The introduction of the first project last class should sufficiently motivate today's lecture. Today we will learn about some computer programming basics that you will need to know in order to complete your first project.

**Programming Basics:**

A computer program is simply a collection of individual statements that can be stored somewhere and executed whenever needed. Typically, programs are given a set of inputs called arguments that are used by the program in its calculations.

In Matlab, programs are known specifically as m-files. To write an m-file, look under the "File" menu and go to "New" and then select "m-file". This should open Matlab's Editor / Debugger window, which is where you will be creating and modifying your m-files. We will learn more about programming in the Matlab environment later. For now, it is important to learn about two types of programming structures that are absolutely essential: conditional expressions and loops.

**Conditional Expressions:**

A conditional expression performs some true or false test and then continues based on the outcome of the test. In Matlab, we will be using a common type of conditional expression known as an "if statement". A basic if statement has the following form:

```matlab
if test
    statement 1
    statement 2
    statement 3
    ...
else
    false statements
    ...
end
```

This if statement can be read as follows: if test returns true then do statements, otherwise skip to end and continue code execution.

Suppose that something needs to be done if test returns true and something else needs to be done if test returns false. In order to do this, the if statement should take the following form:

```matlab
if test
    true statements
    ...
else
    false statements
    ...
end
```
This if statement can be read as follows: if \( test \) returns true then do true statements, otherwise do false statements and then continue code execution.

If more than one test needs to be made, it may be done as follows:

```plaintext
if test 1
    true statements 1
    .
    .
elseif test 2
    true statements 2
    .
    .
elseif test 3
    true statements 3
    .
    .
else
    all false statements
    .
    .
end
```

This if statement can be read as follows: if test 1 returns true then do true statements 1, if test 2 returns true then do true statements 2, if test 3 returns true then do true statements 3, otherwise do all false statements and then continue code execution. You may use as many elseif statements as is necessary.

Now that we know the forms that an if statement can take, we need to take a look at the types of expressions that we can use for the test portion of the if statement. The test portion can contain the following characters, which are used to make true or false comparisons:

<table>
<thead>
<tr>
<th>Relational Operators</th>
<th>Logical Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>==</code></td>
<td><code>&amp;</code></td>
</tr>
<tr>
<td><code>~=</code></td>
<td>`</td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td></td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example Expression</th>
<th>Returns True If:</th>
</tr>
</thead>
<tbody>
<tr>
<td>if variable == 1</td>
<td>variable equals 1</td>
</tr>
<tr>
<td>if variable &lt;= 5</td>
<td>variable is less than or equal to 5</td>
</tr>
<tr>
<td>if variable &gt; 10 &amp; variable &lt; 20</td>
<td>variable is between the values of 10 and 20</td>
</tr>
<tr>
<td>if variable1 == variable2</td>
<td>variable1 equals variable2 or variable1 equals variable3</td>
</tr>
</tbody>
</table>
Loops:

Loops are very useful in performing repetitive tasks in computer programs. A loop allows you to repeat a set of statements a number of times. Two different types of loops that we will be using in this course are for and while loops.

For loops

For loops are useful when you want to perform some calculation a known number of times. For loops have the following structure:

```
for index = startvalue:increment:stopvalue
    statements
    .
    .
    .
end
```

The argument `index` must be a variable name and will contain the values given in the vector that follows it. The vector given by `startvalue:increment:stopvalue` defines the values that `index` will contain. The `statements` section contains a set of computer statements that will be executed for each value of `index`. For illustration of how a for loop executes, consider the following example:

```
for i = 1:6
    y(i) = i^2;
end
```

This for loop creates a vector `y` that contains the squares of the values in the vector `x`. After completion the vector `y` will be \[ \begin{bmatrix} 1 & 4 & 9 & 16 & 25 & 36 \end{bmatrix} \]. The first line of this example is the for statement and can be read as: “for the index variable `i`, which will start at one, increment by one and end at 6, do the following statements”. The second line is the statement that will be repeated for each value of the index variable `i`. On the first pass through the for loop, `i=1` and the statement is computed as `y(1)=x(1)^2`, on the second pass through the for loop, `i=2` and the statement is computed as `y(2)=x(2)^2`, etc. In short, the for loop takes the place of the following Matlab statements:

```
y(1)=1^2;
y(2)=2^2;
y(3)=3^2;
y(4)=4^2;
y(5)=5^2;
y(6)=6^2;
```

After the sixth pass through the for loop, when `i=6`, the program exits the for loop and any statements following the for loop are executed.
While Loops

While loops are useful when you want to perform some calculation as long as some defined condition is true. While loops have the following structure:

```
while test
    statements
    .
    .
    .
end
```

The argument `test` is some expression containing relational and/or logical operators, similar to the argument of an `if` statement. If the `test` returns true, the `statements` are executed and the `test` is checked again. The loop will execute as long as `test` returns true. The variables modified within the loop should include the variables in `test` or the `test` expression will never change and always return true. This creates what is called an infinite loop. As the name suggests, an infinite loop executes forever without end, and is certainly not the desired outcome of your program. Let's look at an example while loop to see how it works:

```
i=1;
while i<=10
    z(i)=i+3;
    i=i+1;
end
```

This while loop creates a vector `z` that contains the values of `i` with 3 added to them. After completion the vector `z` will be \[13\, 12\, 11\, 10\, 9\, 8\, 7\, 6\, 5\, 4\]. The first line of this example initializes the variable `i` to 1. The second line of code is the beginning of our while loop. The while loop tests whether `i` is less than or equal to 10, which it is. The next line creates a vector `z` and calculates its first element. The fourth line of code is where we modify a variable found in our test so that we avoid an infinite loop. This line increments the variable `i` by 1. The end of the while loop has been reached and the test is evaluated once again. This time `i=2`, which is still less than or equal to 10, so the statements are executed again. The loop continues in this fashion until the test `i<=10` returns false. When the test returns false, the program leaves the loop and continues execution with the statements that follow. The loop in our example takes the place of the following Matlab commands:

```
z(1)=1+3;
z(2)=2+3;
z(3)=3+3;
z(4)=4+3;
z(5)=5+3;
z(6)=6+3;
z(7)=7+3;
z(8)=8+3;
z(9)=9+3;
z(10)=10+3;
```