Your homework is due at 11:59 PM on Sept. 12, 2018. You will use the Learn OCaml platform to submit and automatically grade your homework. You can submit your homework as often as you like until the due date and check your grade immediately.

Reminder: Do not change the names of the declared functions. They will be specifically called by the testing code.

1. **Fix Me**

   The two functions pow and fib contain various syntax and type errors. To familiarize yourself with OCaml correct these programs such that OCaml accepts them and they compute the correct answer. Function descriptions and example inputs/outputs are provided in the comments.
2. **Programming Warm Up**

   The Newton-Raphson method can be used to find the roots of a function; in particular, it can be used for computing the square root of a given integer. Given a good initial approximation, it converges rapidly and is highly effective for computing square roots, solving the equation a−x2=0.
   To compute the square root of a, choose a positive number, for example 1.0, as the first approximation. If x is the current approximation, then the next approximation x' is:

   x′=ax+x2.

   Stop as soon as the difference between x and x′ becomes too small.
   Implement the function findroot x acc where x approximates the square root of a with accurracy acc. i.e. return x' when the absolute difference between x and x' is smaller than acc.
   We use epsilon_float as the desired accuracy, which in OCaml is the difference between 1.0 and the smallest exactly representable floating-point number greater than 1.0.
   Note that we made findroot a local function to be defined inside the function square_root.
   Remark: You can compute the absolute value of a floating point number in Ocaml with the library function abs_float : float -> float.
3. Write the function pow_tl : int -> int -> int tail-recursively. The power function raises the integer n to the power of k. The tail-recursive version of the power function is the helper function aux : int -> int -> int -> int that takes as input a base n, an exponent k, and an accumulator acc to build up the result.
4. Implement the function is_prime : int -> bool. This can be done naively as follows: to test whether given n (where n > 1) is prime, we try to find a number that divides n by checking for each number xwhere x * x <= n whether it divides n. For inputs n <= 1, you should raise the exception Domain.
5. Compute the greatest common divisor gcd of two integers a and b using Euclid's algorithm. Euclid noted that a can be written in quotient remainder form where a = b * q + r; moreover, to compute the greatest common divisor of a and b, it suffices to compute the greatest common divisor of b and rinstead until the second number is zero.

   Here is an example of how it works:

To compute gcd 60 42, we first compute gcd 42 18, then gcd 18 6, and finally gcd 6 0. At this point we stop and return 6.
Implement the function gcd : int -> int -> int using this algorithm. If either of the inputs is negative, you should raise the exception Domain.