Creating and editing programming loops

We will create a program loop that finds the square root of x by successive approximation. Start with an initial guess z. If z is too large, x/z will be smaller than the square root of x. If z is too small, x/z will be larger. If z is the square root, z = x/z exactly. If we average z with x/z, the answer will be closer to the square root than z. This is because we have averaged a number that is too large with one that is too small. So we will average z with x/z eight times and the answer will be very close to the square root. We will do this in a programming loop.

Open the programming menu using View|Toolbars|Programming. The loop will be calculated eight times, so the loop will have a statement "for i = 1..8," where i is a counter.
The loop will return to us the answer which we will label *rootx*. First type *rootx:*=, then choose *add line* from the programming menu.

rootx :=

The vertical bar indicates lines in a program.

The first thing to do is to make an initial guess for rootx. For example, z := 2. However, in a programming loop the left arrow is used instead of :=. It is chosen from the programming menu.

rootx :=
z ← 2

The left arrow is used to define values inside a program.
The value of z will not be known outside of the program.
:= cannot be used in a program loop.

Now we add the repeated loop by choosing *for* from the programming menu.

rootx :=
z ← 2
for ∈

We will set x = 49 for now so we do not get an error message saying that x is not defined.

x := 49

Now fill in values for a counting variable, i, and type the expression which defines the new z in terms of the old z.

rootx :=
z ← 2
for i ∈ 1 .. 8
$z \leftarrow \frac{z + \frac{x}{z}}{2}$

The new z is the average of the old z and x/z.

Definitions within a loop are local, not global:

A property of programming loops is that they are not permitted to change the values of any variables defined outside the loop, with one exception.

In Mathcad, the last line of the program loop determines the value that is returned.

This value is put into *rootx* because we start the loop by typing *rootx* := and it is the := that defines variables outside the program loop.

Now let's see how it works by typing *rootx:=*

$rootx = 7$ 7 is the square root of x which we set equal to 49.

Maybe we didn't have to do the loop 8 times. Let's look at the value of z at the end of each pass through the loop. We can make z a vector with 8 elements numbered 0 to 7. It is good practise to initialize the vector to zero. So we will copy the program loop above and add new lines that initialize a vector z.

$$\begin{aligned}
&rootx(x) := \\
&\quad \text{for } i \in 0..7 \\
&\qquad z_i \leftarrow 0 \\
&\quad z_0 \leftarrow 2 \\
&\quad \text{for } i \in 1..7 \\
&\qquad z_i \leftarrow \frac{z_{i-1} + \frac{x}{z_{i-1}}}{2} \\
&\quad z
\end{aligned}$$

To open a new line above an existing line, place the cursor at the LEFT and underneath of that line and choose *add line*. This allows us to put a new for loop at the top. z is now a subscripted variable. In the next to last line, the next value of z_i is defined in terms of the previous value.

You want all elements of the vector z returned, not just z_7.

The thing that Mathcad puts into *rootx* is the thing on the last line of the program. The last line should then be z and should not be z_i.

To open a new line at the bottom (below the *for* loop) we position the blue cursor to the right of and underneath the entire loop, to indicate that we do not want the new line with z to be inside of the loop.

See the next page for more examples.

Also, *rootx* can be made a function of x simply by typing *rootx*(x). Then *rootx*(49) gives the 8 successive approximations to the square root of 49 and x(2) does the same thing for 2. Also we can type *rootx*(2) and find the root of 2 without typing in the program a second time.

Format|Result has been used below to show all 15 decimal places that are available in Mathcad. Also, "show trailing zeroes" has been selected.

This method of finding the square root is very effective.

$$rootx(49) = \begin{pmatrix} 2.000000000000000 \\ 13.250000000000000 \\ 8.474056603773585 \\ 7.128205591060185 \\ 7.001152932064677 \\ 7.000000094930961 \\ 7.000000000000000 \\ 7.000000000000000 \end{pmatrix} \quad rootx(2) = \begin{pmatrix} 2.000000000000000 \\ 1.500000000000000 \\ 1.416666666666667 \\ 1.414215686274510 \\ 1.414213562374690 \\ 1.414213562373095 \\ 1.414213562373095 \\ 1.414213562373095 \end{pmatrix}$$

$z :=$

z is defined inside the program loop only. Outside the loop the value of z is not available. We have put the values of z in the function rootx which is available outside the loop.

For comparison: $\sqrt{2} = 1.414213562373095$

For larger numbers, our initial guess of 2 is not good enough for us to obtain an accurate answer in only eight iterations. For example, the square root of 3025 is:

$rootx\left(55^2\right)^T = (2 \quad 757.25 \quad 380.622 \quad 194.285 \quad 104.927 \quad 66.878 \quad 56.055 \quad 55.01)$

The transpose operation results in the list being printed horizontally and takes up less space.

Opening new lines in a program loop:

This sample loop creates a matrix M with consecutively numbered elements.

```
M := | for i ∈ 0..1
       for j ∈ 0..2
         M[i,j] ← 3·i + j
     M
```

$$M = \begin{pmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \end{pmatrix}$$

To open a line within in a loop:

Position the blue cursor at the LEFT underlining the whole line and select *add line* from the Programming Taskbar.

```
M := | for i ∈ 0..1
       for j ∈ 0..2
         M[i,j] ← 3·i + j
     M
```

Will give you this:

```
M := | for i ∈ 0..1
       for j ∈ 0..2
         |
         M[i,j] ← 3·i + j
     M
```

To open a line in a loop below a line:

Position the cursor at the right of the line, then select *add line*:

```
M := | for i ∈ 0..1
       for j ∈ 0..2
         M[i,j] ← 3·i + j
     M
```

Will give you this:

```
M := | for i ∈ 0..1
       for j ∈ 0..2
         M[i,j] ← 3·i + j
         |
     M
```

To open a line above or below a loop:

Position the blue cursor at the left of the entire loop to open a line above, and to the right of the entire loop to open a line below. For example, to get a line above:

```
M := | for i ∈ 0..1
       for j ∈ 0..2
         M[i,j] ← 3·i + j
     M
```

Will give you this:

```
M := | for i ∈ 0..1
       |
       for j ∈ 0..2
         M[i,j] ← 3·i + j
     M
```

If a loop contains a loop, you must also select the the loop within:

```
M := | for i ∈ 0..1
       for j ∈ 0..2
         M[i,j] ← 3·i + j
     M
```

Will give you this:

```
M := |
     for i ∈ 0..1
       for j ∈ 0..2
         M[i,j] ← 3·i + j
     M
```

Try it: With the cursor positioned on the **right side** of the loops, show that new lines are opened **after** the loops.

Try it: You cannot open a new line with the cursor in the first line of a loop.

blank page