

# REPETITION STRUCTURES

# Repetition Structures

- Usually, sections of computer code can be categorized as ***sequences***, ***selection structures***, and ***repetition structures***.
- As a rule of thumb, if a section of code is repeated more than three times, it is a good candidate for a repetition structure.
- Repetition structures are often called loops. All loops consist of five basic parts.
  - A parameter to be used in determining whether or not to end the loop.
  - Initialization of this parameter.
  - A way to change the parameter each time through the loop. (If you don't change it, the loop will never stop executing.)
  - A comparison, using the parameter, to a criterion used to decide when to end the loop.
  - Calculations to do inside the loop.

# Repetition Structures, ctd.

- MATLAB supports two different types of loops: the for loop and the while loop.
- Two additional commands, break and continue , can be used to create a third type of loop, called a midpoint break loop.
- The for loop is the easiest choice when you know how many times you need to repeat the loop.
- While loops are the easiest choice when you need to keep repeating the instructions until a criterion is met.
- Midpoint break loops are useful for situations where the commands in the loop must be executed at least once, but where the decision to exit the loop is based on some criterion

# for Loops

- The structure of the for loop is simple. The first line identifies the loop and defines an index, which is a number that changes on each pass through the loop and is used to determine when to end the repetitions.
- After the identification line comes the group of commands we want to execute.
- Finally, the end of the loop is identified by the command end.
- Thus, the structure of a for loop can be summarized as

```
for index = [matrix]
    commands to be executed
end
```

- The loop is executed once for each element of the index matrix identified in the first line. Here's a simple example:

```
for k = [1,3,7]
    k
end
```

```
k =
1
k =
3
k =
7
```

# for Loops, ctd.

- The index in this case is  $k$ . Programmers often use  $k$  as an index variable.
- The index matrix can also be defined with the colon operator or, indeed, in a number of other ways as well.
- Here's an example of code that finds the value of 5 raised to powers between 1 and 3:

```
for k = 1:3
    a = 5^k
end
```

- On the first line, the index,  $k$ , is defined as the matrix [1, 2, 3].
- The first time through the loop,  $k$  is assigned a value of 1, and  $5^1$  is calculated.
- Then the loop repeats, but now  $k$  is equal to 2 and  $5^2$  is calculated.
- The last time through the loop,  $k$  is equal to 3 and  $5^3$  is calculated.

```
a =
5
a =
25
a =
125
```

# for Loops, ctd.

```
for k = 1:5
    a(k) = k^2
end
```

- This loop defines a new matrix,  $a$ , one element at a time. Since the program repeats its set of instructions five times, a new element is added to the  $a$  matrix each time through the loop, with the following output in the command

window:

```
a =
1
a =
1 4
a =
1 4 9
a =
1 4 9 16
a =
1 4 9 16 25
```

# for Loops, ctd.

- Most computer programs do not have MATLAB's ability to handle matrices so easily; therefore, they rely on loops similar to the one just presented to define arrays. It would be easier to create the vector  $a$  in MATLAB with the code

```
k = 1:5  
a = k.^2
```

which returns

```
k =  
1 2 3 4 5  
a =  
1 4 9 16 25
```

- This is an example of **vectorizing** the code.

# for Loops, ctd.

- Another common use for a for loop is to combine it with an if statement and determine how many times something is true. For example, in the list of test scores shown in the first line, how many are above 90?

```
scores = [76,45,98,97];  
count = 0;  
  
for k=1:length(scores)  
    if scores(k)>90  
        count = count + 1;  
    end  
end  
  
disp(count)
```



# for Loops, Example-1

## Calculating Factorials With A For Loop

- A factorial is the product of all the integers from 1 to  $N$ .
- For example, 5 factorial is  $1 * 2 * 3 * 4 * 5$ .
- In mathematics texts, factorial is usually indicated with an exclamation point:

5! is five factorial. And  $0! = 1$ .

- By prompting the user to enter a non-negative integer, type your own factorial code using a for loop and test it.
- You can add an if statement to confirm that the input is a non-negative integer and not an array,
- **Input must be a non-negative integer**
- **Input should be a scalar**

# for Loops, Example-2

- Consider the series given by,

$$\sum_{n=0}^N \frac{(-1)^n x^{2n+1}}{(2n+1)!}$$

- Prompt the user to enter values of  $x$  and  $N$  ( $N > 0$ ).
- Calculate the value of the series for entered  $x$  and  $N$ .

```
x = input('Enter the value of x: ');  
N = input('Enter the value of N: ');  
  
S = 0;  
for n = 0:N  
    S = S + (-1)^n*x^(2*n+1)/factorial(2*n+1);  
end
```

# while Loops

- While loops are similar to for loops. The big difference is the way MATLAB decides how many times to repeat the loop. While loops continue until some criterion is met. The format for a while loop is

```
while criterion
    commands to be executed
end
```

- Here's an example:

```
k = 0;
while k < 3
    k = k + 1
end
```

|     |
|-----|
| k = |
| 1   |
| k = |
| 2   |
| k = |
| 3   |

- In this case, we initialized a counter,  $k$ , before the loop. Then the loop repeated as long as  $k$  was less than 3. We incremented  $k$  by 1 every time through the loop, so that the loop repeated three times.

# while Loops, ctd.

- We could also use  $k$  as an index number to define a matrix or just as a counter. Most for loops can also be coded as while loops. Recall the for loop in the previous section used to calculate the first three powers of 5. The following while loop accomplishes the same task:

```
k = 0;
while k<3
    k = k+1;
    a(k) = 5^k
end
```

- Each time through the loop, another element is added to the matrix  $a$ , and the code returns

```
a =
5
a =
5 25
a =
5 25 125
```

# while Loops, ctd.

- As another example, first initialize a:

```
a = 0;
```

- Then find the first multiple of 3 that is greater than 10:

```
while (a<10)
    a = a + 3
end;
```

- The first time through the loop, a is equal to 0, so the comparison is true.
- The next statement ( a = a + 3 ) is executed, and the loop is repeated.
- This time a is equal to 3 and the condition is still true, so execution continues.
- In succession, we have

```
a = 3
```

```
a = 6
```

```
a = 9
```

```
a = 12
```

# while Loops, ctd.

- while loops can be used to calculate series until the terms get sufficiently small. Recall the series we considered earlier.
- We can continue calculating until the terms get really small:

```
x = input('Enter the value of x: ');
N = input('Enter the value of N: ');

n = 0;
term = (-1)^n*x^(2*n+1)/factorial(2*n+1);
S = term;

while abs(term) > 0.001
    n = n + 1;
    term = (-1)^n*x^(2*n+1)/factorial(2*n+1);
    S = S + term;
end
```

# while Loops, ctd.

- Find the difference between these two pieces of code:

```
x = input('Enter the value of x: ');
N = input('Enter the value of N: ');

n = 0;
term = (-1)^n*x^(2*n+1)/factorial(2*n+1);
S = term;

while abs(term) > 0.001
    n = n + 1;
    term = (-1)^n*x^(2*n+1)/factorial(2*n+1);
    S = S + term;
end
```

```
x = input('Enter the value of x: ');
N = input('Enter the value of N: ');

n = 0;
term = (-1)^n*x^(2*n+1)/factorial(2*n+1);
S = 0;

while abs(term) > 0.001
    S = S + term;
    n = n + 1;
    term = (-1)^n*x^(2*n+1)/factorial(2*n+1);
end
```

# while Loops, ctd.

- One common use for a while loop is error checking of user input.
- Consider a program where we prompt the user to input a positive number, and then we calculate the log base 10 of that value.
- We can use a while loop to confirm that the number is positive, and if it is not, to prompt the user to enter an allowed value.
- The program keeps on prompting for a positive value until the user finally enters a valid number.

```
x = input('Enter a positive value of x: ')
while (x<=0)
    disp('log(x) is not defined for negative numbers')
    x = input('Enter a positive value of x: ')
end
y = log10(x);
fprintf('The log base 10 of %4.2f is %5.2f \n',x,y)
```



# break Statement

- The break command can be used to terminate a loop prematurely (while the comparison in the first line is still true). A break statement will cause termination of the smallest enclosing while or for loop. Here's an example:

```
n = 0;
while(n<10)
    n = n+1;
    a = input('Enter a value greater than 0:');
    if(a<=0)
        disp('You must enter a positive number')
        disp('This program will terminate')
        break
    end
    disp('The natural log of that number is')
    disp(log(a))
end
```

# continue Statement

- The continue command is similar to break ; however, instead of terminating the loop, the program just skips to the next pass

```
n = 0;
while(n<10)
    n = n+1;
    a = input('Enter a value greater than 0:');
    if(a<=0)
        disp('You must enter a positive number')
        disp('Enter a value again')
        continue
    end
    disp('The natural log of that number is')
    disp(log(a))
end
```

# break - continue

- In selection and repetition statements, break and continue is used to alter the flow of control.

```
for count = 1:10
% loop 10 times
    if ( count == 5 )
        break; % break loop if x
is 5
    end
    fprintf("%d ", count);}
end
% 1 2 3 4
```

When the compiler encounters break statement, it immediately goes out of the loop

```
for count = 1:10
% loop 10 times
    if ( count == 5 )
        continue;
        % skip remaining code
in loop,
        % continue looping
    end
    fprintf("%d ", count);
end
% 1 2 3 4 6 7 8 9 10
```

When the compiler encounters continue statement, it immediately stops the execution of that iteration of the loop

# Nested Loops

- It is often useful to nest loops inside other loops.
- We can write a code segment to fill the matrix A

```
cols = 5; rows = 5;  
for k=1:cols  
    for j=1:rows  
        a(i,j)=i*j;  
    end  
end
```