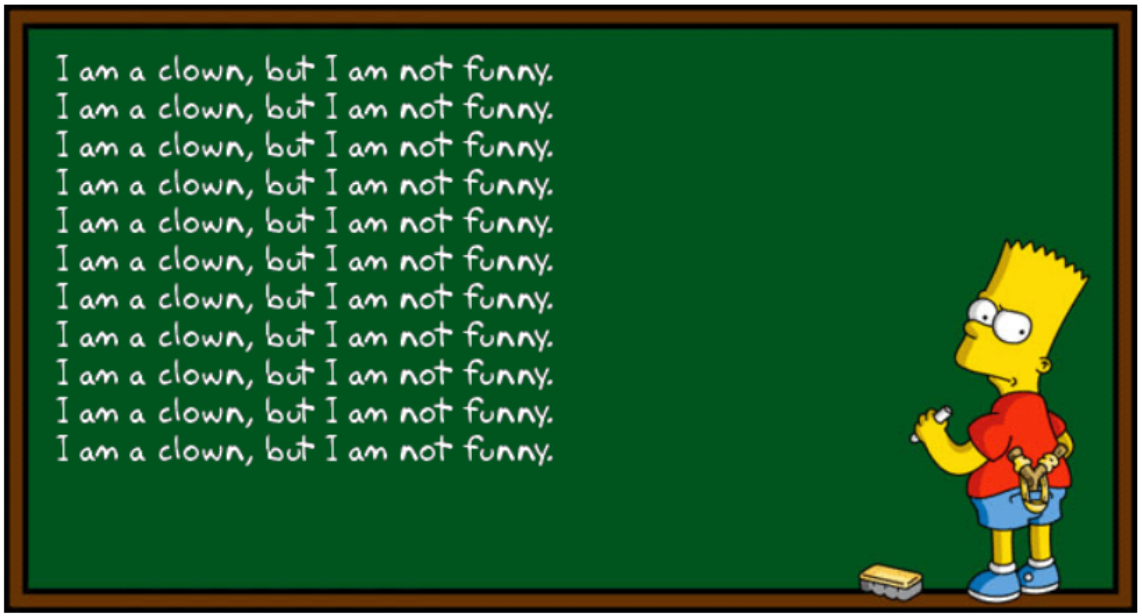
REPETITION STRUCTURES*

*** Some of the contents are adopted from**
MATLAB® for Engineers, Holly Moore, 3rd Ed., Pearson Inc., 2012.

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.
  - Why do we need loops?
    - Write on the screen 1000 times



Repetition Structures, ctd.

- One way of doing this:

```
disp('I am a clown, but I am not funny.');
disp('I am a clown, but I am not funny.');
disp('I am a clown, but I am not funny.');
disp('I am a clown, but I am not funny.');
disp('I am a clown, but I am not funny.');
disp('I am a clown, but I am not funny.');
disp('I am a clown, but I am not funny.');
disp('I am a clown, but I am not funny.');
disp('I am a clown, but I am not funny.');
disp('I am a clown, but I am not funny.');
disp('I am a clown, but I am not funny.');
disp('I am a clown, but I am not funny.');
disp('I am a clown, but I am not funny.');
disp('I am a clown, but I am not funny.');
…
disp('I am a clown, but I am not funny.');
```

Repetition Structures, ctd.

- 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.

Keywords

- **Loops**
  - while
  - for
  - continue
  - break

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:

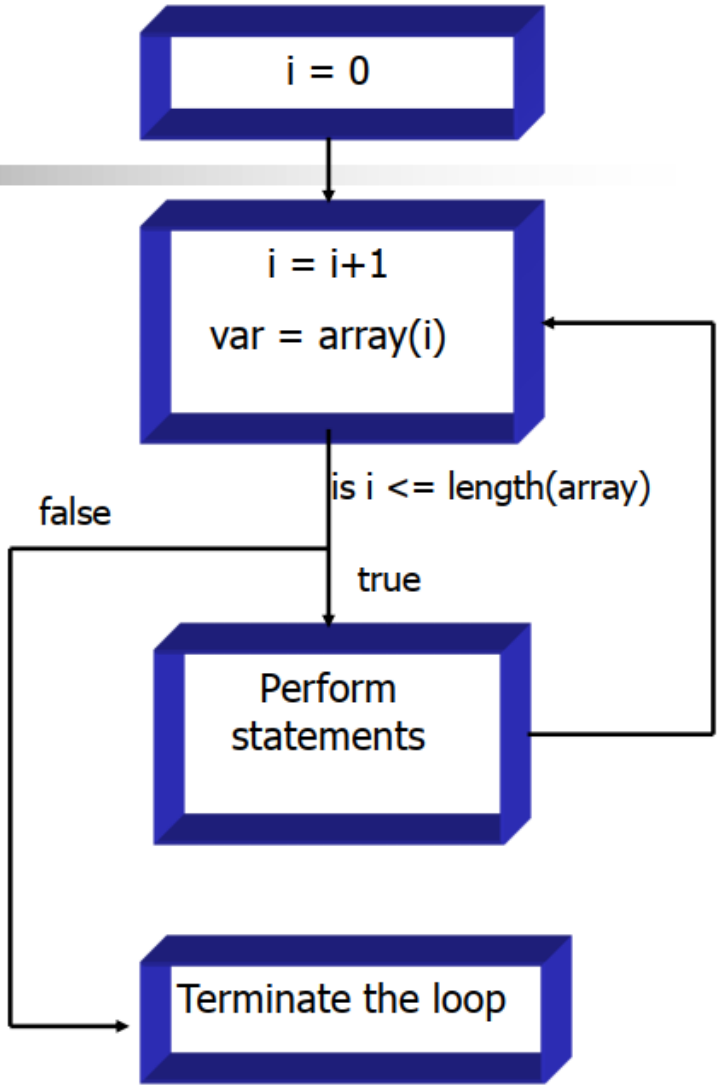
```
for k = [1,3,7]
    k
end
```

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

`for` Loops, ctd.

for loop

```
for var = array
   statement1;
   statement2;
...
end
```



`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 5^3 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

- compute the sum of squares of numbers 1 to 10

```
n = 10; sumsq = 0;

for k = 1 : n
    sumsq = sumsq + k ^ 2;
end
disp(sumsq)
```

Tip: use generic symbols rather than hard coded numbers

- How do we do this without a loop?

```
>> sum((1:10).^2)

ans =

  385
```

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
```

`for` Loops, Example-2

- Write a script called **large_elements** that takes as input an array named **dimensionVector** that is a row matrix or a vector which will store the dimensions of a matrix called as **A**. The script identifies those elements of **A,** which will be generated by **rand()** function, that are greater than the sum of their two indexes. For example, if the element **A(2,3)** is 6, then that element would be identified because 6 is greater than 2 + 3. The output of the script gives the indexes of such elements found in row major order. It is a matrix with exactly two columns. The first column contains the row indexes, while the second column contains the corresponding column indexes. If no such element exists, the script returns an empty array.

`for` Loops, Example-2

```
clc
clear all
close all

dimensionVector = input('Enter # of rows and columns as an array= ');

A = round(rand(dimensionVector(1,1),dimensionVector(1,1))*10);
A
[row col] = size(A);
found = [];
for ii = 1:row
    for jj = 1:col
        if A(ii,jj) > ii + jj % if the element > the sum of its indexes
            found = [found; ii jj]; % add a new row to the output matrix
        end
    end
end
found
```

`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
```

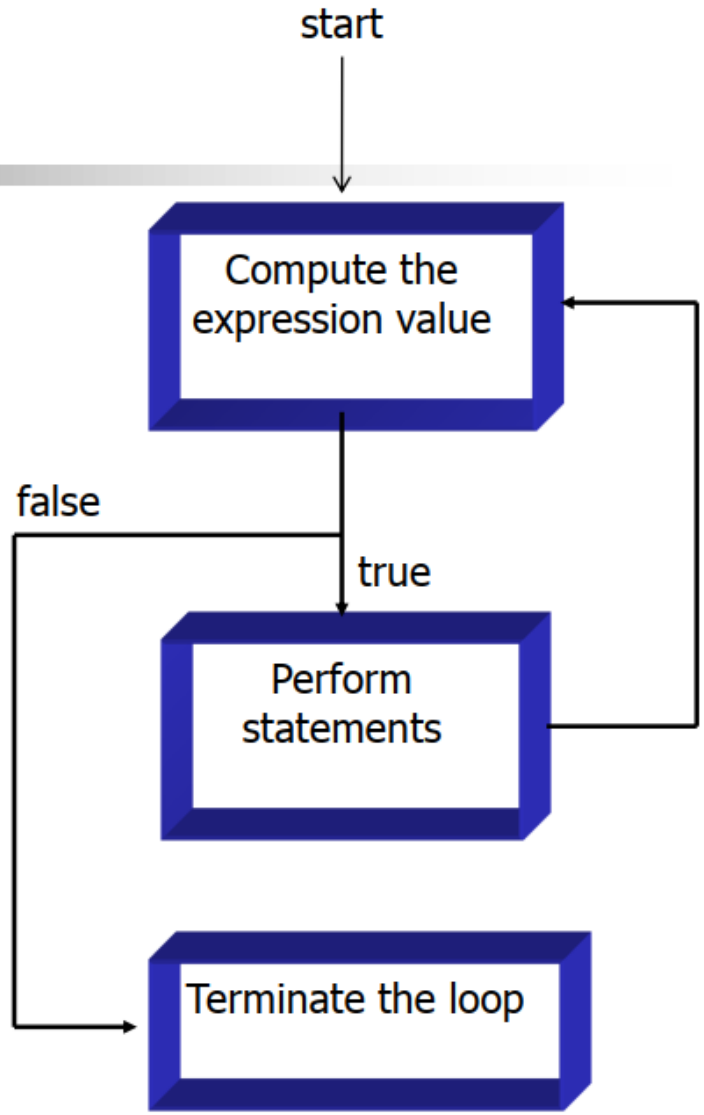
```
k =
1
k =
2
k =
3
```

• 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.

while loop

```
while expression
   statement1;
   statement2;
...
end
```



`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 also be used to count how many times a condition is true by incorporating an `if` statement. Recall the test scores we counted in a `for` loop earlier.
- We can also count them with a `while` loop:

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

`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)
```

`while` Loops, Example 1.

Write a program that generates the following output:

```
1
1  2
1  2  3
1  2  3  4
1  2  3  4  5
1  2  3  4  5  6
1  2  3  4  5  6  7
1  2  3  4  5  6  7  8
1  2  3  4  5  6  7  8  9
1  2  3  4  5  6  7  8  9  10
```

`while` Loops, Example 1, cnt.

- Solution:

```
i = 1;

while i < 11
    disp(1 : i);
    i = i + 1;
end
```

Initial count

Stopping condition

`while` Loops, Example 2.

Write a program that generates the following output:

```
1  2  3  4  5  6  7  8  9  10
1  2  3  4  5  6  7  8  9
1  2  3  4  5  6  7  8
1  2  3  4  5  6  7
1  2  3  4  5  6
1  2  3  4  5
1  2  3  4
1  2  3
1  2
1
```

`while` Loops, Example 2.

- Solution

```
i = 10;

while i > 0
    disp(1 : i);
    i = i - 1;
end
```

`while` Loops, Example 3.

What will the following program print?

```
i=1;

while ~(i == 6)
  disp(i)
  i = i + 2;
end
```

`while` Loops, Example 4.

- Write a program that the computes factorial n
  - For example if n = 10 then n! = 10 * 9 * 8 ... * 1

```
n = 10; res = 1;

while n > 1
   res = res * n;
   n = n - 1;
end

disp(res);
```

`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
```

```
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 break statement, it immediately goes out of the loop

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