COGS 119/219
MATLAB for Experimental Research

Fall 2014 – Week 1

Built-in array functions, Data types

.m files, Flow Control

Built-in array functions

Function	Description	Example
mean(A)	If A is a vector, returns the mean value of the elements of the vector	>> A = [5 9 2 4]; >> mean(A) ans = 5
std(A)	If A is a vector, returns the standard deviation of the elements of the vector	>> A = [5 9 2 4]; >> std(A) ans = 2.9439
sum(A)	If A is vector, returns the sum of the elements of the vector	>> A = [5 9 2 4]; >> sum(A) ans = 20

Built-in array functions

Function	Description	Example
C = max(A)	If A is a vector, C is the largest element in A. If A is a matrix, C is a row vector containing the largest element of each column of A.	>> C = [5 9 2 4 11 6]; >> C = max(A) ans = 11
[d,n] = max(A)	If A is a vector, d is the largest element in A, n is the position of the element.	>> [d,n] = max(A) d= 11 n = 5

Built-in array functions

Function	Description	Example
min(A)	The same as max(A), but for the smallest element.	>> A = [5 9 2 4]; >> min(A) ans = 2
[d,n] = min(A)	The same as [d,n] = max(A), but for the smallest element.	>> [d,n] = min(A) d= 2 n = 3

Built-in array functions

See help of these functions for various useful ways of using them.

size	flipud
all	fliplr
prod	rot90
any	repmat

Try:

```
>> mymat = eye(3) * diag([4 5 7]) + 2;
>> mybigmat = repmat(mymat, [3 6]);
```

Data types

```
>> help datatypes
  Data types and structures.

  Data types (classes)
    double           - Convert to double precision.
    logical          - Convert numeric values to logical.
    cell             - Create cell array.
    struct           - Create or convert to structure array.
    single           - Convert to single precision.
    uint8            - Convert to unsigned 8-bit integer.
    uint16           - Convert to unsigned 16-bit integer.
    uint32           - Convert to unsigned 32-bit integer.
    uint64           - Convert to unsigned 64-bit integer.
    int8             - Convert to signed 8-bit integer.
    int16            - Convert to signed 16-bit integer.
    int32            - Convert to signed 32-bit integer.
    int64            - Convert to signed 64-bit integer.
    inline           - Construct INLINE object.
    function_handle  - Function handle array.
    javaArray        - Construct a Java Array object.
    javaMethod       - Invoke a Java method.
    javaObject       - Invoke a Java object constructor.
    javaMethodEDT    - Invoke a Java method on the Swing Event Dispatch Thread.
    javaObjectEDT    - Invoke a Java object constructor on the Swing Event Dispatch
```

Data types: Examples

```
>> a=1;
>> b = 1.0;
>> whos
```

Data types: Examples

```
>> a = 1;
>> b = 1.0;
>> whos
  Name        Size                Bytes  Class     Attributes

  a           1x1                     8  double
  b           1x1                     8  double
```

Data types: Examples

```
>> a = 1;
>> b = 1.0;
>> whos
  Name        Size                Bytes  Class     Attributes

  a           1x1                     8  double
  b           1x1                     8  double


>> c = round(b);
>> d = uint8(b);
>> whos
  Name        Size                Bytes  Class     Attributes

  a           1x1                     8  double
  b           1x1                     8  double
  c           1x1                     8  double
  d           1x1                     1  uint8
```

Data types: Examples

```
>> e = true;
>> whos
  Name        Size                Bytes  Class     Attributes

  a           1x1                     8  double
  b           1x1                     8  double
  c           1x1                     8  double
  d           1x1                     1  uint8
  e           1x1                     1  logical
```

Data types: Examples

```
>> e = true;
>> whos
  Name        Size                Bytes  Class     Attributes

  a           1x1                     8  double
  b           1x1                     8  double
  c           1x1                     8  double
  d           1x1                     1  uint8
  e           1x1                     1  logical
```

```
>> f = 'true';
>> f2 = 'cool';
>> g = '1';
>> h = 'this is easy!';
>> whos
  Name        Size                Bytes  Class     Attributes

  a           1x1                     8  double
  b           1x1                     8  double
  c           1x1                     8  double
  d           1x1                     1  uint8
  e           1x1                     1  logical
  f           1x4                     8  char
  f2          1x4                     8  char
  g           1x1                     2  char
  h           1x13                   26  char
```

Data types: Examples

```
>> f + f2

ans =

   215   225   228   209
```

Data types: Examples

```
>> f + f2

ans =

   215   225   228   209


>> h1 = int8(f2)

h1 =

   99  111  111  108
```

Data types: Examples

```
>> f + f2

ans =

   215   225   228   209


>> h1 = int8(f2)

h1 =

   99  111  111  108


>> f2 = f + g

f2 =

   165   163   166   150
```

Data types: Examples

```
>> whos
  Name        Size                Bytes  Class     Attributes

  a           1x1                     8  double
  ans         1x4                    32  double
  b           1x1                     8  double
  c           1x1                     8  double
  d           1x1                     1  uint8
  e           1x1                     1  logical
  f           1x4                     8  char
  f2          1x4                    32  double
  g           1x1                     2  char
  h           1x13                   26  char
  h1          1x4                     4  int8
```

Data types: Examples

```
>> whos
  Name        Size                Bytes  Class     Attributes

  a           1x1                     8  double
  ans         1x4                    32  double
  b           1x1                     8  double
  c           1x1                     8  double
  d           1x1                     1  uint8
  e           1x1                     1  logical
  f           1x4                     8  char
  f2          1x4                    32  double
  g           1x1                     2  char
  h           1x13                   26  char
  h1          1x4                     4  int8
```

```
>> f2(2)

ans =

   163
```

Data types: Examples

```
>> whos
  Name        Size                Bytes  Class     Attributes

  a           1x1                     8  double
  ans         1x4                    32  double
  b           1x1                     8  double
  c           1x1                     8  double
  d           1x1                     1  uint8
  e           1x1                     1  logical
  f           1x4                     8  char
  f2          1x4                    32  double
  g           1x1                     2  char
  h           1x13                   26  char
  h1          1x4                     4  int8
```

```
>> f2(2)

ans =

   163
```

```
>> f(2)

ans =

r
```

Data types: Examples

```
>> whos
  Name        Size                Bytes  Class     Attributes

  a           1x1                     8  double
  ans         1x4                    32  double
  b           1x1                     8  double
  c           1x1                     8  double
  d           1x1                     1  uint8
  e           1x1                     1  logical
  f           1x4                     8  char
  f2          1x4                    32  double
  g           1x1                     2  char
  h           1x13                   26  char
  h1          1x4                     4  int8
```

```
>> f2(2)

ans =

   163


>> f(2)

ans =

r
```

```
>> f(1,1)

ans =

t

>> f(1)

ans =

t
```

Strings

```
>> first = 'Matlab is so';
>> last = 'cool';
>>
>> full = first + last
Error using  +
Matrix dimensions must agree.
```

Strings

```
>> first = 'Matlab is so';
>> last = 'cool';
>>
>> full = first + last
Error using  +
Matrix dimensions must agree.


>> full = [first last]

full =

Matlab is socool
```

Strings

```
>> first = 'Matlab is so';
>> last = 'cool';
>>
>> full = first + last
Error using  +
Matrix dimensions must agree.
```

```
>> full = [first last]

full =

Matlab is socool
```

```
>> full2 = [first '_' last]

full2 =

Matlab is so_cool
```

Strings

```
>> first = 'Matlab is so';
>> last = 'cool';
>>
>> full = first + last
Error using  +
Matrix dimensions must agree.
```

```
>> full = [first last]

full =

Matlab is socool
```

```
>> full2 = [first '_' last]

full2 =

Matlab is so_cool
```

```
>> full3 = [first '' last]

full3 =

Matlab is socool
```

Strings

```
>> first = 'Matlab is so';
>> last = 'cool';
>>
>> full = first + last
Error using  +
Matrix dimensions must agree.
```

```
>> full = [first last]

full =

Matlab is socool
```

```
>> full2 = [first '_' last]

full2 =

Matlab is so_cool
```

```
>> full3 = [first '' last]

full3 =

Matlab is socool
```

```
>> full4 = [first ' ' last]

full4 =

Matlab is so cool
```

Strings

```
>> full5 = [first 'bloody' last]

full5 =

Matlab is sobloodycool
```

Strings

```
>> full5 = [first 'bloody' last]

full5 =

Matlab is sobloodycool


>> full6 = [first ' bloody ' last]

full6 =

Matlab is so bloody cool
```

Strings

```
>> full5 = [first 'bloody' last]

full5 =

Matlab is sobloodycool


>> full6 = [first ' bloody ' last]

full6 =

Matlab is so bloody cool


>> full7 = [first ' "bloody" ' last]

full7 =

Matlab is so "bloody" cool
```

Strings

```
>> full5 = [first 'bloody' last]

full5 =

Matlab is sobloodycool
```

```
>> full6 = [first ' bloody ' last]

full6 =

Matlab is so bloody cool
```

```
>> full7 = [first ' "bloody" ' last]

full7 =

Matlab is so "bloody" cool
```

```
>> full8 = full2'

full8 =

M
a
t
l
a
b

i
s

s
o

c
o
o
l
```

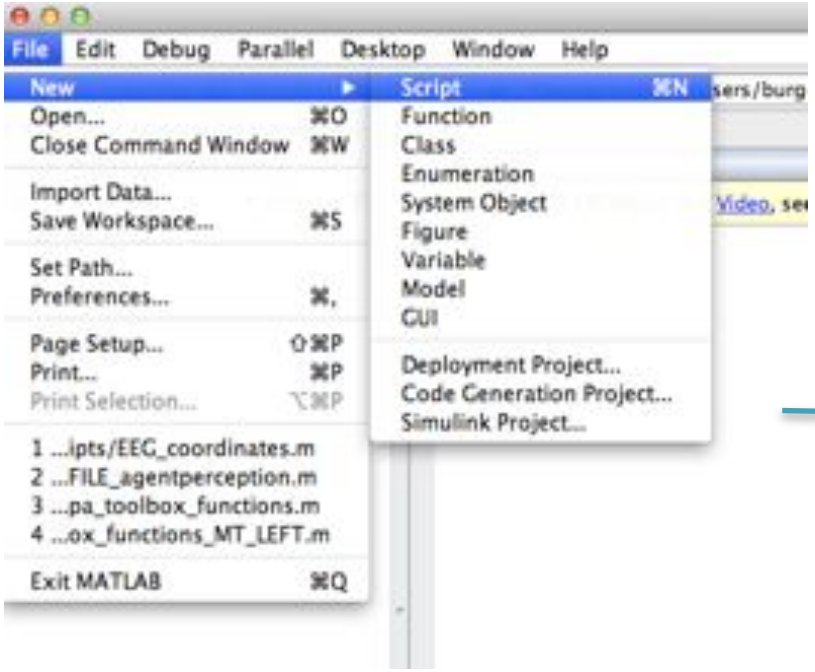
Strings

```
>> whos
  Name          Size                Bytes  Class     Attributes

  first         1x12                   24  char
  full          1x16                   32  char
  full2         1x17                   34  char
  full3         1x16                   32  char
  full4         1x17                   34  char
  full5         1x22                   44  char
  full6         1x24                   48  char
  full7         1x26                   52  char
  full8        17x1                    34  char
  last          1x4                     8  char
```

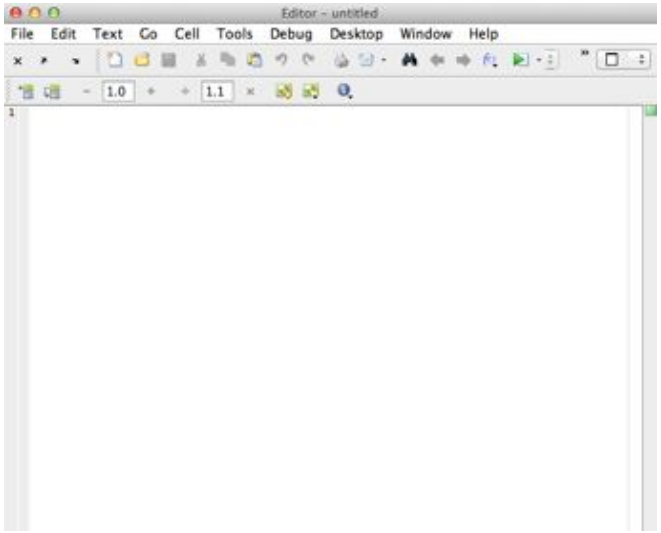
.m files

.m files

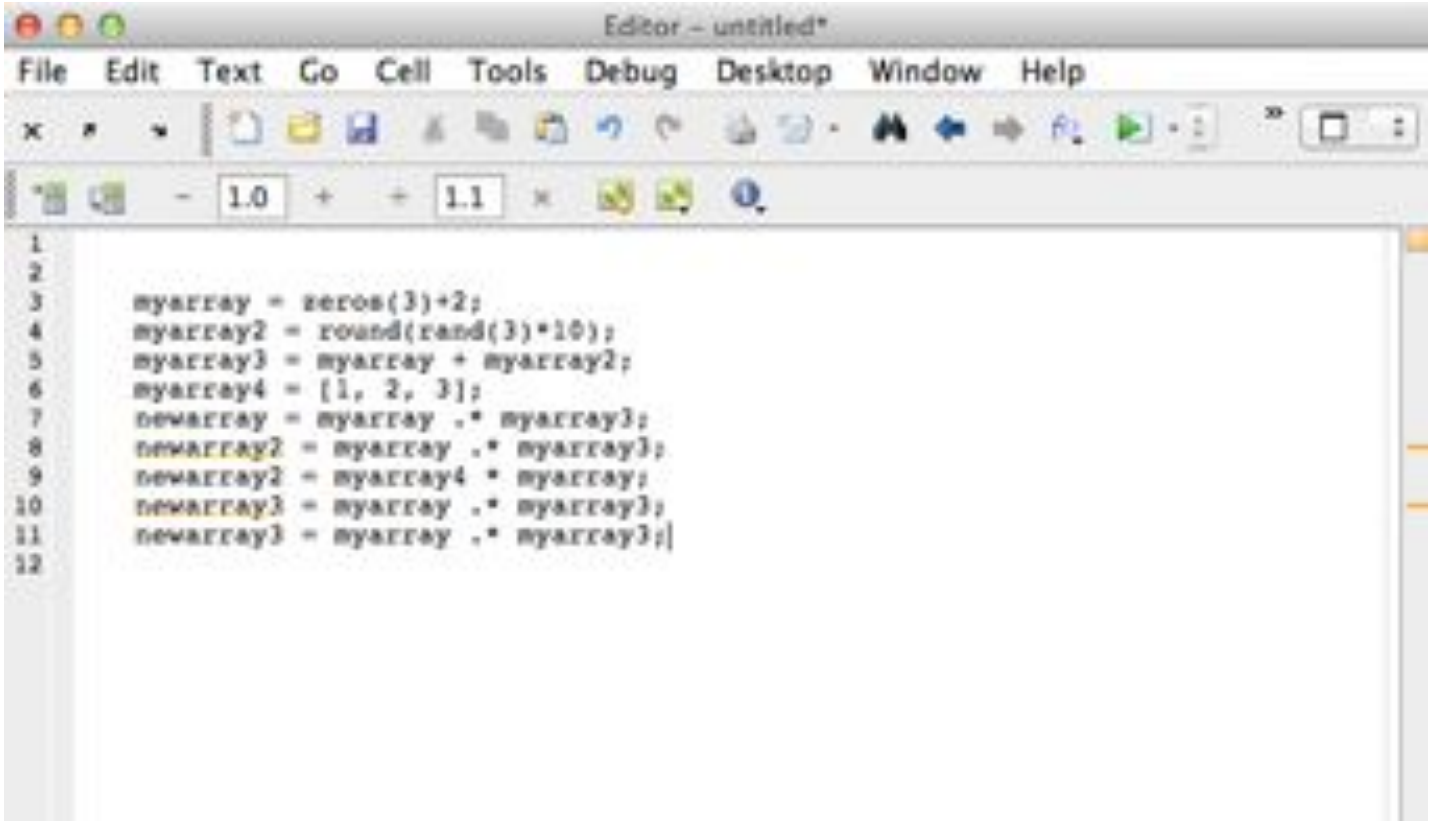
- We can write the MATLAB commands that we type at the command window in a file: extension .m



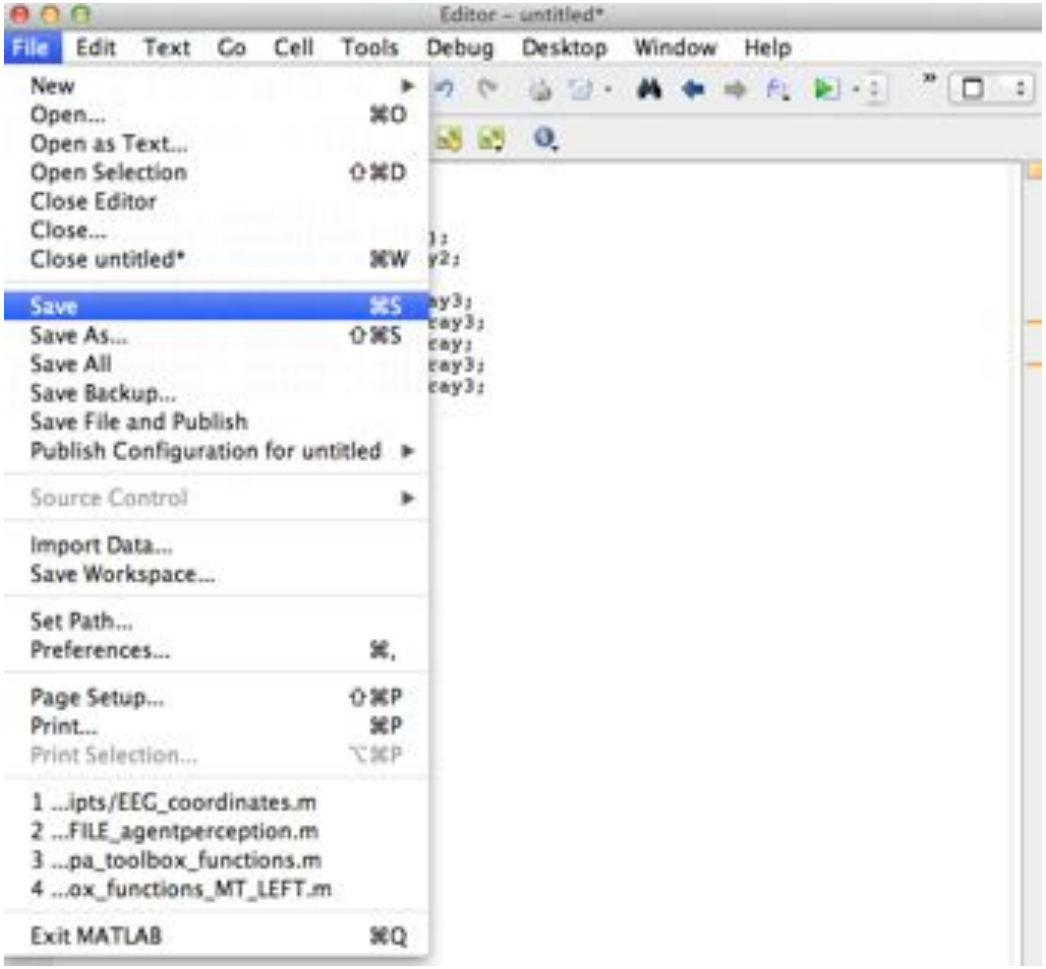
Editor window for the .m file



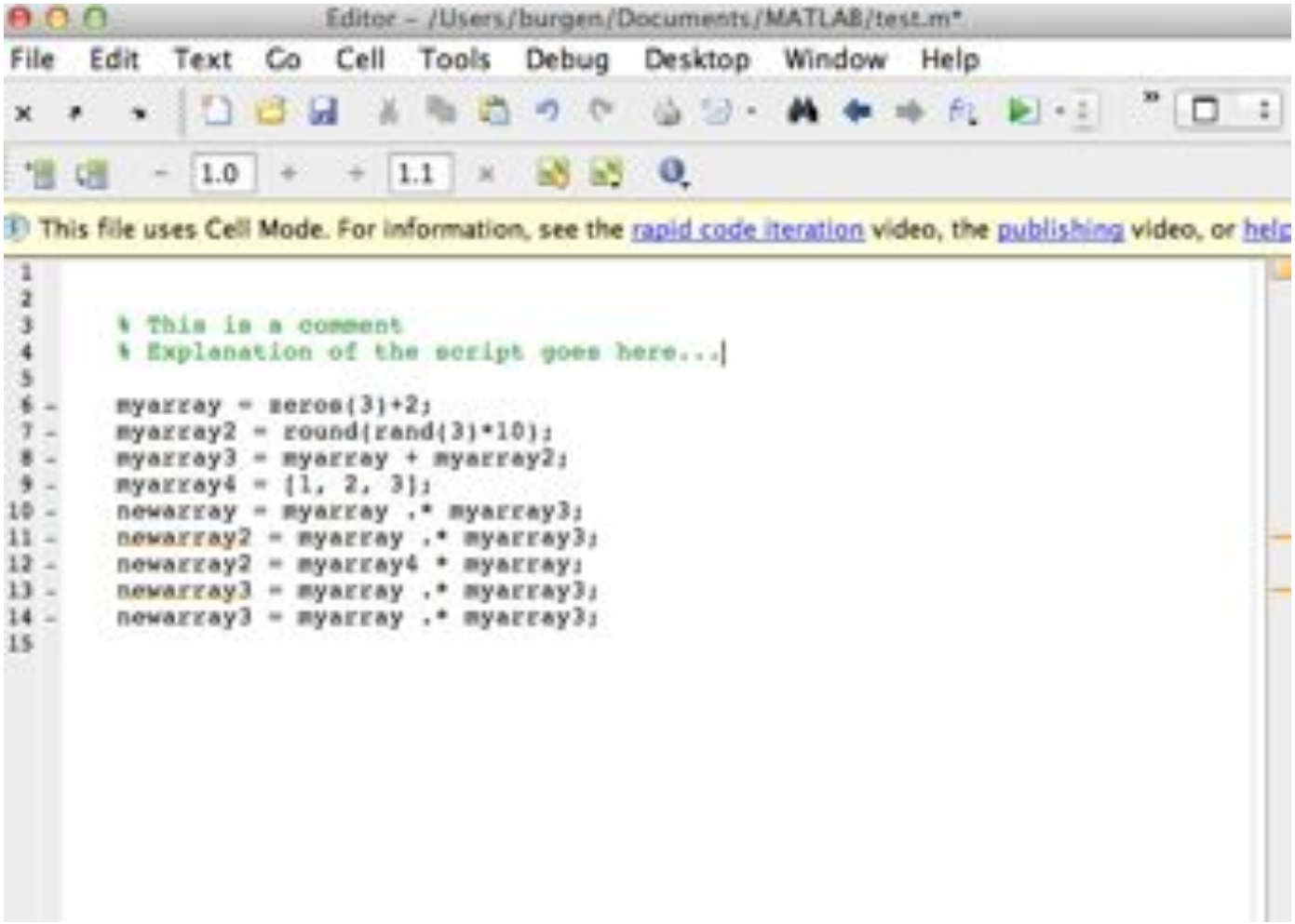
.m files


Editor – untitled*
File Edit Text Go Cell Tools Debug Desktop Window Help
1.0
1.1
myarray = zeros(3)+2;
myarray2 = round(rand(3)*10);
myarray3 = myarray + myarray2;
myarray4 = [1, 2, 3];
newarray = myarray .* myarray3;
newarray2 = myarray .* myarray3;
newarray2 = myarray4 * myarray;
newarray3 = myarray .* myarray3;
newarray3 = myarray .* myarray3;

.m files


Editor – untitled*
File Edit Text Go Cell Tools Debug Desktop Window Help
New
Open... ⌘O
Open as Text...
Open Selection ⇧⌘D
Close Editor
Close...
Close untitled* ⌘W
Save ⌘S
Save As... ⇧⌘S
Save All
Save Backup...
Save File and Publish
Publish Configuration for untitled
Source Control
Import Data...
Save Workspace...
Set Path...
Preferences... ⌘,
Page Setup... ⇧⌘P
Print... ⌘P
Print Selection... ⌥⌘P
1 ...ipts/EEG_coordinates.m
2 ...FILE_agentperception.m
3 ...pa_toolbox_functions.m
4 ...ox_functions_MT_LEFT.m
Exit MATLAB ⌘Q

Comments


Editor – /Users/burgen/Documents/MATLAB/test.m*
File Edit Text Go Cell Tools Debug Desktop Window Help
1.0
1.1
This file uses Cell Mode. For information, see the rapid code iteration video, the publishing video, or help
% This is a comment
% Explanation of the script goes here...
myarray = zeros(3)+2;
myarray2 = round(rand(3)*10);
myarray3 = myarray + myarray2;
myarray4 = [1, 2, 3];
newarray = myarray .* myarray3;
newarray2 = myarray .* myarray3;
newarray2 = myarray4 * myarray;
newarray3 = myarray .* myarray3;
newarray3 = myarray .* myarray3;

FLOW CONTROL

FLOW CONTROL

- In a simple program, the commands are executed one after the other in the order they are typed.

FLOW CONTROL

- In a simple program, the commands are executed one after the other in the order they are typed.

- Many situations require more sophisticated programs in which commands are not necessarily executed in the order they are typed.

FLOW CONTROL

- In a simple program, the commands are executed one after the other in the order they are typed.

- Many situations require more sophisticated programs in which commands are not necessarily executed in the order they are typed.

- Matlab provides several tools:

  1. Conditional statements (if-else)
  2. Loops (for and while loops)

Conditional statements

- A conditional statement is a command that allows MATLAB to make a decision of whether to execute a group of commands that follow the conditional statement, or to skip these commands.

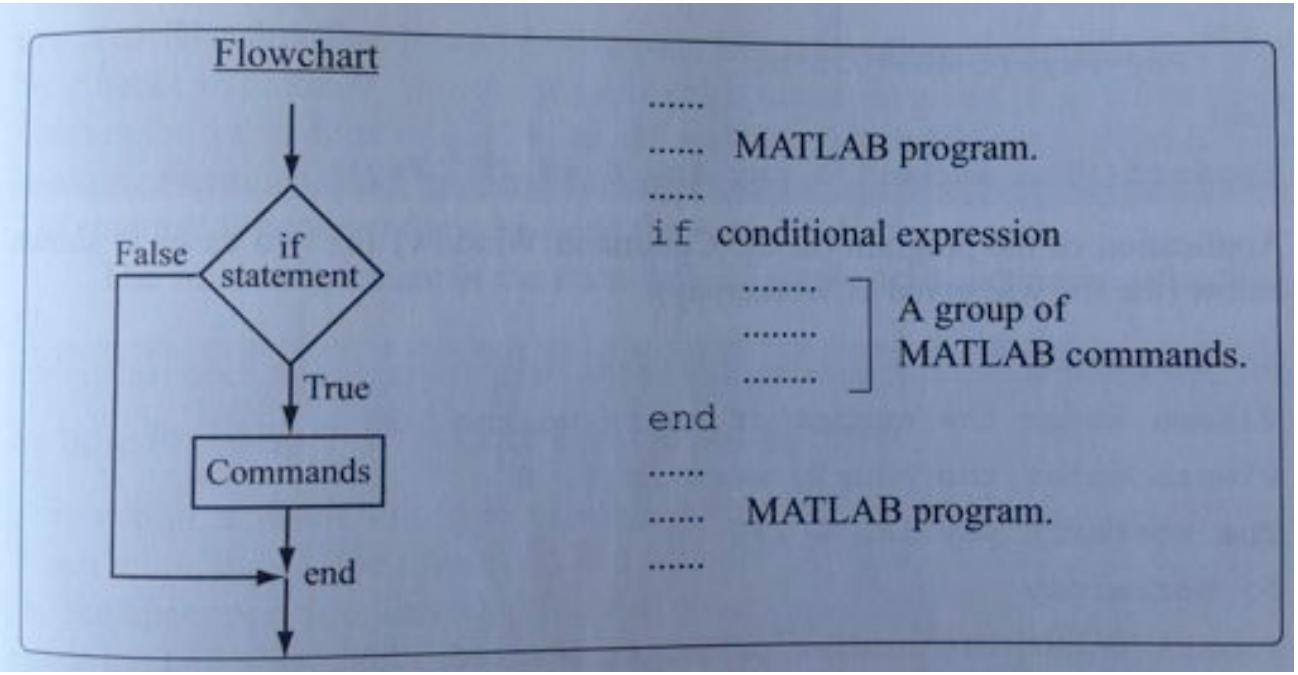
- A conditional statement can be in three forms:

  if – end

  if – else – end

  if – elseif – else - end

if-end statement



If the conditional expression is true, the program continues to execute the commands that follow the if statement.

If the conditional expression is false, the program skips the commands between if and end, and continues to execute the commands that follow end.

Conditional expressions

- Conditional expressions consist of relational and/or logical operators.

Relational Operators	Logical Operators
<	& (AND)
>	| (OR)
<=	~ (NOT)
>=	
==	
=~	

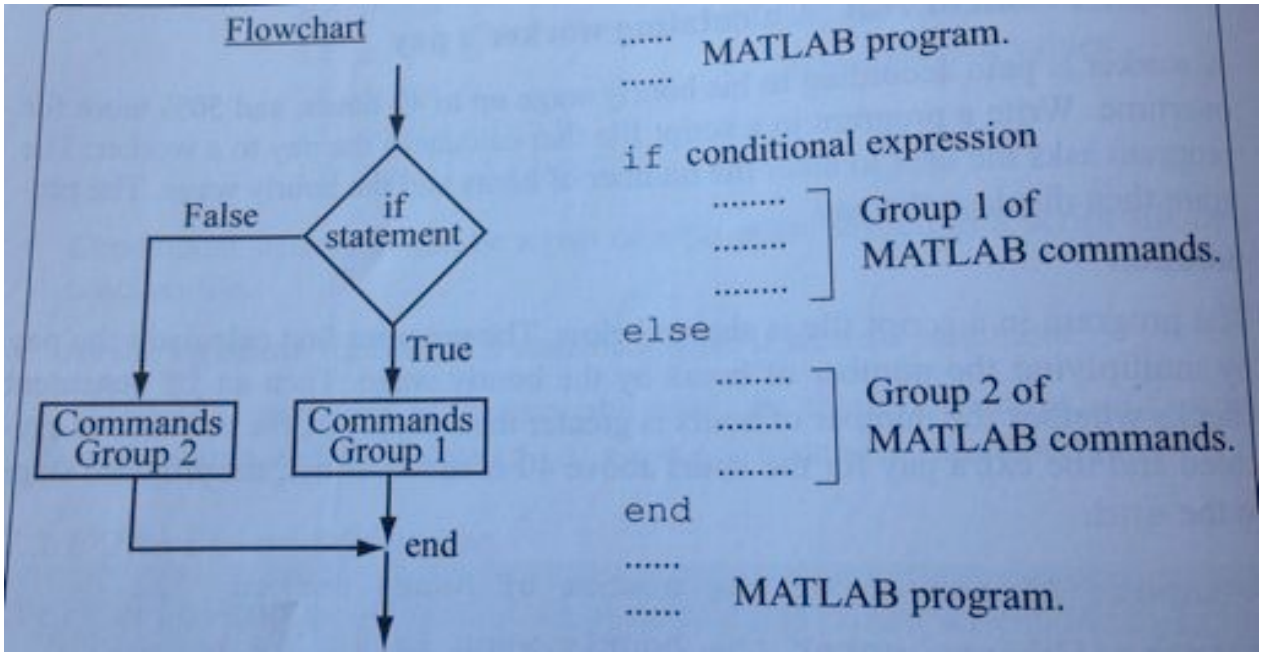
if-end statement

```
x = 5;
y = -3;
s = '';

if x > 0
  s = 'x is a positive number';
end
```

if-end statement

```
x = 5;
y = -3;
s = '';

if x > 0
  s = 'x is a positive number';
end

if x > 0 & y < 0
   s = 'x is positive, and y is negative';
end
```

if-else-end statement



If the conditional expression is true, the program executes group 1 of the commands between if and the else statements and then skips to the end.

If the conditional expression is false, the program skips to the else, and then executes group 2 of commands between the else and the end.

if-else-end statement

```
x = 5;
s = '';

if x > 0
    s = 'x is a positive number';
else
    s = 'x is 0 or a negative number';
end
```

if-else-end statement

```
x = 5;
s = '';

if x > 0
    s = 'x is a positive number';
else
    s = 'x is 0 or a negative number';
end
```

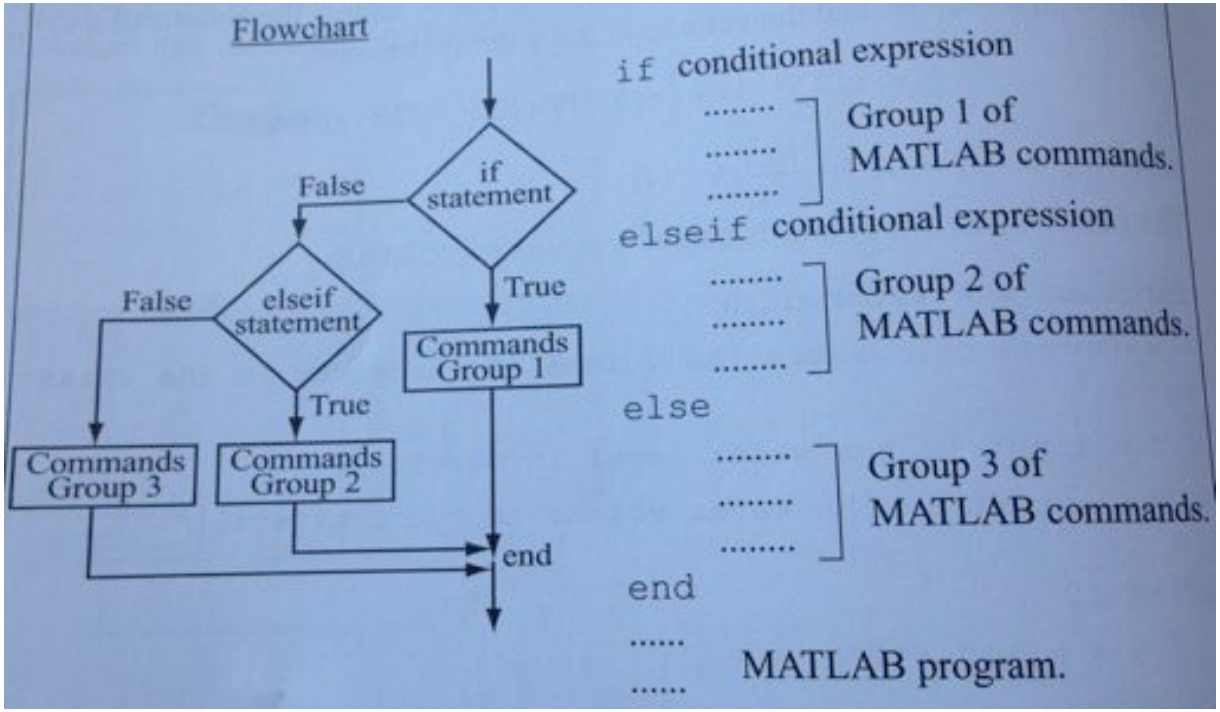
if-else-end statement

```
x = -4;
s = '';

if x > 0
    s = 'x is a positive number';
else
    s = 'x is 0 or a negative number';
end
```

if-else-end statement

```
x = -4;
s = '';

if x > 0
    s = 'x is a positive number';
else
    s = 'x is 0 or a negative number';
end
```

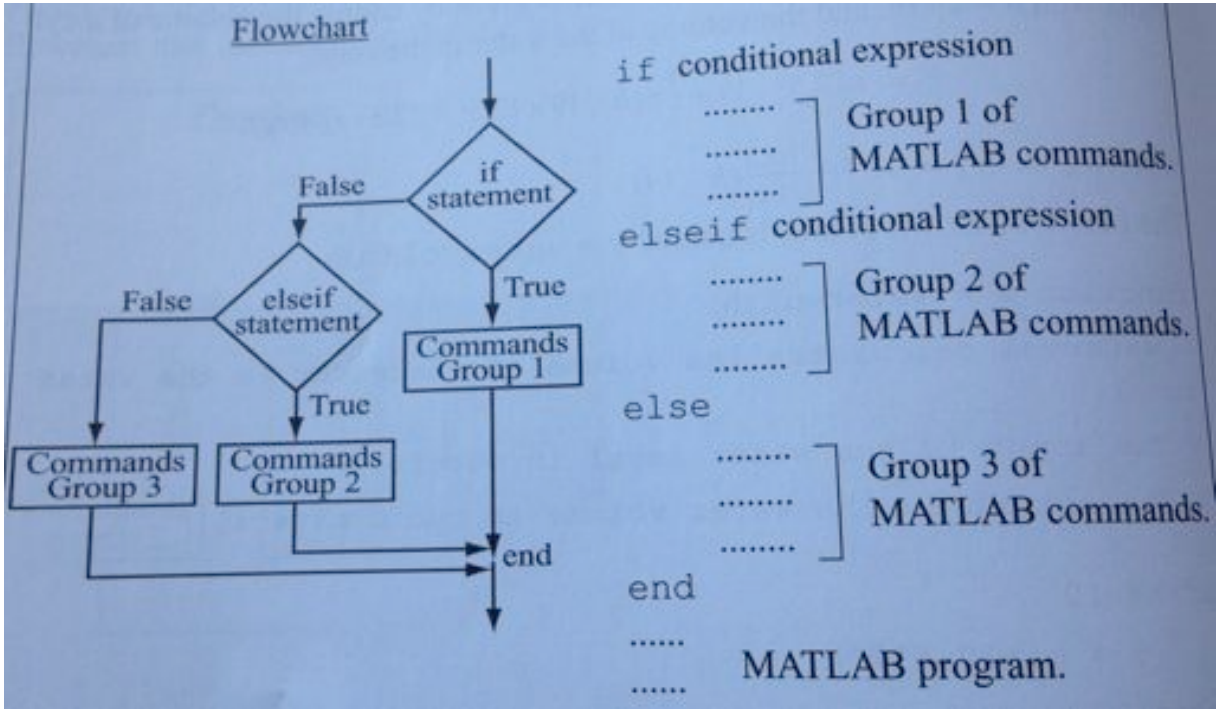
if-elseif-else-end statement



If the conditional expression is true, the program executes group 1 commands between the if and the elseif statement, and then skips to the end.

If the conditional expression in the if statement is false, the program skips to the elseif statement. If the conditional expression in the elseif statement is true, the program executes group 2 of commands between the elseif and the else and then skips to the end.

if-elseif-else-end statement



If the conditional expression in the elseif statement is false, the program skips to the else and executes group 3 of commands between the else and the end.

if-elseif-else-end statement

```
x = 5;
s = '';

if x > 0
    s = 'x is a positive number';
elseif x < 0
    s = 'x is a negative number';
else
    s = 'x is 0.'
end
```

if-elseif-else-end statement

```
x =-3 ;
s = '';

if x > 0
    s = 'x is a positive number';
elseif x < 0
    s = 'x is a negative number';
else
    s = 'x is 0.'
end
```

if-elseif-else-end statement

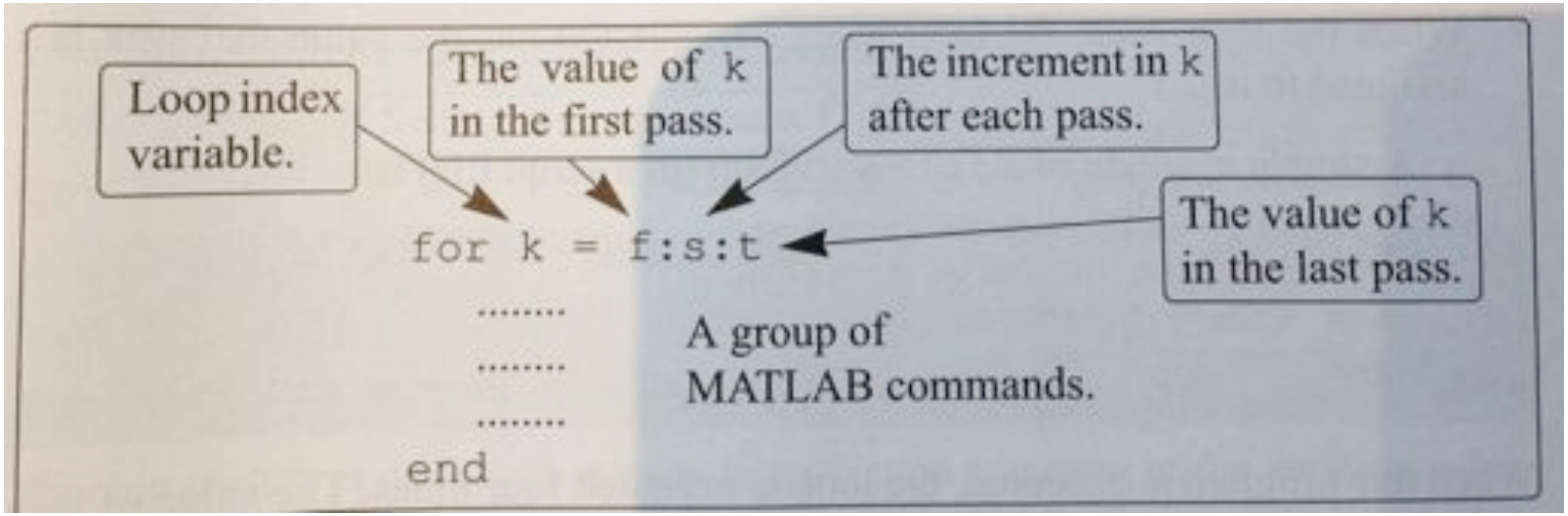
```
x = 0 ;
s = '';

if x > 0
    s = 'x is a positive number';
elseif x < 0
    s = 'x is a negative number';
else
    s = 'x is 0.'
end
```

LOOPS

- Another method to alter the flow of a computer program.
- In a loop, the execution of a command, or a group of commands, is repeated several times consecutively.
- In each pass, at least one variable defined within the loop is/are assigned new values.
- MATLAB has two kinds of loops: for-end loops and while-end loops.

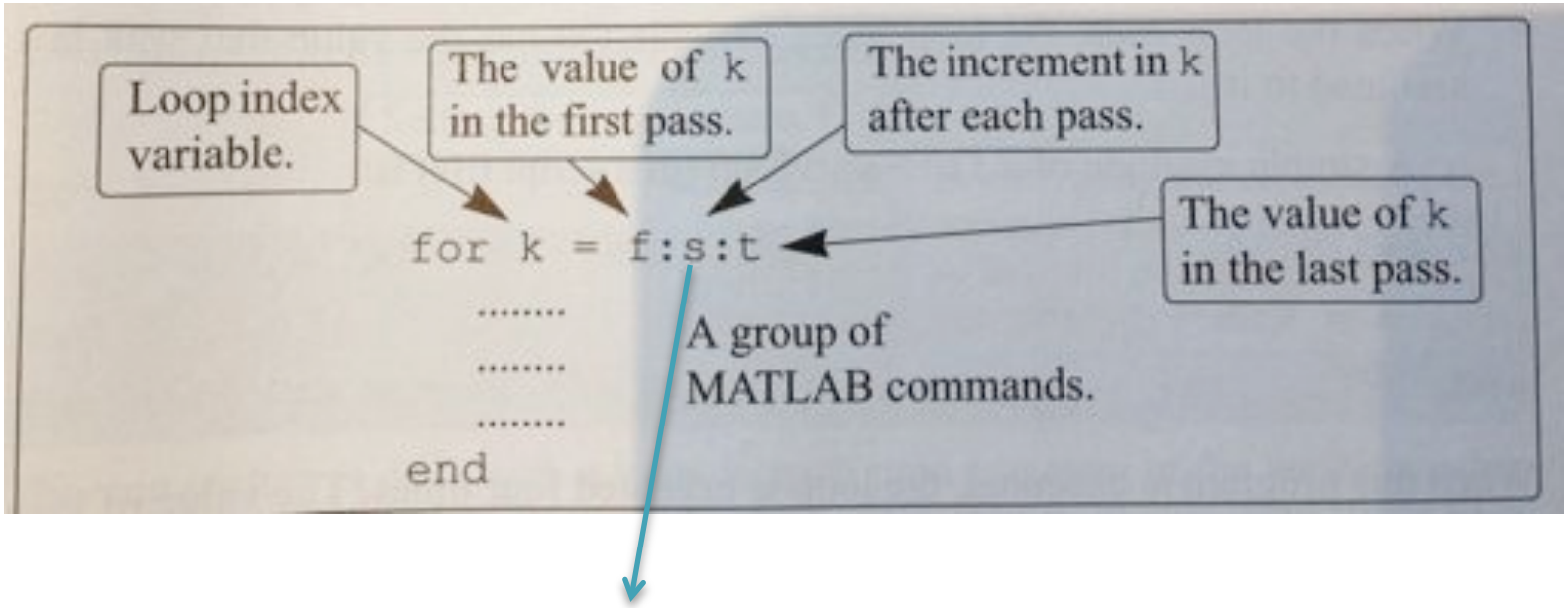
for loops

- The execution of a command, or a group of commands, is repeated a predetermined number of times.



for loops

- The execution of a command, or a group of commands, is repeated a predetermined number of times.



Most of the time, the increment value is omitted.
The default value of s is 1 (increment 1 in each iteration).

for loops: Example

```
for k = 1:4
    x = k^2;
end
```

for loops: Example

```
for k = 1:4
    x = k^2;
end
```

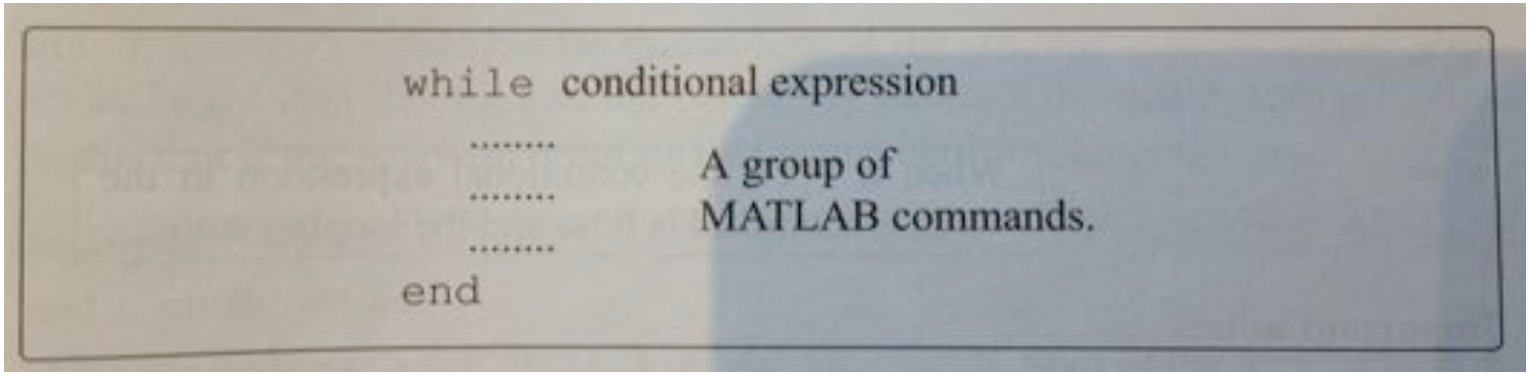
The loop is executed 4 times. Here is how the loop works:

1. In the first iteration, k = 1. Then the program executes x = 1^2, which is 1
2. In the second iteration, k = 2. Then the program executes x = 2^2, which is 4.
3. In the third iteration, k = 3. Then the program executes x = 3^2, which is 9.
4. In the fourth iteration, k = 4. Then the program executes x = 4^2, which is 16.

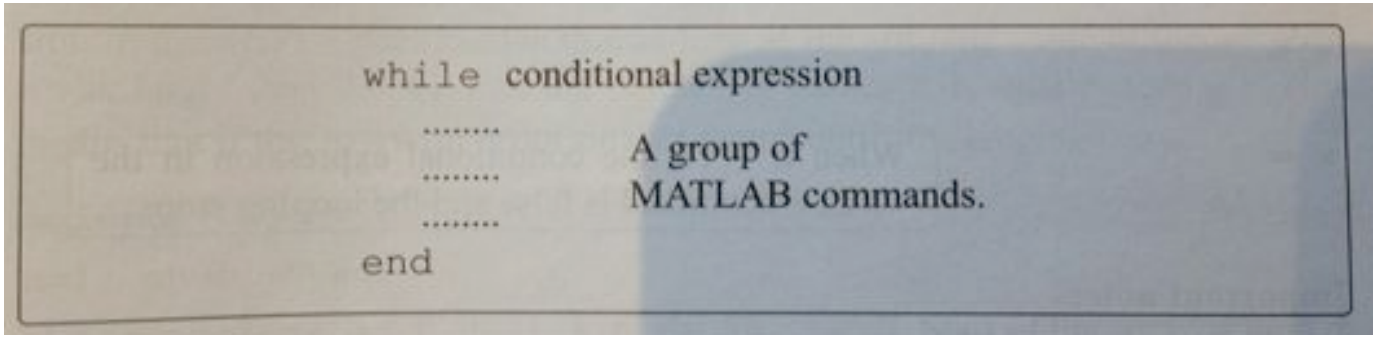
while loops

- while loops are used in situations when looping is needed but the number of iterations is not known ahead of time.

- In while loops, the number of iterations is not specified when the looping starts. Instead, the looping process continues until a stated condition is satisfied.



while loops



Conditional expression is checked.

If it is false, MATLAB skips to the end statement and continues with the program.

If it is true, MATLAB executes the group of commands that follow between the while and end command.

Then, MATLAB jumps back to the while command and checks the conditional expression.

This looping process continues until the conditional expression is false.

while loops: Example

```
x = 1;
while x <= 7
        x = 2*x ;
end
```

while loops: Example

```
x = 1;
while x <= 7
        x = 2*x ;
end
```

Here is how the program above works:

1. x is assigned to 1.
2. The conditional statement x<=7 is checked. Since it is true (1<=7), the following command x = 2*x is executed, which makes x = 2.
3. It jumps back to while statement and checks the conditional statement. Since 2 <= 7, the following command, x = 2*x is executed, which makes x = 4.
4. It jumps back to the while statement and checks 4 <= 7. Since it is true, it executes x = 2*x, which makes x = 8.
5. It jumps back to the while statement and checks 8 <= 7. Since it is false, it jumps to the end without executing the statements between while and the end.

Note about while loops

- You have to be sure that the variable (or variables) that are in the conditional expression are assigned new values during the looping process, and will eventually be assigned values that make the conditional expression false.

- Otherwise, the looping will continue indefinitely.

Note about while loops

- You have to be sure that the variable (or variables) that are in the conditional expression are assigned new values during the looping process, and will eventually be assigned values that make the conditional expression false.

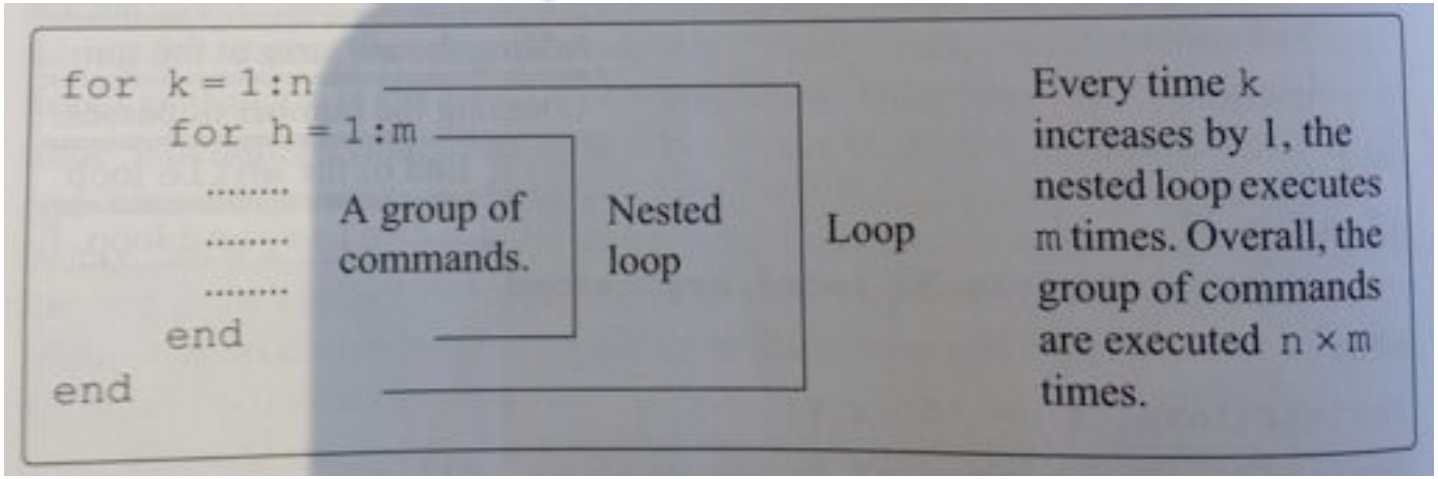
- Otherwise, the looping will continue indefinitely.
- For instance, consider

```
x = 3;
while x > 2
        x = 2*x ;
end
```

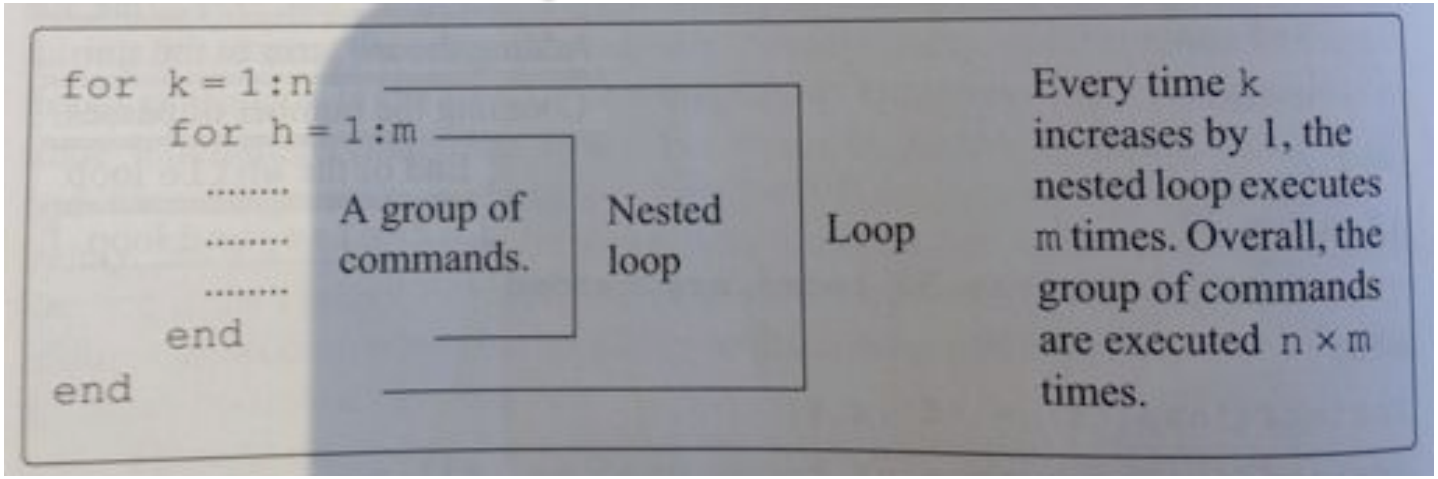
Nested Loops and conditional statements

- Loops and conditional statements can be nested within themselves and each other.
- This means that a loop and/or a conditional statement can start (and end) within another loop and/or conditional statement.
- There is no limit to the number of loops and conditional statements that can be nested.

Nested loops



Nested loops



For example, if n = 3 and m = 4:

1. k = 1, and the nested loop executes 4 times with h = 1, 2, 3, 4.
2. k = 2, and again the nested loop executes 4 times with h = 1, 2, 3, 4.
3. k = 3, and again the nested loop executes 4 times with h = 1, 2, 3, 4.

break command

- When inside a loop (for and while), the break command terminates the execution of the loop (the whole loop, not just the last iteration).
- MATLAB jumps to the end command of the loop and continues with the next command.
- If the break command is inside a nested loop, only the nested loop is terminated.
- When a break command appears outside of a loop in a script file, it terminates the execution of the file.

continue command

- The continue command can be used inside a loop (for and while) to stop the present iteration and start the next iteration in the looping process.

- The continue command is usually a part of a conditional statement. When MATLAB reaches the continue command, it does not execute the remaining commands in the loop, but skips to the end command of the loop, and then starts a new iteration.