Lecture 3 Functions and Control Structures

Functions

We had a brief look at creating functions in the previous lecture. These functions will be a necessary part of learning and using the three basic control structures (decisions, counted loops, conditional loops).

What is a function? A function is a group of statements that can be accessed within a MathCAD worksheet. Functions are designed to perform a specific task or tasks repeatedly by giving a name to the series of statements. Users can create their own functions that are accessed in the same way as the built in functions, as many as you need.
It's very important to understand why we use functions and how to define those functions that we need to develop. Consider a basic structure of MathCAD function in definition.

Function Structure

```
FunctionName(InputList) := | "function statements"
                           | "including controls (if, for, while) if needed"
                           | for i ∈ 1.. 10
                           |     "this separate line separates the for loop statements"
                           |     "from the rest of the function"
                           | "output variable or vector"
```

The 'InputList' (dummy arguments) can contain as many variables and vectors as you want to pass into the function. It's good idea to use generic variable names, not the same ones used in the worksheet. In HW#3, you may have named a variable 'vector', but in the input list to a function you might want to call it something else ('in', 'input' ...). The names in the input list are the same ones to be used inside the function. If I use "FunctionName(x) :=" , then x is the name inside the function being used to represent that information from the main worksheet. What comes out of the function is only what is listed on the last line of the function. This can be multiple pieces of information placed into a single vector or array as we will see it in the following examples.

Rules of thumb for creating functions

You can give any commands you wish within a function. Generally speaking, there are no restrictions in terms of *creating and using functions*. However there are some rules of thumb about using functions that you should follow to make life easier.

1) KEEP THE INPUT / OUTPUT IN THE MAIN PROGRAM

=> The main worksheet should control all input and output of information. Functions should perform operations on data that has already been read in the main program and passed to the function through the input list. Functions are miniature programs inside the main program.

2) DO NOT GIVE YOUR FUNCTION THE SAME NAME AS VARIABLES IN YOUR MAIN PROGRAM OR IN THE FUNCTION. DO NOT USE THE NAME OF AN EXISTING FUNCTION.

=> Keep the name of the function separate from the names you choose for variables. For example, I named a function in a homework solution as 'stress'. There should have been no variables in the main program or function called 'stress'. There is also no built-in MathCAD function called 'stress'. How can I quickly check this...? Well, we can go to the "Help" and type the function name in 'index' that we are about to define. We will see whether there are any built-in constant or built-in function already defined for the name that we've just picked up.

The examples in the following sections use control structures written within functions. First, let's learn control structures in Mathcad programming.

How to define a function in Mathcad

A function is a rule which performs certain operations to its arguments and returns a numerical result. To define a function of x and y such as f(x,y)=x^2+2*xy+y^2:

1. Type the function name followed by a left parenthesis.
2. Type a list of arguments separated by commas followed by a right parenthesis.
3. Type ":" to see the definition symbol, ":=" and a placeholder.
4. Type an expression or a string in the placeholder.
5. Make sure any variables you use in right-hand expression are either

- part of the argument list or
- defined beforehand.

If a variable in the right-hand expression doesn't satisfy one of these conditions, you'll see it marked in red as undefined..
When you've defined a function, the function is not evaluated it until you use it later on in the worksheet.

Control Structures

Three ways to control the flow of a program by controlling which commands are executed, how many times, and even how to manipulate executions.
1) Decision (if-statements)
2) Counted loop (for-loops)
3) Conditional loop (while-loops)

1. If Statements - Decision (no loop)

Do this if (condition true) single line command to be executed
Do this other thing otherwise.

if "condition is true"

"statement 1"

"statement 2"

2. For-loop - a counted loop (repeat statements a pre-determined number of times)

for variable ∈ start , next .. end

"statement 1"

"statement 2"

3. While loop - conditional loop (repeat statements as long as condition remains true)

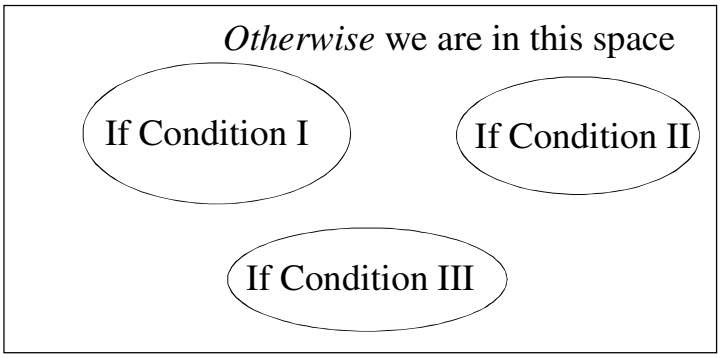
while "condition is true"

"statement 1"

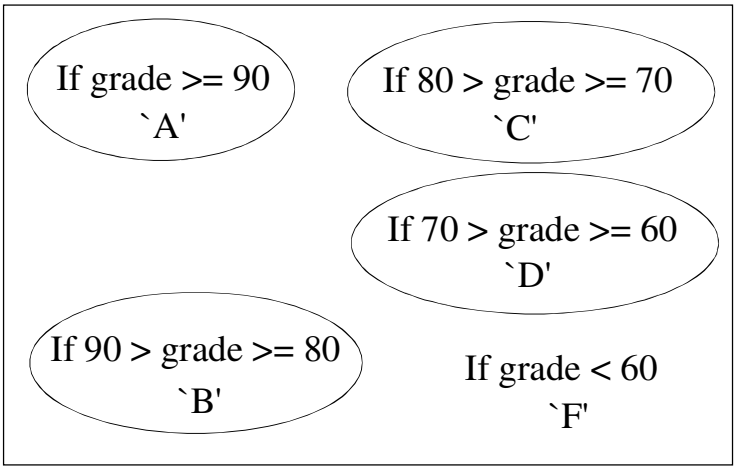
"statement 2"

1. IF-Statements

Venn Diagram of branching



Example #1: Assigning grades



Write MathCAD code to express the above grade classifications
This structure uses 5 separate if - statements with upper and lower limits
These if - statements - in - series do not interface with each other

Example #1: NOTE: Keystroke for IF (})

$$\text{grade}(x) := \begin{array}{l} \text{out} \leftarrow \text{"you get an A"} \quad \text{if } (x \geq 90) \\ \text{out} \leftarrow \text{"you get a B"} \quad \text{if } (x \geq 80) \wedge (x < 90) \\ \text{out} \leftarrow \text{"you get a C"} \quad \text{if } (x \geq 70) \wedge (x < 80) \\ \text{out} \leftarrow \text{"you get a D"} \quad \text{if } (x \geq 60) \wedge (x < 70) \\ \text{out} \leftarrow \text{"you fail"} \quad \text{if } (x < 60) \\ \text{out} \end{array}$$

$\text{grade}(50) = \text{"you fail"}$

$\text{grade}(61) = \text{"you get a D"}$

$\text{grade}(75) = \text{"you get a C"}$

$\text{grade}(88) = \text{"you get a B"}$

$\text{grade}(93) = \text{"you get an A"}$

Each decision structure is evaluated separately

Even if one condition is found true, all other conditions are still checke

Example #2: grades: another way

We can just use a series of if - statements

$$\text{grade}(x) := \left| \begin{array}{l} \text{out} \leftarrow \text{"you get a D"} \quad \text{if } x \geq 60 \\ \text{out} \leftarrow \text{"you get a C"} \quad \text{if } x \geq 70 \\ \text{out} \leftarrow \text{"you get a B"} \quad \text{if } x \geq 80 \\ \text{out} \leftarrow \text{"you get a A"} \quad \text{if } x \geq 90 \\ \text{out} \leftarrow \text{"you fail"} \quad \text{if } x < 60 \\ \text{out} \end{array} \right.$$

$\text{grade}(50) = \text{"you fail"}$

$\text{grade}(61) = \text{"you get a D"}$

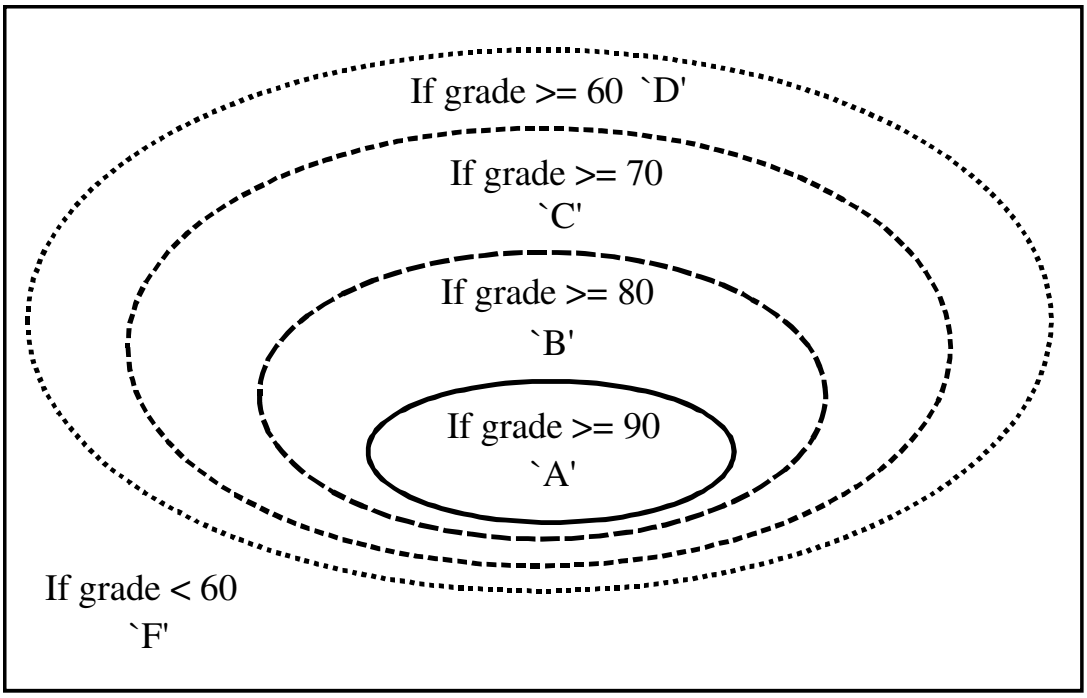
$\text{grade}(75) = \text{"you get a C"}$

$\text{grade}(88) = \text{"you get a B"}$

$\text{grade}(93) = \text{"you get a A"}$

Example #3: Nested if-statements

Now we will NEST the if - statements instead of using them in series. In this case, the next if - statement will only be evaluated if the previous one is true. In this way, we won't overlap assignments. Let's look at the Venn diagram first (on the next page).



As we see, there are 5 different conditions, but they are organized in a more specific way such as:

```
grade(x) := | if x ≥ 60
            |     out ← "you get a D"
            |     if x ≥ 70
            |         out ← "you get a C"
            |         if x ≥ 80
            |             out ← "you get a B"
            |             out ← "you get an A"   if x ≥ 90
            | out ← "you fail"   otherwise
            | out
```

grade(50) = "you fail"

grade(61) = "you get a D"

grade(75) = "you get a C"

grade(88) = "you get a B"

grade(93) = "you get an A"

Each condition is evaluated based on the result of the previous condition.

So if x < 60, none of the IF-statements are seen, and the 'otherwise' action is triggered.

2. For-loops (Ctrl+") : counted loops

Introduction: Look back to the top of page 20, where we used a range variable to create a vector using multiple commands. Suppose we want to create lots of vectors with varying contents. We could issue these same lines of code over and over again for each new vector. An easier way is to create a miniature program (a function) within the worksheet that contains the commands to create a vector. In order to use the commands, we just have to refer to the name we give the function. We will create a function that creates a vector given

the starting and stopping points, and the increment between points. How do we do that? Let's talk about this during the class.

Three Basic elements used to define a for-loop

1. for ▮ ∈ ▮ (From the top menu, Insert>>programming tool bar>>click "for"

2. A range variable that dictates how (e.g., how many times, when it terminates the loop, and what pattern it should follows) MathCAD performs the following command line(s). For example, if we type a range variable "i" and the range [1,5] (a sequence of integer values ranging from 1 to 5 by increment of 1), then we will have:

$$\text{for } i \in 1..5$$

▮

This is exactly same as what we have already been familiar with, i.e., how to create a range variable.

3. Command line(s): let's say, we want to create an array "vector" whose contents are integers of 1,2,3,4, and 5. Then we will have:

$$\text{for } i \in 1..5$$

$$\text{vector}_i \leftarrow i$$

Therefore, *remember* that a for-loop consists of three things: (1) a frame of for-loop (2) range variable(s) and (3) command line(s)

Example #1: Defining a variable that contains the sum of the first 5 integers.

A working MathCAD code	Using a built-in function (just for comparison purpose)	Dead wrong! Then, what's the problem?
$x := \begin{\|} s \leftarrow 0 \\ \text{for } i \in 0..4 \\ \quad s \leftarrow s + i \end{\|}$ $x = 10$	$i := 0..4$ $x := \sum_i i$ $x = 10$	$s := 0$ $x := \text{for } i \in 0..4$ $\quad s \leftarrow s + i$ $x = 4$

Example #2: Defining a function in order to find the sum of the first *n* integers (equivalent to using MathCAD's summation operator).

This code below works properly.	In a more advanced way, we can write:
$\text{sum}(n) := \left\| \begin{array}{l} s \leftarrow 0 \\ \text{for } i \in 0..n \\ \quad s \leftarrow s + i \\ s \end{array} \right.$ $\text{sum}(4) = 10$	$\text{sum}(\text{begin}, \text{end}) := \left\| \begin{array}{l} s \leftarrow 0 \\ \text{for } i \in \text{begin}..\text{end} \\ \quad s \leftarrow s + i \\ s \end{array} \right.$ $\text{sum}(0,4) = 10 \qquad \text{sum}(1,10) = 55$

Example #3: Let's combine the control structures such as "If-statement" and "for-loop" together and see what we can do with them

When we want to sum up the *absolute* values	Can you see what the answer would be?
$\text{sum}(\text{begin}, \text{end}) := \left\| \begin{array}{l} s \leftarrow 0 \\ \text{for } i \in \text{begin}..\text{end} \\ \quad \left\| \begin{array}{l} s \leftarrow s + \lvert i \rvert \quad \text{if } i < 0 \\ s \leftarrow s + i \quad \text{otherwise} \end{array} \right. \\ s \end{array} \right.$ $\text{sum}(-4,4) = 20$	$\text{sum}(\text{begin}, \text{end}) := \left\| \begin{array}{l} s \leftarrow 0 \\ \text{for } i \in \text{begin}..\text{end} \\ \quad \left\| \begin{array}{l} s \leftarrow s + \lvert i \rvert \quad \text{if } i < 0 \\ \text{continue} \quad \text{otherwise} \end{array} \right. \\ s \end{array} \right.$ $\text{sum}(-4,4) =$

Now we'll take the previous example up a notch on the following.

Write a program that processes a vector of grades to count how many students pass and how many fail, and calculate the average overall score

Example #4: Another combination of two control structures such as "If-statement" and "for-loop" with an array.

1. Input phase:
Input: individual student grades

output: number of failing students
number of passing students

average of all grades

2. Design phase:
Map: read the input vector > determine the total number of data that is used to determine the range of a range variable> check whether the component (at the current step) is equal or larger than 60 > add "1" to the counters to keep track of either "pass" or "fail"> sum up the numbers> average them>output

Pseudocode:
1. Enter the vector of student grades
2. Start a for-loop that executes once for each student
    2-1. Decide if grade is passing or failing
        if passing, add one to the passing variable
        if failing, add one to the failing variable
    2.2 Add grade to running total regardless of pass or fail so we can calculate average

3. Go back to top of the loop to get next grade and repeat substeps 2-1 and 2-2 until all the components of the input vector are processed.

4. 'average' is the sum of all grades divided by the total number of grades entered
5. Output the results in a vector form (the total number of "pass", the total number of "fail", and the average of grades)

Let's program this in MathCAD.

$$Grades := \begin{pmatrix} 75 \\ 84 \\ 23 \\ 96 \\ 43 \end{pmatrix}$$

$$class(x) := \left| \begin{array}{l} sum \leftarrow 0 \\ pass \leftarrow 0 \\ fail \leftarrow 0 \\ \text{for } i \in 1..length(x) \\ \quad \left| \begin{array}{l} pass \leftarrow pass + 1 \text{ if } x_i \geq 60 \\ fail \leftarrow fail + 1 \text{ otherwise} \\ sum \leftarrow sum + x_i \end{array} \right. \\ ave \leftarrow \dfrac{sum}{length(x)} \\ out \leftarrow \begin{pmatrix} fail \\ pass \\ ave \end{pmatrix} \end{array} \right.$$

$result := class(Grades)$

$failures := result_1$

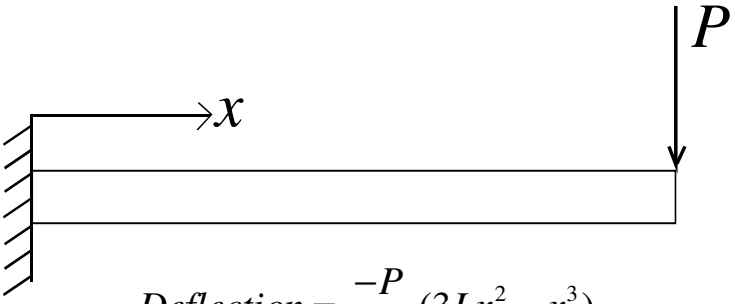
$passes := result_2$

$average := result_3$

$failures = 2 \qquad passes = 3 \qquad average = 64.2$

(Advanced) Example #5: Consider the cantilevered beam illustrated to the below with a variable valued point load at the end. An equation is provided, which describes the beam deflection at any point along the beam.

The program below uses a for-loop to: a) create a vector of x-locations along the length of the beam, b) create a vector with the corresponding deflections, c) Use the vectors to compute the deflections along the beam. NP is the number of points at which we want to calculate the deflection between 0 and the total length L.



$$Deflection = \frac{-P}{6EI}(3Lx^2 - x^3)$$

Solution #1

$I := 600 \quad E := 29000 \quad P := 20 \quad L := 20 \quad NP := 10$

$$\text{BeamDefl}(L, NP, E, I, P) := \left| \begin{array}{l} \text{inc} \leftarrow \dfrac{L}{(NP - 1)} \\ \text{for } i \in 1..NP \\ \quad \left| \begin{array}{l} x_i \leftarrow 0 + (i - 1)\text{inc} \\ \text{defl}_i \leftarrow \dfrac{-P}{6 \cdot E \cdot I} \cdot \left[3 \cdot L \cdot (x_i)^2 - (x_i)^3 \right] \end{array} \right. \\ \begin{pmatrix} x \\ \text{defl} \end{pmatrix} \end{array} \right.$$

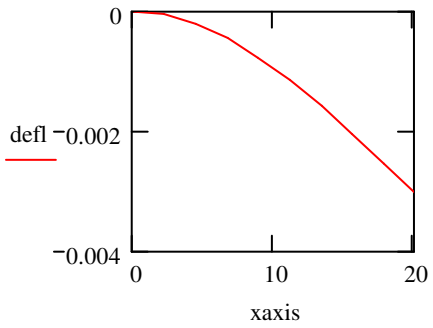
$\text{results} := \text{BeamDefl}(20, 10, E, I, P)$

$$\text{results} = \begin{pmatrix} \{10,1\} \\ \{10,1\} \end{pmatrix}$$

We've set up the output from the function to contain the two vectors:
x-coordinates and deflections at those coordinates.
They are placed as individual elements in a 2x1 vector.
Note that when we display the contents of 'results',
it tells us that each of the two elements are a 10x1 vector.

Thus 'results' is called a 'data structure'.
The difference is that each element in a data structure can be a vector or matrix,
not just a scalar.
These elements are indexed just like a vector (with a subscript).
In order to display and use the contents of the data structure,
we need to assign a name to each of the elements in the data structure.

$\text{xaxis} := \text{results}_1 \qquad \text{defl} := \text{results}_2$



Solution #2

$I := 600 \quad E := 29000 \quad P := 20 \quad L := 20 \quad NP := 10$

$$\text{BeamDefl}(L, NP, E, I, P) := \left| \begin{array}{l} \text{inc} \leftarrow \dfrac{L}{(NP - 1)} \\ i \leftarrow 1 \\ \text{for } x \in 0, \text{inc} .. L \\ \quad \left| \begin{array}{l} \text{xaxis}_i \leftarrow x \\ \text{defl}_i \leftarrow \dfrac{-P}{6 \cdot E \cdot I} \cdot \left(3 \cdot L \cdot x^2 - x^3\right) \\ i \leftarrow i + 1 \end{array} \right. \\ \begin{pmatrix} \text{xaxis} \\ \text{defl} \end{pmatrix} \end{array} \right.$$

$\text{results} := \text{BeamDefl}(20, 10, E, I, P)$

$$\text{results} = \begin{pmatrix} \{10,1\} \\ \{10,1\} \end{pmatrix}$$

What difference do you see between Solution #1 and Solution #2?

Would the output be same?

3. While – loop; conditional loop (decision and loop)

A loop is a program statement that causes one or more statements (the body of the loop) to execute repeatedly until a particular condition occurs. There are two kinds of loops:

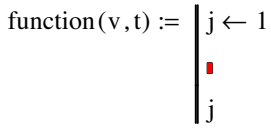
"For" loops are useful when you know exactly how many times the body of the loop should execute.

"While" loops are useful when you want to stop execution upon the occurrence of a condition but you don't know exactly when that condition will occur.

Therefore, "While" loops are useful when you want to stop execution upon occurrence of a condition and you don't know exactly when that condition will occur. If you know exactly how many iterations you want, use a "for" loop instead.

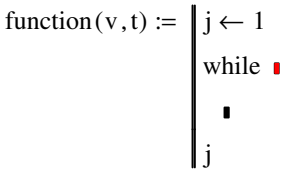
1. To create a "While" loop;

A. Click in the placeholder into which you want to place the "while" loop.

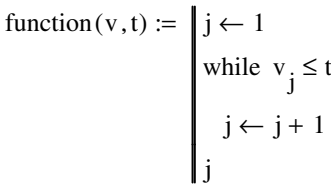


B. Click on the Math toolbar to open the Programming toolbar containing the programming operators.

C. Click on the "while" button or press **Ctrl+]**. Do *not* just type the word "while":



D. In the placeholder to the right of the "while", type a Boolean expression. Click the "Add Line" button on the Programming toolbar to insert placeholders for additional statements if necessary.



2. How it works;

$$x := \begin{pmatrix} 1 \\ 3 \\ 5 \\ 7 \end{pmatrix}$$

Let's consider a vector "x" given above. When we invoke the function "function" with the vector "x" and a number '4'; "function(x,4) = ", what will be the returning value?

3. What would be equivalent to a "while" loop?

$$\text{function02}(v,t) := \left| \begin{array}{l} j \leftarrow 1 \\ \text{for } i \in 1..\text{length}(v) \\ \quad \left| \begin{array}{l} j \leftarrow j + 1 \text{ if } v_i \le t \\ \text{break otherwise} \end{array} \right. \\ j \end{array} \right.$$

$$\text{function02}(x,4) = 3$$

4. What's new in the above code? -- Using a "break" statement in a loop whenever you want to halt execution in a loop.

   When MathCAD encounters a "break" statement in the body of a "for" or "while" loop:

   A. The loop ceases execution and return the most recent value computed.
   B. The program execution then continues with the next line of the program after the loop.

5. However, we actually want a function that computes a total number of components of a vector, which is equal and less than a user's input number ("t"; the second dummy argument in the list). The function "function" always returns a value that is one larger than the actual number of the components, which are equal to or smaller than "t". So, we can use another variable "counter" to keep track of the actual number:

```
function03(v,t) := | counter ← 0
                   | j ← 1
                   | while v_j ≤ t
                   |     | counter ← counter + 1
                   |     | j ← j + 1
                   | counter
```

function03(x,4) = 2

6. What will happen if we invoke the function "function03" with 't=8'; function03(x,8) = ? We will get an error message, saying "Value of subscript or superscript is too big (or too small) for this array" Why? It will be explained during the class.

7. The solution for the problem in Item #6 would be:

```
function04(v,t) := | counter ← 0
                   | j ← 1
                   | while v_j ≤ t
                   |     | counter ← counter + 1
                   |     | break if j = length(v)
                   |     | j ← j + 1
                   | counter
```

function04(x,8) = 4

Using "Break" statement with a "If-statement" will provide a condition to MathCAD that terminates the loop before "j" becomes larger than the total number of components inside a vector.

Use a "while" loop whenever you want a set of statements to keep executing until a condition is met. Make sure you have a statement somewhere that makes the condition false. Otherwise the loop will execute indefinitely and you will need to stop it by pressing **Esc**. This loop continues to repeat as long as the condition(s) is (are) true. Statements inside loop must be able to change variable(s) used in the condition.

Example #1. Locating the first occurrence of a specific element in a vector.

We have a long vector of numbers, and there is only one value less than zero in that vector. We want to find where in that list the negative value is.

```
Find_Negative(vec) := | stop ← 1
                      | location ← 1
                      | while stop = 1
                      |     if vec_location < 0
                      |         out_location ← location
                      |         stop ← 0
                      |     location ← location + 1
                      | out_location
```

$$list := \begin{pmatrix} 9 \\ 7 \\ 3 \\ 90 \\ -3 \\ 12 \end{pmatrix}$$

result := Find_Negative(list)

result = 5

Note: notice the condition in the while statement above. We are asking if 'stop' is equal to 1 each time through the loop. If that condition is true, the loop is executed again. We are not assigning 1 to the variable 'stop'; we are comparing the existing variable 'stop' to a value. That equal sign is in bold, which means it's a comparison (Boolean Operator). We get that one by using Ctrl =

'location' is what we are using to index the vector. It has to be integer values, and is set up to be 1 first time through the loop, then 2, etc. so that in the if statement we are sequentially comparing the numbers in the vector vs. 0 one at a time. When we find the negative value, the value of 'location' is pointing to the place where the negative value resides in the vector, so we save it into 'out_location', which is listed at the bottom of the function as the variable that is output from the function. We also re-assign 'stop' to 0, which will cause the while loop to stop executing (it only continues if stop = 1). Thus our algorithm will keep looking until it finds a negative value, save the location of that negative value, and then exit the loop, and we're done.

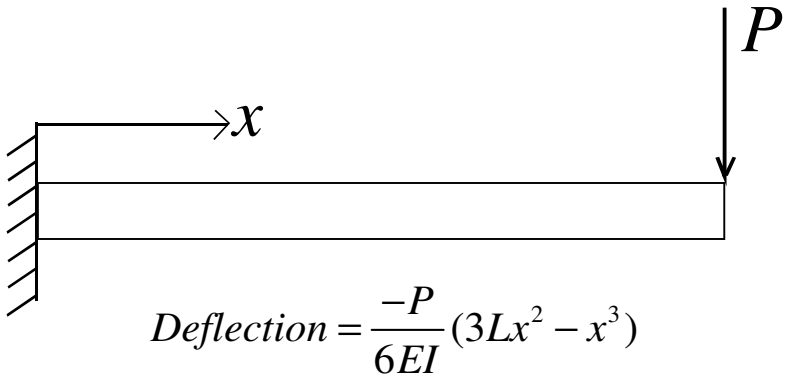
Example #2. We'll change the previous example of calculating the deflection of a cantilevered beam so that it is now a design problem. Let's say that following parameters are fixed (non-negotiable values): I, NP, P, E
And the length of the beam L is to be designed such that the maximum deflection is close to 0.06 inches
Obviously, the longer the beam is, the greater the tip deflection will be. We will iteratively increase the length of the beam until the deflection is greater than 0.06 inches.

What we will do is calculate the tip (maximum) deflection starting with the length of 20 inches. If the tip deflection is less than 0.06 inches, we will increase the length of the beam by a factor of 5 % of its current value. This will be done as many times as necessary until the tip deflection exceeds 0.06 inches.

Pseudocode:
Receive the input (length of beam, number of points along the beam).
Calculate the maximum deflection at the tip of the beam

if the deflection magnitude is less than 0.06 inches, increase length by 5% and recalculate the tip deflection and repeat using a while loop

Now that length has been chosen so that tip deflection exceeds 0.06 in, calculate the increment
Start a loop to calculate the x-axis values and deflection at these locations
Place the output into a data structure.



$$Deflection = \frac{-P}{6EI}(3Lx^2 - x^3)$$

What is the length of the beam for the maximum deflection = 0.06 inches?

$kips := 1000lbf \qquad ksi := 1\frac{kips}{in^2}$

$E := 29000ksi \qquad P := -20kips \qquad I := 50in^4 \qquad L := 20in \qquad NP := 10$

$BeamDefl(L, NP, P, E, I) :=$

$maxdefl \leftarrow \frac{-P \cdot L^3}{3 \cdot E \cdot I}$

while $|maxdefl| < 0.06\,in$

$\quad L \leftarrow L \cdot 1.05$

$\quad maxdefl \leftarrow \frac{-P \cdot L^3}{3 \cdot E \cdot I}$

$i \leftarrow 1$

$inc \leftarrow \frac{L}{(NP - 1)}$

for $x \in 0, inc .. L$

$\quad xaxis_i \leftarrow x$

$\quad defl_i \leftarrow \frac{-P}{6 \cdot E \cdot I} \cdot \left(3 \cdot L \cdot x^2 - x^3\right)$

$\quad i \leftarrow i + 1$

$\begin{pmatrix} xaxis \\ defl \end{pmatrix}$

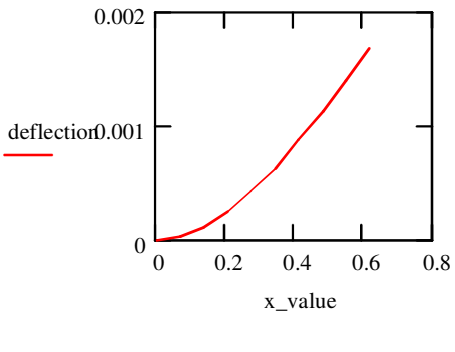
This "While" control structure is designed to adjust 'L' (length of the beam) until 'maxdefl' becomes less than 0.06 in. (Constraint I impose on 'L')

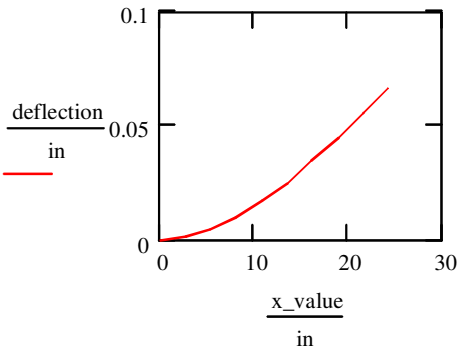
Since maxdefl has a unit of length, compare 'maxdefl' to other numbers that have a unit of length.
NOTE: Consistency!!!

The "for" control structure uses the new 'L' value and computes deflection along the whole beam.

How many times?
Well, in this case, 'x' has an increment 'inc' that contains real number. Therefore, I need to create an index to locate values of 'x' and 'defl'.
As shown in two lines above, I assigned one to a variable 'i'. 'i' is being used as the indexing variable right here.

$\begin{pmatrix} x\_value \\ deflection \end{pmatrix} := BeamDefl(L, NP, P, E, I)$





To plot vectors (or 1-D arrays) having units that are different from default units, you need to divide the vectors by the unit you want. Otherwise, Mathcad returns a plot with a default unit (The left plot above)