PROCEDURAL PROGRAMMING:
LOOPS

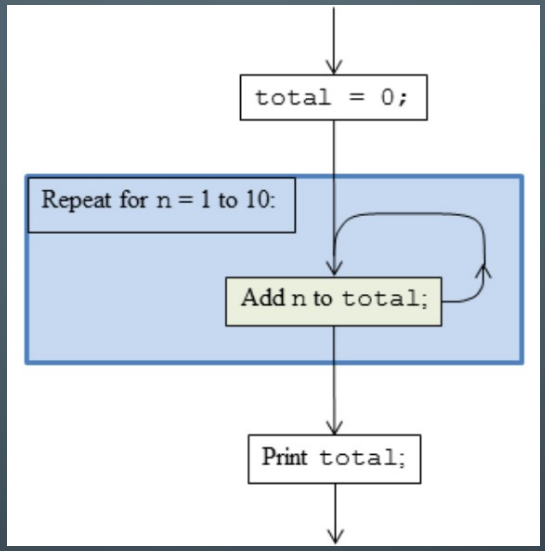
ASSOC. PROF. TUNÇ DURMAZ
tdurmaz@yildiz.edu.tr
JUNE 07, 2021

LOOPS – The Loop Concept

- Without loops, computers would be little more than fancy calculators. Loops give a computer its greatest power power—whether that computer is your laptop, your cell phone, or your automobile's automatic braking system.
- In the previous lecture, we learned about the control constructs—if, if-else, if-elseif, if-elseif-else, each of which stipulates whether a block of statements is to be executed one time or zero times.
- The loop construct can stipulate any number of executions of a block of statements—zero, one, a hundred, a hundred thousand, or any other number.

LOOPS – The Loop Concept

- illustrates a loop for the addition of the integers from 1 to 10.



- The loop is outlined in a blue box.
- The flow of operations enters at the top of the figure, follows the arrows and exits at the bottom of the figure.
- Inside this loop are two statements.
  - The first is "Repeat for n = 1 to 10". This statement is a control statement, which means that it controls the execution of another statement or statements.
  - The second statement in the loop is "Add n to total", which we have highlighted in light green. This statement is the body of the loop. The body of a loop contains the statements that are repeated.
- The repetition of the body of the loop continues until the condition has been met. The green statement is repeated 10 times, and then the loop is over.

LOOPS – The Loop Concept

- Here is a formal definition of the word "loop" as it is used in computer programming:
  - 1a. (noun) a set of statements that is repeated until some condition is met.
  - 1b. (noun) a control construct that causes a block of statements to be executed repeatedly (i.e., zero, one, two, or more times).
  - 2. (non-transitive verb) repeat a set of statements until some condition is met.
- One execution of the body of a loop is called an iteration of the loop.
- As we mentioned above, the body of the loop in the figure (previous slide) is controlled by the statement—"Repeat for n = 1 to 10".
  - It repeatedly changes the value of the variable "n".

LOOPS – The Loop Concept

- The variable n, which is being updated each time through the loop, is a critical part of the loop, and it has a special name.
- A variable that is changed by the control statement of a loop is called a **loop index**.
- This is a new use of the word “index”, which, up until now, has been used in the programming sense to denote a positive integer that enumerates the elements of a vector or matrix.

LOOPS – The for Loop

- For-loop is another control construct and the name “for-loop” is based on the presence of the word “for” in the control statement.
- Here is the MATLAB implementation of a for-loop:

```
total = 0;
for n = 1:10          <-- control statement
    total = total + n;    <-- body
end
fprintf('total = %d\n', total);
```

(the loop = the `for` … `end` block)

- and here is the result of running it:

```
total = 55
```

LOOPS – The for Loop

- When n = 1:10 appears in the control statement of a for-loop, it means this:

```
Assign the first element of [1 2 3 4 5 6 7 8 9 10] to n.
Execute the body of the for-loop.
Assign the next element of [1 2 3 4 5 6 7 8 9 10] to n.
Execute the body again.
Assign the next element of [1 2 3 4 5 6 7 8 9 10] to n.
Execute the body again.
. . .
Assign the last element of [1 2 3 4 5 6 7 8 9 10] to n.
Execute the body for the last time.
```

- The general form of the for-loop is as follows:

```
for index = values
   block
end
```

- In our example:

EXAMPLE PHRASE	GENERAL FORM
n	index
1:10	values
total = total + n	block

LOOPS – The for Loop

- It might be helpful to note the similarities between the syntax of the for-loop and the syntax of the if-statement:
  - Both constructs begin with a control statement and end with the keyword end.
  - The control statement for each of them begins with a keyword—for in the for-loop and if in the if-statement.
  - The semantics of the two constructs are similar as well in that each one has a body that is controlled by the control statement and each one has a control statement that stipulates the **number** of times that the body will be executed.
  - The difference is that in the case of the if-statement that **number** is limited to zero or one, whereas in the case of the for-loop the number is zero or some positive integer.

LOOPS – The for Loop

- Another control construct can be used in the body of the for-loop.

```
N = 5;
list = rand(1,N); % assigns a row vector of random numbers
for x = list
   if x > 0.5
      fprintf('Random number %f is large.\n',x)
   else
      fprintf('Random number %f is small.\n',x)
   end
end
```

LOOPS – The for Loop

```
N = 5;
list = rand(1,N); % assigns a row vector of random numbers
for x = list
   if x > 0.5
      fprintf('Random number %f is large.\n',x)
   else
      fprintf('Random number %f is small.\n',x)
   end
end
```

Here is what happened:

1. N was assigned the number 5.
2. rand was called with the arguments 1 and 5, causing it to generate a row vector of five random numbers, [0.14189 0.42176 0.91574 0.79221 0.95949], and that row vector was assigned to list.
3. The control statement assigned the 1st element of list, 0.14189, to x.
4. The if-statement found that 0.14189 is less than 0.5, triggering the 1st fprintf
5. The end was reached and the flow of execution returned to the beginning of the loop.
6. The control statement assigned the 2nd element of list, 0.42176, to x.
7. The if-statement found that 0.42176 is less than 0.5, triggering the 1st fprintf .
8. The end was reached and the flow of execution returned to the beginning of the loop.
9. The control statement assigned the 3rd element of list, 0.91574, to x.
10. The if-statement found that 0.91574 is greater than 0.5, triggering the 2nd fprintf.
11. The **end** was reached and the flow of execution returned to the beginning of the loop.
12. The control statement assigned the 4th element of **list**, **0.79221**, to **x**.
13. The if-statement found that **0.79221** is greater than **0.5**, triggering the 2nd **fprintf** .
14. The **end** was reached and the flow of execution returned to the beginning of the loop.
15. The control statement assigned the 5th element of **list**, **0.95949**, to **x**.
16. The if-statement found that **0.95949** is greater than **0.5**, triggering the 2nd **fprintf** .
17. The **end** was reached and the loop ended because the last element of **list** had been assigned to **x**.

LOOPS – The for Loop

```
N = 5;
list = rand(1,N); % assigns a row vector of random numbers
for x = list
   if x > 0.5
      fprintf('Random number %f is large.\n',x)
   else
      fprintf('Random number %f is small.\n',x)
   end
end
```

- We can learn some things from this second example:

1. The values assigned to the loop index do not have to be
   - integers,
   - regularly spaced, or
   - assigned in increasing order.
2. Another control construct can be used in the body of the for-loop.

LOOPS – The for Loop

- An interesting question arises when the loop index is assigned a value not only by the control statement but also by a statement within the body of the loop: What is the new value of the index on the next iteration?

```
total = 0;
for n = 1:10
   n
   n = n + 1;
   total = total + n;
end
fprintf('%d\n',total);
```

- At the beginning of the nth iteration of every for-loop, the loop's control statement will assign the loop index the nth term in its list of values, regardless of any value that may have been assigned to the loop index within the body of the loop during the previous iteration.
- Assignments to the loop index inside the body of a loop are temporary.

LOOPS – The for Loop

- Every array operation and every matrix operation can be translated into an equivalent for-loop version.
- No reason to do it because the array operation will typically run faster and it is easier to program than the equivalent version using explicit looping, but it makes the point that an array operation always requires looping.

```
>> u = [5 4 8 8 2];

>> v = [5 5 7 8 8];

>> w = u - v

w =
     0    -1     1     0    -6
```

```
for ii = 1:length(u)
   w(ii) = u(ii) - v(ii);
end
```

LOOPS – Nested for Loops

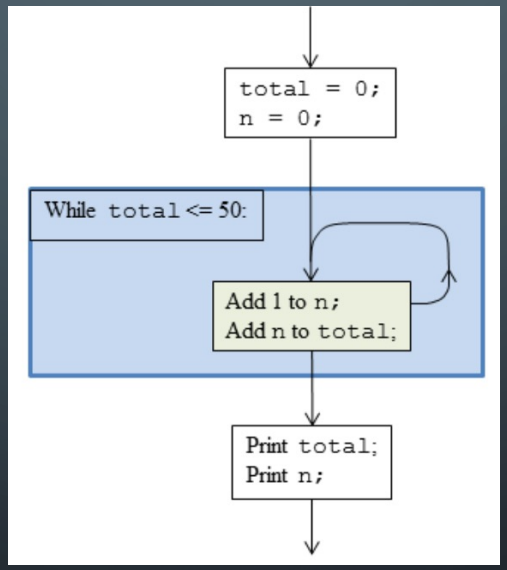
- As mentioned in the previous section, MATLAB allows the nesting of any control construct inside any other construct.
- Nested for-loops often occur when we are doing things with two-dimensional arrays.
- Here is code that uses explicit loops to find the multiplication of an array with itself:
- Example: **A = randi(10,3,4)**

```
for m = 1:size(A,1)
    for n = 1:size(A,2)
        P(m,n) = A(m,n)*A(m,n);   <-- body of inner loop
    end
end
```

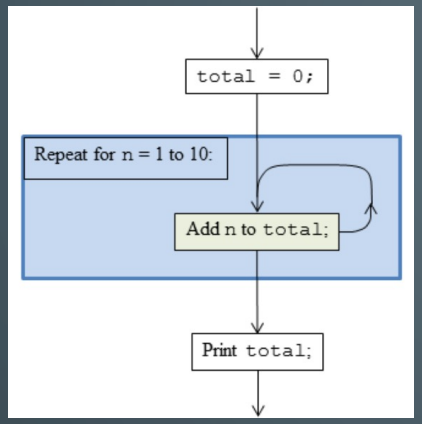
body of outer loop

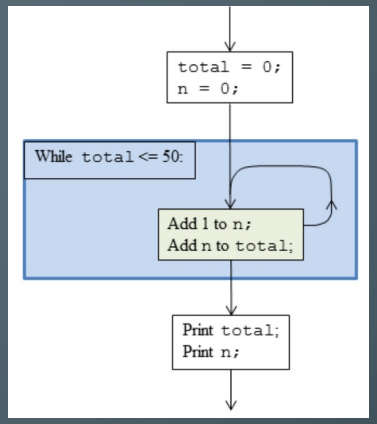
LOOPS – The while Loop

- Suppose instead we wanted to sum the positive integers until we reached the first sum that is greater than 50. Unless we knew that we needed to stop at 10, we could not do this with a for-loop. What we need to solve this problem is a while-loop.



LOOPS – The while Loop





- A comparison between the for-loop and the while-loop reveals the major difference between the two:
- Unlike the for-loop, the while-loop has no formal loop index. We have chosen to use the same variable n in both the for-loop and the while-loop, but it is formally a loop index only in the for-loop. In the while-loop, it may be convenient to think of n as a loop index, and we will sometimes refer to a counter like n that enumerates the number of iterations as a "loop index", but it is not part of the syntax of the while-loop, as it is for the for-loop. In particular, it is not a required part of the control-statement of the while-loop. As a result, there are three important differences between the two types of loops concerning a counter, such as n:
  - n must be initialized before the while-loop is entered, whereas initialization is unnecessary for the for-loop;
  - n must be incremented in the body of the while-loop, whereas incrementing is unnecessary in body of the for-loop;
  - the control-statement of the while-loop does not assign values to n, whereas the control-statement of the for-loop does.
- While-loops often have nothing that resembles a loop index.

LOOPS – The while Loop

- Here is the MATLAB implementation of the loop in our example:

```
total = 0;
n = 0;
while total <= 50
    n = n + 1;
    total = total + n;
end
fprintf('total = %d\n', total);
fprintf('n = %d\n', n);
```


- The general form of the while-loop is as follows:

```
while conditional

   block

end
```

LOOPS – The while Loop

- The general form of the while-loop is as follows:

```
while conditional

   block

end
```

- This form is very similar to the simple if statement:

```
if conditional
   block
end
```

- As for the if-statement, the conditional in the while-statement determines whether the statements in the body, will be executed. The key difference between the if-statement and the while-statement is that, after the body is executed, the if-statement ends, whereas in the while-statement, the conditional is evaluated again. As long as the conditional is true (i.e., is nonzero), the body will be executed, and re-executed, and re-executed, etc.

LOOPS – Infinite Loops and Control+C

- The following piece of code tries to find an approximation to the square root of x:

```
y = x
while abs(y^2 - x) > 0.001*x
   y = (x/y + y)/2
end
```

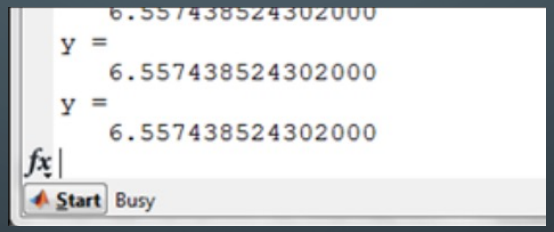
- We might hope to get an ideal square root by setting the acceptable level to zero. We can try that by changing the control statement to this:

```
while abs(y^2 - x) > 0
```

- UPDATE: while abs(y^2 - x) > -0.00001
- Unfortunately, this idea does not work because the value of abs (y^2 – x) will never get to zero.
- A loop that continues iterating without any possibility of stopping is called an infinite loop.

LOOPS – Infinite Loops and Control+C

- When MATLAB is caught in an infinite loop and nothing is being printed to the Command Window, it may at first glance appear that the MATLAB system has died.
- Fortunately, in this circumstance MATLAB displays a small sign of life that shows that is still with us. While MATLAB is working on your program, whether it is making good progress or is caught in an infinite loop, it displays the word "Busy" just to the right of the Start button at the bottom left of the Command Window as a signal that it is working.



- Any time that we suspect that it is wasting time instead of making progress, we can tell MATLAB to stop running the program that we have given it without killing MATLAB itself. It is done with a Control+C. A control command is issued by holding the Ctrl key and hitting the c key. Its meaning is roughly, "Abort!".

LOOPS – Changing the Loop Flow with Break and Continue

- Sometimes during an iteration of a for-loop or a while-loop, part of the calculation should be skipped.
- For example, we want to set all the values in the vector named "readings" to zero until we reach the first value that exceeds 100. For example, suppose the vector has these values:

```
readings = [32 100, 0, 8, 115, 123, 277 92, 14, 87 0 8];
```

- The first value that exceeds 100 is 115, so in this case we would want to set the values before 115 to zero:

```
readings = [ 0,  0, 0, 0, 115, 123, 277 92, 14, 87 0 8];
```

We could do this with a while-loop as follows:

```
ii = 1;
while ii < length(readings) && readings(ii) <= 100
   readings(ii) = 0;
   ii = ii + 1;
end
```

LOOPS – Changing the Loop Flow with Break and Continue

- But there is better approach:

```
for ii = 1:length(readings)
   if readings(ii) > 100
      break;
   else
      readings(ii) = 0;
   end
end
```

- The break-statement can appear only inside a loop.
  - In fact, MATLAB will stop the program and print an error message, if it encounters the keyword break anywhere that is not inside a loop.
- The meaning of the break-statement is that the loop is ended and control continues at the next statement following that loop.

LOOPS – Changing the Loop Flow with Break and Continue

- The break-statement can be used in a while-statement as well. Here is an alteration of the while-statement above that incorporates a break-statement to accomplish the same goal:

```
ii = 1;
while ii < length(readings)
   if readings(ii) <= 100
      readings(ii) = 0;
   else
      break;
   end
   ii = ii + 1;
end
```

LOOPS – Changing the Loop Flow with Break and Continue

- A common misunderstanding involving the break-statement appears when it is used in nested-loops. The break applies only to the innermost loop, meaning that it will cause the loop it appears in to terminate, but the outer loop will continue.
- Here is an example. Suppose we have the following array.

```
A =
    81    10    16    15    65    76    70    82
    90    28    97    42     4    74     4    69
    13    55    95    91    85    39    28    32
    91    95    49    79    93    65     5    95
    63    96    80    95    68    17    10     4
```

- We wish to look at each element in row-major order, and set them to zero until we find the first value that is greater than 90. Here is the result we want:

```
A =
     0     0     0     0     0     0     0     0
     0     0    97    42     4    74     4    69
    13    55    95    91    85    39    28    32
    91    95    49    79    93    65     5    95
    63    96    80    95    68    17    10     4
```

LOOPS – Changing the Loop Flow with Break and Continue

- To look at each element in row-major order, we need a nested for-loop, so let's write one and include a break-statement to stop the processing when we reach a value greater than 90:

```
for ii = 1:size(A,1)
   for jj = 1:size(A,2)
      if A(ii,jj) <= 90
         A(ii,jj) = 0;
      else
         break;
      end
   end
end
```

- When we hit a value greater than 90, we break out of the loop. Sounds good, but which loop do we break out of? Only the inner loop. This does not accomplish what we wanted. Here is the result:

```
A =
     0     0     0     0     0     0     0     0
     0     0    97    42     4    74     4    69
     0     0    95    91    85    39    28    32
    91    95    49    79    93    65     5    95
     0    96    80    95    68    17    10     4
```

LOOPS – Changing the Loop Flow with Break and Continue

- MATLAB has no mechanism for specifying that multiple loops be terminated when a break-statement is encountered. The only way to cause the outer loop to terminate is to include a statement in the inner loop that writes a note and another statement in the outer loop that reads it. That note, which in common programming terminology is called a **flag** which is a value indicating a special condition. In this case the special condition is that the value we have been looking for has been found.

```
found = false;
for ii = 1:size(A,1)
   for jj = 1:size(A,2)
      if A(ii,jj) <= 90
         A(ii,jj) = 0;
      else
         found = true;
         break;
      end
   end
   if found
      break;
   end
end
```

- This flag-setting scheme will always work for you when you want to break out of more than one loop. If the loops are nested three deep, then you need two additional if-statements containing break-statements; deeper nesting requires more if-statements and break-statements, but only one flag is needed, no matter how deep the nesting.

LOOPS – Changing the Loop Flow with Break and Continue

- The break-statement has a complementary construct, called the continue statement.
- The continue-statement, which consists of the single keyword continue, causes the innermost loop to continue to the next iteration without completing the current one. It is used when it is determined, while executing the statements in the body of the loop, that all the subsequent statements in the body of the loop should be skipped. Unlike the break-statement, the continue-statement has no effect on the number of iterations that are carried out.

LOOPS – Changing the Loop Flow with Break and Continue

- Suppose, for each number x we wish to print x, x^2, x^4, $x^{1/2}$, and $x^{1/4}$. However, we are not interested in complex numbers (for some reason), so, if the number is negative, since the last two values would be complex, we do not want to print (or calculate) them. Here is way to do it without using a continue-statement:

```
for ii = 1:length(numbers)
   x = numbers(ii);
   fprintf('x = %d\n', x);
   fprintf('    x^2 = %d\n', x^2);
   fprintf('    x^4 = %d\n', x^4);
   if x >= 0
      fprintf('    x^(1/2) = %f\n', x^(1/2));
      fprintf('    x^(1/4) = %f\n', x^(1/4));
   end
end
```

- In this simple example, the first three fprintf statements will be executed. for each number in the list, but the last two will be executed only if x is nonnegative.
- Here is how we do it with a continue-statement:

```
for ii = 1:length(numbers)
   x = numbers(ii);
   fprintf('x = %d\n', x);
   fprintf('    x^2 = %d\n', x^2);
   fprintf('    x^4 = %d\n', x^4);
   if x < 0
      continue
   end
   fprintf('    x^(1/2) = %f\n', x^(1/2));
   fprintf('    x^(1/4) = %f\n', x^(1/4));
end
```

RECALL

- Control constructs determine the flow of execution of the program. With no control constructs, statements execute sequentially. Some control constructs are *selection statements* meaning that it selects which code to execute. Other control constructs are *loops*, where the control construct determines how many times the loop will execute.
- A control statement is a statement that determines whether other statements will be executed.
  - An if statement decides whether to execute another statement, or decides which of two statements to execute.
  - A loop decides how many times to execute another statement.