1. Write a function \( \text{gcd}(a, b) \) to compute the greatest common divisor of two integers \( a \) and \( b \), following the \textit{Euclidean algorithm} that we discussed in class:

(a) If \( a < b \), switch them so that \( a \geq b \).

(b) Compute \( r = a \mod b \).

(c) If \( r = 0 \) then \( b \) divides \( a \), so return \( b \).

(d) Otherwise, replace \( a \) with \( b \) and \( b \) with \( r \), and keep going.

Check that your function gives the right answer for various choices of \( a \) and \( b \).

2. Fermat’s little theorem states that if \( p \) is prime and \( a \) is an integer not divisible by \( p \), then

\[
a^{p-1} \equiv 1 \pmod{p}.
\]

We will discuss the proof on Wednesday, but for now:

(a) Search online for a list of 3- or 4-digit primes, and do a little experiment to satisfy yourself that this is true.

(b) Satisfy yourself that it is not true when \( p \) is not prime.

(c) You may find that when your numbers get big, it’s slow to compute \( x^{**} \mod z \). Rewrite your code using the Python function \texttt{pow(x,y,z)}, which does the same thing much faster.
3. This is a warm-up for RSA encryption.

(a) Choose a 3-digit prime number $p$ and a 4-digit prime number $q$, and let $n = pq$ and $k = (p - 1)(q - 1)$. Choose a small number $e$ such that $\gcd(e, k) = 1$, and write some code to find a number $d$ such that

$$de \equiv 1 \pmod{k}.$$

(b) Let $m = 1234$, where $m$ stands for message. Find $c$, which stands for ciphertext, such that $c \equiv m^d \pmod{n}$.

(c) The original message should satisfy $m \equiv c^e \pmod{n}$. Check that this is true, and if it’s not then find and correct your mistake.

4. Choose a partner with whom to exchange secret messages.

(a) Write a short secret message. Break it into blocks of 3 letters, and encode each one as a 6-digit number using the code

\[
\begin{align*}
A &= 01 & B &= 02 & C &= 03 & D &= 04 & E &= 05 & F &= 06 & G &= 07 \\
H &= 08 & I &= 09 & J &= 10 & K &= 11 & L &= 12 & M &= 13 & N &= 14 \\
O &= 15 & P &= 16 & Q &= 17 & R &= 18 & S &= 19 & T &= 20 & U &= 21 \\
V &= 22 & W &= 23 & X &= 24 & Y &= 25 & Z &= 26 & \text{period} &= 27 \\
& & & & & & \text{space} &= 00.
\end{align*}
\]

You could write a function to encode and decode messages, or you could do it by hand.

(b) Encrypt your 6-digit numbers using $c \equiv m^d \pmod{n}$. Send an email to your colleague containing $n$ and $e$ (your “public key”) and the sequence of 6- or 7-digit numbers $c$. But do not send $p$, $q$, $k$, or $d$ (your “private key”), nor the unencrypted messages $m$.

(c) Have your partner decrypt the message using $m \equiv c^e \pmod{n}$, and decode it using the code above.

(d) Have your partner write a secret message in reply, encode it as sequence of 6-digit numbers, and encrypt it using your public key: that is, $c \equiv m^e \pmod{n}$. Have them send it to you, and decrypt it using your private key: $m \equiv c^d \pmod{n}$. Then decode it and see what they said.

(To that the roles of $d$ and $e$ are switched in this last part. Your partner still doesn’t know your private key.)
5. Do it again with a 6-digit prime for $p$ and a 7-digit prime for $q$. Probably there is a bottleneck around computing $d$. Next week we’ll discuss a faster way to compute $d$, using the extended Euclidean algorithm. If you get to this problem before then, either think about a clever way to find $d$, or go read about that algorithm.

For real-life encryption, like sending your credit card number to a website, we use primes that are hundreds of digits long. Can your code handle primes that big?