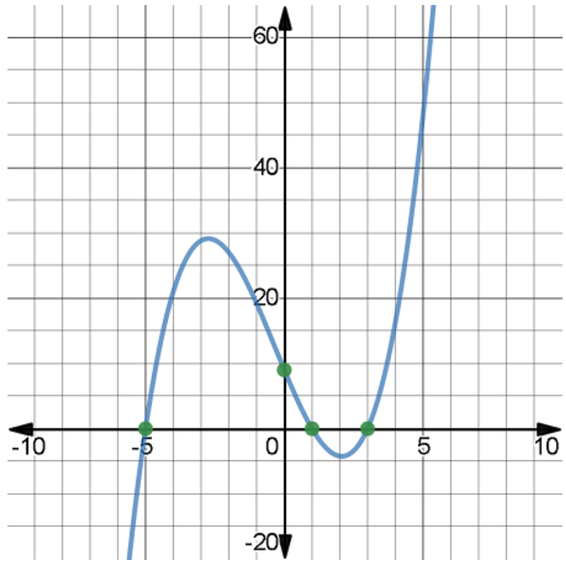
Take-home test for Unit 7

Intro



A **cubic polynomial** is a function of $\boldsymbol{x}$ of the form $\boldsymbol{A\,x^3 + B\,x^2 + C\,x + D}$, where parameters $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{D}$ are real numbers. Consider three concrete examples:

$$P_1(x) = x^3 - x^2 + 4\,x - 10$$

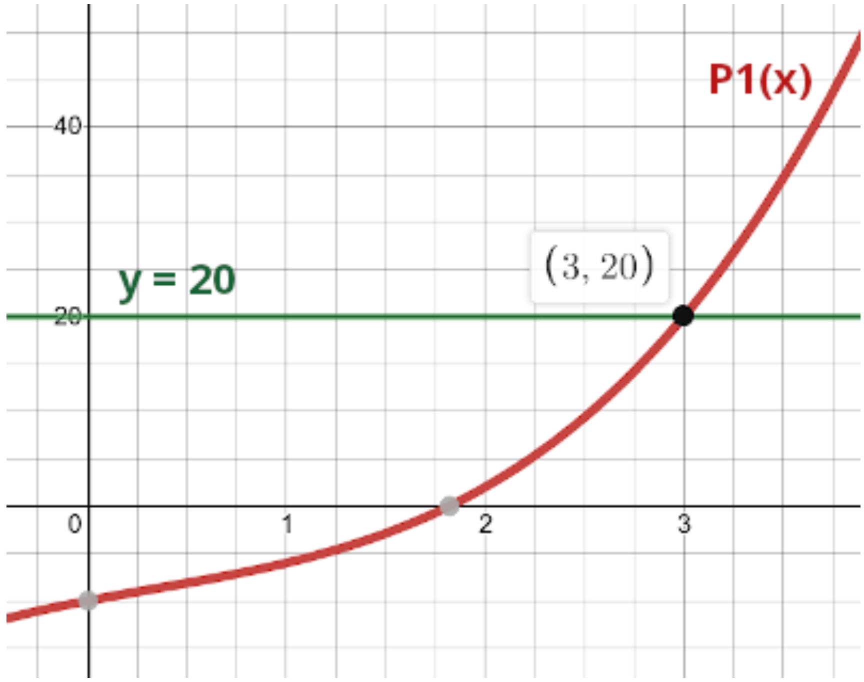
$$P_2(x) = x^3 + 0.5\,x^2 + x - 6$$

$$P_3(x) = 3\,x^3 + 13.6\,x^2 + 13.2\,x + 37.8$$

A **root** of a cubic polynomial equation $\boldsymbol{P(x) = y}$ is a value $\boldsymbol{r}$ such that $\boldsymbol{P(r) = y}$.

For example, $\boldsymbol{r = 3}$ is a root of the equation $\boldsymbol{P_1(x) = 20}$:

$$P_1(3) = 3^3 - 3^2 + 4 \cdot 3 - 10 = 27 - 9 + 12 - 10 = 20$$



In Python, a polynomial can be implemented as a function. The three examples shown above can be written as:

```
def P1(x):
    return x*x*x - x*x + 4*x - 10

def P2(x):
    return x*x*x + 0.5*x*x + x - 6

def P3(x):
    return 3*x*x*x + 13.6*x*x + 13.2*x + 47.8
```

Roots of a polynomial equation ***P(x)*** = ***y*** can be found using function `goalSeek` if you supply the tested polynomial function as the `function` parameter, set the `target` parameter equal to ***y***, and start with a good `LowLimit` and `HighLimit` interval enclosing the root.

Note that `goalSeek` requires that the tested function `f` is such that `f(LowLimit)` ≤ `target` ≤ `f(HighLimit)`. All polynomial equations provided for this task will satisfy this requirement.

For this test, you are provided with the file `poly.txt` containing coefficients of 25 polynomial equations. The first few lines of the file look as follows:

```
# A     B       C       D       y       Lo      Hi      Equation
1.5     -3.1    9.5     -16.23  9.45    -4.56   6.28    1.5 x^3 - 3.1 x^2 + 9.5 x - 16.23 = 9.45
3       22.1    14.4    3.39    -3.31   -11.77  2.08    3 x^3 + 22.1 x^2 + 14.4 x + 3.39 = -3.31
3       33.8    55      67.32   0.12    -15.66  -4.08   3 x^3 + 33.8 x^2 + 55 x + 67.32 = 0.12
2       -9.2    3.9     -41.05  4.85    -0.97   8.45    2 x^3 - 9.2 x^2 + 3.9 x - 41.05 = 4.85
...
```

The first line is a header, you will have to skip it when reading the file. Each of the following lines starts with the numbers `A`, `B`, `C`, `D`, and `y`, uniquely determining a polynomial equation:

$$A\,x^3 + B\,x^2 + C\,x + D = y.$$

The equation coefficients are followed by the suggested `Lo` and `Hi` limits. Each provided polynomial equation is guaranteed to have exactly one root in the interval `Lo` $\leq x \leq$ `Hi`. We will use `goalSeek` to find this root for each of the provided equations.

The last column is a conventional representation of the equation, which you can be pasted in WolframAlpha (https://www.wolframalpha.com/) to confirm that your program correctly finds the roots. For example, here is the response (https://www.wolframalpha.com/input/?i=1.5+x%5E3+-3.1+x%5E2%2B+9.5+x+-16.23%3D+9.45) for the first polynomial in the file, in particular it says that the real root is equal to *2.4*.

Task

In this task, we are going to write a program `test7.py` that finds the roots of cubic polynomial equations listed in the file `poly.txt` using `goalSeek` function.

Step-by-step implementation:

1. Use `goalSeek` function to find the root of the polynomial equations $P_1(x) = 20$ shown in the introduction. The expected answer is: *3*. Choose the low and high limits to contain the root you are looking for (`-5` and `5` would suffice). Confirm that your program is finding the root correctly.
2. Copy the provided file `poly.txt` in the same folder with your script. Read the file. Discard any line that starts with a `#` symbol (thus skipping the header).

   > You can use operator `!=` to check if two values are not equal. For example, the condition `line[0] != '#'` is `True` if the first character in `line` is not a `#`.

   For each non-header line, split it and use `float` function to extract `A`, `B`, `C`, `D`, `y`, `Lo`, and `Hi`. Print them out to confirm that your program correctly extracts these parameters.
3. Write a function `makePoly` that **generates** a Python function representation of a cubic polynomial from its coefficients `A`, `B`, `C`, `D`. For example, the polynomial function `P1` we used earlier:

```
def P1(x):
    return x*x*x - x*x + 4*x - 10
```

could be created with the generator function as follows:

```
P1 = makePoly(1, -1, 4, -10)
```

4. For each polynomial you read from the file, use `makePoly` to generate its Python function representation. Run `goalSeek` on this function with `y` and the given `Lo` and `Hi` limits to find the root of the equation. You can use WolframAlpha (https://www.wolframalpha.com/) to check that the roots are correct.

   After that, for each polynomial, print out its coefficients `A`, `B`, `C`, `D` and `y`, followed by the root your found. Format the output nicely making sure the columns line up. Also, add `=` and `at` to clearly separate the coefficients, the target value `y`, and the root:

```
    1.50    -3.10     9.50   -16.23  =      9.45  at      2.40
    3.00    22.10    14.40     3.39  =     -3.31  at     -6.70
    3.00    33.80    55.00    67.32  =      0.12  at     -9.60
    2.00    -9.20     3.90   -41.05  =      4.85  at      5.10
    3.00    61.80   118.60   130.24  =      0.04  at    -18.60
    3.00     6.70    13.90    26.59  =      3.79  at     -1.90
    0.30    -1.13   -10.96   -42.47  =     -9.71  at      9.10
    3.00   -25.80   -57.60   -71.13  =     -7.53  at     10.60
    0.60    10.68    24.06    81.36  =      0.78  at    -15.80
    1.40     9.92    18.34    53.55  =      7.73  at     -5.80
    2.50    26.35    37.13    74.81  =     -2.38  at     -9.30
    1.20   -13.86    -2.47  -109.63  =     -3.85  at     12.30
    2.80   -12.88    -4.68   -60.55  =      1.05  at      5.60
    2.00    19.40    35.80    43.29  =      4.79  at     -7.70
    2.00     2.60     8.80     2.61  =     -3.79  at     -0.80
    2.20    -6.50     1.60   -44.27  =     -1.07  at      4.00
    2.00   -11.20     4.90   -76.68  =     -9.58  at      6.10
    1.40    25.96    35.84   143.02  =      3.82  at    -17.40
    2.90    31.74    14.60    39.74  =     -2.66  at    -10.60
    2.00    24.80    17.90    68.66  =     -2.74  at    -11.90
    2.80    26.06    42.85    63.66  =      0.52  at     -7.70
    1.50    30.45   100.25   102.83  =      0.53  at    -16.50
    1.30   -20.72    -8.54   -12.76  =      8.56  at     16.40
    1.80   -28.60    -9.85  -136.92  =      0.03  at     16.50
    0.90    10.63    28.73   109.43  =      9.52  at     -9.70
```

> If the roots you compute look similar but slightly different, try to choose a smaller `maxError`, and make sure to use `format` function when outputting the numbers.