m\mathbb{Z})^\times$ for all *integers* n (positive, negative, or zero). Remember that negative powers in modular arithmetic are defined in terms of the modular inverse.

Note. The reason that I have added "$\%m$" to the end of several expression above is to conform to our convention: the elements of $\mathbb{Z}/m\mathbb{Z}$ are taken to be the integers $\{0, 1, \cdots, m-1\}$, so we take remainders to place numbers within that set.

3. Textbook exercise 1.24, part (a). (Solving two simultaneous congruences)

4. This problem shows one way in which congruences are compatible with exponentiation. The important point to notice is that **a different modulus is relevant in the exponent than in the base**. Suppose that m is a positive integer, and $g \in (\mathbb{Z}/m\mathbb{Z})^\times$.

   (a) Suppose that n is any integer such that $g^n \equiv 1 \pmod{m}$, and a, b are two integers such that $a \equiv b \pmod{n}$. Prove that $g^a \equiv g^b \pmod{m}$.

   (b) The **order of g modulo m** is the *minimum* positive integer n such that $g^n \equiv 1 \pmod{m}$. Prove that, if n is the order of g modulo m, and a is *any other* positive integer such that $g^a \equiv 1 \pmod{m}$, then $n \mid a$.

   Hint. Use exercise 1.19, which was Problem 6 on the last problem set.

   (c) Suppose now that n is the order of $g \pmod{m}$, and assume that g is a unit modulo m. Prove that for any two integers a, b,

   $$g^a \equiv g^b \pmod{m} \text{ if and only if } a \equiv b \pmod{n}.$$

   (For one direction of the "if and only if," you simply need to cite part (a). The other direction requires a separate argument, which will require part (b).)

   Hint: use the fact that k has an inverse modulo $p-1$, and apply Problem 4, part (a).

5. Textbook exercise 1.33, part (a). (a way to find order-q elements)
   Also read part (b) and think about it; you don't need to write up a solution, but the statement may help you think about how to solve Programming Problem 3).

Note. (a) To solve this problem, you will need to use **Fermat's little theorem,** which you can find on page 30 of the textbook (we will also discuss it in class soon).

(b) The statement of this problem uses the notation $\mathbf{F}_p$ as an alternative to $\mathbb{Z}/p\mathbb{Z}$; see Remark 1.23 in the text. When using the notation $\mathbf{F}_p$, the text write $=$ instead of $\equiv \pmod{p}$, and always leaves "take remainders" implicit. For example, they write $a^{(p-1)/q}$ rather than $a^{(p-1)/q}\%p$, but this remainder is always taken implicitly. You are free to do this when writing your solutions as well, as long as you are clear about the fact that you are doing all arithmetic in $\mathbf{F}_p$.

Programming problems:

1. Write a function `modpow(a,n,m)` that takes integers a, n, m and returns $a^n \pmod{m}$. The exponent n may be positive or negative (it will be positive in at least half of the test cases), but you may assume that if $n < 0$, then a is a unit modulo m (you'll need to use your code from the previous problem to handle the $n < 0$ case). **For this problem, please do not use Python's built-in function for modular powers. You should implement the fast-powering algorithm yourself to see how it works. However, you may look up and use the built-in function on all programming assignments after this one.** The test cases will range in size, with a and n 256-bit integers in the largest case.

2. Write a function `dh(g,p,B)` that performs Alice's role in Diffie-Hellman key exchange, using a randomly chosen secret number a. More precisely: you are given a prime number p, an element $g \in \mathbb{Z}/p\mathbb{Z}$, and Bob's transmission B. Your function should return a pair of two numbers A, S, where A is the number you transmit to Bob, and S is the shared secret. You should look up how to generate random numbers in Python (don't worry about finding a "cryptographically secure" random number generator, just use the standard one in Python). The prime p will be between 16 and 256 bits long, and the autograder will run your code several times on each input to make sure that it appears to be choosing the secret number a randomly.

3. Write a function `findOrderQ(q,p)` that takes two prime numbers q and p and returns an element $g \in \mathbb{Z}/p\mathbb{Z}$ of order q if possible. See Written Problem 5 for a way to do this. If it is *n*ot possible to find such an element g, your function should `return None`.