# Loop Statement

**Chul Min Yeum**
Assistant Professor
Civil and Environmental Engineering
University of Waterloo, Canada

**AE121: Computational Method**

UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING

**Last updated: 2019-06-04**

# For-Loop

- used as a counted loop
- repeats an action a specified number of times
- an iterator or loop variable specifies how many times to repeat the action
- general form:

```
for loopvar = range
    action
end
```

- the range is specified by a vector
- the action is repeated for every value of the loop variable in the specified vector

# Input in a For-Loop

- If it is desired to repeat the process of prompting the user and reading input a specified number of times (N), a for loop is used:

```
for ii = 1:N
    % prompt and read in a value
    % do something with it!
end
```

- If it is desired to store the values entered in a vector, the most efficient method is to preallocate the vector first to have N elements

# Example: How For-Loop Works

```
sum = 0
for ii = 1:3
    sum = sum + ii;
end
```

1) ii is 1.
2) sum = 0 + 1; (‘sum’ becomes 1)
3) end
4) ii is 2.
5) sum = 1 + 2; (‘sum’ becomes 3)
6) end
7) ii is 3.
8) sum = 3 + 3; (‘sum’ becomes 6)
9) End
10) No more ii -> exit the loop.

‘sum’ becomes 6.
‘ii’ becomes 3.

# Pre-allocating a Vector

- Preallocating sets aside enough memory for a vector to be stored
- The alternative, extending a vector, is very inefficient because it requires finding new memory and copying values every time
- Many functions can be used to preallocate, although it is common to use **zeros**
- For example, to preallocate a vector *vec* to have N elements:

```
vec = zeros(1,N);
```

(Highly recommended) if you knew the array size that you allocate values computed from a loop !!!

# Example: Simple Example I

## Generate a row vector having scalar values from 11 to 20 with 1 interval

```matlab
% THINK ABOUT THE ANSER BEFORE LOOKING AT SAMPLE CODES


row_vec1 = 11:20;

for ii=1:10
    row_vec2(ii) = ii+10;
end
% not recommend: the size of row_vec2 is changed on every loop

row_vec3 = zeros(1, 10);
for ii=1:10
    row_vec3(ii) = ii+10;
end
% recommended if you need to use for-loop

row_vec4 = zeros(1, 10);
for ii=[1:10]
    row_vec4(ii) = ii+10;
end
% you can assign any vector as a range

row_vec5 = [];
for ii=11:20
    row_vec5 = [row_vec5 ii];
end
% not recommend: the size of row_vec5 is changed on every loop
```

**Refereeing: Error**

**Assigning : Okay**

**matlab**

```matlab
% all vectors are identical
row_vec1
```

```
row_vec1 = 1x10
    11    12    13    14    15    16    17    18    19    20
```

```matlab
row_vec2
```

```
row_vec2 = 1x10
    11    12    13    14    15    16    17    18    19    20
```

```matlab
row_vec3
```

```
row_vec3 = 1x10
    11    12    13    14    15    16    17    18    19    20
```

```matlab
row_vec4
```

```
row_vec4 = 1x10
    11    12    13    14    15    16    17    18    19    20
```

```matlab
row_vec5
```

```
row_vec5 = 1x10
    11    12    13    14    15    16    17    18    19    20
```

# Example: Simple Example II

Sum 1 to 10 and assign its value to a variable

```
% THINK ABOUT THE ANSER BEFORE LOOKING AT SAMPLE CODES



A1 = 1+2+3+4+5+6+7+8+9+10;

A2 = 0;
for ii=1:10
    A2 = A2 + ii;
end

A3 = (1+10)*10/2; % use an equation

A4 = sum(1:10); % use a built-in function
```

```
% all values are identical
A1
```

```
A1 = 55
```

```
A2
```

```
A2 = 55
```

```
A3
```

```
A3 = 55
```

```
A4
```

```
A4 = 55
```

# Nested For-Loops

- A nested **for** loop is one inside of ( as the action of) another **for** loop
- General form of a nested **for** loop:

```
for loopvarone = rangeone
    actionone:
  for loopvartwo = rangetwo
     actiontwo
  end
end
```

- The inner loop action is executed in its entirety for every value of the outer loop variable

```
for ii=1:3
    for jj=1:2
        [ii jj]
    end
end
```

```
ans = 1x2

     1     1
ans = 1x2

     1     2
ans = 1x2

     2     1
ans = 1x2

     2     2
ans = 1x2

     3     1
ans = 1x2

     3     2
```

# Use of Nested For-Loops

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 1 | 2 | 3 | 4 |
| 1 | 2 | 3 | 4 |

mat1

```
mat1 = [1 2 3 4; 1 2 3 4; 1 2 3 4];

[num_row, num_col] = size(mat1);

sum_values_all = 0;
for ii=1:num_row
    for jj=1:num_col
        sum_values_all = sum_values_all + mat1(ii,jj);
    end
end
```

# Combining For-Loop(s) and If

- **for** loops and **if** statements can be combined
  - the action of a loop can include an **if** statement
  - the action of an **if** statement can include a **for** loop
- This is also true for nested **for** loops; **if** statements can be part of the action(s) of the outer and/or inner loops
- This is done if an action is required on an element (of a vector or matrix) only if a condition is met

# Example: Logical Operation

## Change 1 to 5, 2 to 7, and the rest to 10 in a given vector

```
vec = [1 1 2 1 3 1 6 7 5];

vec_new1 = zeros(size(vec));
% vec_new1 = vec; vec_new1 = 0; % same result
for ii=1:numel(vec_new1)
    if vec(ii)==1
        vec_new1(ii) = 5;
    elseif vec(ii)==2
        vec_new1(ii) = 7;
    else
        vec_new1(ii) = 10;
    end
end

vec_new2 = zeros(size(vec));
vec_new2(vec==1) = 5;
vec_new2(vec==2) = 7;
vec_new2((vec~=1)&(vec~=2)) = 10;
% recommend: logical indexing can clearn out your code.
```

```
vec_new3 = ones(size(vec))*10;
vec_new3(vec==1) = 5;
vec_new3(vec==2) = 7;
% we can save one more line

% all vectors are identical
vec_new1
```

```
vec_new1 = 1x9
     5     5     7     5    10     5    10    10    10
```

```
vec_new2
```

```
vec_new2 = 1x9
     5     5     7     5    10     5    10    10    10
```

```
vec_new3
```

```
vec_new3 = 1x9
     5     5     7     5    10     5    10    10    10
```

# Example: Find

## Find indexes of a vector where 5 is located

```
vec = [1 5 6 4 8 5 3 7 8 5 9 10];

loc1 = find(vec == 5); % it is starightforward to use a 'find' function

% note that we do not know the size of loc2
loc2 = [];
for ii=1:numel(vec)
    if vec(ii)==5
        loc2 = [loc2 ii];
    end
end
% the size of loc2 is changed on every loop

loc3 = zeros(1, numel(vec));
count = 0;
for ii=1:numel(vec)
    if vec(ii)==5
        count = count + 1;
        loc3(count) = ii;
    end
end
loc3 = loc3(1:count);
% If we know the maximum size that it could possibly be, we can preallocate to a size of the loc3.
% And then delete the “unused” elements.
% In order to do that, you would have to count the number of elements that are actually used.
```

```
loc1 = 1x3
     2     6    10

loc2 = 1x3
     2     6    10

loc3 = 1x3
     2     6    10
```

# Break

- 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.

```
for ii=1:10
    ii
    if ii==5
        break;
    end
end
```

```
ii = 1
ii = 2
ii = 3
ii = 4
ii = 5
```

## Quiz: Value Replacement

If values in ‘vec1' are larger than 0 and less than 50, replace the values to 10. Otherwise, replace them to 5.

vec1 = [ 1 10 70 80 2]

vec1 = [10 10 5 5 10]

# Quiz: Value Replacement (Answer)

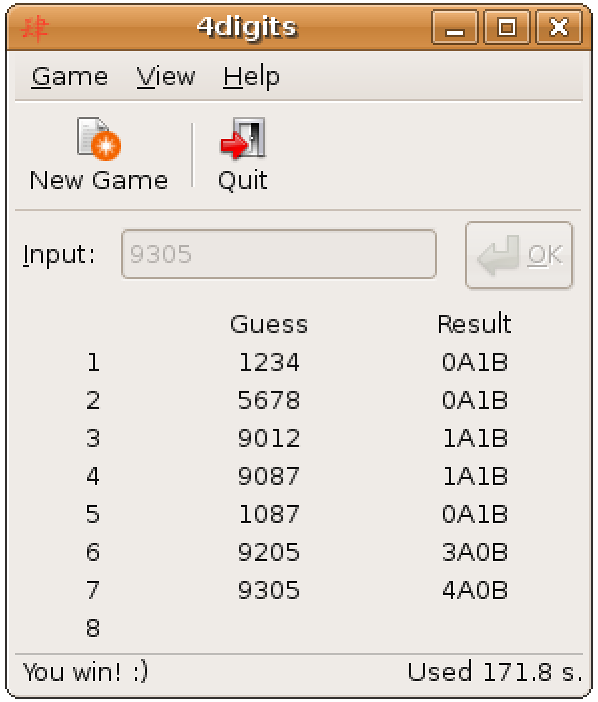
If values in 'vec1' are larger than 0 and less than 50, replace the values to 10. Otherwise, replace them to 5.

```
for ii=1:numel(vec1)
    test_value = vec1(ii);
    if test_value > 0
        if test_value<50
            repl_value = 10;
        else
            repl_value = 5;
        end
    else
        repl_value = 5;
    end
    vec1(ii) = repl_value;
end
```

```
vec2 = vec_org;
logi_10 = and(vec_org>0, vec_org<50);
vec2(logi_10) = 10;
vec2(~logi_10) = 5;
```

# Quiz: Bulls and Cows

Bulls and Cows is a mind game played by two players. In the game, a random, 4-digit number is chosen and it's values are compared to those of another trial number. **All four digits of the number are different**. If any digit in the chosen number is the exact same value and in the exact same position as any digit in the trial number, this is called a bull. If the digit is present in both the trial number and chosen number, but is not in the same location, this is called a cow.




**Use selection and loop statements to solve this problem**

# While-Loop

- used as a conditional loop
- used to repeat an action when ahead of time it is not known how many times the action will be repeated
- general form:

```
while condition
    action
end
```

- the action is repeated as long as the condition is true
- an *infinite loop* can occur if the condition never becomes false (Use **Ctrl-C to break out** of an infinite loop)
- Note: since the condition comes before the action, it is possible that the condition will be false the first time it is evaluated and therefore the action will not be executed at all

# Counting in a While-Loop

- it is frequently useful to count how many times the action of the loop has been repeated
- general form of a while loop that counts:

```
counter = 0;
while condition
    % action
    counter = counter + 1;
end
% use counter – do something with it!
```

```
% do you know what this code is?
count = 0;
val = 0;
while count~=11
    val = val + count;
    count = count + 1;
end

val
```

```
val = 55
```

# Example: Find (Advanced)

Find an first 4th index of 5 in a given vector

```
vec = [1 5 2 3 5 6 7 4 5 5 6 3 4 5 6 8 5];

tmp = find(vec==5, 4);
idx1 = tmp(end); clearvars tmp
% it's a good practice to delete variables that you will not  use anymore

count = 0;
idx2 = 0;
while count ~=4
    idx2 = idx2 + 1;
    if vec(idx2)== 5
        count = count + 1;
    end
end

count = 0;
idx3 = 1;
while count ~=4
    if vec(idx3)== 5
        count = count + 1;
    end
    idx3 = idx3 + 1;
end
idx3 = idx3 - 1;
% Compare the code for idx2 and idx3
% You should pay attention to the location of the "idx" phrase. Depending
% on its location, the value is varied.
```

```
count = 0;
idx4 = 0;
while 1
    idx4 = idx4 + 1;
    if vec(idx4)== 5
        count = count + 1;
        if count==4
            break;
        end
    end
end
% More often, there are many exit conditions in your while loop. In this
% case, you will introduce break. (however, in this example, there is one
% exit


count = 0;
idx5 = 0;
for ii=1:numel(vec)
    if vec(ii) == 5
        count = count + 1;
        if count==4
            idx5 = ii;
            break;
        end
    end
end
% this example is to introduce how you can design a for-loop to do the same
% operation from a while-loop
```

All ‘idx’ are 10.

# Use MATLAB Wisely!!

- Using **for** loops with vectors and matrices is a very important programming concept, and is necessary when working with many languages
- However… Although **for** loops are very useful in MATLAB, **they are almost not necessary** when performing an operation on every element in a vector or matrix!
- This is because MATLAB is written to work with matrices (and therefore also vectors), so functions on matrices and operations on matrices automatically iterate through all elements – no loops needed!

# Vectorizing

- The term ***vectorizing*** is used in MATLAB for re-writing code using loops in a traditional programming language to matrix operations in MATLAB
- For example, instead of looping through all elements in a vector vec to add 3 to each element, just use scalar addition:

```
vec = vec + 3;
```

# Efficient Code

- In most cases, code that is faster for the programmer to write in MATLAB is also faster for MATLAB to execute.
- **Faster to write = less making mistakes = less time for debugging**
- Keep in mind these important features:
  - Scalar and array operations
  - Logical vectors
  - Built-in functions
  - Preallocation of vectors

# Operations on Vectors & Matrices

- Can perform numerical operations on vectors and matrices, e.g. vec + 3
- Scalar operations e.g. mat * 3
- Array operators operate term-by-term or element-by-element, so must be same size
- Addition + and subtraction -
- Array operators for any operation *based on* multiplication require dot in front .* ./ .\ .^

# **Example:** Dot Product (Theory)

## 4.6 Dot Product

The dot product operation between two compatible vectors is defined as

$$\vec{a} \bullet \vec{b} = \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix} \bullet \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix} = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n = \sum_{k=1}^{n} a_k b_k$$

which is the linear combination of all elements in both vectors. The result is a scalar. Interestingly, this linear combination is the same as

$$\vec{a}^T \vec{b} = \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix}^T \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix} = \begin{bmatrix} a_1 & \cdots & a_n \end{bmatrix} \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix} = \begin{bmatrix} a_1 b_1 + a_2 b_2 + \cdots + a_n b_n \end{bmatrix}$$

The dot product is also known as the scalar product or inner product, and sometimes uses the syntax $\left(\vec{a}, \vec{b}\right) = \left\langle \vec{a}, \vec{b} \right\rangle = \vec{a} \bullet \vec{b}$.

AE115 lecture note (Dr. Brush)

# Example: Dot Product (Code)

Compute a dot product between two row vectors (A and B):

$$\boldsymbol{A} = [3\ 4\ 5\ 6\ 8]$$
$$\boldsymbol{B} = [5\ 6\ 7\ 7\ 8]$$

```
AB1 = dot(A,B); % use of a built-in function

AB2vec = zeros(1,5); % again, do not set zeros(5) because it is a 5 x 5 matrix
for ii=1:numel(A)
    AB2vec(ii) = A(ii)*B(ii);
end
AB2 = sum(AB2vec);
% it is direct convesion of the dot operation.

AB3 = 0;
for ii=1:5
    AB3 = AB3 + A(ii)*B(ii);
end
% we can make shorter the code like this.

AB4 = 0;
for ii=1:numel(A)
    AB4 = AB4 + A(ii)*B(ii);
end
```

# Example: Dot Product (Continue)

Compute a dot product between two row vectors (A and B):

$$\boldsymbol{A} = [3\ 4\ 5\ 6\ 8]$$
$$\boldsymbol{B} = [5\ 6\ 7\ 7\ 8]$$

```
AB5 = sum(A.*B);
% I expect that you will write a code like this at the end of course.
% In this case, in most cases, there is a built-in function (here, 'dot'),
% but not always. You need to learn a skill to write this code based on the
% definition of the dot product.

AB6 = A*B';
% Super! Can you see the difference between * and .*? and understand how
% this is working? You are amazing.
```

## Common Pitfalls

- Forgetting to initialize a running sum or count variable to `0` or a running product to 1
- Not realizing that it is possible that the action of a **while** loop will never be executed
- Not error-checking input into a program (debugging using a MATLAB workspace helps)
- Not taking advantage of MATLAB; not vectorizing! (Do matrix operations!!!)

# Programming Style Guidelines

- Use loops for repetition only when necessary
  - **for** statements as counted loops
  - **while** statements as conditional loops (because we need to have an exist condition)
- Do not use *i* or *j* for iterator variable names if the use of the built-in constants **i** and **j** is desired (recommendation: use ii or jj)
- Indent the action of loops (use a smart indent – Ctrl + I)
- Preallocate vectors and matrices whenever possible (when the size is known ahead of time).
- If the loop variable is just being used to specify how many times the action of the loop is to be executed, use the colon operator 1:n

# Slide Credits and References


- Stormy Attaway, 2018, Matlab: A Practical Introduction to Programming and Problem Solving, 5th edition
- Lecture slides for “Matlab: A Practical Introduction to Programming and Problem Solving”
- Holly Moore, 2018, MATLAB for Engineers, 5th edition