Fibonacci Series

Question 1. (30 points)

The Fibonacci sequence is defined as a recursive equation where the current number is equal to the sum of the previous two numbers:

$$F_n = F_{n-1} + F_{n-2}$$

By definition, the first two Fibonacci numbers are always: $F_0 = 0$, $F_1 = 1$. The remaining numbers in the sequence are calculated from the above equation. Please note that the **n** in the equation represents a particular Fibonacci number, not some mathematical constant. Here is a list up to F_{11}:

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}
0	1	1	2	3	5	8	13	21	34	55	89

Goal: Ask the user for a number and check if it is in the Fibonacci sequence!

Program Inputs

- **Enter a value to check:**
  - *The user will always enter 1 or higher, no error checking needed!*

Program Outputs

- **XXX is a Fibonacci number**
  - *Replace XXX with the original value*
- **XXX is NOT a Fibonacci number; try YYY or ZZZ.**
  - *Replace XXX with the original value, YYY and ZZZ with the Fibonacci numbers around the user's guess*

Test Case 1:
```
Enter a value to check: 75025
75025 is a Fibonacci number!
```

Test Case 2:
```
Enter a value to check: 2
2 is a Fibonacci number!
```

Test Case 3:
```
Enter value to check: 9
9 is NOT a Fibonacci number, try 8 or 13.
```

Test Case 4:
```
Enter value to check: 12
12 is NOT a Fibonacci number, try 8 or 13.
```

Test Case 5:
```
Enter value to check: 500000
500000 is NOT a Fibonacci number, try 317811 or 514229.
```

Newton's Method

Question 2. (70 points)

Implement Newton's Method to find roots of 3rd order polynomials given an initial guess (`http://mathworld.wolfram.com/NewtonsMethod.html`).

Key programming concepts: while loops, if statements, conditions, and, or
Approximate lines of code: 23 (does not include comments, white space
Prohibited code/commands/functions: Any automatic root finders, return, root, poly

Program Inputs

- **Enter xˆ3 coeff:**
  Enter xˆ2 coeff:
  Enter xˆ1 coeff:
  Enter xˆ0 coeff:
  - *User will always enter numeric values for each coefficient*
- **Enter initial guess:**
  - *User will always enter a numeric value*

Program Outputs

- **Improper polynomial; no possible roots.**
  - *Display this message if given polynomial doesn't have any roots*
- **Solution not found in 5000 iterations.**
  - *Display this message if Newton's Method takes over 5000 iterations (typically happens when roots are imaginary)*
- **Found root of XXX in YYY iterations.**
  - *Replace XXX with the found root to 3 decimal places and YYY with the number of algorithm iterations*

Assignment Details

Newton's Method is a common numerical analysis method to approximate roots of polynomials. Remember that a polynomial is a mathematical expression of variables raised to powers and multiplied by coefficients, for example:

$$f(x) = x^2 + 7x + 10 \tag{0..1}$$
$$f(x) = x^3 + 4x^2 - 11x - 30 \tag{0..2}$$

It is often necessary to find the roots of a polynomial, which are the values of the independent variable (x) that evaluate the polynomial to zero, e.g. $f(x) = 0$. For example, the first polynomial $f(x) = x^2 + 7x + 10$ has two roots at -2 and -5. We can solve for these values programmatically by iteratively solving Newton's equation. The method starts with an initial guess (x_{old}) and uses the given polynomial and its derivative to to find a new guess (x_{new}):

$$x_{new} = x_{old} - \frac{f(x_{old})}{f_{derivative}(x_{old})}$$

This is best illustrated with an example. The polynomial $f(x) = x^2 + 3x + 2$ has the derivative $f_{derivative}(x) = 2x + 3$, and we can find one of its roots by starting from an initial guess of 0. The update formula is used every loop iteration until a root is found or too many tries have been made. A root is reached when x_{new} stops changing based on the following equation:

$$\frac{|x_{new} - x_{old}|}{|x_{new}|} < 10^{-9}$$

Here is a complete example:

$$f(x) = 0x^3 + x^2 + 3x + 2 \quad and \quad f_{derivative}(x) = 2x + 3$$

x_{old}	$x_{new} = x_{old} - \frac{f(x_{old})}{f_{derivative}(x_{old})}$	$\frac{absolute(x_{new} - x_{old})}{absolute(x_{new})} < 10^{-9}$	Solution?
$x_{old} = 0$	$-0.666666667 = 0 - \frac{2}{3}$	$\frac{0.666666667}{0.666666667} < 10^{-9}$	No
$x_{old} = -0.666666667$	$-0.933333333 = -0.666666667 - \frac{0.444444444}{1.66666667}$	$\frac{0.266666667}{0.933333333} < 10^{-9}$	No
$x_{old} = -0.933333333$	$-0.996078431 = -0.933333333 - \frac{0.0711111111}{1.13333333}$	$\frac{0.062745098}{0.996078431} < 10^{-9}$	No
$x_{old} = -0.996078431$	$-0.999984741 = -0.996078431 - \frac{0.00393694733}{1.00784314}$	$\frac{0.00390630961}{0.999984741} < 10^{-9}$	No
$x_{old} = -0.999984741$	$-1 = -0.999984741 - \frac{1.52592547e-05}{1.00003052}$	$\frac{1.52587891e-05}{1} < 10^{-9}$	No
$x_{old} = -1$	$-1 = -1 - \frac{2.32830866e-10}{1}$	$\frac{2.32830755e-10}{1} < 10^{-9}$	Yes

Sample Output

The following test cases do not cover all possible scenarios (**develop your own!**) but should indicate if your code is on the right track. **To guarantee full credit, your program's output should exactly match the output below.**

Test Case 1:

```
Enter x^3 coeff: 1
Enter x^2 coeff: 0
Enter x^1 coeff: 0
Enter x^0 coeff: 0
Enter initial guess: 1
Found root of 0.000 in 614 iterations
```

Test Case 2:

```
Enter x^3 coeff: 0
Enter x^2 coeff: 0
Enter x^1 coeff: 0
Enter x^0 coeff: 4
Improper polynomial; no possible roots.
```

Test Case 3:

```
Enter x^3 coeff: 1
Enter x^2 coeff: -3
Enter x^1 coeff: -4
Enter x^0 coeff: 12
Enter initial guess: 7
Found root of 3.000 in 9 iterations
```

Test Case 4:

```
Enter x^3 coeff: 1
Enter x^2 coeff: -3
Enter x^1 coeff: -4
Enter x^0 coeff: 12
Enter initial guess: 1.5
Found root of 2.000 in 5 iterations
```

Test Case 5:

```
Enter x^3 coeff: 0
Enter x^2 coeff: 1
Enter x^1 coeff: 3
Enter x^0 coeff: 2
Enter initial guess: 0
Found root of -1.000 in 6 iterations
```

Test Case 6:

```
Enter x^3 coeff: 0
Enter x^2 coeff: 10
Enter x^1 coeff: -1
Enter x^0 coeff: 5
Enter initial guess: 5
Solution not found in 5000 iterations...
```