Software Engineering: Assignment 1
"Looking for Fermat's Last Theorem Near Misses"

HW1 is a "tiny group" programming assignment. You will either choose a partner, or randomly assigned a partner (if by the end of week 2 you didn't had partner yet), and the two of you will work together on this assignment. You may NOT work with anyone else on this assignment.

The Setup: The 17th century mathematician Pierre Fermat's sparked 400 years of nightmares for mathematicians by claiming he had a proof for his "last theorem," but then not providing the proof. Fermat's last theorem states that there are no natural numbers (1, 2, 3,…) x, y, and z such that $x^n + y^n = z^n$, in which n is a natural number greater than 2. After mathematicians spent centuries of futility trying to devise a proof, the English mathematician Andrew Wiles published a proof of Fermat's last theorem in 1995, a proof that is generally believed to be correct (though at about 130 pages, it is pretty difficult to understand). If you'd like to know more about Fermat's Last Theorem, and pop culture references to it (like by the Simpson's and Star Trek), please see **https://www.youtube.com/watch?v=ReOQ300AcSU.**

Anyway, your HW1 assignment is to write a program that helps an interactive user search for "near misses" of the form (x, y, z, n, k) in the formula $x^n + y^n = z^n$, where x, y, z, n, k are positive integers, where 2< n <12, where 10 <= x <= k, and where 10 <= y <= k. When your program starts, it should ask the user for n (the power to use in the equation) and ask the user of k (which limits the range of x and y possibilities to test). For now I am not limiting k, except to say that it should be > 10. You may want to impose an upper limit on k if you find that your program is crashing due to overflowing integer variables, but don't make it TOO small, or you will limit the kinds of near misses you can find.

Your program should then look for "near misses" of the form $x^n + y^n <> z^n$. A "near miss" is a RELATIVELY small difference between $(x^n + y^n)$ and z^n for some integers x, y, and z. Your program should systematically search for x, y, and z combinations that are "almost right." NOTE WELL: Fermat's last theorem tells us that there should NOT be any $x^n + y^n = z^n$ combinations that are EXACTLY right for any n > 2. Also notice that although x and y are constrained above, z is NOT constrained directly.

For each possible x,y combination, I suggest that you calculate $(x^n + y^n)$, and then look for whole numbers z and z + 1 that "bracket" $(x^n + y^n)$, so that $z^n < (x^n + y^n) < (z+1)^n$. Find out which one (either z^n or $(z+1)^n$) is closer to $(x^n + y^n)$, and determine the "miss" as the smallest of these two values: $[(x^n + y^n) - z^n]$ or $[(z+1)^n - (x^n + y^n)]$. Then divide that miss by $(x^n + y^n)$ to obtain the RELATIVE size of the miss. Do this for all the possible combinations available, always keeping track of the smallest relative miss so far. Every time you find a new smallest relative miss, type out to the screen the current x, y, z, the actual miss (an integer), and the relative miss (a percentage or fraction). Make sure the interactive user can tell what the numbers mean as they are printed out; that is, label them well. When you have exhausted all the x, y, z triples possible, end the program. The screen should show the smallest possible miss as the last thing printed on the screen. Make sure that the IDE pauses so that the interactive user can carefully examine the output to the screen.

PROGRAMMING TIP: Depending on which programming language you use, and what n and k the interactive user picks, you may have problems with variables overflowing. Please do your best to minimize those problems, but it is not a tragic thing if that happens for large n and k for programming languages that have trouble with big integers.

It is better to hand in something simple that works, and on time, rather than handing in something fancy that is either late or doesn't work. The best programs will work, be on time, be well documented (see below), and will make it clear to the interactive user what's going on using screen output.

TEAMWORK TIPS: Each person in your tiny group is vital to the team effort. Make certain that you listen carefully to your partner. Make sure that all members of your team know EXACTLY what is expected of them, and what the other members of the team are doing. DESIGNING your solution BEFORE doing any coding is always a sensible idea in software engineering, but it is an even BETTER idea when working as part of a software development team. Communication is key in successful software development; practice that for HW1.

The specification above mentions “well documented.” Here are some hints of what good documentation might look like:

Start your program with an “opening comment.” You may include more items than the items listed here, but you may NOT leave out any of the items mentioned here. If you DO leave any of these out, or if you do a poor job on any of them, you will lose points. If one of these items is not applicable to your program (for example, if it doesn’t use any external files), then include the name of the item in your opening comment but add “N/A” or “none” to indicate that this isn’t applicable to your program.

Items to include in your opening comment:

- a title for your program
- the name of the file that holds your program
- a list of any external files necessary to run your program
- a list of external files your program creates (If you list any external files, briefly explain what each of them contains.)
- the names of any programmers working on the program
- email address of all programmers
- the course number and section number of the course you’re writing this program for
- the date you finished the program and submitted it
- an explanation of what the program does
- any resources you used to complete the program (Always give credit where credit is due; for example, if you used a website to check on an algorithm, list that here.)

Other comments that are required:

- each declaration should include a comment that explains its use
- each subprogram (function, subroutine, object,…) should have an opening comment describing its purpose)
- each loop should be preceded by a comment that describes its purpose
- any statement that is particularly unclear or “tricky” should have a comment clarifying its use to the human reader

What programming language to use?

- Java, C++, or Python. If you want to use something else, please contact me.

What to submit?

- Create an account on github so that you and your partner can share the code development easily. Submit only the clone command. I will download your code and run it according to the readme file that should be available on github.
- Your github project should include the following:
    - The source code
    - The readMe file
    - The executable file that I will run (jar file if using Java, exe file if Python or C++).
    - Any other necessary library I need to run your code
- Do not forget to share the repository with me (fwedyan@lewisu.edu), or make it public