Loops in MATLAB

Lecture 5

Dr. Görkem Saygılı

Department of Biomedical Engineering
Ankara University

Introduction to MATLAB, 2017-2018 Spring

Outline:

In this lecture, we will learn:

- for loop and while loop,
- break and continue statements,
- making loops more efficient in MATLAB,
- logical indexing.

Loops in Built-in Functions:

You may have not realized it yet but you have been using loops in MATLAB already!

For example, sum(...) function which we already used to find the sum of elements inside a vector internally uses a loop inside to do its calculation.

Similarly, max(...), min(...) functions also uses loops to find the maximum and minimum of a vector, respectively.

Iteration and Loop Index:

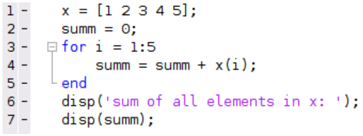
Each single execution of a loop is called an iteration.

If a code contains at least one loop, we can call that code as an “iterative” code.

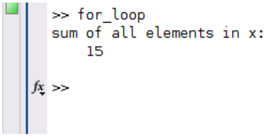
The variable that is defined at the loop initialization is called the “loop index”.

Calculating the Sum of Elements Inside a Vector:

Let's sum the elements inside a 1×5 vector using a for loop:

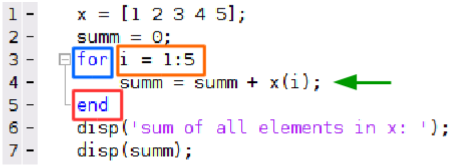


```
x = [1 2 3 4 5];
summ = 0;
for i = 1:5
    summ = summ + x(i);
end
disp('sum of all elements in x: ');
disp(summ);
```



```
>> for_loop
sum of all elements in x:
    15

>>
```

For Loop in MATLAB:



```
x = [1 2 3 4 5];
summ = 0;
for i = 1:5
    summ = summ + x(i);
end
disp('sum of all elements in x: ');
disp(summ);
```

Blue Rectagle: Starting with a “for” command.

Orange Rectagle: Control statement.

Green Arrow: Loop Body.

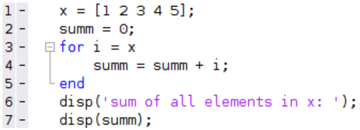
Red Rectangle: “end” command to indicate the end of the loop.

Remember: End Keyword for Functions and If Statements:

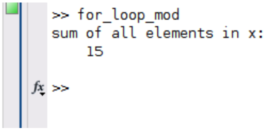
Remember that while writing functions, we do not have to use the “end” keyword. Whereas, for if statements, we have to use “end” to indicate the final line of the statement body.

Similar to if statements, we have to use “end” keyword for for loops as well.

For loop - A Different Approach:



```
x = [1 2 3 4 5];
summ = 0;
for i = x
    summ = summ + i;
end
disp('sum of all elements in x: ');
disp(summ);
```



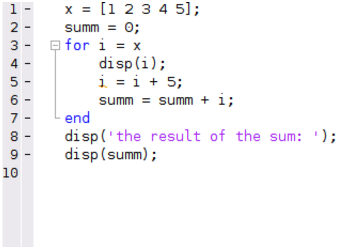
```
>> for_loop_mod
sum of all elements in x:
    15

>>
```

Instead of accessing elements in x using loop index i, we can assign i to each element of x at the start of the loop.

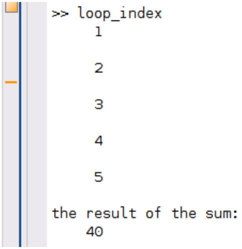
Hence, the values assigned to the loop index do not have to be integers.

Changing Loop Index in the Body:

What happens if the loop index is modified in the loop body?



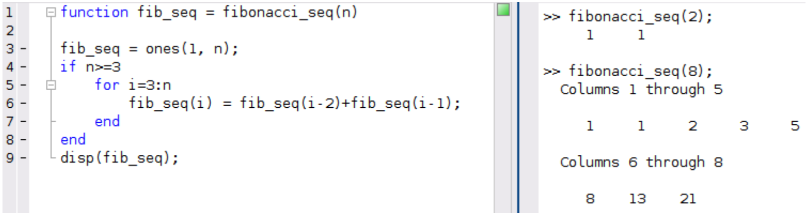
```
x = [1 2 3 4 5];
summ = 0;
for i = x
    disp(i);
    i = i + 5;
    summ = summ + i;
end
disp('the result of the sum: ');
disp(summ);
```



```
>> loop_index
     1

     2

     3

     4

     5

the result of the sum:
    40
```

The changes on loop index in the body is temporary. It has no effect on the next iteration.

Generating Fibonacci Sequence using For Loop:

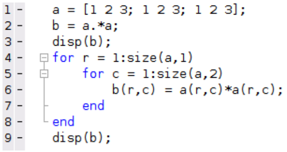


```
function fib_seq = fibonacci_seq(n)

fib_seq = ones(1, n);
if n>=3
    for i=3:n
        fib_seq(i) = fib_seq(i-2)+fib_seq(i-1);
    end
end
disp(fib_seq);
```

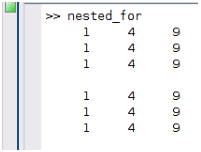
```
>> fibonacci_seq(2);
     1     1

>> fibonacci_seq(8);
  Columns 1 through 5

     1     1     2     3     5

  Columns 6 through 8

     8    13    21
```

Nested For Loops:

MATLAB allows nested loops similar to selection statements. Nested for loops are especially useful when we are working on matrices:

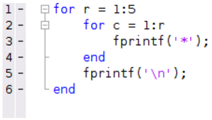


```
a = [1 2 3; 1 2 3; 1 2 3];
b = a.*a;
disp(b);
for r = 1:size(a,1)
    for c = 1:size(a,2)
        b(r,c) = a(r,c)*a(r,c);
    end
end
disp(b);
```

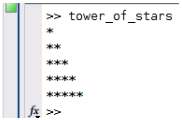


```
>> nested_for
     1     4     9
     1     4     9
     1     4     9

     1     4     9
     1     4     9
     1     4     9
```

Tower of Stars:



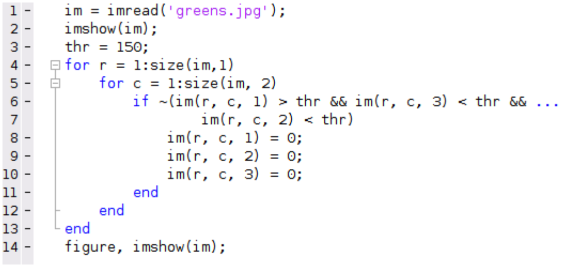
```
for r = 1:5
    for c = 1:r
        fprintf('*');
    end
    fprintf('\n');
end
```



```
>> tower_of_stars
*
**
***
****
*****
>>
```

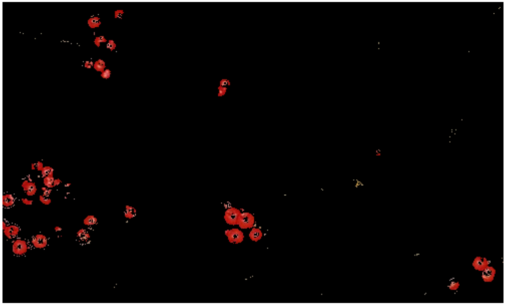
Images:

Segmentation:



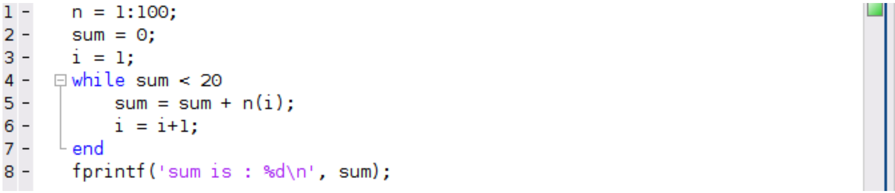
```
im = imread('greens.jpg');
imshow(im);
thr = 150;
for r = 1:size(im,1)
    for c = 1:size(im, 2)
        if ~(im(r, c, 1) > thr && im(r, c, 3) < thr && ...
                im(r, c, 2) < thr)
            im(r, c, 1) = 0;
            im(r, c, 2) = 0;
            im(r, c, 3) = 0;
        end
    end
end
figure, imshow(im);
```

Segmented Fruits:



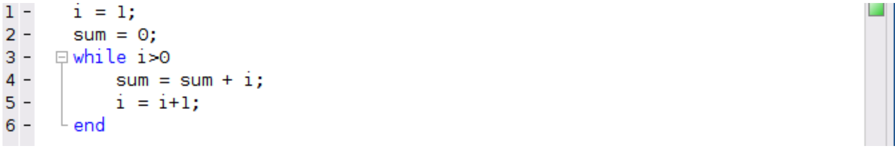
While Loop:

We have another control construct: while.



```
n = 1:100;
sum = 0;
i = 1;
while sum < 20
    sum = sum + n(i);
    i = i+1;
end
fprintf('sum is : %d\n', sum);
```

Infinite Loop:

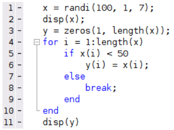


```
i = 1;
sum = 0;
while i>0
    sum = sum + i;
    i = i+1;
end
```

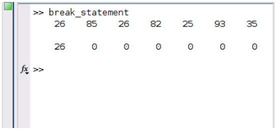
In such cases, you can use Cntrl + C to terminate the execution of the code.

Break Statement:

We can also use break statement in while loop as we did in if statements.

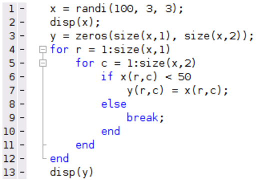


```
x = randi(100, 1, 7);
disp(x);
y = zeros(1, length(x));
for i = 1:length(x)
    if x(i) < 50
        y(i) = x(i);
    else
        break;
    end
end
disp(y)
```

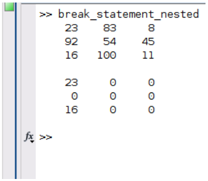


```
>> break_statement
    26    85    26    82    25    93    35

    26     0     0     0     0     0     0

>>
```

Break Statements in Nested Loops:

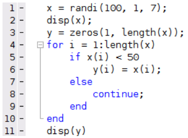


```
x = randi(100, 3, 3);
disp(x);
y = zeros(size(x,1), size(x,2));
for r = 1:size(x,1)
    for c = 1:size(x,2)
        if x(r,c) < 50
            y(r,c) = x(r,c);
        else
            break;
        end
    end
end
disp(y)
```

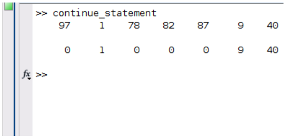


```
>> break_statement_nested
    23    83     8
    92    54    45
    16   100    11

    23     0     0
     0     0     0
    16     0     0

>>
```

Continue:



```
x = randi(100, 1, 7);
disp(x);
y = zeros(1, length(x));
for i = 1:length(x)
    if x(i) < 50
        y(i) = x(i);
    else
        continue;
    end
end
disp(y)
```



```
>> continue_statement
    97     1    78    82    87     9    40

     0     1     0     0     0     9    40

>> 
```