**MATLAB STRINGS AND MORE PROGRAMMING**

* Strings, for example text strings, are strings of characters surrounded by single quotes, e.g.,

s='hello ', t='there you all'

They are really vectors but not as you have used so far. Try, for example,

size(s)

and you will get

ans = 1 7 % note the two spaces after 'o'

just like a vector.

* Even a number can be treated as a string, e.g.,

x='12753'

sets *x* equal to a string of digits which is now not a number.

x(1:3)

abs(x) % returns the ascii value of each entry of the string

Now try, for example,

1+x

and you will get a strange answer which is '1' plus the ascii code of each entry in the string.

* On the other hand, try

sort(x)

and the characters will be sorted to give *12357* and

fliplr(x)

These can be very useful in programming.

* Strings can be put together just like other vectors can (recall that we have defined the strings *s* and *t* above):

u=[s,t,' come back and see us, hear']

would build the string *u= Hello there you all come back and see us, hear*

* Suyppose you want to find the number of *a*'s in the string *u*
* num=findstr('a',u) %K=findstr(S1,S2) finds the locations of
* % occurrences of string S1 in string S2
* length(num) % This will give the number of occurrences of a in u
* There are useful commands in matlab for converting strings to numbers and numbers to strings and also evaluating strings that might contain matlab commands:
  + The statement

n=17

defines *n* to be the integer *17* and

ns=int2str(n)

converts this integer to the string *17*.

* + The statement

x=17.45

defines *x* to be the floating point number *17.45* and

xs=num2str(x)

converts this number to the string *17.45* which, remember, is now a string of characters, six of them to be exact.

* To convert a string to a number or to simply evaluate a string (one that makes sense to evaluate) use the *eval* command.

eval([ns,'+',xs])

or

str2num(ns)+str2num(xs)

would give the number *34.45* which is now a real floating point number.

* Here is another use of eval and strings: suppose you wanted to build a set of values *x1, x2 ,..., xn* with values *1^2, 2^2, 3^2, ... , n^2* (note, I am not saying *x(1)* and *x(2)*, etc. -- In other words I am not building a vector). You could do this as follows:
* n=input(' input a positive integer, n = ')
* for i=1:n
* eval(['x',int2str(i),' = ',int2str(i^2)])

end

* Strings are used extensively in labeling graphics, e.g.,
* a=3
* b=2.5
* c=1
* x=0:.05:4;
* plot(x,a\*sin(b\*x+c),2\*(pi-1)/5,0,'o')
* grid
* xlabel('x-axis')
* ylabel('y-axis')
* title(['a = ',int2str(a),' b = ',num2str(b),' c = ',int2str(c)])
* text(2\*(pi-1)/5+.1,.1,'first zero')
* Note that in using the input statement the value you ask for can be a matrix as well as a number. So, in particular, it can be a vector, e.g.,

v=input(' input a vector , v = ')

When you execute this you might do the following:

input a vector , v = [1 2 3 4]

then you will get *v=[1 2 3 4]*

* It is also possible to input strings. Here the syntax is slightly different:

input(' some test ', 's')

Note the additional comma and *'s'* to indicate that the input is to be considered as a string of characters.

* As an Suppose in a program you want to have a choice of whether to proceed *Y* or stop *N* and you want the default to be *Y* which is returned if you don't choose *N* and don't want to enter the default.
* i=input('Do you want to continue, Y/N [Y]:','s')
* if isempty(i) % the isempty command returns 1 (true) if i is not empty
* % and 0 if it is
* i='Y';

end

* Take any three digit number, with at least two distinct digits, and form the largest integer and smallest integer by sorting the digits in desending and ascending order. Then substract to obtain a new three digit integer. Repeat this procedure and eventually you will arrive at *495*. Here is a Matlab program that tests this for the input integer *335*. Try a few others and then modify the program so that you can run it by inputing a number without changing the program.
* count=0;
* s='335'
* while str2num(s) ~= 495
* count=count+1;
* m=sort(s);
* M=fliplr(m);
* d=eval([M,'-',m]);
* ds=int2str(d);
* lds=length(ds);
* if lds < 3
* ds =[ds,'0'\*ones(1,3-lds)];
* end
* s=ds
* count
* end
* Exercise: Write a program to input an arbitrary string of *N* digits (i.e., an integer of length *N*) and then carry out the algorithm described in the last example. Note for *N* equal *4* limit is always *6174* but for higher *N* you must be more cleaver in the test because there can be limit points and limit cycles, Hint: Build a vector that keeps track of all the previous answers and tests, at each step, to see whether a value has repeated. You might want to use "any" or "all". In case of a limit cycle print out the whole cycle of values.
* Note the various uses of strings in the next example. This example also gives you a brief introduction to the polynomial features of Matlab. Note a polynomial in Matlab in just the vector of coeficients of the polynomial.
* p=[1 -6 11 -6];
* x=0:.25:4; rand('normal'); % a string is used to declare normal distribution
* y=polyval(p,x)+rand(size(x));
* c=polyfit(x,y,3) % interpolates a cubic poly to the data x, y
* % returns the 4 coefficients of the poly.
* fit=polyval(c,x);
* plot(x,fit,x,y,'o'); % a string is used to put circles at the data points
* Here is some more polynomial stuff clear a=[1 2 3;4 5 6;7 8 0] p=poly(a) % returns the coefficients of characteristic poly of a. r=roots(p) %gives the eigenvalues p2=poly(r) % this gives a polynomial with r as roots so should give p.
* Here is another example and an exercise.
* a=poly(1:5) % the a Wilkinson polynomial of degree 5, i.e., has roots 1,2,3,4,5

r=roots(a)

Now write a for that builds Wilkinson polynomials p1 p2 p3 p4 p5 p6 p7 p8 p9. Then plot the polynomials (showing their zeros and then saves the polynomials in files named fn1 fn2 fn3, etc. (hint: Suppose that

j=4

eval(['h',int2str(j),'=figure'])

eval(['p',int2str(j),' = poly(1:j)'])

eval(['x=.95:.05:(',int2str(j),'+.05);'])

eval(['plot(x,polyval(p',int2str(j),',x))'])

title(['Wilkinson polynomial of degree ',int2str(j)])

grid

eval(['save fn',int2str(j),' p',int2str(j),' -ascii'])

* Many times it is useful to display numbers in a different format. Strings also enter in this arena. All variables in matlab are stored in double precision floating point format. The way matlab displays these numbers can be set many ways.
* format short % displays in 5-digit fixed point notation.
* format short e % displays in 5-digit floating point notation.
* format long % displays in 15-digit fixed point notation.
* format long e % displays in 15-digit floating point notation.

% hex displays in hexadecimal notation.

* For more detailed formating of numbers you can use *fprintf* which gives much more control than int2str and num2str: Try the following
* fprintf('%.0e\n',pi)
* fprintf('%.1e\n',pi)
* fprintf('%.3e\n',pi)
* fprintf('%.5e\n',pi)
* fprintf('%.10e\n',pi)
* fprintf('%.0f\n',pi)
* fprintf('%.1f\n',pi)
* fprintf('%.3f\n',pi)
* fprintf('%.5f\n',pi)
* fprintf('%.10f\n',pi)
* fprintf('%.0g\n',pi)
* fprintf('%.1g\n',pi)
* fprintf('%.3g\n',pi)
* fprintf('%.5g\n',pi)
* fprintf('%.10g\n',pi)
* fprintf('%8.0g\n',pi)
* fprintf('%8.1g\n',pi)
* fprintf('%8.3g\n',pi)
* fprintf('%8.5g\n',pi)
* fprintf('%8.10g\n',pi)
* Exercise: Build a table of the numbers *n*, *log(n)/n* and *10^n\*exp(n)* with *n=1 2 3 4 5 6 7 8 9 10*. If you just do it without thought the output looks terrible. Try using fprintf to get a better look.
* clc
* disp([' ']')
* disp(['n log(n)/n 10^n\*exp(n)'])
* disp('------------------------------')
* disp([' ']' )
* fprintf('%.4g %.5f %.8g\n',8,log(8)/8,10^8\*exp(8))

[RETURN TO MAIN DOC](http://texas.math.ttu.edu/%7Egilliam/mlhtml_1/m4330_ml_1.html)