Course reader: *Control statements*

- Control statements allow you to add flexibility to programming. Programming would be pretty useless without control statements.

- All control statements have the same primordial origins:

  Conditional
  content
  alternatives
  end

- `if` statements are very intuitive: If it's raining, then I'll bring an umbrella. Here is an example:

```
v = 4;
if v>3 % main conditional
   disp([ num2str(v) '>3' ]) % content
elseif v<0
   disp([ num2str(v) ' is negative' ]) % alternative
else
   disp([ num2str(v) ' is between 0 and 3' ]) % rest
end
```

- `for` loops are for repeating some code a number of times. Example:

```
for i=1:4 % loop from 1 to 4 in steps of 1
   fprintf( 'Iteration %g: %g^2=%g\n' ,i,i,i^2);
end
```

- `while` loops are similar to for-loops, but you have to break out of the while loop when some condition is satisfied, typically using a toggle. Another important difference is that the looping variable is *not* automatically updated in a `while`-loop (it is automatically updated in a `for`-loop).

```
toggle = true;
counter = 0;
while toggle
   counter = counter+1;
   disp([ 'Counter: ' num2str(counter) ])
   if counter>4, toggle=false; end
end
```

  Notice in this example that the entire if-statement was on a single line. That's OK only if you have a really short and easily understandable line of code.

- `try` statements mean that MATLAB will try to run the code, and if there is an error, it will store the error message and then keep running the code.

```
vec = [1 3 2 5];
try
```

```
    vec(5); % gives an error but MATLAB keeps going
catch me; % 'me' contains information about the error
end
```

- Very often, different control statements can solve the same problem. Therefore, you should use the control statement you feel most comfortable with.
- That said, there are some guidelines:
  - `for`: Use when you know the exact number of iterations.
  - `while`: Use when you don't know in advance the exact number of iterations.
  - `try`: As useful as it might seem, I recommend against using `try` statements. If your code crashes, that usually means something is wrong and it's best to know about it so you can fix it.
  - `if`: Don't use in place of loops.

- Control statements are most powerful when combined! (Like the power-puff girls, I think.) Consider the following:

```
for i=1:7
   if i^2 > 20
      fprintf('%g^2 is greater than 20\n',i);
   end
end
```

- Give your looping variables sensible names, particularly for long and/or embedded loops. I recommend ending looping variables with `i` for *index*. and put a short comment after `end` to see which loop is ending. Use one-letter variables names only for small loops with very little code, when there is minimal possibility for confusion.

```
for rowi=1:num_rows % loop over rows
    %<lots of code here>
    for triali=1:num_trials % loop over trials
        %<lots of code here>
        for sensi=1:num_sensors % loop over sensors
            %<lots of code here>
        end % end sensor loop
    end % end trials loop
    %<lots of code here>
end % end loop over rows
```

- Use smart indentation to make code easier to read. Consider the following code, which is the same as the above code but without spacing. Which do you think is easier to debug?

```
for rowi=1:num_rows % loop over rows
%<lots of code here>
for triali=1:num_trials % loop over trials
%<lots of code here>
for sensi=1:num_sensors % loop over sensors
```

```
%<lots of code here>
end % end sensor loop
end % end trials loop
%<lots of code here>
end % end loop over rows
```

You can apply smart indent by selecting some text in the MATLAB editor window and pressing Ctrl-i (or apple-i) or right-clicking and selecting Smart Indent.

Exercises

1. Create a 15-element vector of normally distributed random numbers. In a loop, test whether each element is greater than 1 and print out the result. The output should be, for example:

`The 2nd element is -2.435 and is not greater than 1.`

Make sure your code has exceptions to print "1st", "2nd", etc. instead of "1th", "2th" etc.

2. Find and fix the error in the following code.

```
for i=.1:.2:2
    res(i) = i*4+log(i);
end
```

3. Under what circumstances (if any) would the following code print "I'm here!"?

```
v = randn;
if v>1
  v=-v;
elseif v<0
  disp('v<0')
else
  disp('I''m here!') % note two single-quotes for text!
end
```

4. What (if anything) is the difference between the following for-loop vs. while-loop? If they are the same, change the line order inside the while-loop to make them different; if they are already different, change the line order to make them the same.

```
for i=1:6
   disp([ 'Iteration ' num2str(i) ])
end


j=1;
toggle = true;
while toggle
   disp([ 'Iteration ' num2str(j) ])
   j=j+1;
   if j>6, toggle=false; end
end
```

5. Before running this code in MATLAB, guess what messages will print out, and guess what the error message in variable `me` will be. Then run it in MATLAB to confirm.

```
vec = 1:6;
try
   for i=1:7
      disp([ 'Number ' num2str(vec(i)) ])
```

```
   end
catch me;
end
```

6. Will the following code print out a message? Think of your answer first, then test it in MATLAB. Finally, fix this code.

```
clear
try
   varname == 4;
   disp([ 'varname is ' num2str(varname) ])
catch me;
end
```

7. This code is awful in many ways. Count the problems and fix them.

```
for i=1:6
i=3;varA(i)=i*43; end
```

8. Which of the following lines of code produce errors, and how can you fix them?

```
i=3; i=rand(i);
i=[3 2]; r=rand(i); r(3,3)
i=[3 2]; i(1), i(2)
i=3; i(1), i(2)
```

9. Rewrite the following code to get the same result without a loop. In general, you should always see whether it's possible to avoid loops when programming.

```
nums = round(logspace(1,3,15));
numlogs = zeros(size(nums));
for numi=1:length(nums)
   numlogs(numi) = log(nums(numi));
end
```

10. Lines 2-4 below differ only by spaces. Do they produce the same results?

```
qt=40; qm=42;
[qt - qm]
[qt -qm]
[qt-qm]
```

11. Before doing this problem, you need to know that you can press Ctrl-c to break out of a loop. Now run the code and figure out what's wrong with it.

```
j=1;
toggle = true;
while toggle
   disp([ 'Iteration ' num2str(j) ])
   if j>10, toggle=false; end
end
```

Answers

1. I thought it would be handy to break the message into two parts, one that does, and one that does not, depend on the outcome of the test.

```
vect = randn(15,1);
for vi=1:length(vect)
   % figure out ending
   if vi==1
      e='st';
   elseif vi==2
      e='nd';
   elseif vi==3
      e='rd';
   else
      e='th';
   end

   % print first part of message
   fprintf('The %g%s element is %g and is ',vi,e,vect(vi));

   % second part of message
   if vect(vi)>1
      fprintf('bigger than 1\n');
   else
      fprintf('smaller than 1\n');
   end
end
```

2. The problem is that the index into `res` must be integer. Here are two solutions.

```
asdf = .1:.2:2;
for i=1:length(asdf)
   res(i) = asdf(i)*4+log(asdf(i));
end

%% or:
cnt = 1;
for i=.1:.2:2
   res(cnt) = i*4+log(i);
   cnt = cnt+1;
end
```

3. If `v` is between 0 and 1. I put that `v=-v` just to confuse you—changing the value of `v` won't affect the behavior of the if-else statements.

4. They key insight here is that in a `while`-loop, you have to be careful about *where* you update the counting variable.

5. It crashes on the 7th iteration because `vec` has only 6 elements. The error message will be about "Index exceeds matrix dimensions," which is your tip that something is wrong with accessing part of a matrix that doesn't exist.

6. The double-equals (==) sign is only for *asking the question* whether the left and right sides are equal. The single-equals (=) is used to assign a value to a variable. `varname` doesn't exist, so you cannot test whether it equals 4.

7. I get a headache just looking at this code! Maybe this helps:

```
for i=1:6
  varA(i) = i*43;
end
```

8. The bugs here are again related to accessing elements of the vector that do not exist (the 3rd position of a 2-element vector).
This problem also nicely illustrates that you can simply overwrite a variable without having to explicitly clear and re-initialize it. This kind of behavior is what people refer to when they say that "MATLAB is a forgiving high-level language."

9. Getting rid of loops (also called "vectorizing") is deeply satisfying. A good loop-deletion is a great way to end the day on Friday. Don't you agree?

```
nums = round(logspace(1,3,15));
numlogs = log(nums);
```

10. MATLAB will interpret `[-qm]` as the negative of `qm` and `[- qm]` as subtracting `qm`.

11. The counting variable `j` is never updated, so the toggle never becomes false. The loop continues to run until the sun blows up and swallows the Earth.