Using functions as arguments

Your assignment this week is to write a routine (actually, several functions) which finds the root of a function. For example, suppose the function is

\[ y = \sqrt{x} - 3.8 \]

You will need to write a Scilab routine which evaluates this function repeatedly, and takes some action based on the results. But how do to that?

One way would be to **hardwire** the function into your routine, like this:

```scilab
// set the bounds of the interval
starting_x = 0;
ending_x  = 10;

// now evaluate the function in the middle of the interval
mid_x = (starting_x + ending_x) / 2;
y = sqrt(mid_x) - 3.8;

// take the appropriate action ...
```

But such a routine will work ONLY for the function `sqrt(x) - 3.8`. If you need to find the roots of a different function, like `cos(x) - sqrt(x)`, then you need to edit your routine and change the line at which `y` is calculated.

Editing source code every time you need to run a routine in a slightly different way is poor programming style. It’s much better to find a way to leave the source code as-is, and act upon some input supplied by the user.

So, it would be nice if we could just call an arbitrary function in the root-finding routine, like this:

```scilab
// set the bounds of the interval
starting_x = 0;
ending_x  = 10;
```

/ now evaluate the function in the middle of the interval
mid_x = (starting_x + ending_x) / 2;
y = call_the_function_I_want(mid_x);

// take the appropriate action ...

So, in order to find the root of \( \sqrt{x} - 3.8 \), one would somehow write

\[
call\_the\_function\_I\_want(x) \quad \text{means} \quad \sqrt{x} - 3.8
\]

and to find the root of \( \cos(x) - \sqrt{x} \), somehow tell Scilab that now

\[
call\_the\_function\_I\_want(x) \quad \text{means} \quad \cos(x) - \sqrt{x}
\]

There is a way to do this! The built-in Scilab routine \texttt{feval} will call a function for you, and it will even supply the appropriate arguments to the function. All you have to do is to provide it with

- the first argument*: the values to be passed to that function
- the second argument: the name of a function to evaluate

* Actually, the \texttt{feval} function may take either one or two initial numerical arguments, before the name of the function to evaluate. See the on-line documentation.

It will do all the work, and then return to you the results.

---

Example of using \texttt{feval}

Suppose we know we'll have to find the roots of three different functions:

1. \texttt{quadratic\_function} involves a quadratic equation
2. function \( y = \text{quadratic\_function}(x) \)
3.
4. \( y = x^2 - 4x + 3; \)
5.

6. \texttt{cubic\_function} involves a cubic equation
7. function \( y = \text{cubic\_function}(x) \)
8.
9. \( y = 12x^3 + x^2 - 4x + 3; \)
10.
11. `trig_function` involves a trigonometric equation
12. function `y = trig_function(x)`
13.
14. `y = sin(x)*cos(x) - sin(x)*sin(x);`
15.

We create a general root-finding program -- perhaps it uses the bisection method. We design this program so that it takes 3 arguments: the starting and ending points of the interval on which to seek a root, and the name of the function whose root we're seeking.

```
function y = root_finder(starting_x, ending_x, routine_to_root)
```

Inside this program is a loop, which iterates to get closer and closer to the true root. It contains a section which looks something like this:

```
// find the middle of the interval
mid_x = (starting_x + ending_x) / 2;

// and evaluate the function at that point
y = feval(mid_x, routine_to_root);

// take the appropriate action ...
```

Now, suppose we are interested in the interval from 0 to 25. To look for a root of the quadratic function, we would type into the MATLAB command window

```
>> root_finder(0, 25, quadratic_function)
```

To find the root of the cubic function,

```
>> root_finder(0, 25, cubic_function)
```

To find the root of the trigonometric function,

```
>> root_finder(0, 25, trig_function)
```
Unfortunately, in Scilab, we can’t use this facility to call built-in functions such as `sqrt` or `sin`. Rats. In MATLAB, on the other hand, the `feval` function will do this job as well.

If you want some practice ...

1. write a function called `square` which takes one argument, calculates the square of the argument, and returns it.
2. write another function called `cube` which takes one argument, calculates the cube of the argument, and returns it.
3. write a function called `plusone` which takes one argument, calculates the sum of the argument plus one, and returns it.
4. now, write a function called `calculate`, which takes two arguments:
   - the first is the name of a function
   - the second is a value at which to evaluate the function

You should be able to call `calculate` like this:

   calculate(cube, 5.5)
   calculate(square, 10)
   calculate(plusone, -33.3)

What is the result of each?

Here are my versions of these programs:

- `square.sci`
- `cube.sci`
- `plusone.sci`
- `calculate.sci`

Source: http://spiff.rit.edu/classes/phys317/lectures/funcs_as_args.html