Control Flow

Scientific Computing
Fall, 2019
Paul Gribble

Here we will learn above several ways to specify the flow of information as your code gets executed. We will learn about *loops*, which are constructs that allow you to repeat blocks of code multiple times, typically while changing the values of variables inside the repeating block. We will learn about *conditionals*, which allow you to execute different branches of code depending on the values of variables. We will see how to *pause* your code, and to *break* out of a code block.

1 Loops

Loops are used when you have a chunk of code that you need to repeat over and over again, each time changing one (or more) parameters. Here is a simple example for the purposes of demonstration. Let's say you want to load data from 5 files, named `data1.txt`, `data2.txt`, ..., `data5.txt`. Let's say each file contains a one-dimensional array of 10 values. Let's say you want to take the average of each data file and then report the overall mean and overall variance of those 5 values. Here's one way to do it:

```
d1 = load('data1.txt');
d1m = mean(d1);
d2 = load('data2.txt');
d2m = mean(d2);
d3 = load('data3.txt');
d3m = mean(d3);
d4 = load('data4.txt');
d4m = mean(d4);
d5 = load('data5.txt');
d5m = mean(d5);
%
% report overall mean and overall variance of 5 data file means
alldata = [d1m d2m d3m d4m d5m];
datamean = mean(alldata);
datavar = var(alldata);
disp(sprintf('mean=%.3f and variance=%.3f', datamean, datavar))
```

You can see that there is a lot of repetition in this code. What if we had to load data from 1000 data files? There would be a lot of copying and pasting of code chunks. This is error prone and inefficient. Instead let's use a *for loop*. A for loop allows you to repeat a block of code some predetermined number of times, and includes a counter so that you know which iteration of the loop is currently running. Here is what the code above would look like if we used a for loop:

```
nfiles = 5;
alldata = ones(1,nfiles)*NaN; % pre-allocate array and fill with NaN
for i=1:nfiles
    d = load(['data',num2str(i),'.txt']);
    alldata(i) = mean(d);
end
datamean = mean(alldata);
datavar = var(alldata);
disp(sprintf('mean=%.3f and variance=%.3f', datamean, datavar))
```

Now all we would need to change if we have 1000 data files (or one million) is the value of our variable `nfiles=1000;` or `nfiles=1e6;`—nothing else in the code would have to change. This makes our code much more resilient against programming errors.

You can see a for loop begins with the keyword `for` followed by a name of a variable (your choice) that will keep track of which iteration of the loop is currently running. Then the equal sign = followed by a list of values to be iterated through. This list can be a constructed list using the colon operator (as in the example above) or it can be a variable such as an array containing several values as in the example below. Next is the block of code to be repeated. The end of this code block is denoted by the `end` keyword.

```
x = 1:3:15;
for i=x
    disp(sprintf('i=%d', i))
end
```

which prints out:

```
i=1
i=4
i=7
i=10
i=13
```

For loops are executed in a serial fashion, one repetition after another. When we talk later about parallel programming we will see that one can pretty easily *parallelize* a for loop in MATLAB so that different iterations are distributed over multiple cores of a CPU (or indeed over multiple CPUs in different machines over a network).

There is a second sort of loop called a *while loop*. This kind of loop is typically used when the number of iterations is not known in advance. A while loop keeps repeating until the value of a logical expression changes from TRUE to FALSE (changes from 1 to 0). As a little demo, here is an example of a while loop that prints out successive integers starting from 1, until they exceed a critical value, in this case 10:

```
i=1;
while (i<9)
    disp(sprintf('i=%d', i))
    i=i+1;
end
```

which displays:

```
i=1
```

```
i=2
i=3
i=4
i=5
i=6
i=7
i=8
```

The expression that determines whether a while loop will continue repeating can be any valid MATLAB expression that evaluates to 0 (FALSE) or 1 (TRUE). In fact, in MATLAB 0 is FALSE and any other value is considered to be TRUE.

For both for loops and while loops, there are two keywords to know about that can break you out of a loop (`break`) and can move you to the next iteration of the loop (`continue`). I tend not to use these, but you can see the MathWorks online help to read more about them:

break

continue

2 Conditionals

So far we have seen how to use loops to repeat sections of code over and over again as needed. The other major control flow mechanism in high-level languages such as MATLAB is the *conditional*, which allows you to specify how code branches in one direction or another depending on some logical condition.

So for example let's say you ask the user to enter a number, and if the number is even you output "The number is even." and if the number is odd you output "The number is odd.":

```
x = input('Enter a number: ');
if (mod(x,2)==0)
    disp('The number is even.');
elseif (mod(x,2)~=0)
    disp('The number is odd.');
end
```

which produces:

```
Enter a number: 15
The number is odd.
```

and

```
Enter a number: 12
The number is even.
```

In general one can string together any number of `elseif` branches (including none of them). For example:

```
x = input('Enter a number: ');
if (x<10)
    disp('The number is less than 10');
elseif (x<20)
```

```
    disp('The number is less than 20');
elseif (x<30)
    disp('The number is less than 30');
else
    disp('I don''t have anything to say');
end
```

You don't need any `else` statements at all if is suits your needs:

```
x = input('Enter a number: ');
if (x<0)
    disp('That is a negative number');
end
```

The MathWorks online documentation has a page on conditional statements here:

Conditional Statements

3 Switch statements

The other type of conditional you might come across (though I rarely use them) is called a *switch statement*. Typically these are used when there is some relatively large list of potential cases and for each you have a defined (and different) course of action. Here is an example adapted from the MathWorks help page on conditionals:

```
d = input('Enter a day of the week: ','s');
switch d
    case 'Monday'
        disp('First day of the week')
    case 'Tuesday'
        disp('Day 2')
    case 'Wednesday'
        disp('Day 3')
    case 'Thursday'
        disp('Day 4')
    case 'Friday'
        disp('Last day of the work week')
    otherwise
        disp('Weekend!')
end
```

Side note: can you spot any potential problem(s) with the above code?

4 Pause, break, continue, return

There are some keywords in MATLAB that give you finer control over the flow of a program.

The `pause` keyword by itself will simply cause MATLAB to stop at that point in the code, and wait until the user strikes any key—then MATLAB will continue. Try this:

```
for i=1:10
```

```
    if (i==5)
        disp('i is 5! hit any key to continue');
        pause
    else
        disp(sprintf('i is not 5, continuing... (i is %d)', i));
    end
end
```

which produces:

```
i is not 5, continuing... (i is 1)
i is not 5, continuing... (i is 2)
i is not 5, continuing... (i is 3)
i is not 5, continuing... (i is 4)
i is 5! hit any key to continue
i is not 5, continuing... (i is 6)
i is not 5, continuing... (i is 7)
i is not 5, continuing... (i is 8)
i is not 5, continuing... (i is 9)
i is not 5, continuing... (i is 10)
```

The `break` keyword in MATLAB will terminate the execution of a loop. Any code appearing after the `break` keyword will not be executed. In nested loops the `break` keyword only exits from the loop in which it appears. Here is an example of how `break` works:

```
for i=1:10
    if (i==5)
        disp('i is 5! stopping...');
        break
    else
        disp(sprintf('i is not 5, continuing... (i is %d)', i));
    end
end
```

which produces:

```
i is not 5, continuing... (i is 1)
i is not 5, continuing... (i is 2)
i is not 5, continuing... (i is 3)
i is not 5, continuing... (i is 4)
i is 5! stopping...
```

I tend not to use `break` to exit out of loops but rather write code that more gracefully determines what to do. It's a personal preference.

The `continue` keyword passes control to the next iteration of a loop, and skips any code appearing below where it appears. Like the `break` keyword, when `continue` appears in nested loops it only applies to the loop in which it appears. Here is an example of how `continue` works:

```
for i=1:10
    if mod(i,2)==0
        continue
```

```
    end
    disp(sprintf('the number %d is odd', i));
end
```

which produces:

```
the number 1 is odd
the number 3 is odd
the number 5 is odd
the number 7 is odd
the number 9 is odd
```

Again, I tend not to use `continue` but this is perhaps a personal preference.

The `return` keyword forces MATLAB to return control to the "invoking function", which means that when used within a function, `return` will exit the function without executing any of the remaining code. See the notes on Functions for more about how functions work in MATLAB.