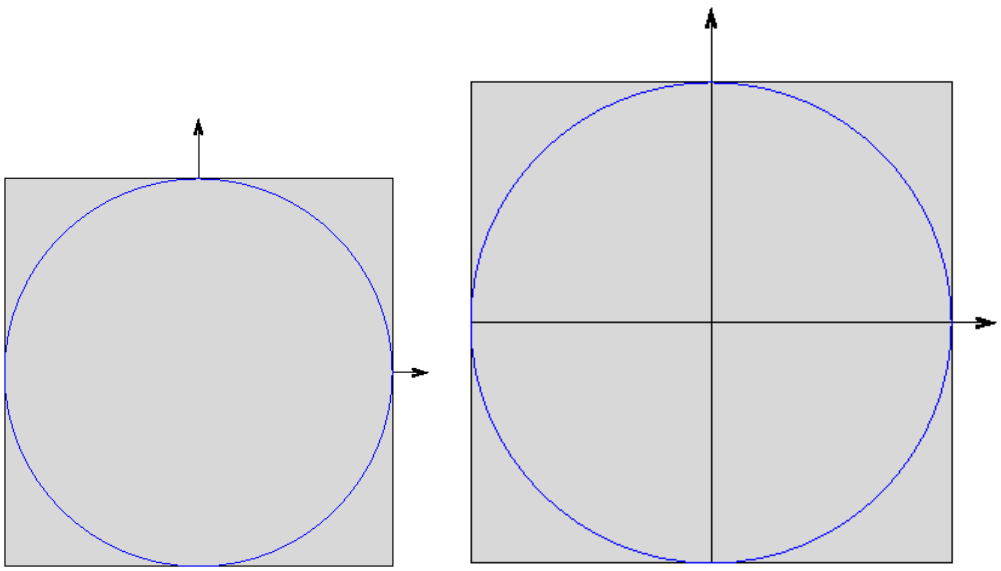
Placing multiple functions in a single source code file

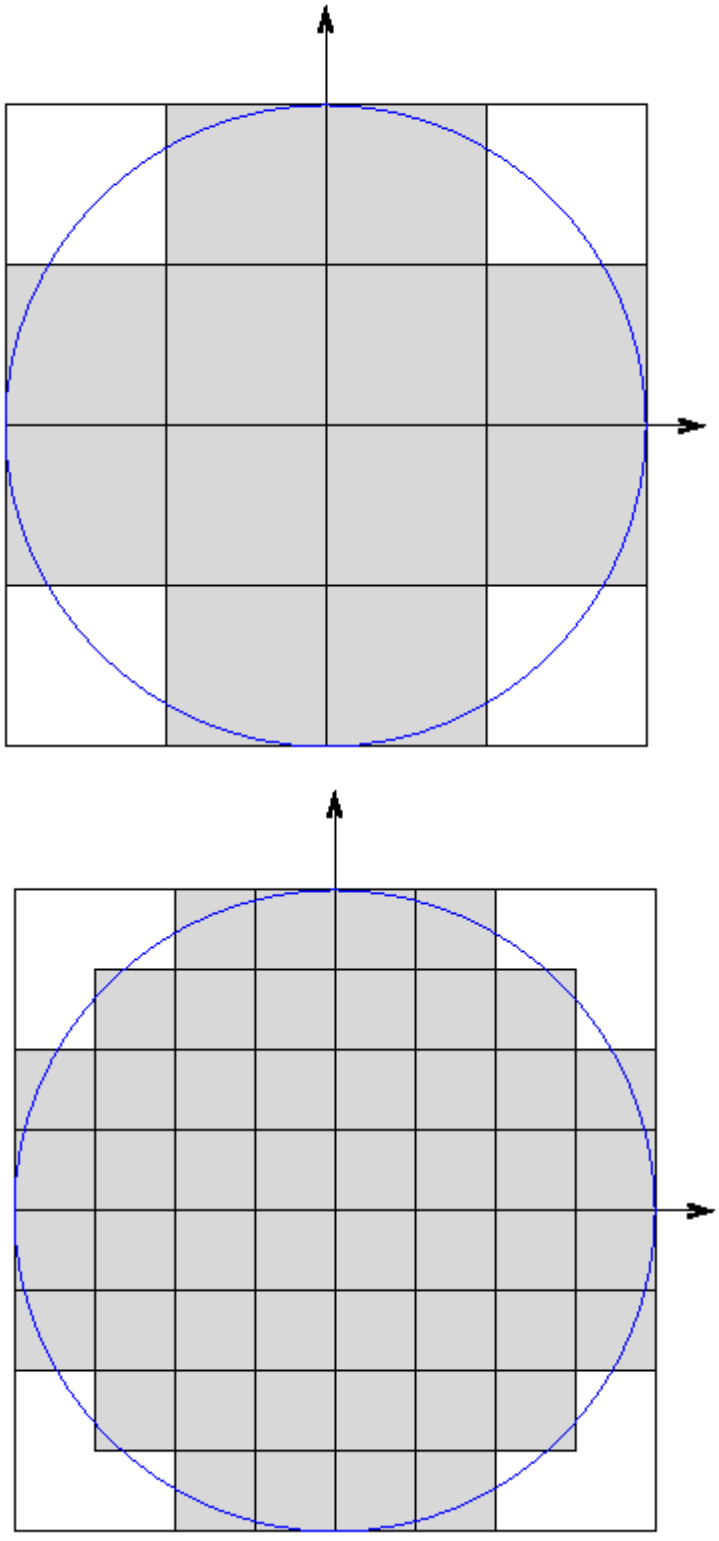
When programs grow more than 20 or 30 lines in size, it often helps to break them up into pieces (sometimes called subroutines, functions, or modules). It helps to simplify the design of the code, and makes it much easier to read, too.

As an example, let's consider a program which calculates the value of pi in a silly way: it uses square blocks to approximate the area of a circle, then finds pi from the area:

First and second approximations:



Third and fourth approximations:



Here's the Scilab source code, all in a single file, with no subroutines.

- silly_pi.sci -- no subroutines

Now, notice this complicated block inside the main loop of the program (I've removed some of the comments and debug statements for clarity):

```
 box_length = (2.0*radius)/num_boxes;
 box_rad = box_length*0.5;
 box_area = box_length*box_length;

 area_sum = 0;
 for (xi = 1 : num_boxes)
  xc = -1 + box_rad + box_length*(xi - 1);

  for (yi = 1 : num_boxes)
   yc = -1 + box_rad + box_length*(yi - 1);

   dist = sqrt((xc*xc) + (yc*yc));
   if (dist < radius)
    area_sum = area_sum + box_area;
   end

  end
 end
```

This block of code calculates the area inside all the boxes which fall within the radius of the circle. The only things this block needs to know are the radius of the circle, and the number of blocks in each direction. So, we could make a routine out of it:

```
function area_sum = area_inside(radius, num_boxes)
//
// Calculate the area inside all boxes which fall inside the a circle
//   of the given radius.
//
 box_length = (2.0*radius)/num_boxes;
 box_rad = box_length*0.5;
 box_area = box_length*box_length;

 area_sum = 0;
 for (xi = 1 : num_boxes)
```

```
    xc = -1 + box_rad + box_length*(xi - 1);

    for (yi = 1 : num_boxes)
      yc = -1 + box_rad + box_length*(yi - 1);

      dist = sqrt((xc*xc) + (yc*yc));
      if (dist < radius)
        area_sum = area_sum + box_area;
      end

    end
  end

endfunction
```

The main loop now looks much simpler:

```
for (iter = 1 : num_iters)

  area_sum = area_inside(radius, num_boxes);

  // check to see how much the sum has changed in the past iteration
  fractional_change = abs((area_sum - old_sum)/area_sum);

  // print out a nice line summarizing this iteration's result
  mypi = area_sum/(radius*radius);
  mprintf('  iter %5d  boxes %8d  pi %10.6f  frac_change %9.4e \n', ...
            iter, num_boxes*num_boxes, mypi, fractional_change);

  // prepare for next iteration
  old_sum = area_sum;
  num_boxes = num_boxes*2;
  iter = iter + 1;

end
```

But how do we arrange this source code?

Option 1: Put subroutines in separate source code files.

We could put the main program in one source code file all by itself, and the subroutine in a second file.

- silly_pi_main.sci -- main program only
- area_inside.sci -- subroutine only

Option 2: Put subroutines into the same source code file as main program

Another option is to put the subroutines into the same source code file as the main program. We simply add lines after the end of the main program; the first line must be a function declaration, like so:

```
  (main program is here ...)

// now calculate pi
mypi = area_sum/(radius*radius);

// end of main program
endfunction

function area_sum = area_inside(radius, num_boxes)
//
// Calculate the area inside all boxes which fall inside the a circle
//   of the given radius.
//
 box_length = (2.0*radius)/num_boxes;

   (etc.)
```

- silly_pi_2.sci -- main program plus subroutine, all in one file

This is probably the better choice, in most cases. All the code needed is in a single file, so it's easy to transfer from one place to another. But we still gain the simplicity of breaking a big program up into separate pieces.

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