Question 1 Approach

The goal of this question is to generate a random prime number with exactly d digits (where 100 <= d <= 1000). Testing the primality of such large numbers using deterministic algorithms is computationally expensive, so we use Miller-Rabin to test for primality. In Miller Rabin we need to compute x = a^d mod n so we use modular exponentiation with repeated squaring to compute it efficiently without overflowing. The algorithm repeatedly squares the base and multiplies it into the result only when the current bit is 1. This is done in the mod_exp(base, exponent, modulus) function. It computes (base^exponent) mod modulus. The actual Miller-Rabin is done in is_probable_prime(n, k=20) to check whether a number n is probably prime. A prime number p has to satisfy Fermat's Little Theorem meaning for any such that 1 < a < p, a^(p-1) congruent 1(mod p). Composite numbers can also satisfy this by accident so they are Fermat liars. Miller-Rabin makes the test more accurate by decomposing p-1 as 2^r * d with d odd and checking a sequence of squarings. Basically the algorithm runs as following:

1. write n - 1 = 2^r * d with d odd
2. repeat k rounds:
    a. pick a random base a
    b. compute x = a^d mod n
    c. if x = 1 or x = n - 1, n passes this round
    d. otherwise square x repeatedly up to r-1 times:
        i. if any square equals n - 1, n passes
        ii. if none, n is composite
3. If n passes all k rounds, then it is probably prime

Finally, in generate_d_digit_prime(d), we create random d digit candidate numbers and test them until 1 is found to be prime. We get the lower and upper bounds for the number with d digits then generate a number within those bounds. We take only odd numbers since the only prime even number is 2. We test if the number is prime using is_probable_prime(). We repeat until a prime is found.