AN INTRODUCTION TO MATLAB

Jie Zhang

Feb. 14, 2013 to April 09, 2013

Adopted from the Presentation by

Joseph Marr, Hyun Soo Choi, and Samantha Fleming
George Mason University
School of Physics, Astronomy and Computational Sciences

2011, version 1

NOTE: Subject to frequent revisions!

Last Revision:
February 11, 2013

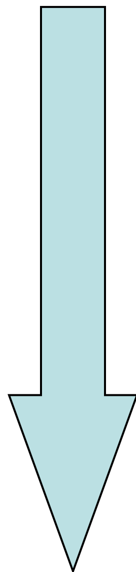
Table of Contents

CHAPTER 1: Prologue (why MATLAB)
CHAPTER 2: The MATLAB environment
CHAPTER 3: Assignment, Variables, and Intrinsic Functions
CHAPTER 4: Vectors and Vector Operations
CHAPTER 5: Matrices (Arrays) and Matrix Operations
CHAPTER 6: Iteration I: FOR Loops
CHAPTER 7: Basic Graphs and Plots
CHAPTER 8: Writing a Matlab Program
CHAPTER 9: Iteration II: Double Nested FOR Loops (DNFL)
CHAPTER 10: Conditionals: IF Statements

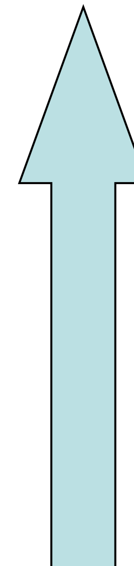
CHAPTER 1

PROLOGUE

FIRST…WHERE IS MATLAB W/R/T PROGRAMMING LANGUAGES?



- **Matlab** (“scripting language”)
- Python
- Java
- Fortran, C, C++
- Assembly language
- Machine code (binary!)



Increasing
program size
(fairly objective)

Increasing
ease-of-use
(fairly subjective)

WHY MATLAB?

"*Softer*" Reasons . . .

- Most scientists and engineers know it
- Widely used in industry, academia and government
- Easy to learn (it's a "scripting" language)
- **LOTS** of pre-developed application packages
- Very good technical support/user community

"*Harder*" Reasons . . .

- Excellent graphics capabilities
- Includes many modern numerical methods
- Optimized for scientific modeling
- Usually, ***FAST*** matrix operations
- Interfaces well with other languages

GEORGE MASON UNIVERSITY

WHAT WE PLAN TO STUDY: The Building Blocks of Programs

Sequence
- ASSIGNMENT: single variables
- ASSIGNMENT: vectors and matrices (arrays)

Repetition
- ITERATION (FOR loops, WHILE loops, and the all-important double nested FOR loops – DNFL)

Selection
- SELECTION (IF statements, single and multi-way)

WHAT THE EXPERTS HAVE SAID:

"The only way to learn a new programming language is by writing programs in it."

Brian Kernighan & Dennis Richie
"*The C Programming Language*"

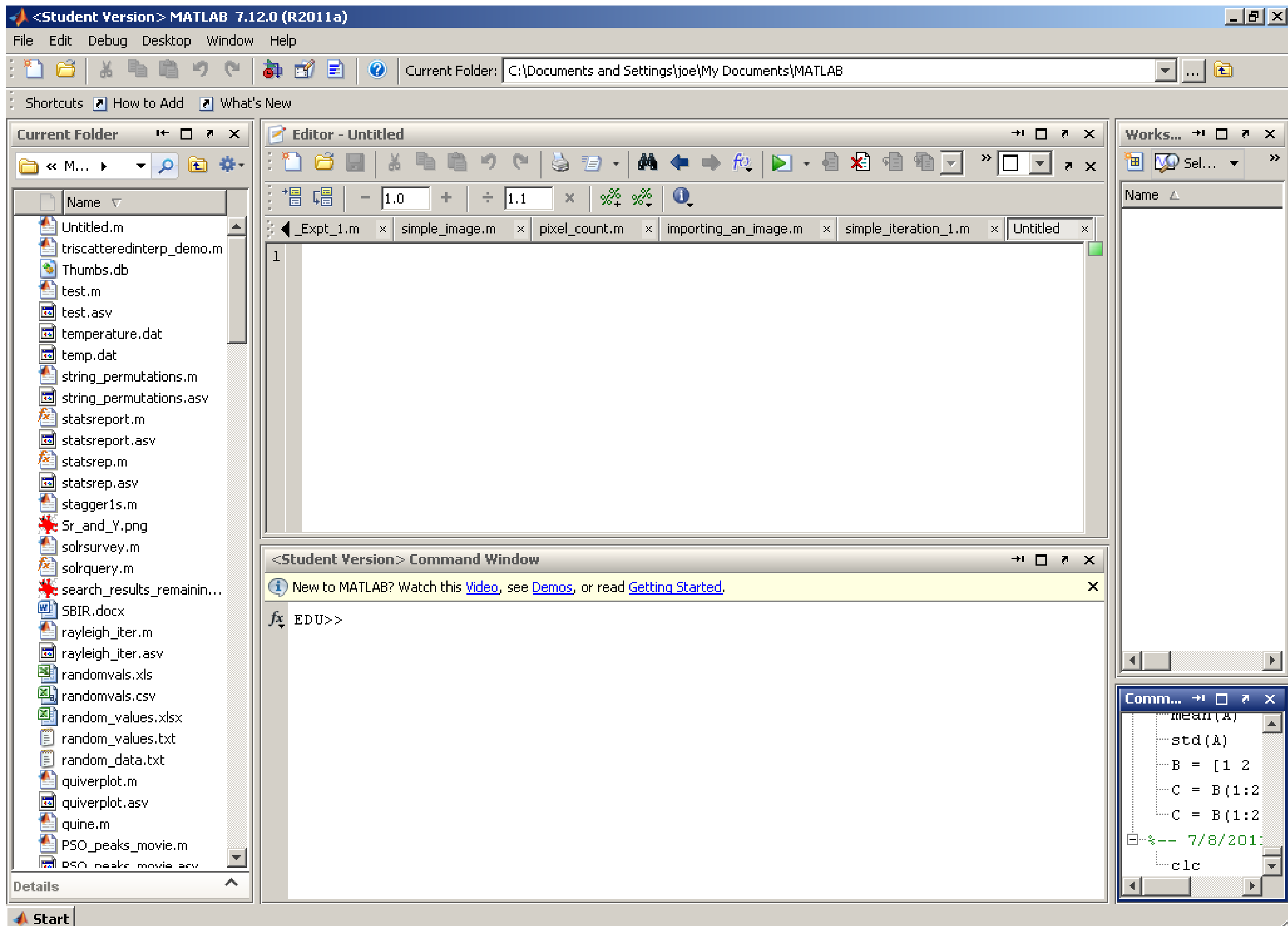
[NOTE: red emphasis added]

GEORGE MASON UNIVERSITY

CHAPTER 2

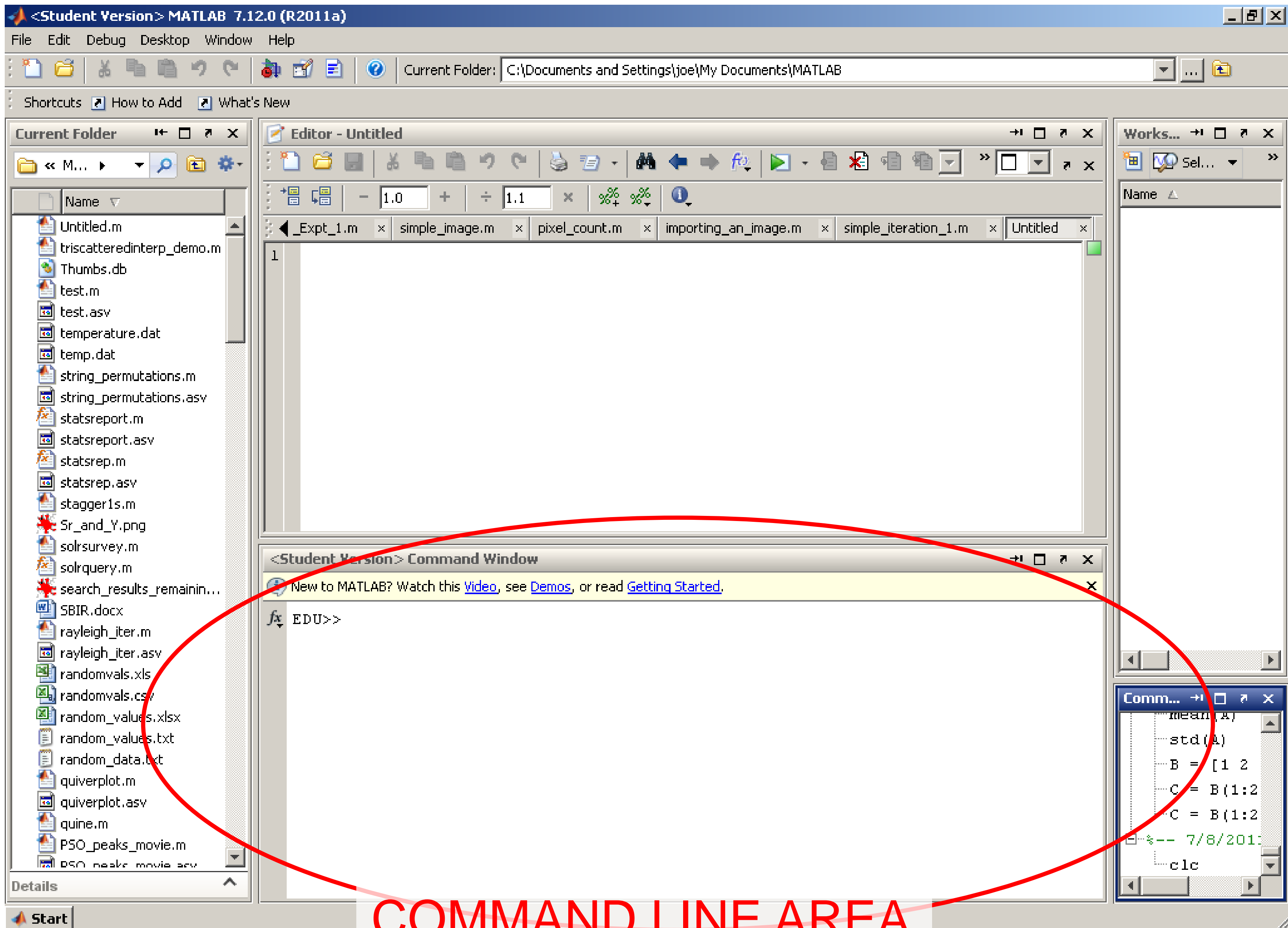
THE MATLAB ENVIRONMENT

THE MATLAB DESKTOP



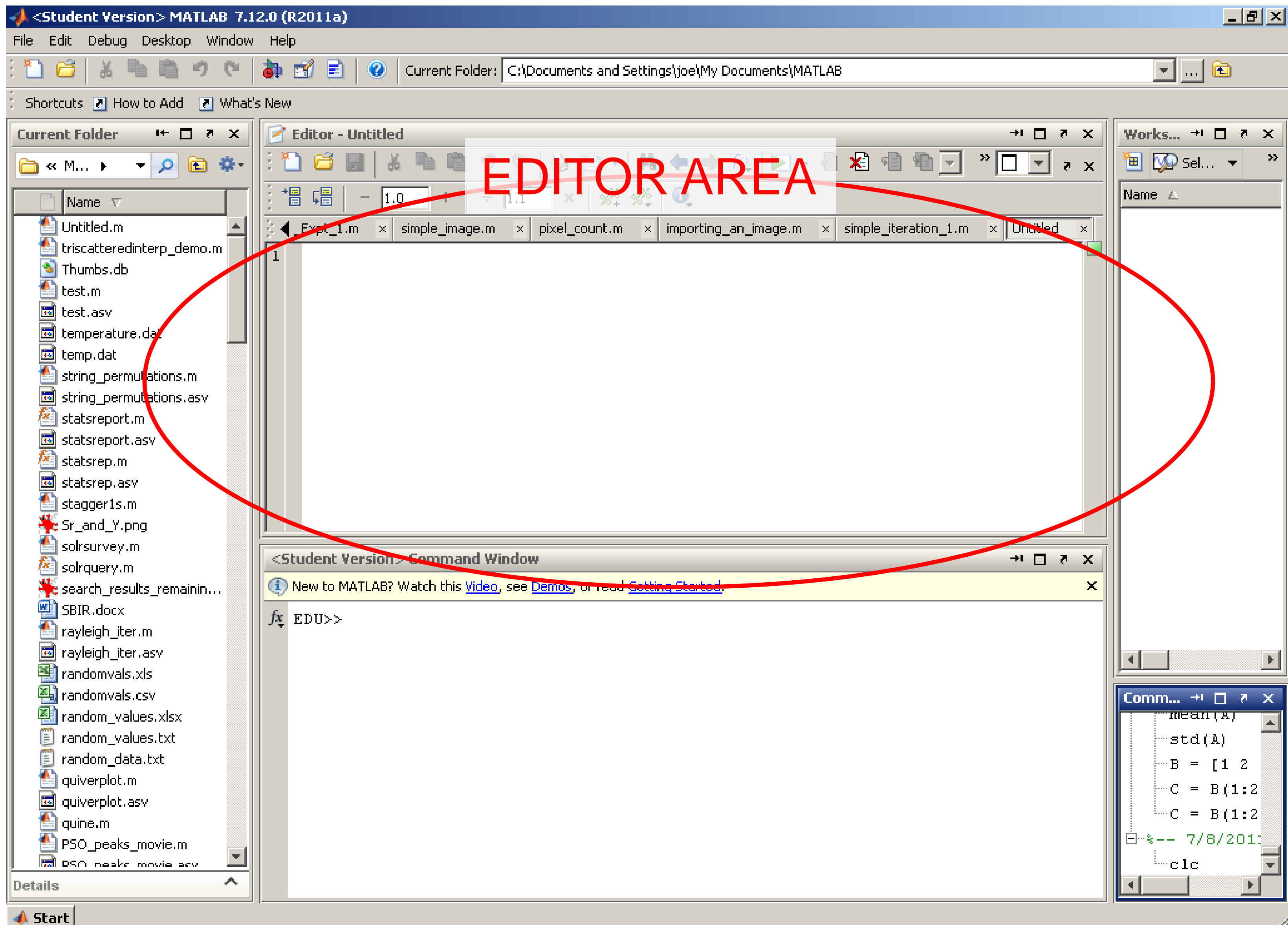
GEORGE MASON UNIVERSITY

THE MATLAB DESKTOP

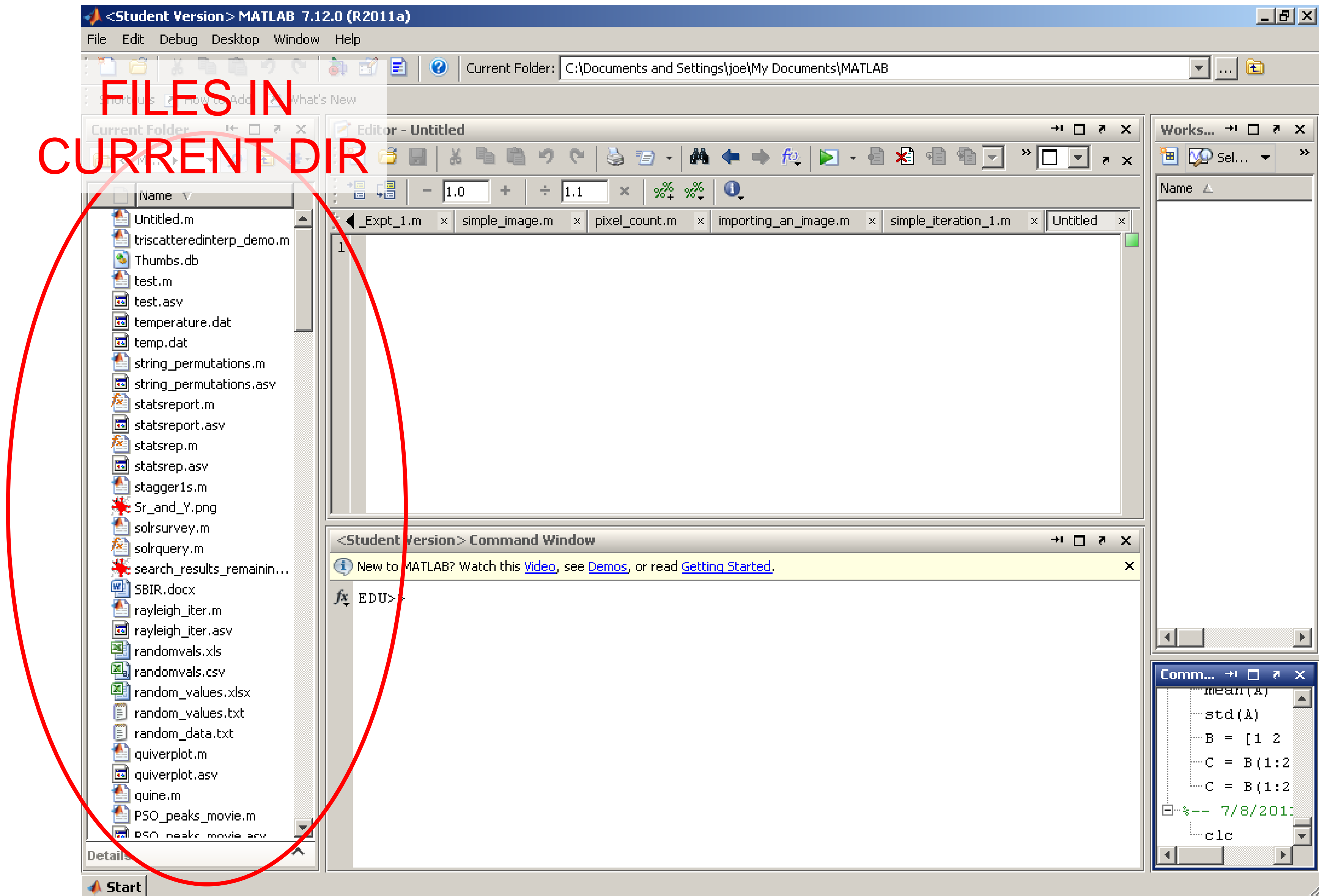


GEORGE MASON UNIVERSITY

THE MATLAB DESKTOP

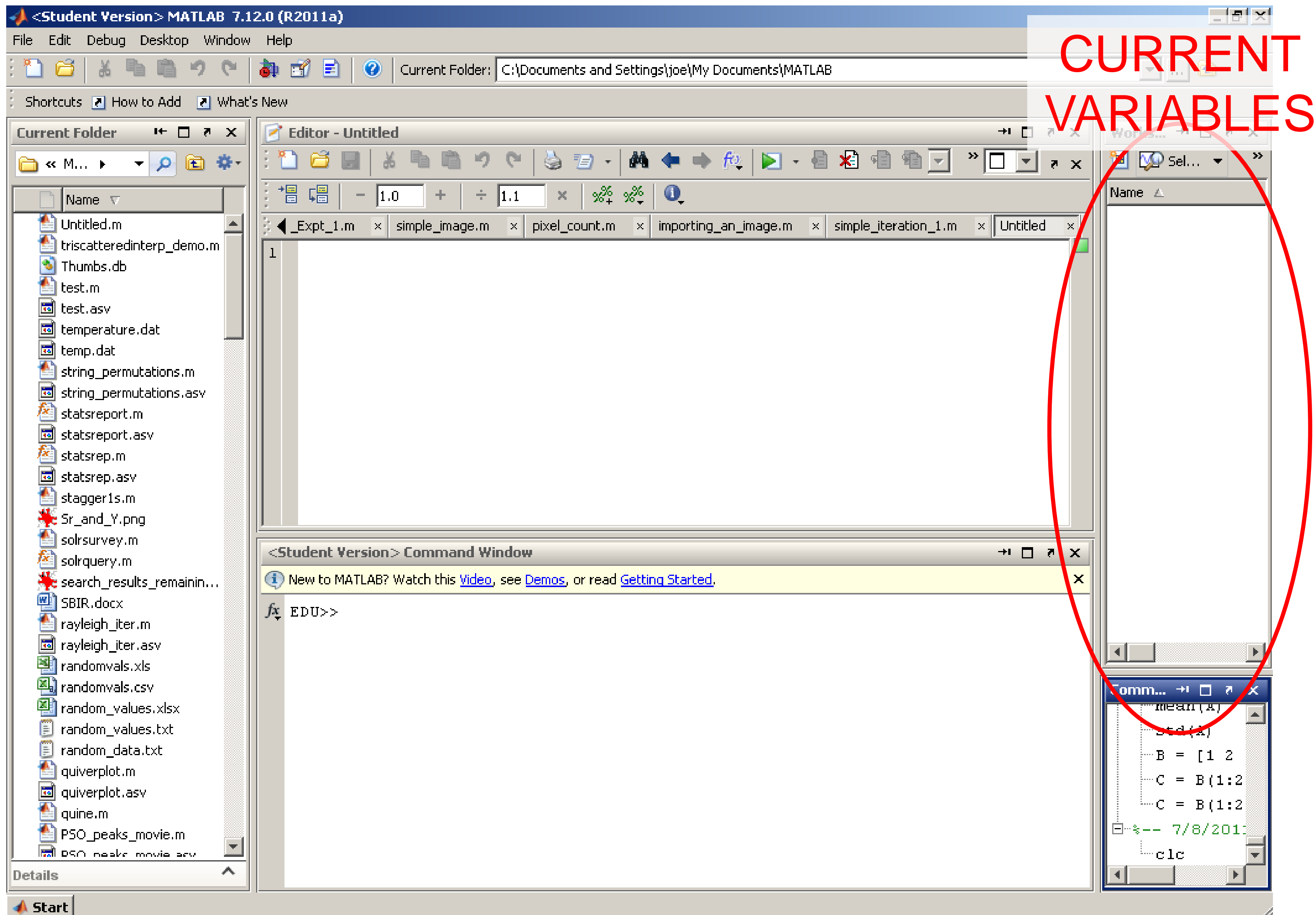


THE MATLAB DESKTOP



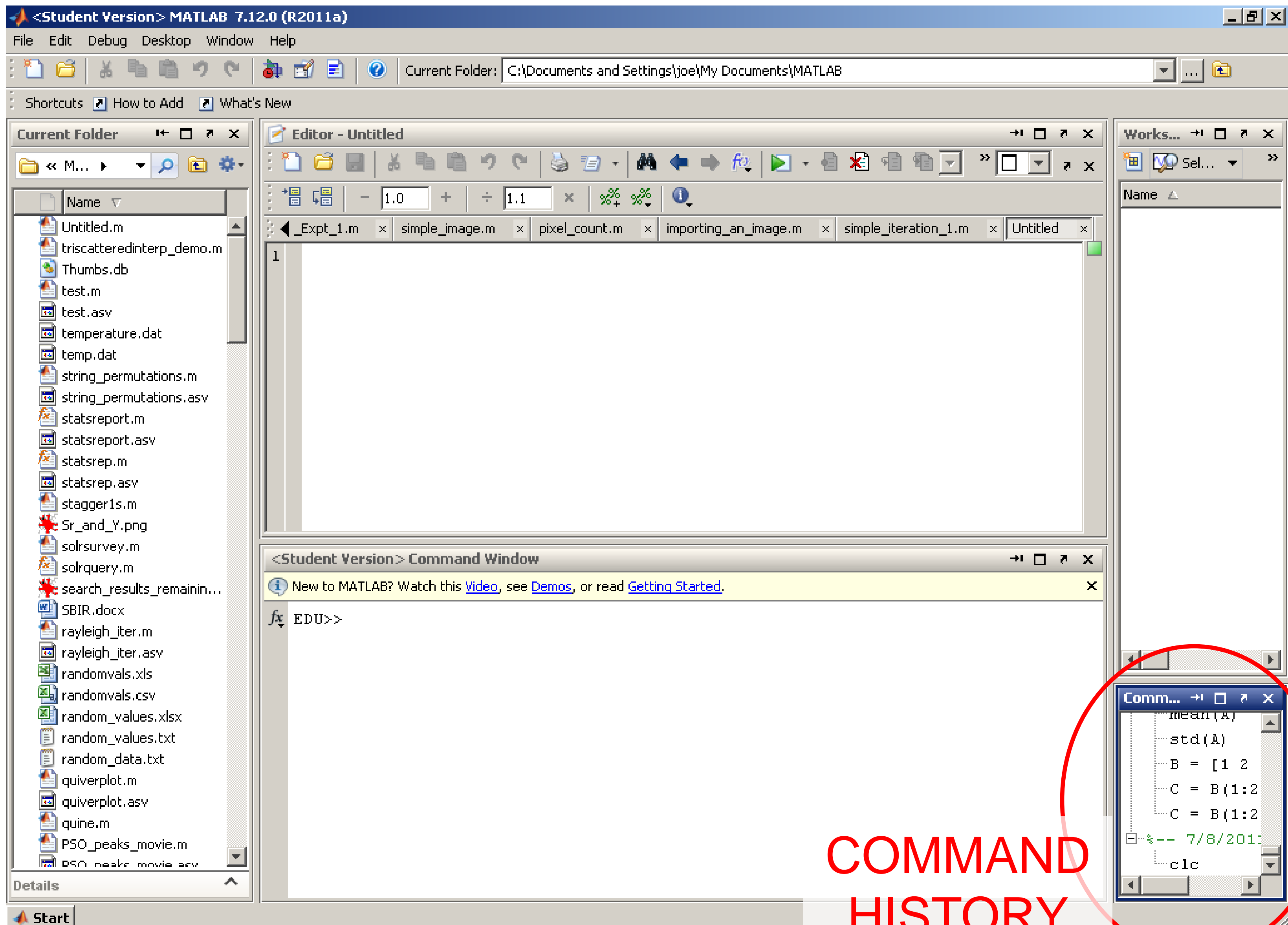
GEORGE MASON UNIVERSITY

THE MATLAB DESKTOP



GEORGE MASON UNIVERSITY

THE MATLAB DESKTOP



CHAPTER 3

ASSIGNMENT and INTRINSIC FUNCTIONS

GEORGE MASON UNIVERSITY

ASSIGNMENT: VARIABLES

• Think of a labeled box. We can put "stuff" inside that box. "Stuff" here means ***values***.

• Box labels have names written on them.

• Those names enable us to refer to specific boxes, at a later time, as needed – to go straight to them.

• These ***labeled boxes*** are what we call ***variables***. A variable is a labeled memory box.

• Putting "stuff" inside that box is called ***assigning a value to a variable***, or simply, ***assignment***.

ASSIGNMENT: VARIABLES

- “MATLAB variable names must begin with a letter, which may be followed by any combination of letters, digits, and underscores. MATLAB distinguishes between uppercase and lowercase characters, so `A` and `a` are not the same variable.” (http://www.mathworks.com/help/techdoc/matlab_prog/f0-38052.html)

- Valid variable names:
  `A,   a,   Aa,   abc123,   a_b_c_123`

- Invalid variable names:
  `1A, _abc, ?variable, abc123?`

ASSIGNMENT: VARIABLES

- To do assignment in Matlab:
    1. Write a variable name
    2. Write the "=" symbol
    3. Write the value that you want to store in the variable

- Examples:

```
A = 5                          (an integer value)
a123 = 1.0                     (a floating point value)
abc_123 = 1.0e-02              (an exponential value)
myVariable = 'Joe'             (a string value)
```

ASSIGNMENT: VARIABLES

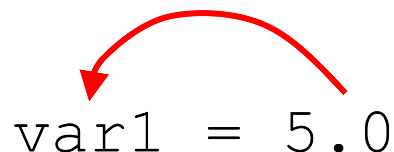
Rules of Assignment:

- The "=" symbol ***DOES NOT MEAN EQUALS***! It means assignment: Assign the ***value on the right*** of the "=" symbol to the ***variable on the left*** of the "=" symbol.
- To access what's "in the box"—that is, the value currently held by the variable—simply type the name of the variable alone on a line.

ASSIGNMENT: VARIABLES

REMEMBER:

- **Value on the right of "="** gets stored into **variable on the left of "=":**



```
var1 = 5.0
```

- Example: Valid assignment (creates var2, assigns it the **value contained in var1**):

```
var2 = var1
```

- Example: Invalid assignment (generates error: **var3 not previously declared** – <u>holds no value</u>)

```
var2 = var3
```

ASSIGNMENT: VARIABLES

- **Rules of Assignment** (cont):
  - Variables can be used in assignment statements to assign values to other variables: Since placing a variable on the right side of "=" retrieves its current value, we can subsequently assign that value to yet another variable:

var1 = 3.0 (assigns the value 3.0 to var1)
var2 = var1 (retrieves 3.0 from var1 and stores that value into var2)

  - We can do math with variables, too:

Examples:
```
var3 = var2 + var1    (here, 3.0 + 3.0)
var4 = var3 * var2    (here, 6.0 * 3.0)
var5 = var4 / var1    (here, 18.0 / 3.0)
var6 = var2 ^ var1    (here, 3.0 ^ 3.0)
```

ASSIGNMENT: VARIABLES

- **Rules of Assignment** (cont):

  - We can also "update" a variable by constantly reassigning new values to it. Updating a variable by adding 1 to it, and then assigning the new value back into the same variable is called "incrementing". Example:

```
var7 = 1
var7 = var7 + 1
```

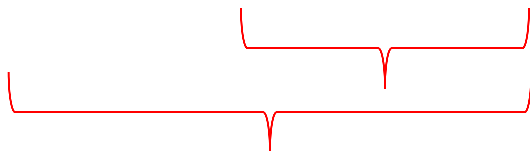
Incrementing a variable is a VERY IMPORTANT thing to do, because, doing so enables us to **count** effectively.

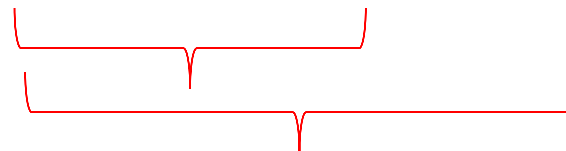
ASSIGNMENT: VARIABLES

- **Rules of Arithmetic Operator Precedence**:
  - The preceding arithmetic examples raise a question: In what order are the arithmetic operations performed in an assignment statement?
  - Like in algebra: Anything in parentheses first, followed by exponentiation, followed by multiplication/division, followed by addition/subtraction

Examples:

```
var3 = var2 + var1 * var4     var3 = (var2 + var1) * var4
```

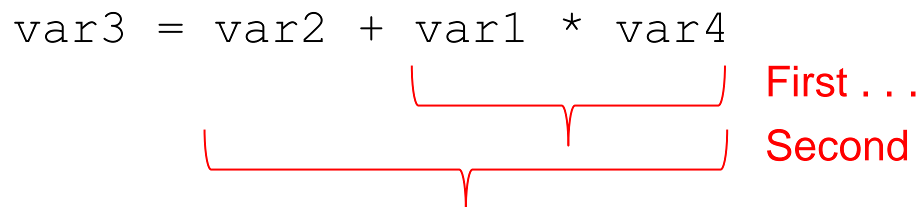


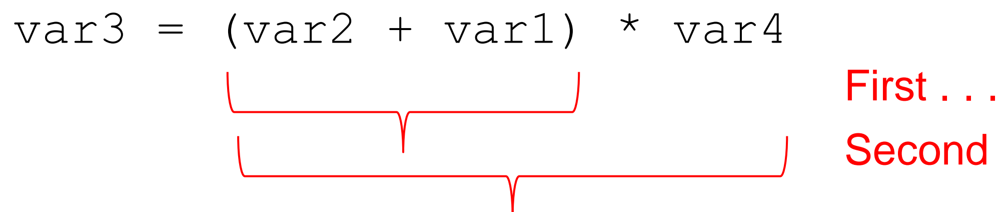


ASSIGNMENT: VARIABLES

- **Rules of Arithmetic Operator Precedence :**

Examples:





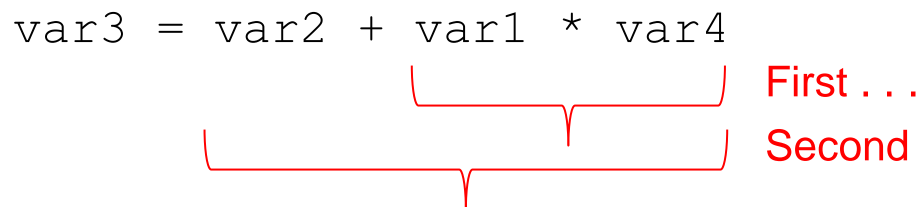
But what if the operators are of equal precedence?

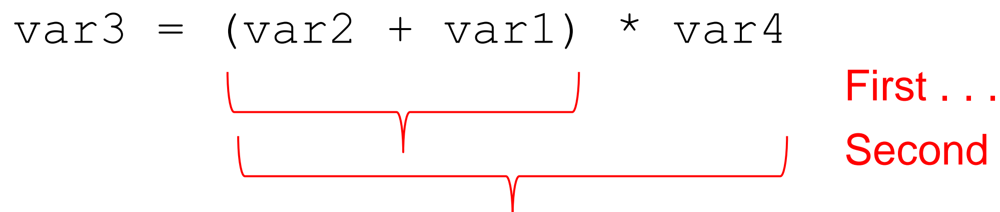
```
var3 = var2 / var1 * var4

???
```

ASSIGNMENT: VARIABLES

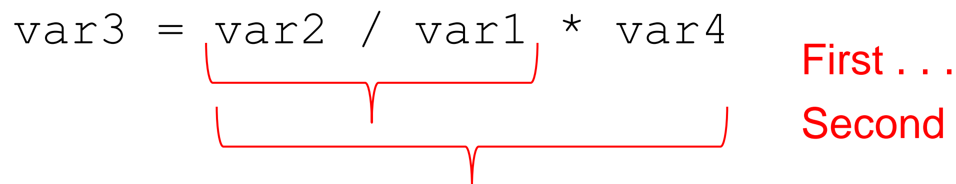
- **Rules of Arithmetic Operator Precedence :**

  Examples:





When operators are of equal precedence, associate **LEFT TO RIGHT:**



ASSIGNMENT: VARIABLES

- **Rules of Arithmetic Operator Precedence** :

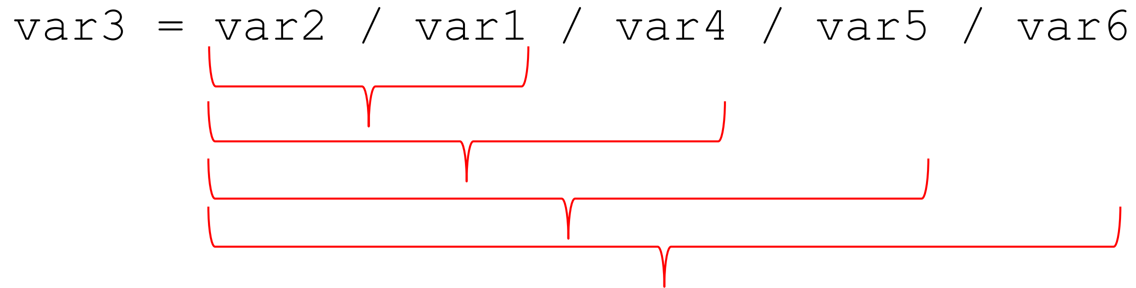
Examples:

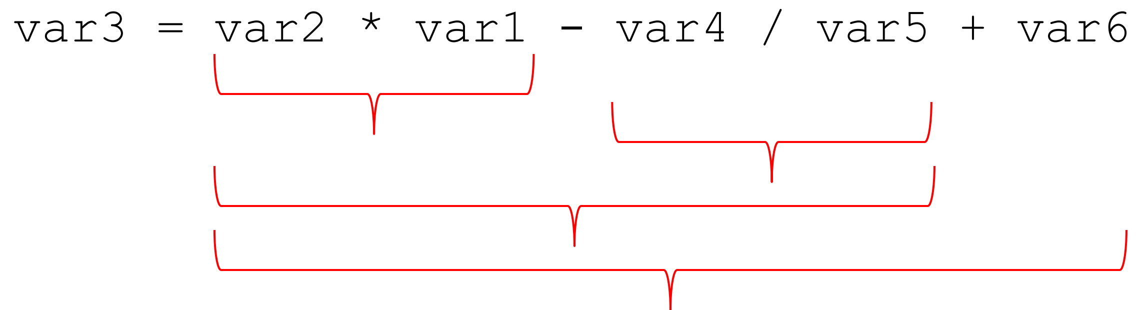
```
var3 = var2 / var1 / var4 / var5 / var6

                  ???


var3 = var2 * var1 - var4 / var5 + var6

                  ???


var3 = var2 / var1 * var4 / var5 ^ var6

                  ???
```

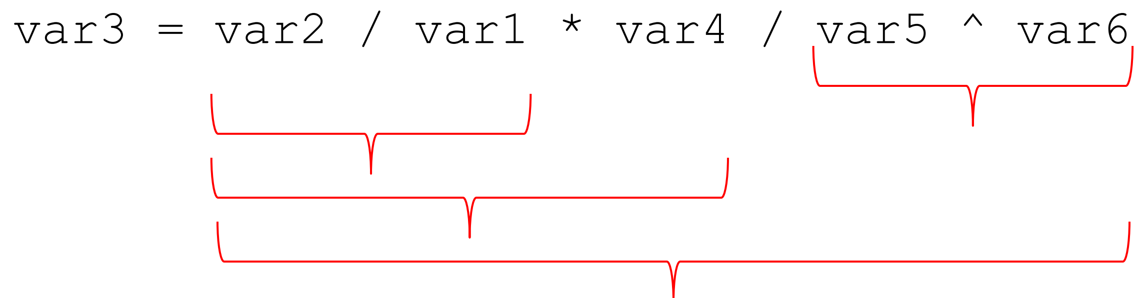
ASSIGNMENT: VARIABLES

- **Rules of Arithmetic Operator Precedence :**

Examples:







George Mason University

APPEARANCE OF OUTPUT

- We can change the way numbers are printed to the screen by using Matlab's "format" command, followed by the appropriate directive, "short" or "long" (the format directive is *persistent*!)

```
>> pi
>> ans = 3.1416

>> format short
>> pi
>> ans = 3.1416

>> format long
>> pi
>> ans = 3.141592653589793

>> sqrt(2)
>> ans = 1.414213562373095
```

GEORGE MASON UNIVERSITY

SOME BUILT-IN FUNCTIONS

Here are a few of Matlab's built-in (intrinsic) functions, to get you started:

π:

```
>> pi
>> ans = 3.1416
```

sine(x):

```
>> sin(pi)
>> ans = 1.2246e-016
```

e^N :

```
>> exp(1)
>> ans = 2.7183
```

$\sqrt{N}$:

```
>> sqrt(2)
>> ans = 1.4142
```

cosine(x):

```
>> cos(pi)
>> ans = -1
```

natural log (N):

```
>> log(2)
>> ans = 0.6931
```

remainder ($\frac{a}{b}$):

```
>> mod(5,2)
>> ans = 1
```

tangent(x):

```
>> tan(pi)
>> ans = 1.2246e-016
```

base 10 log (N):

```
>> log10(2)
>> ans = 0.3010
```

GEORGE MASON UNIVERSITY

SOME BUILT-IN FUNCTIONS

We can use Matlab's built-in functions on the right hand side of an assignment statement, to produce a value that we then assign to a variable:

π:

```
>> x = pi
>> x = 3.1416
```

sine(x):

```
>> x = sin(pi)
>> x = 1.2246e-016
```

e^N :

```
>> x = exp(1)
>> x = 2.7183
```

$\sqrt{N}$:

```
>> x = sqrt(2)
>> x = 1.4142
```

cosine(x):

```
>> x = cos(pi)
>> x = -1
```

natural log (N):

```
>> x = log(2)
>> x = 0.6931
```

remainder ($\frac{a}{b}$):

```
>> x = mod(5,2)
>> x = 1
```

tangent(x):

```
>> x = tan(pi)
>> x = 1.2246e-016
```

base 10 log (N):

```
>> x = log10(2)
>> x = 0.3010
```

GEORGE MASON UNIVERSITY

INTERLUDE: FOCUS ON "MOD"

remainder $(\frac{a}{b})$:

```
>> x = mod(5,2)
>> x = 1
```

The "mod" function is very important. It comes up again and again, and is quite useful.

It is simply this: The **INTEGER remainder** after long division.

Remember long division, and the remainder?

GEORGE MASON UNIVERSITY

INTERLUDE: FOCUS ON "MOD"

"Ten divided by three is three remainder one"

or

mod(10,3) = 1

"Twelve divided by seven is one remainder five"

or,

mod(12,7) = 5

INTERLUDE: FOCUS ON "MOD" (cont)

"Eight divided by two is three remainder zero"

or

mod(8,2) = 0

"Twenty nine divided by three is nine remainder two"

or,

mod(29,3) = 2

INTERLUDE: FOCUS ON "MOD" (cont)

"Three divided by five is zero remainder three"

or

mod(3,5) = 3

"Eight divided by eleven is zero remainder eight"

or,

mod(8,11) = 8

INTERLUDE: FOCUS ON "MOD"
YOUR TURN!

mod(8,3) = ??? (in words, then the value)

mod(4,5) = ??? (in words, then the value)

mod(4,2) = ??? (in words, then the value)

mod(10,7) = ??? (in words, then the value)

mod(10,5) = ??? (in words, then the value)

mod(10,2) = ??? (in words, then the value)

INTERLUDE: FOCUS ON "MOD" YOUR TURN! (ANSWERS)

mod(8,3) : "Eight divided by three is two remainder two"

mod(4,5) : "Four divided by five is zero remainder four"

mod(4,2) : "Four divided by two is two remainder zero"

mod(10,7) : "Ten divided by seven is one remainder three"

mod(10,5) : "Ten divided by five is two remainder zero"

mod(10,2) : "Ten divided by two is five remainder zero"

INTERLUDE: FOCUS ON “MOD” YOUR TURN! (ANSWERS)

mod(8,3) = 2

mod(4,5) = 4

mod(4,2) = 0

mod(10,7) = 3

mod(10,5) = 0

mod(10,2) = 0

Decimal to Binary Conversion Function

>>dec2bin(16)

>>dec2bin(1024)
>>dec2bin(1023)

ASSIGNMENT: YOUR TURN!

Example 1: Create a variable called **x** and assign it the value 3. Create another variable called **y** and assign it the value 4. Compute the product of **x** and **y**, and assign the result to a third variable called **z**.

Example 2: Now square **z**, and assign the result to a fourth variable called **a**. Take the base 10 logarithm of **z** and assign that to a fifth variable called **b**. Reassign the value of **b** to **x**. Cube **b**, and assign the result to a sixth variable called **c**.

Example 3: Print out the final values of **x**, **y**, **z**, **a**, **b** and **c**.

ASSIGNMENT: YOUR TURN! (ANSWERS)

Example 1: Create a variable called **x** and assign it the value 3. Create another variable called **y** and assign it the value 4. Compute the product of **x** and **y**, and assign the result to a third variable called **z**.

```
x = 3;
y = 4;
z = x * y;
```

NOTE: A semi-colon at the end of a Matlab statement suppresses output, i.e., tells Matlab to "be quiet!"

GEORGE MASON UNIVERSITY

ASSIGNMENT: YOUR TURN! (ANSWERS)

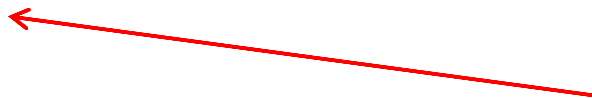
Example 2: Now square **z**, and assign the result to a fourth variable called **a**. Take the base 10 logarithm of **z** and assign that to a fifth variable called **b**. Reassign **x** the value of **b**. Cube **b**, and assign the result to a sixth variable called **c**.

```
a = z^2;
b = log10(z);
x = b;
c = b^3;
```

ASSIGNMENT: YOUR TURN! (ANSWERS)

Example 3: Print out the final values of **x**, **y**, **z**, **a**, **b** and **c**.

x
y
z
a
b
c



NOTE: Since a semi-colon at the end of a Matlab statement suppresses output, <u>to get printed output</u>, simply don't put a semi-colon at the end.

GEORGE MASON UNIVERSITY

We stopped here on

February 14, 2013

February 19, 2013

Review

Question:

Use Matlab to do the following operation

(1) Assign   A = 7
(2) Assign   B = 236
(3) Find   A * B
(4) Find   B mod A
(5) Find   sin(A)
(6) Find $\log_{10}(B)$
(7) Find   ln(B)
(8) Find e^A
(9) Find $A + B + A^2 + B^{1/2}$

Review

Answer

```
(1) >> A = 7
(2) >> B = 236
(3) >>  A * B                    % ans = 1652
(4) >> mod(B, A)                 % ans = 5
(5) >> sin(A)                    % ans = 0.6570
(6) >> log10(B)                  % ans = 2.3729
(7) >> log (B)                   % ans = 5.4638
(8) >> exp(A)                    % ans = 1.0966e+03
(9)  >> A + B + A^2 + sqrt(B)  % ans = 307.3623
```

GEORGE MASON UNIVERSITY

CHAPTER 4

VECTORS and VECTOR OPERATIONS

VECTORS

- Think of a "VECTOR" as a bunch of values, all lined up (here we have a "**row vector**"):

Vector **A**:

1	2	3	4	5

- We create **row vector** A like this (all three ways of writing the assignment statement are equivalent):

```
A = [1 2 3 4 5];
A = [1, 2, 3, 4, 5];
A = [1:5];
```

VECTORS

- Vectors are convenient, because by assigning a vector to a variable, we can manipulate the ENTIRE collection of numbers in the vector, just by referring to the variable name.

- So, if we wanted to add the value 3 to each of the values inside the vector A, we would do this:

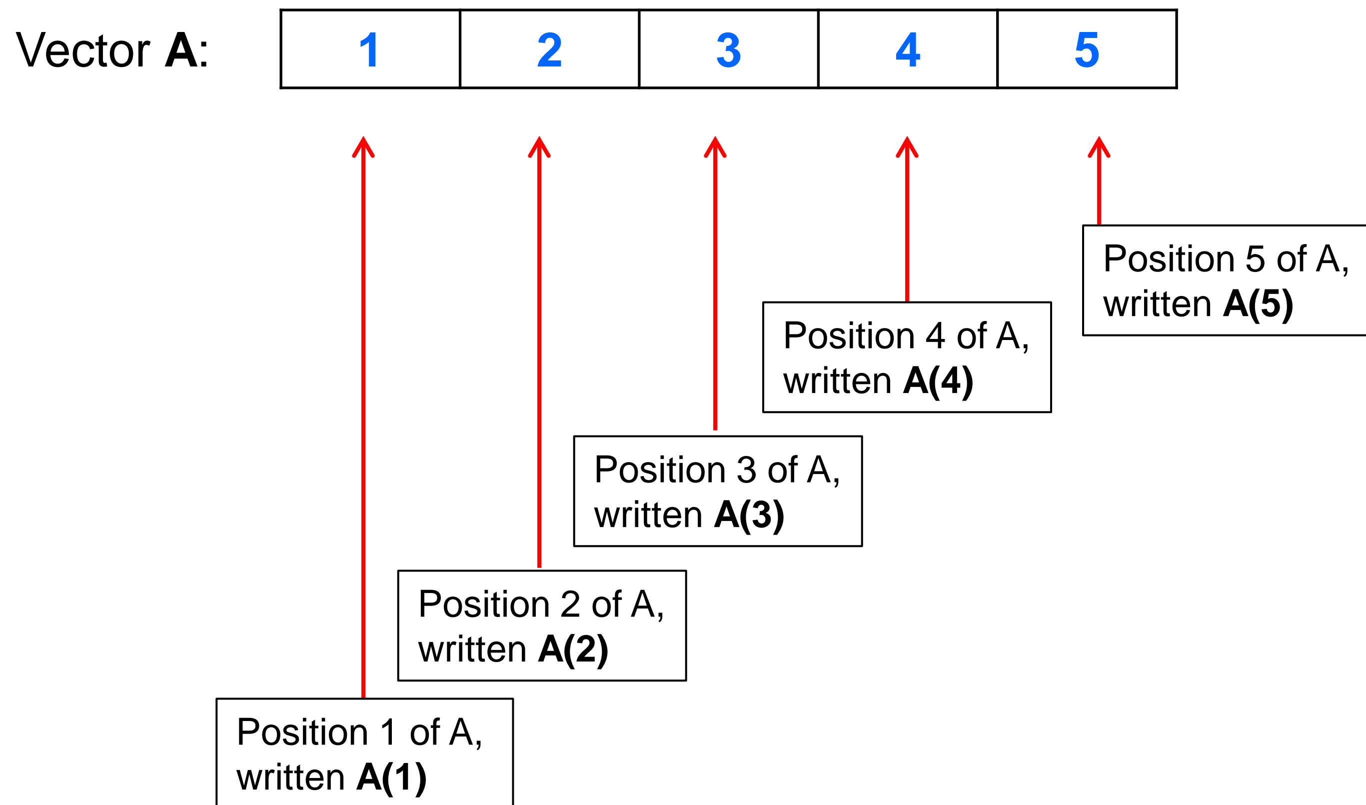
A + 3

Which accomplishes this:

A + 3:	**1+3**	**2+3**	**3+3**	**4+3**	**5+3**
	4	**5**	**6**	**7**	**8**

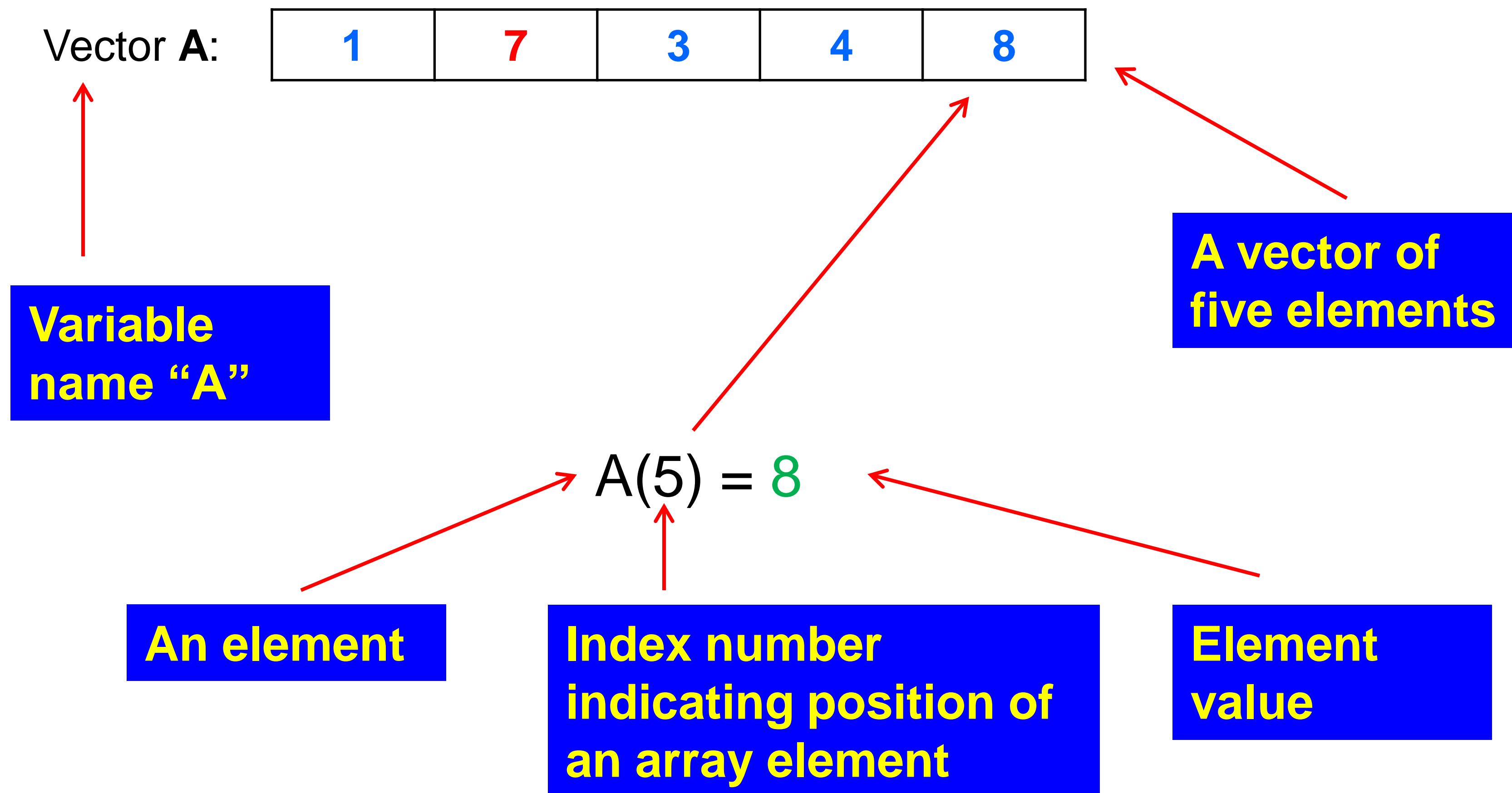
VECTORS

- We can refer to EACH POSITION of a vector, using what's called "subscript notation" or "index notation"



VECTORS

```
>> A=[1,7,3,4,10]  ; %comment: assign values of a vector
```



VECTORS

- **KEY POINT**: Each POSITION of a vector can act like an independent variable. So, for example, we can reassign different values to individual positions.

Before the first assignment:

Vector **A**:	**1**	**2**	**3**	**4**	**5**

A(2) = 7;

After the first assignment:

Vector **A**:	**1**	**7**	**3**	**4**	**5**

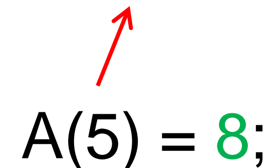
VECTORS

- **KEY POINT**: Each POSITION of a vector can act like an independent variable. So, for example, we can reassign different values to individual positions.

Before the second assignment:

Vector **A**:

1	7	3	4	5



A(5) = 8;

After the second assignment:

Vector **A**:

1	7	3	4	8

VECTORS

• **ANOTHER KEY POINT**: Because each position in a vector can act like an independent variable, we can do all the things with vector positions that we can do with independent variables like we did previously with **x**, **y**, **z**, **a**, **b** and **c**.

Vector **A**:	1	7	3	4	8

So, given vector A above, if we type "A(3)" at Matlab's command line, we will get the value **3** returned:

```
EDU>> A(3)
ans =
          3
```

GEORGE MASON UNIVERSITY

VECTORS: YOUR TURN!

Instructions:

For the next several examples, please try to work out the answers without running the code in Matlab. This is essential, as it will enable you to develop your “Matlab intuition” and also to visualize the sequence of a computation (thus developing your ability to think algorithmically). Furthermore, you will not be allowed to use Matlab software on exams or quizzes and so it’s better to get the practice now rather than wait until later! You may, however, use scratch paper to work out answers.

GEORGE MASON UNIVERSITY

VECTORS: YOUR TURN!

Vector **A**:

1	7	3	4	8

- **Example 4**: If "A(2)" typed at Matlab's command line, what value is printed?
- **Example 5**: If "A(2) + A(3)" is entered at Matlab's command line, what value is printed?
- **Example 6**: If "A(4) * A(5)" is entered at Matlab's command line, what value is printed?
- **Example 7**: If "A * 5" is entered at Matlab's command line, what is printed? Why?

VECTORS: YOUR TURN! ANSWERS

Vector **A**:

1	7	3	4	8

- **Example 4**: If "A(2)" typed at Matlab's command line, what value is printed? **7**

- **Example 5**: If "A(2) + A(3)" is entered at Matlab's command line, what value is printed? **10**

- **Example 6**: If "A(4) * A(5)" is entered at Matlab's command line, what value is printed? **32**

VECTORS: YOUR TURN! ANSWERS

Vector **A**:	1	7	3	4	8

- **Example 7**: If “A * 5” is entered at Matlab’s command line, what is printed? Why?

```
EDU>> A*5

ans =

     5    35    15    20    40
```

Each position of A is multiplied by 5.

VECTORS: YOUR TURN!

Vector **A**:	1	7	3	4	8

- **Example 8**: If "A(2) = A(3) * A(4)" typed at Matlab's command line, what does vector A look like now?

- **Example 9**: Assume vector A, as shown above, and also assume that the following sequence is entered at Matlab's command line. What does the vector A look like after this sequence?

```
A(1) = A(2) – ( A(3) + A(4) );
A = A * A(1);
```

VECTORS: YOUR TURN! ANSWERS

Vector **A**:

1	7	3	4	8

- **Example 8**: If "A(2) = A(3) * A(4)" typed at Matlab's command line, what does vector A look like now?

```
EDU>> A(2) = A(3) * A(4)

A =

     1    12     3     4     8
```

VECTORS: YOUR TURN! ANSWERS

Vector **A**:	1	7	3	4	8

- **Example 9**: Assume vector A, as shown above, and also assume that the following sequence is entered at Matlab's command line. What does the vector A look like after this sequence?

```
A(1) = A(2) – ( A(3) + A(4) );
A = A * A(1)
```

```
A =

        0        0        0        0        0
```

GEORGE MASON UNIVERSITY

VECTOR OPERATIONS

Assume we have a vector, A, as follows:

Vector **A**:	1	7	3	4	8

We can add a single number (a scalar) to every element of A, all at once, as follows:

B = A + 5 =	6	12	8	9	13

We can also multiply every element of A by a single number (a scalar) as follows:

B = A * 5 =	5	35	15	20	40

VECTOR OPERATIONS

Vector **A**:

1	7	3	4	8

We can subtract single number (a scalar) from every element of A, all at once, as follows:

B = A - 5 =

-4	2	-2	-1	3

We can also divide every element of A by a single number (a scalar) as follows:

B = A / 5 =

0.2	1.4	0.6	0.8	1.6

GEORGE MASON UNIVERSITY

VECTOR OPERATIONS

Vector **A**:	1	7	3	4	8

BUT:

If we want to ***exponentiate*** every element of A by a number (i.e., raise each element of A to a particular power), then we have to write this in a special way, using the ***DOT*** operator:

B = A**.^**2 =	1	49	9	16	64

NOT: B = A^2

VECTOR OPERATIONS

We can also do mathematical operations on two vectors ***OF THE SAME LENGTH***‡. For example, assume we have two vectors, A and B, as follows:

Vector **A**:	1	7	3	4	8
Vector **B**:	-1	7	10	6	3

Then:

C = A – B =	2	0	-7	-2	5
C = A + B =	0	14	13	10	11

‡: If A and B are not the same length, Matlab will signal an error

GEORGE MASON UNIVERSITY

VECTOR OPERATIONS

Vector **A**:	1	7	3	4	8
Vector **B**:	-1	7	10	6	3

Multiplication, division, and exponentiation, BETWEEN TWO VECTORS, ELEMENT-BY-ELEMENT, is expressed in the DOT notation:

C = A .* B =	-1	49	30	24	24
C = A ./ B =	-1	1	0.3	0.666	2.666
C = A .^ B =	1^{-1}	7^{7}	3^{10}	4^{6}	8^{3}

GEORGE MASON UNIVERSITY

OTHER USEFUL VECTOR OPERATIONS

Vector **A**:

1	7	3	4	8

e^x of each element of A, where x is an element of A:

```
>> exp(A)
ans =
 1.0e+003 *

   0.0027    1.0966    0.0201    0.0546    2.9810
```

Square root of each element of A:

```
 >> sqrt(A)
ans =
    1.0000    2.6458    1.7321    2.0000    2.8284
```

OTHER USEFUL VECTOR OPERATIONS

Vector **A**:	1	7	3	4	8

Natural logarithm of each element of A:

```
>> log(A)
ans =
        0    1.9459    1.0986    1.3863    2.0794
```

Base ten logarithm of each element of A:

```
>> log10(A)
ans =
        0    0.8451    0.4771    0.6021    0.9031
```

OTHER USEFUL VECTOR OPERATIONS

Vector **A**:

1	7	3	4	8

Cosine of each element of A (elements interpreted as radians):

```
 >> cos(A)
ans =
    0.5403    0.7539   -0.9900   -0.6536   -0.1455
```

Sine of each element of A (elements interpreted as radians):

```
 >> sin(A)
ans =
    0.8415    0.6570    0.1411   -0.7568    0.9894
```

OTHER USEFUL VECTOR OPERATIONS

Vector **A**:	1	7	3	4	8

Mean (average) of ALL elements of A:

```
 >> mean(A)
ans =
    4.6000
```

Standard deviation of ALL elements of A:

```
 >> std(A)
ans =
    2.8810
```

VECTORS: VARIABLE INDEXING

Let's say that I wanted to create a vector A containing all the odd integers between 1 and 10 (inclusive). How would I do this?

Here's one way:

A = [1, 3, 5, 7, 9];

I can simply list them all out. BUT, we could do this another way . . .

VECTORS: VARIABLE INDEXING

I could do it this way:

A = [1:2:10]

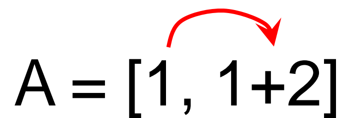
This says, "Begin at 1, then to generate all the remaining values in the vector, keep adding 2 to the previous value. Stop when we reach 10. Don't add the value if it exceeds 10." So . . .

Begin at 1:

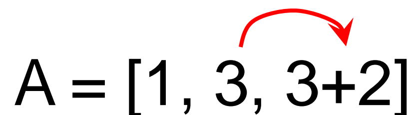
A = [1]

VECTORS: VARIABLE INDEXING

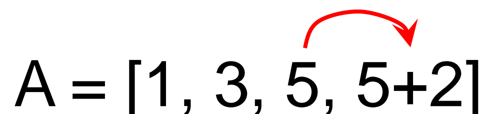
Add 2 to the previous value (i.e., to 1) to generate the next value:



Add 2 to the previous value (i.e., to 3) to generate the next value:

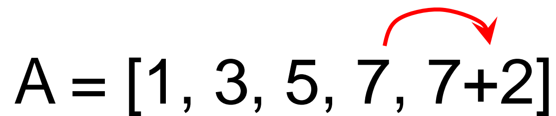


Add 2 to the previous value (i.e., to 5) to generate the next value:



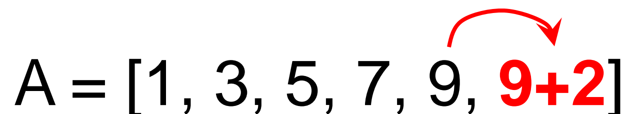
VECTORS: VARIABLE INDEXING

Add 2 to the previous value (i.e., to 7) to generate the next value:



A = [1, 3, 5, 7, 7+2]

Then . . . add 2 to the previous value (i.e., to 9) to generate the next value??



A = [1, 3, 5, 7, 9, **9+2**]

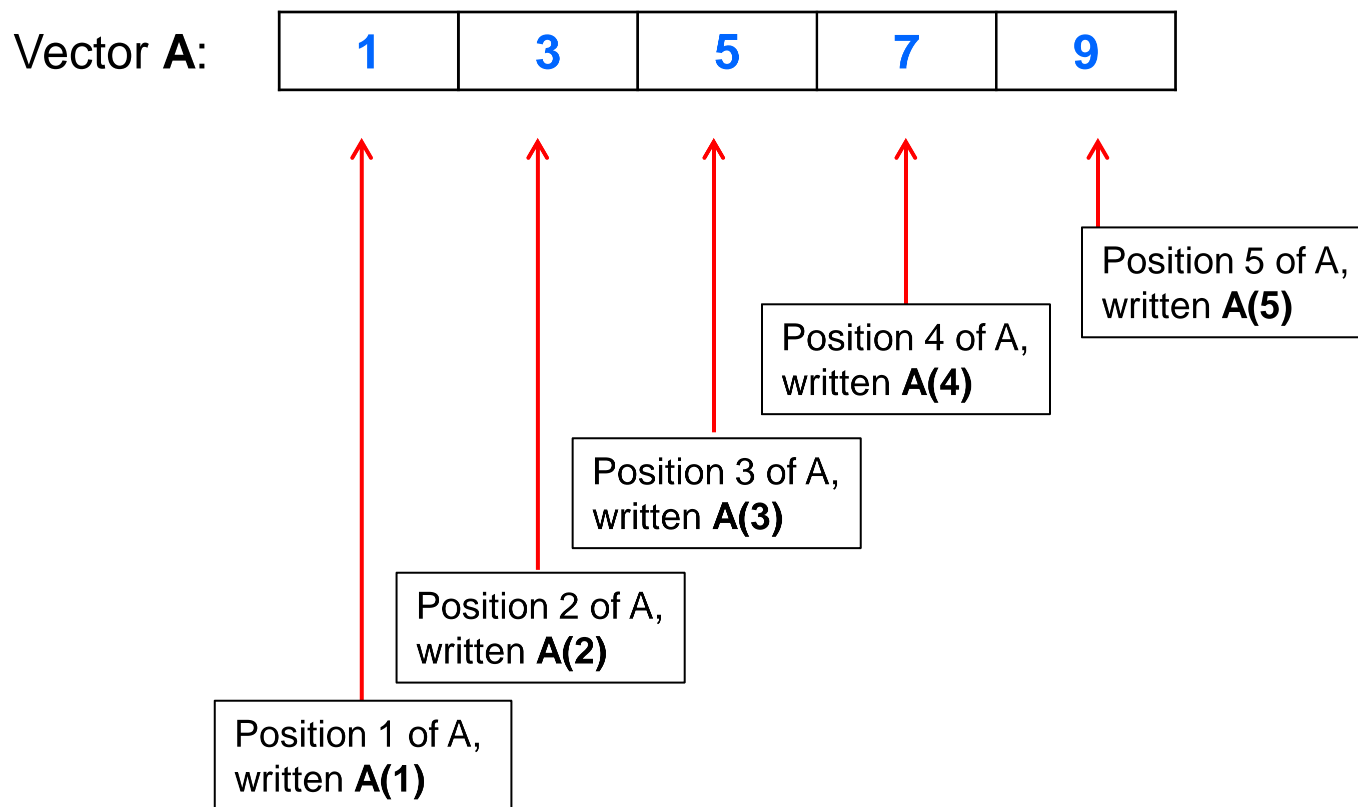
WAIT! 9+2 = 11 and 11 > 10, which is the end of the vector. So since we must stop at 10, we DO NOT insert 11 into the vector A. Thus, the vector A is now constructed as desired:

A = [1, 3, 5, 7, 9]

GEORGE MASON UNIVERSITY

VECTORS: VARIABLE INDEXING

NOTE WHAT WE NOW HAVE:



VECTORS: VARIABLE INDEXING

THUS:

Vector **A**:	1	3	5	7	9
	A(1)=1	**A(2)=3**	**A(3)=5**	**A(4)=7**	**A(5)=9**

VECTORS: VARIABLE INDEXING

WHAT'S THE ADVANTAGE of doing it this way? Why not list out all the members of the vector? Isn't it easier that way?

Actually, no. What if I wanted to create a vector B containing all the odd, positive integers between, say, 1 and 1000 (inclusive). How would I do that? I could list them out, one by one, but that's pretty tedious (not to mention time consuming). Here's how to do it all in one statement:

```
B = [1:2:1000];
```

Remember to end with a semi-colon, or else you're going to see a LOT of output!

That's it!

VECTORS: VARIABLE INDEXING

How about creating a vector C containing all the even, positive integers between 1 and 1000 (inclusive). How would I do that?

Begin at 2 and then add 2:

```
C = [2:2:1000];
```

We can start anywhere we want. We start at 2 because of course, 1 is odd and will not be a member of the vector! The vector will begin with the value 2.

VECTORS: VARIABLE INDEXING

Let's say that I want to find the average of all the even, positive integers between 1 and 1000. How would I do that?

Here's how:

```
C = [2:2:1000];
mean(C)
```

That's it!

CHAPTER 5

MATRICES (ARRAYS) and MATRIX OPERATIONS

MATRICES (ARRAYS)

- Think of a MATRIX as a partitioned box: The box itself has a label (its variable name), and, we can store MULTIPLE things inside the box, each inside its own partition:

Matrix A:

- We can manipulate the ENTIRE COLLECTION at once, just by referring to the matrix's variable name.

GEORGE MASON UNIVERSITY

MATRICES (ARRAYS)

- A matrix has **DIMENSIONS**: In the 2 dimensional case, rows and columns.
- Each partition is referred to by both its row number and its column number.
- **Rule**:
    - In Matlab, both row and column numbers **START AT 1**:

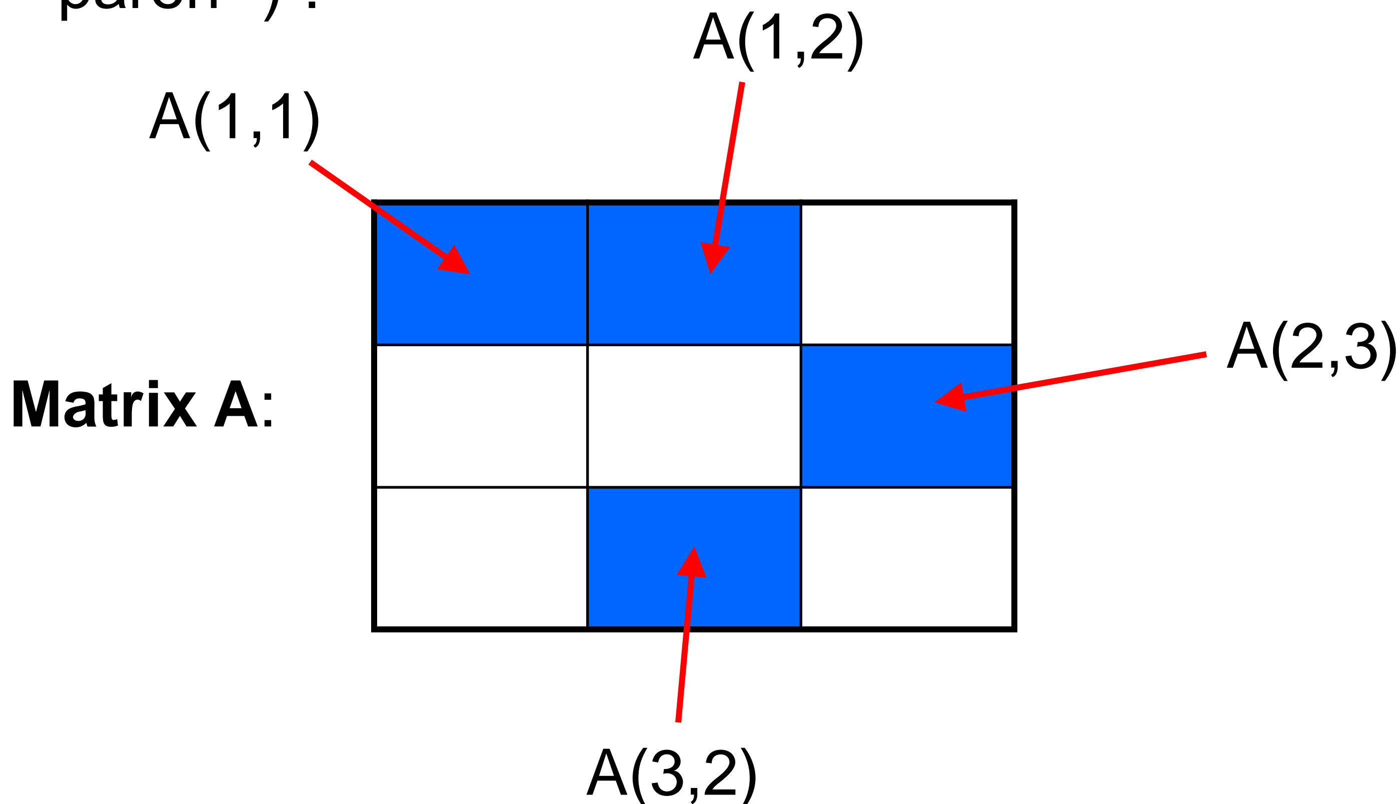
(ROW, COLUMN)

Matrix A:

(1,1)	(1,2)	(1,3)	**Row 1**
(2,1)	(2,2)	(2,3)	**Row 2**
(3,1)	(3,2)	(3,3)	**Row 3**
Column 1	**Column 2**	**Column 3**	

MATRICES (ARRAYS)

- We can access individual partitions (called “elements”) by writing the matrix name, followed by a left paren “(“, the row number, a comma, the column number, and a right paren “)”:



MATRICES (ARRAYS)

- **Rule**:
  The name for an individual matrix element acts **just like a variable**.

A(1,3)

(ROW, COLUMN)

Matrix A:

A(1,1)	A(1,2)	A(1,3)
A(2,1)	A(2,2)	A(2,3)
A(3,1)	A(3,2)	A(3,3)

(ACTUAL VALUES)

4	2	0
13	7	12
1	3	-10

Typing "`A(1,1)`" at the command line, will result in 4
Typing "`A(2,2)`" at the command line, will result in 7
Typing "`A(3,3)`" at the command line, will result in -10

GEORGE MASON UNIVERSITY

MATRICES (ARRAYS)

- **Rules**:
  - Each matrix element acts like a variable and so can be used like variables

Matrix A:

(ROW, COLUMN)

A(1,1)	A(1,2)	A(1,3)
A(2,1)	A(2,2)	A(2,3)
A(3,1)	A(3,2)	A(3,3)

(ACTUAL VALUES)

4	2	0
13	7	12
1	3	-10

Examples:

```
varX  =  A(1,1) + A(1,2)   (so, varX = 4 + 2)
varX  =  A(2,2) * A(1,2)   (so, varX = 7 * 2)
varX  =  A(3,1) - A(2,3)    (so, varX = 1 - 12)
varX  =  A(1,2) ^ A(3,2)    (so, varX = 2^3 )
```

GEORGE MASON UNIVERSITY

MATRICES (ARRAYS)

- As a result, you can assign values to specific matrix elements, too:

(ROW, COLUMN)

Matrix A:

A(1,1)	A(1,2)	A(1,3)
A(2,1)	A(2,2)	A(2,3)
A(3,1)	A(3,2)	A(3,3)

(NEW VALUES)

100	2	0
13	200	12
1	3	300

Examples:

```
A(1,1) =  100;
A(2,2) =  200;
A(3,3) =  300;
```

MATRICES (ARRAYS)

- Matrices can be different dimensions.

Here's a 1-D matrix (called a "row vector"):

(ROW, COLUMN)

Matrix B:

B(1,1)	B(1,2)	B(1,3)	B(1,4)	B(1,5)

```
B(1,1) = 10;
B(1,2) = 8;
B(1,3) = 6;
B(1,4) = 4;
B(1,5) = 2;
```

(ACTUAL VALUES)

Matrix B:

10	8	6	4	2

GEORGE MASON UNIVERSITY

MATRICES (ARRAYS)

- Another 1-dimensional matrix (called a "column vector"):

(ROW, COLUMN)	(ACTUAL VALUES)
C(1,1)	10
C(2,1)	8
C(3,1)	6
C(4,1)	4
C(5,1)	2

Matrix C:

```
C(1,1) = 10;
C(2,1) = 8;
C(3,1) = 6;
C(4,1) = 4;
C(5,1) = 2;
```

MATRICES (ARRAYS)

- **Creating matrices in Matlab (cont)**:
  - **Method #1**: Write it out explicitly (separate rows with a semi-colon):

```
>> D = [1; 2; 3]

D =

      1
      2
      3

>> D = [1, 2, 3; 4, 5, 6; 7, 8, 9]

D =

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

Note: commas are optional

MATRICES (ARRAYS)

- **Creating matrices in Matlab**:
  - **Method #2**: Like variable assignment:
  - Assign the last element in a 1-D matrix, or the "lower right corner element" in a 2-D matrix to be some value.
  - Now you have a matrix filled with zeros and whatever you assigned to the last element:

```
>> D(1,5) = 0

D =   0      0      0      0      0

>> D(3,3) = 10

D =     0      0      0
        0      0      0
        0      0     10
```

MATRICES (ARRAYS)

- **Creating matrices in Matlab (cont)**:
  - **Method #2**: Keep assigning pieces to it:

```
>> D(1,1) = 5    ;assigning value to D(1,1)

>> D(1,2) = 6    ;assigning value to D(1,2)

.......

>> D(2,3) = 7    ;assigning value to D(2,3)
```

MATRICES (ARRAYS)

Method #2, while computationally expensive inside Matlab, is nevertheless perfectly suited for inclusion inside a *FOR loop* – particularly when you do not know the exact number of iterations that will be done (and thus, you don't know the exact size of the matrix involved).

FOR loops describe the second topic we will investigate: *ITERATION*.

SOME BASIC MATRIX OPERATIONS

Some of Matlab's built-in (intrinsic) functions work the way we would expect, on an element-by-element basis for an input matrix (here called "A"):

cosine:

```
>> cos(A)
```

returns cosine of each element of A

sqrt

```
>> sqrt(A)
```

returns the sqrt of each element of A

base 10 log:

```
>> log10(A)
```

returns base 10 logarithm of each element of A

sine:

```
>> sin(A)
```

returns sine of each element of A

natural log:

```
>> log(A)
```

returns the natural logarithm of each element of A

Multiplication by a number (scalar):

```
>> A*5
```

returns matrix A with each element multiplied by 5.

BASIC MATRIX OPERATIONS (cont)

Some arithmetic operators also operate this way, and in particular, if give two matrix arguments "A" and "B", these operators compute on an element-by-element basis:

+ :

```
>> A + B
```

returns a matrix containing the sum of each element of A and its corresponding element in B.

- :

```
>> A - B
```

returns a matrix containing the difference of each element of A and its corresponding element in B.

BASIC MATRIX OPERATIONS (cont)

BUT Multiplication and division operate differently, when we are dealing with a matrix multiplied by a matrix. In order to do element-by-element multiplication or division, we need to use the “dot” operator:

```
>> A .* B
```

returns a matrix containing the product of each element of A and its corresponding element in B.

```
>> A ./ B
```

returns a matrix containing the result of dividing each element of A by its corresponding element in B.

We stopped here on

February 19, 2013

February 19, 2013

Review: Array and Matrix

Question:

(1) Create a row array with five elements "10,20,30,40,50"

(1) Create a column array with five elements "10,20,30,40,50"

Review: Array and Matrix

Answer:

>> A=[10,20,30,40,50] ;row array

>>A=[10;20;30;40;50]   ;column array

Review: Array and Matrix

Question:

(1) Create a 3 X 3 matrix A with the following elements

1	2	3
4	5	6
7	8	9

(2) Create a matrix B, and B=A*10

(3) Calculate A+B, A-B, A multiply B (element by element), A divide B (element by element)

Review: Array and Matrix

Answer:

```
>>A=[1,2,3;4,5,6;7,8,9]  %explicit method

%alternative method, but tedious
>> A(3,3) = 0                    % element assignment method
>> A(1,1)=0
>> A(1,2)=1
>> A(1,3)=3
>>A(2,1)=4
>>A(2,2)=5
>>A(2,3)=6
>>A(3,1)=7
>>A(3,2)=8
>>A(3,3)=9
```

Review: Array and Matrix

Answer (continued):

```
>> B=A*10

>> A+B

>>A-B

>>A.*B  %comment: use “.”operator, different from
    A*B

>>A./B  %comment: use “.”operator
```

Review: Array and Matrix

Question:

Create a 11 X 11 matrix A with all elements equal 10? Then change the value at the center to 100?

GEORGE MASON UNIVERSITY

Review: Array and Matrix

Answer:

```
>> A(11,11)=0 %create an 11 X 11 matrix with all
     elements equal to 0

>>A=A+10      % all elements in A are added by 10

>>A(6,6)=100  %element A(6,6) is assigned to 100
```

Review: Array and Matrix

Questions:

(1) For array A=[3,4,5,7,10], what is A(4)?

(2) For the following matrix A

2	8	10
9	1	3
6	20	5
14	7	6

(a) How many rows and columns in this matrix?
(b) What is A (2,3)?
(c) What is A (3,2)?

Review: Array and Matrix

Answer

(1) A(4) Ans = 7

(2)
(a) Four rows and three columns
(b) A (2,3) Ans = 3
(c) A (3, 2) Ans = 20

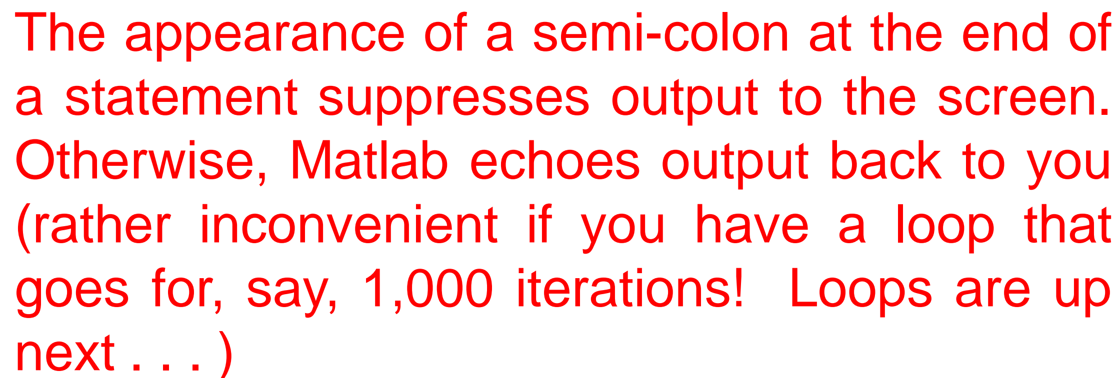
Getting Matlab to “Be Quiet!”

By the way . . .

```
>> D(1,5) = 0

D =   0     0     0     0     0
```

But . . .

```
>> D(1,5) = 0;

>>
```



CHAPTER 6

ITERATION I: FOR LOOPS

ITERATION

- Often times, we'll want to repeat doing something again, and again, and again, and again…maybe millions of times…or maybe, just enough times to access and change each element of a matrix.

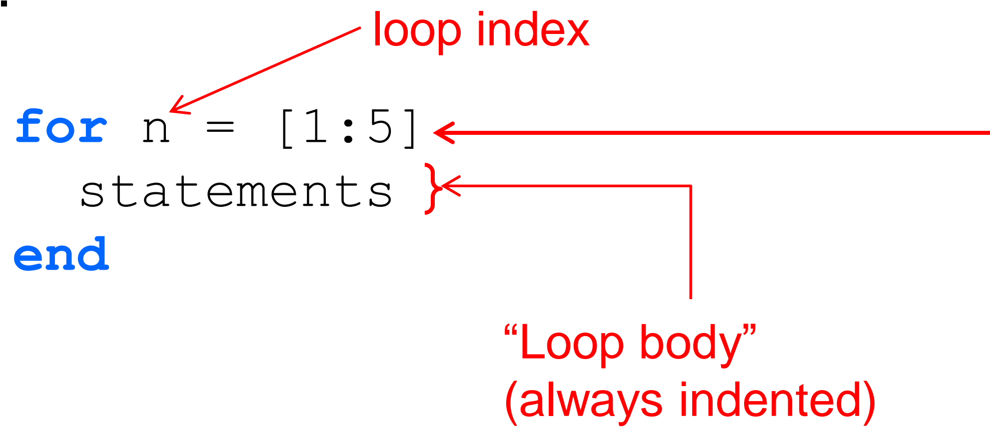
- Computers, of course, are great at doing things over and over again

- This is called *ITERATION*.

- Iteration is executed with a *FOR loop,* or, with a *WHILE loop.*

ITERATION (FOR loops)

- **Syntax**: ***As shown, and always the same***. **NOTE**: The keywords FOR and END come in a pair—***never one without the other***.



- **What's happening**:
  - n first assigned the value 1
  - the statements are then executed
  - END is encountered
  - END sends execution back to the top
  - Repeat: n = 2, 3, 4, 5
  - Is n at the end of the list? Then STOP.

SIDE NOTE:
n = [1:5]
same as
n = [1,2,3,4,5],
same as
n = 1:5

ITERATION (FOR loops)

- **Syntax**: ***As shown, and always the same***. **NOTE**: The keywords FOR and END come in a pair—***never one without the other***.

loop index

```
for n = [1:5]
   statements
end
```

"Loop body"
(always indented)

- **Key Features**:
  - The FOR loop executes for a finite number of steps and then quits.
  - Because of this feature, it's hard to have an infinite loop with a FOR loop.
  - You can NEVER change a FOR loop's index counter (here, called "n").

SIDE NOTE:
n = [1:5]
same as
n = [1,2,3,4,5],
same as
n = 1:5

ITERATION (FOR loops)

- So, a FOR loop is doing "implicit assignment" to the loop index n: each time "through the loop" (or, one "trip"), the loop index n has a particular value (which can't be changed by you, only by the FOR loop). After the trip is complete (when execution reaches END), the loop index is reassigned to the next value in the 1-D matrix, from left to right. When the rightmost (last) value is reached, the FOR loop stops executing.

And then statements AFTER the FOR loop (following the END keyword) are executed -- that is, the FOR loop "exits".

ITERATION (FOR loops) – YOUR TURN!

Instructions:

For the next several examples, please try to work out the answers without running the code in Matlab. This is essential, as it will enable you to develop your "Matlab intuition" and also to visualize the sequence of a computation (thus developing your ability to think algorithmically). Furthermore, you will not be allowed to use Matlab software on exams or quizzes and so it's better to get the practice now rather than wait until later! You may, however, use scratch paper to work out answers.

ITERATION (FOR loops) – YOUR TURN!

Example 10:

```
for  n = [1:5]
    n
end
```

This will print out the value of n, for each iteration of the loop. Why? Because the statement `n` is NOT terminated with a semi-colon:

```
ans =  1
ans =  2
ans =  3
ans =  4
ans =  5
```

ITERATION (FOR loops) – YOUR TURN!

Example 11:

```
for  n = [1:5]
    n^2
end
```

This FOR loop will print out:

```
ans =  1
ans =  4
ans =  9
ans =  16
ans =  25
```

GEORGE MASON UNIVERSITY

ITERATION (FOR loops) – YOUR TURN!

Example 12:

```
for  n = [1:5]
    n^3 - 5
end
```

What will this FOR loop print out?

```
ans =  ?
ans =  ?
ans =  ?
ans =  ?
ans =  ?
```

ITERATION (FOR loops) – YOUR TURN!

Example 12:

```
for  n = [1:5]
     n^3 - 5
end
```

What will this FOR loop print out?

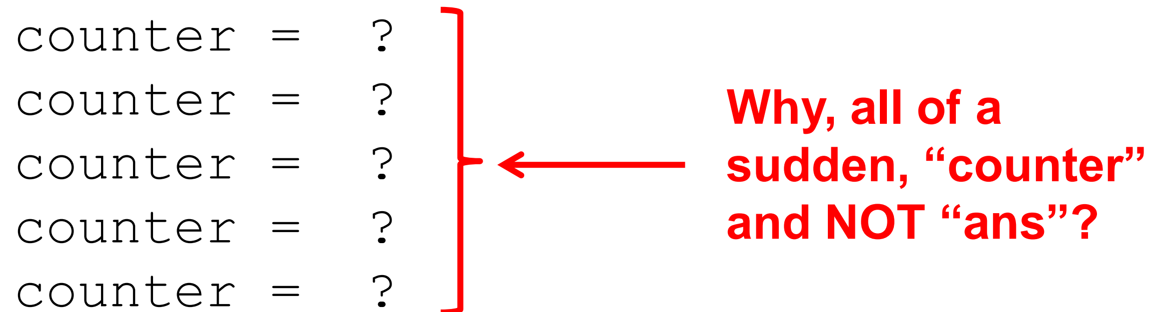
```
ans =  -4
ans =  3
ans =  22
ans =  59
ans =  120
```

ITERATION (FOR loops) – YOUR TURN!

Example 13:

```
counter = 1;
for  n = [1:5]
    counter = counter + n
end
```

What will this FOR loop print out?



ITERATION (FOR loops) – YOUR TURN!

Example 13:

```
counter = 1;
for  n = [1:5]
    counter = counter + n
end
```

What will this FOR loop print out?

```
counter =  2
counter =  4
counter =  7
counter =  11
counter =  16
```

Why, all of a sudden, "counter" and NOT "ans"?

ITERATION (FOR loops) – YOUR TURN!

Example 14:

```
counter = 1;
for  n = [1:5]
    counter = counter - n
end
```

What will this FOR loop print out?

```
counter =  ?
counter =  ?
counter =  ?
counter =  ?
counter =  ?
```

GEORGE MASON UNIVERSITY

ITERATION (FOR loops) – YOUR TURN!

Example 14:

```
counter = 1;
for  n = [1:5]
    counter = counter - n
end
```

What will this FOR loop print out?

```
counter =  0
counter =  -2
counter =  -5
counter =  -9
counter =  -14
```

ITERATION (FOR loops) – YOUR TURN!

Example 15:

```
counter = 1;
for  n = [1:5]
    counter = counter*n
end
```

What will this FOR loop print out?

```
counter =  ?
counter =  ?
counter =  ?
counter =  ?
counter =  ?
```

ITERATION (FOR loops) – YOUR TURN!

Example 15:

```
counter = 1;
for  n = [1:5]
    counter = counter*n
end
```

What will this FOR loop print out?

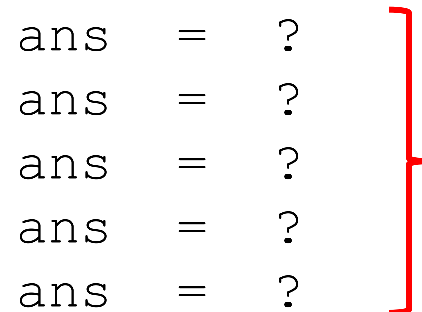
```
counter =  1
counter =  2
counter =  6
counter =  24
counter =  120
```

ITERATION (FOR loops) – YOUR TURN!

Example 16:

```
A = [5 4 3 2 1];
for  n = [1:5]
      A(1,n)
end
```

What will this FOR loop print out?

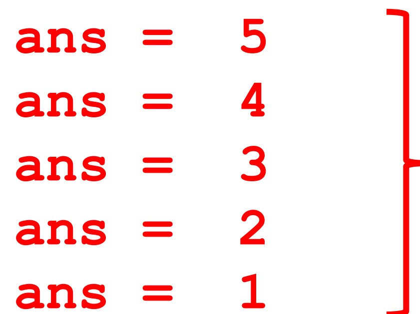


ITERATION (FOR loops) – YOUR TURN!

Example 16:

```
A = [5 4 3 2 1];
for  n = [1:5]
    A(1,n)
end
```

What will this FOR loop print out?



ITERATION (FOR loops) – YOUR TURN!

Example 17:

```
A = [5 4 3 2 1];
counter = 0;
for  n = [1:5]
     A(1,n)= A(1,n) + counter;
     counter = counter + 1;
end
A
```

What will print out?

```
A  =  ?
```

ITERATION (FOR loops) – YOUR TURN!

Example 17:

```
A = [5 4 3 2 1];
counter = 0;
for  n = [1:5]
    A(1,n)= A(1,n) + counter;
    counter = counter + 1;
end
A
```

What will print out?

```
A =
    5     5     5     5     5
```

ITERATION
(FOR loops with variable indexing, I)

• **Syntax**: In the case of variable indexing, the loop index steps forward by an amount other than 1:

```
for n = [1:2:15]
  statements
end
```

• **Key Features**:

• In this FOR loop, the index variable `n` first takes on the value 1 and then, each trip through the loop, it is increased by 2, up to the limit 15. What this means is that the index variable `n` takes on the following values: 1, 3, 5, 7, 9, 11, 13, 15

ITERATION
(FOR loops with variable indexing, II)

• **Syntax**: In this second case of variable indexing, the loop index steps ***backward*** by an amount other than 1:

```
for n = [15:-2:1]
  statements
end
```

• **Key Features**:

  • In this second FOR loop, the index variable `n` first takes on the value 15 and then, each trip through the loop, it is ***decreased*** by 2, down to the limit 1. What this means is that the index variable `n` takes on the following values: 15, 13, 11, 9, 7, 5, 3, 1

ITERATION
(FOR loops with variable indexing, III)

- **Syntax**: In this third case of variable indexing, the loop index steps ***forward,*** but by an amount ***less than*** 1:

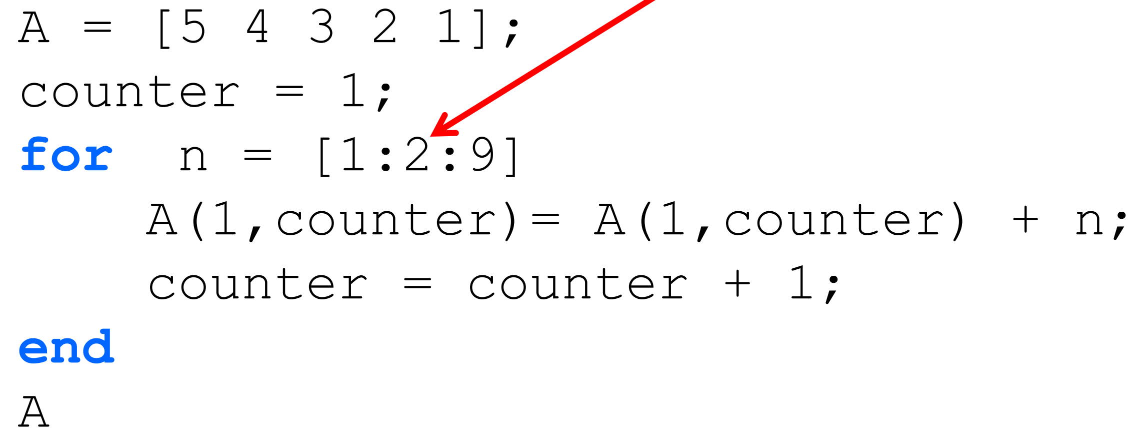
```
for n = [0.1: 0.1 : 1.0]
  statements
end
```

- **Key Features**:

In this third FOR loop, the index variable `n` first takes on the value 0.1 and then, each trip through the loop, it is ***increased*** by 0.1, up to the limit 1.0. What this means is that the index variable `n` takes on the following values: 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0 . Thus, the index variable `n` need not be assigned integer values! Furthermore, we could do the above in reverse, too, as follows: `n = [1.0: -0.1 : 0.1]`

ITERATION
(FOR loops; variable indexing)—YOUR TURN

Example 19:




```
A = [5 4 3 2 1];
counter = 1;
for  n = [1:2:9]
     A(1,counter)= A(1,counter) + n;
     counter = counter + 1;
end
A
```

What will print out?

```
A  =  ?
```

ITERATION
(FOR loops; variable indexing)—YOUR TURN

Example 19:

```
A = [5 4 3 2 1];
counter = 1;
for  n = [1:2:9]
     A(1,counter)= A(1,counter) + n;
     counter = counter + 1;
end
A
```

TRICKY!!

What will print out?

```
A =
       6      7      8      9     10
```

ITERATION (FOR loops)

Study the preceding discussions and example problems **VERY CAREFULLY** to ensure that you understand completely what each FOR loop is doing, AND WHY!

These problems are VERY REPRESENTATIVE of those you might encounter in the future . . .

GEORGE MASON UNIVERSITY

We stopped here on

February 21, 2013

GEORGE MASON UNIVERSITY

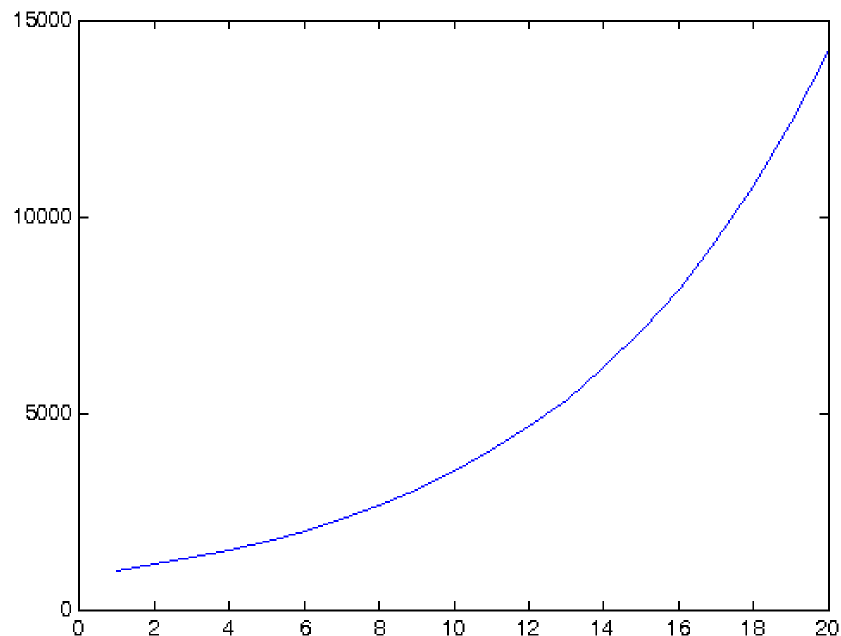
CHAPTER 7

BASIC GRAPHS AND PLOTS

BASIC GRAPHS and PLOTS

Let's begin by creating the graph on the right with the Matlab code on the left . . .

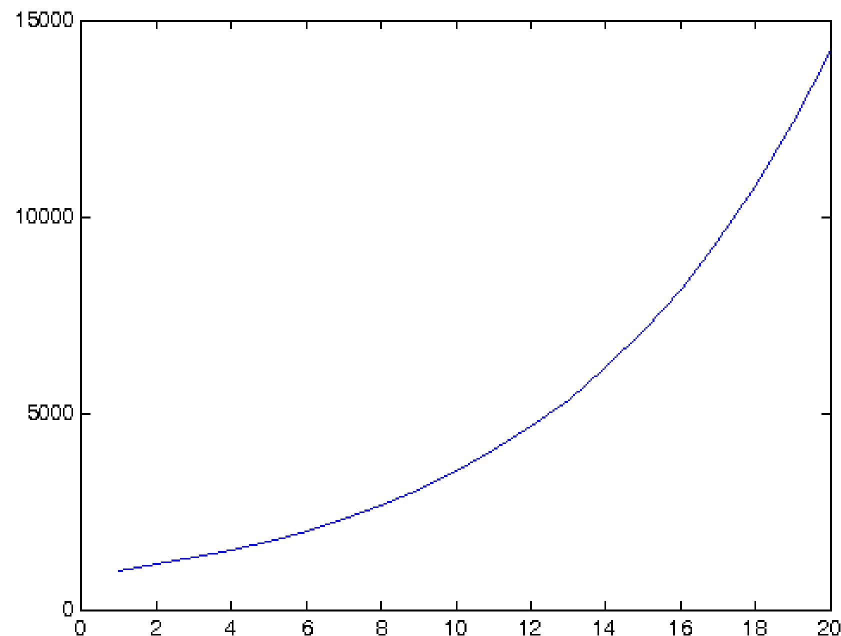
```
clear;clc
birth = 0.15;
death = 0.000;
deltat = 1;
P(1) = 1000;
for t = [1:19]
   P(t+1) = P(t)+birth*P(t);
end
plot(1:20,P)
```



BASIC GRAPHS and PLOTS

First, give the graph a name (internal to Matlab) by using the **figure** command:

```
clear;clc
birth = 0.15;
death = 0.000;
deltat = 1;
P(1) = 1000;
for t = [1:19]
   P(t+1) = P(t)+birth*P(t);
end
figure(1)
plot(1:20,P)
```

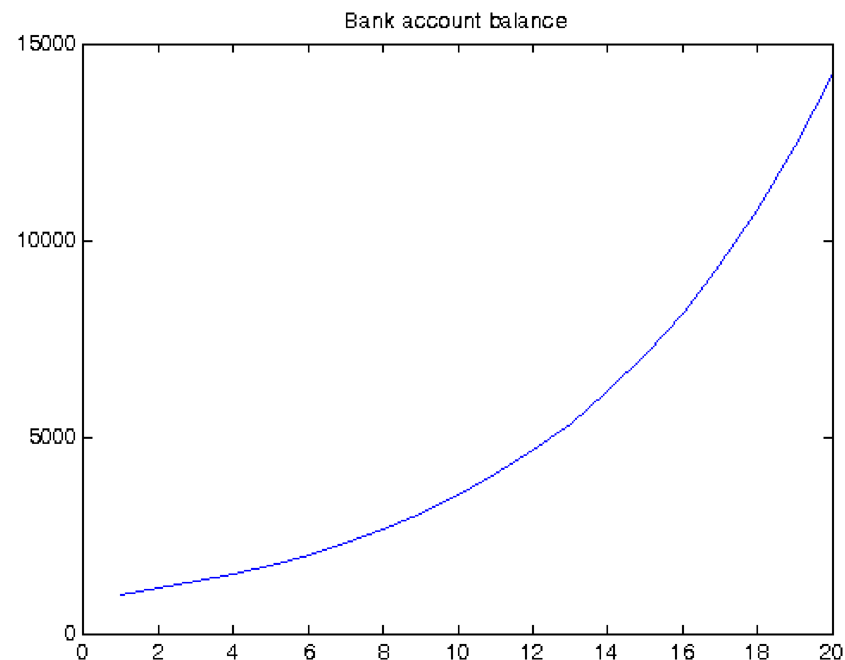


When you re-run this code, there's no visible effect, but the plot now has a name (meaning it won't be overwritten later!)

BASIC GRAPHS and PLOTS

Next, give the graph a title
by using the **title** command:

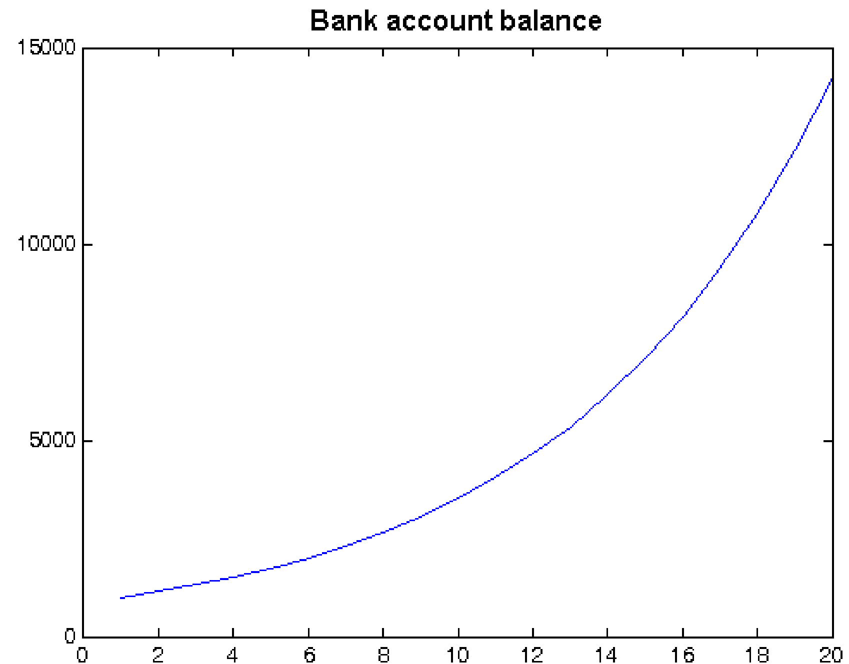
```
clear;clc
birth = 0.15;
death = 0.000;
deltat = 1;
P(1) = 1000;
for t = [1:19]
   P(t+1) = P(t)+birth*P(t);
end
figure(1)
plot(1:20,P)
title('Bank account balance')
```



BASIC GRAPHS and PLOTS

Change the title's font: continue on next line and use the **FontName, FontSize** and **FontWeight** commands:

```
clear;clc
birth = 0.15;
death = 0.000;
deltat = 1;
P(1) = 1000;
for t = [1:19]
   P(t+1) = P(t)+birth*P(t);
end
figure(1)
plot(1:20,P)
title('Bank account balance', ...
    'FontName', 'Arial', ...
    'FontSize', 12, ...
    'FontWeight', 'Bold')
```

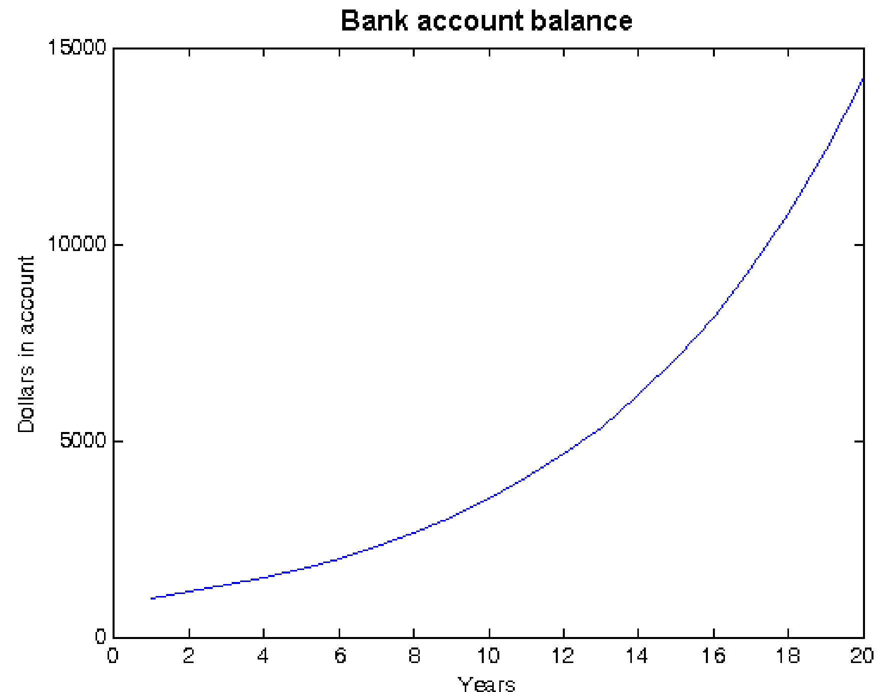


Continue on the next line with three dots in a row

BASIC GRAPHS and PLOTS

Add X and Y axis labels:
use the **xlabel** and **ylabel** commands:

```
clear;clc
birth = 0.15;
death = 0.000;
deltat = 1;
P(1) = 1000;
for t = [1:19]
   P(t+1) = P(t)+birth*P(t);
end
figure(1)
plot(1:20,P)
title('Bank account balance', ...
    'FontName', 'Arial', ...
    'FontSize', 12, ...
    'FontWeight', 'Bold')
xlabel('Years')
ylabel('Dollars in account')
```

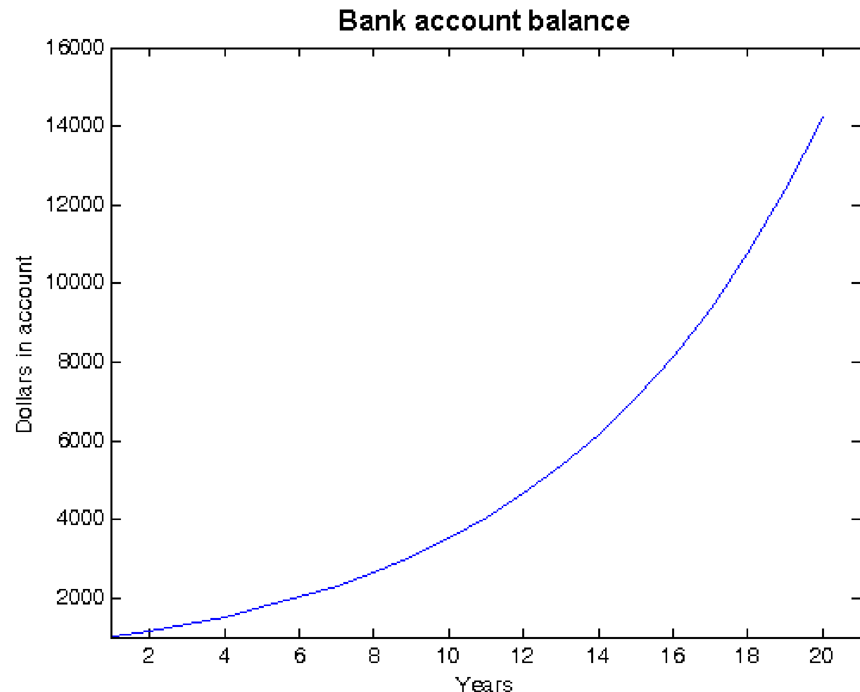


You can also change X and Y axis fonts, sizes, etc, just like for the title

BASIC GRAPHS and PLOTS

Change X and Y axis ranges:
use the **xlim** and **ylim** commands:

```
clear;clc
birth = 0.15;
death = 0.000;
deltat = 1;
P(1) = 1000;
for t = [1:19]
   P(t+1) = P(t)+birth*P(t);
end
figure(1)
plot(1:20,P)
title('Bank account balance', ...
    'FontName', 'Arial', ...
    'FontSize', 12, ...
    'FontWeight', 'Bold')
xlabel('Years')
ylabel('Dollars in account')
xlim([1 21])
ylim([1000 16000])
```

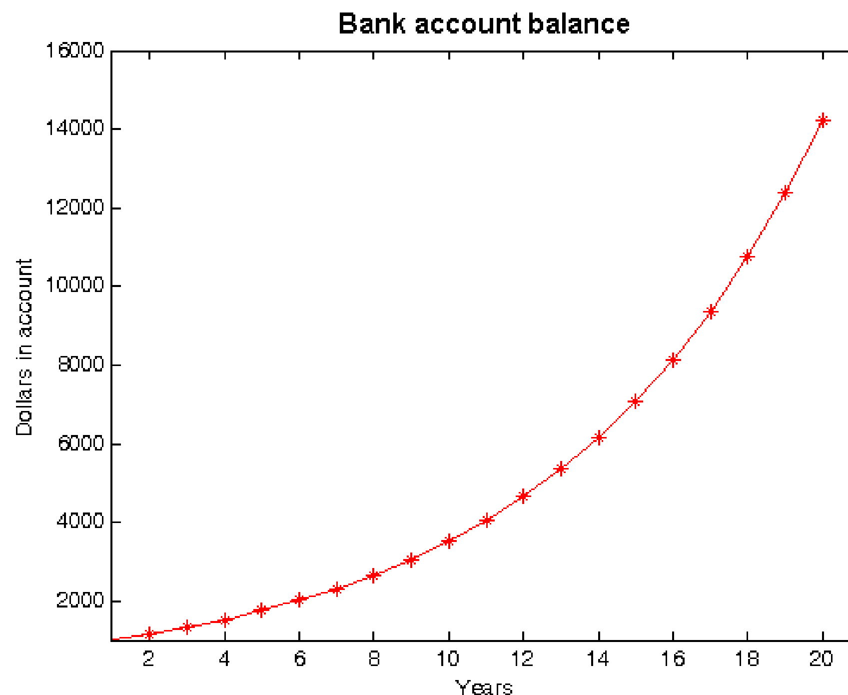


BASIC GRAPHS and PLOTS

Now, change the plot:

Let it have a red line and * datamarkers:

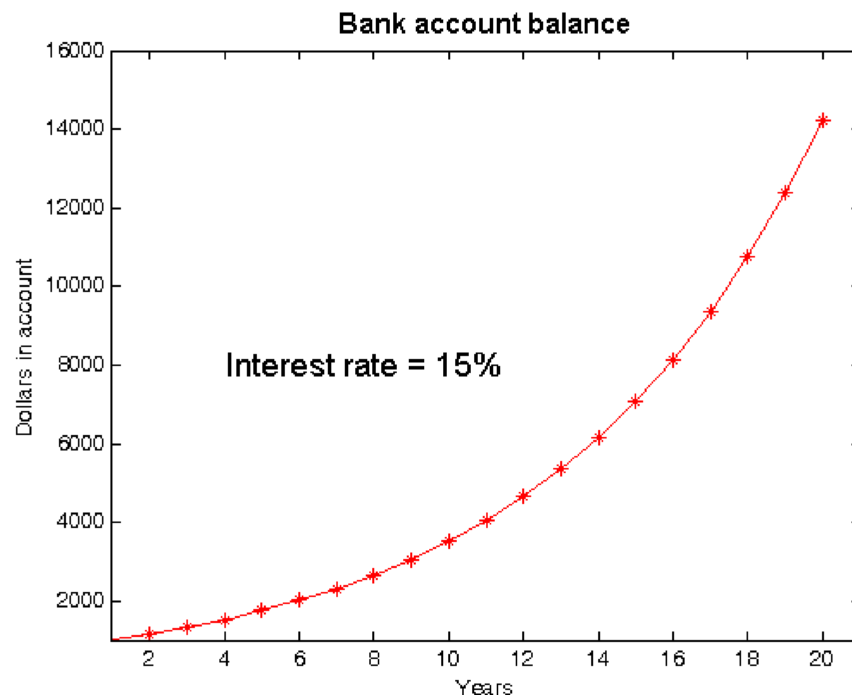
```
clear;clc
birth = 0.15;
death = 0.000;
deltat = 1;
P(1) = 1000;
for t = [1:19]
   P(t+1) = P(t)+birth*P(t);
end
figure(1)
plot(1:20,P,'-*r')
title('Bank account balance', ...
    'FontName', 'Arial', ...
    'FontSize', 12, ...
    'FontWeight', 'Bold')
xlabel('Years')
ylabel('Dollars in account')
xlim([1 21])
ylim([1000 16000])
```



BASIC GRAPHS and PLOTS

Finally, insert some text annotation on the graph using the **text** command, and control its properties:

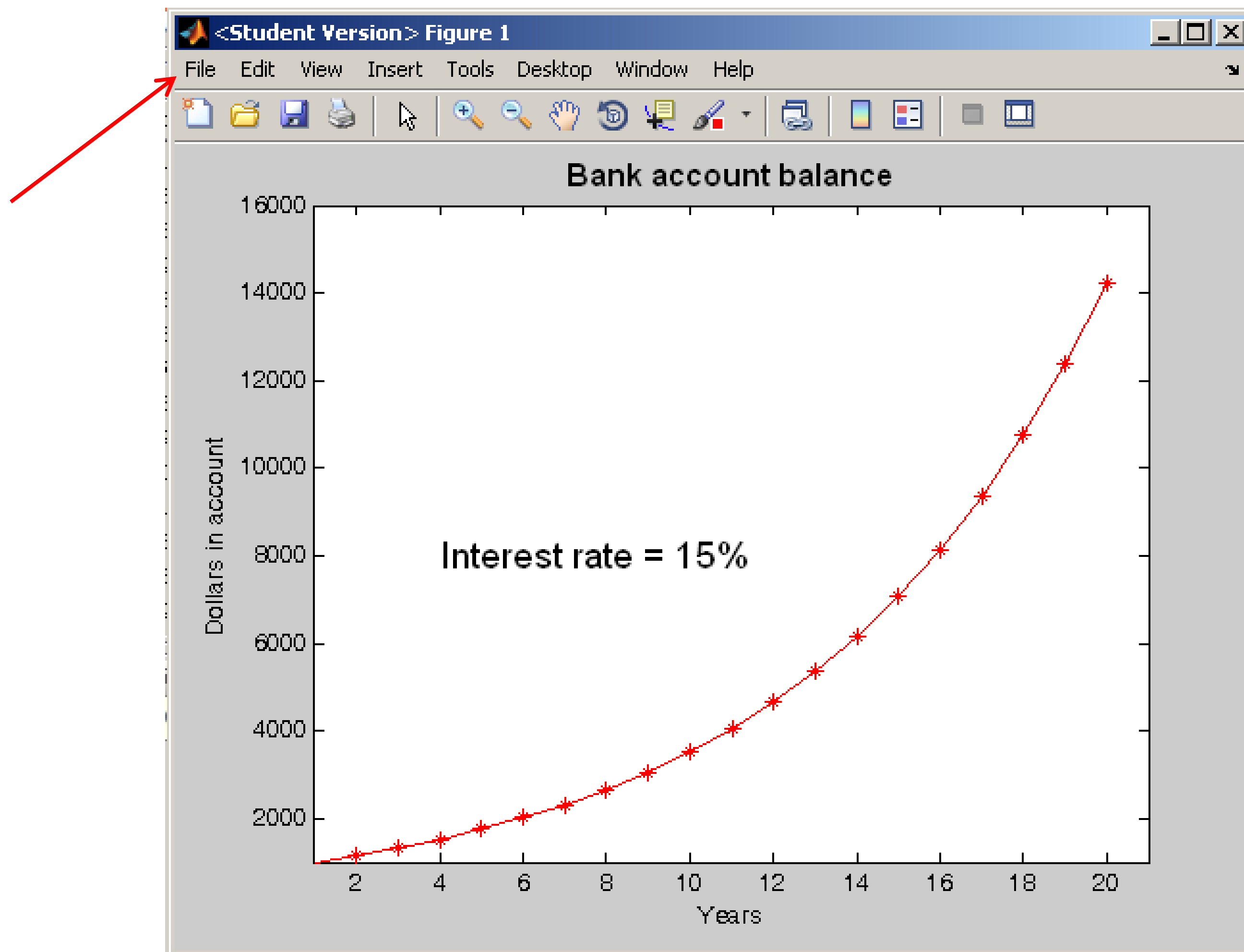
```
clear;clc
birth = 0.15;
death = 0.000;
deltat = 1;
P(1) = 1000;
for t = [1:19]
   P(t+1) = P(t)+birth*P(t);
end
figure(1)
plot(1:20,P,'-*r')
title('Bank account balance', ...
    'FontName', 'Arial', ...
    'FontSize', 12, ...
    'FontWeight', 'Bold')
xlabel('Years')
ylabel('Dollars in account')
xlim([1 21])
ylim([1000 16000])
text(4,8000,'Interest rate = 15%', ...
    'FontName','Arial','FontSize',14)
```



Again, font size, weight, etc. can be controlled just like the title command

BASIC GRAPHS and PLOTS

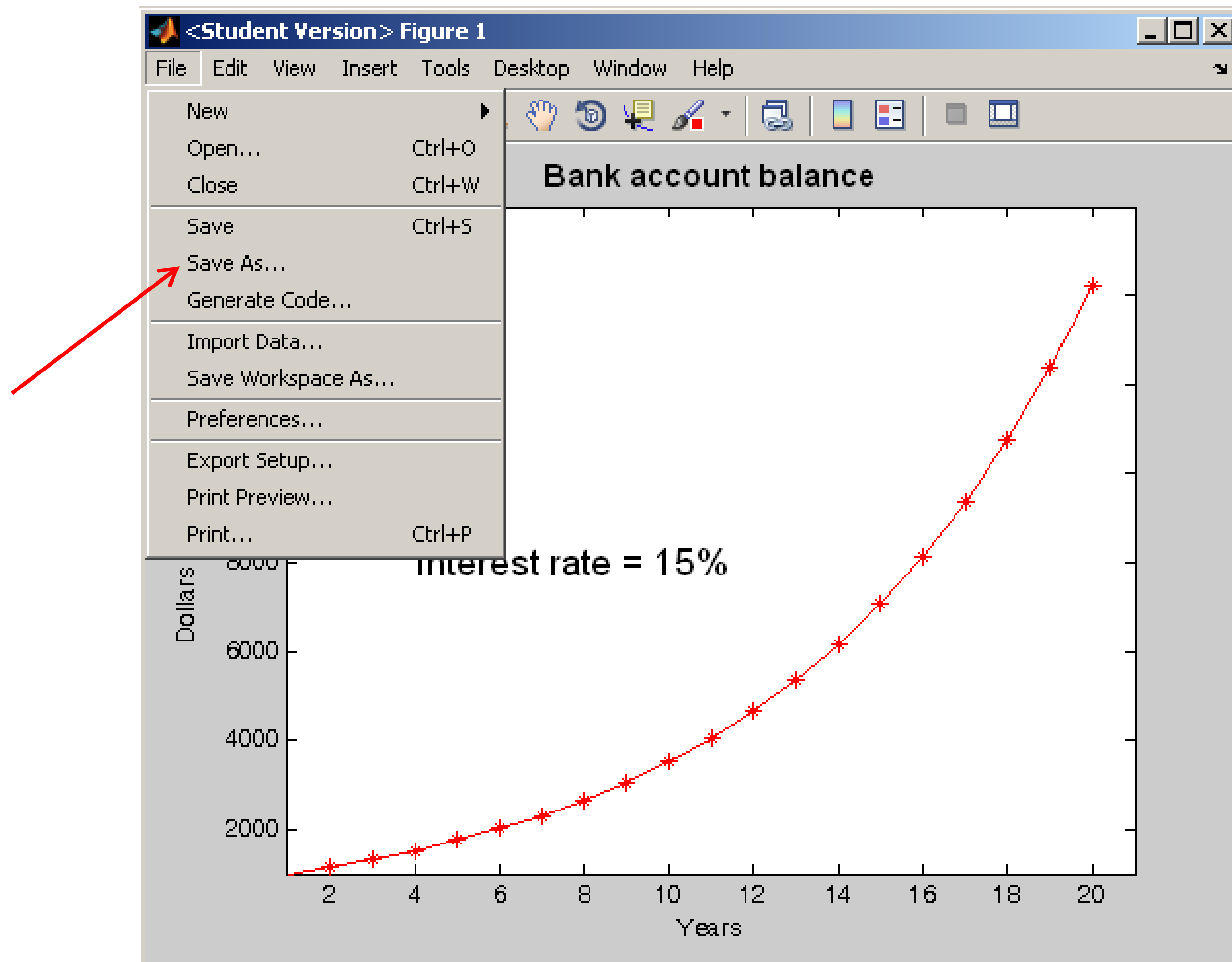
Now, switch to the window containing the figure and click on “File”:



GEORGE MASON UNIVERSITY

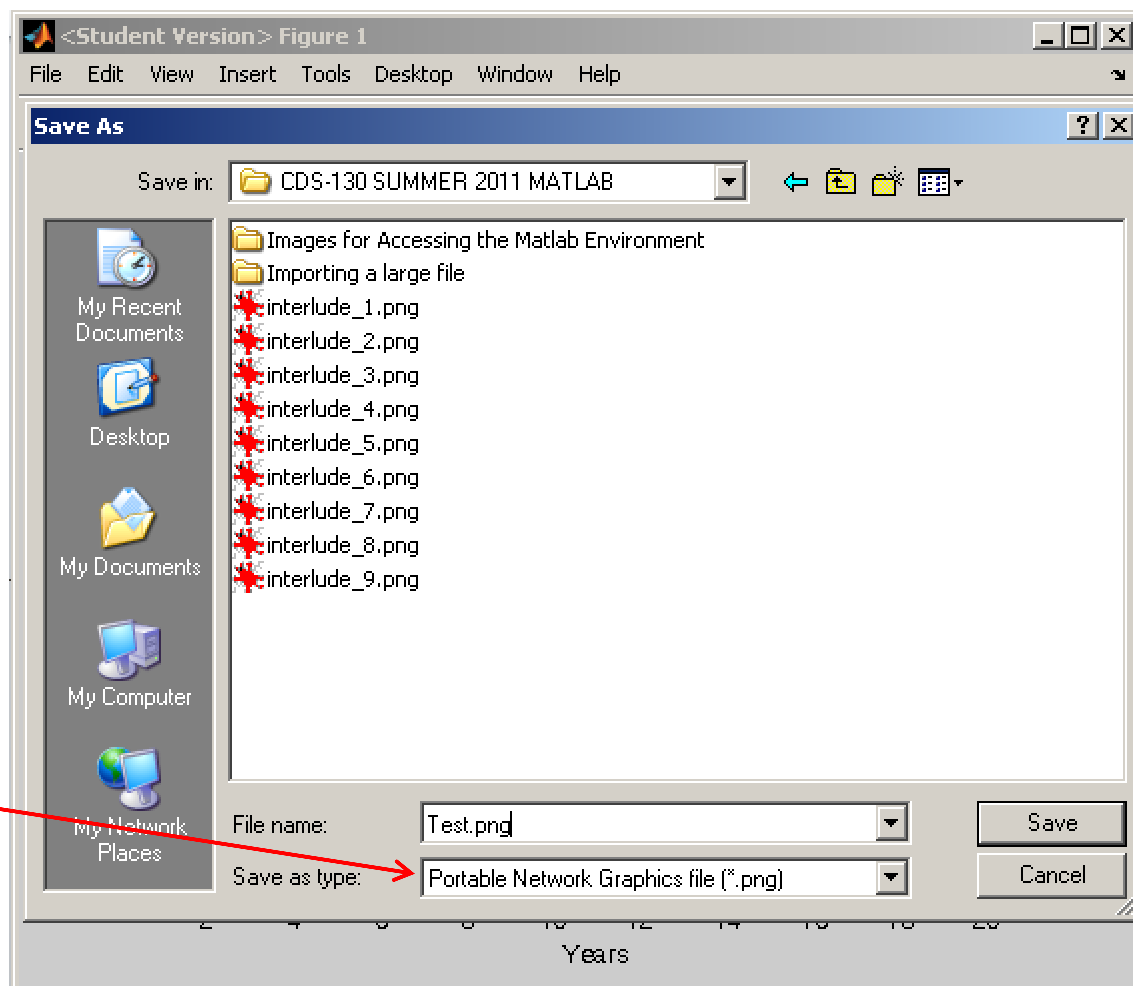
BASIC GRAPHS and PLOTS

Select “Save As”:



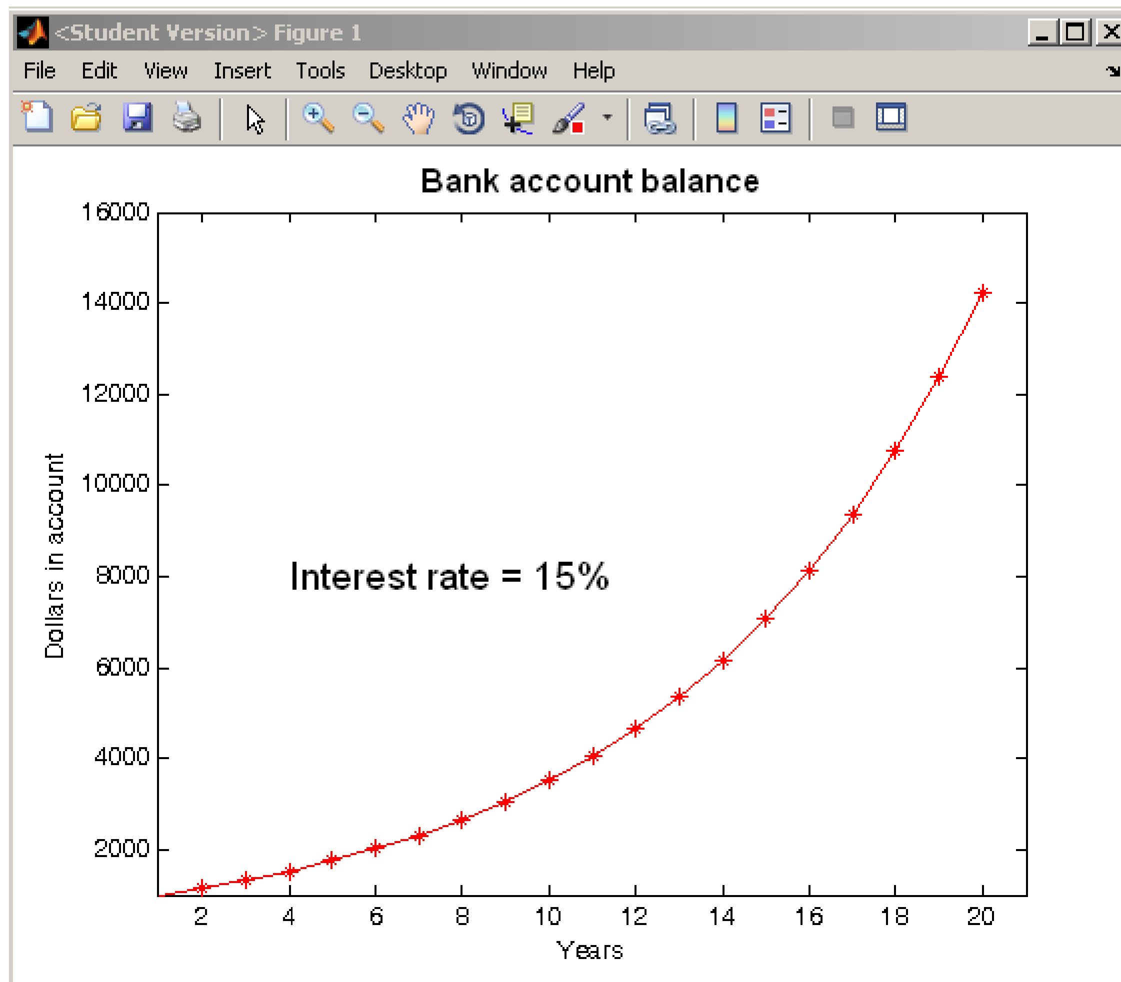
BASIC GRAPHS and PLOTS

Navigate to the directory where you want to save the file, enter a filename, and ensure “Save as Type” is “PNG”:



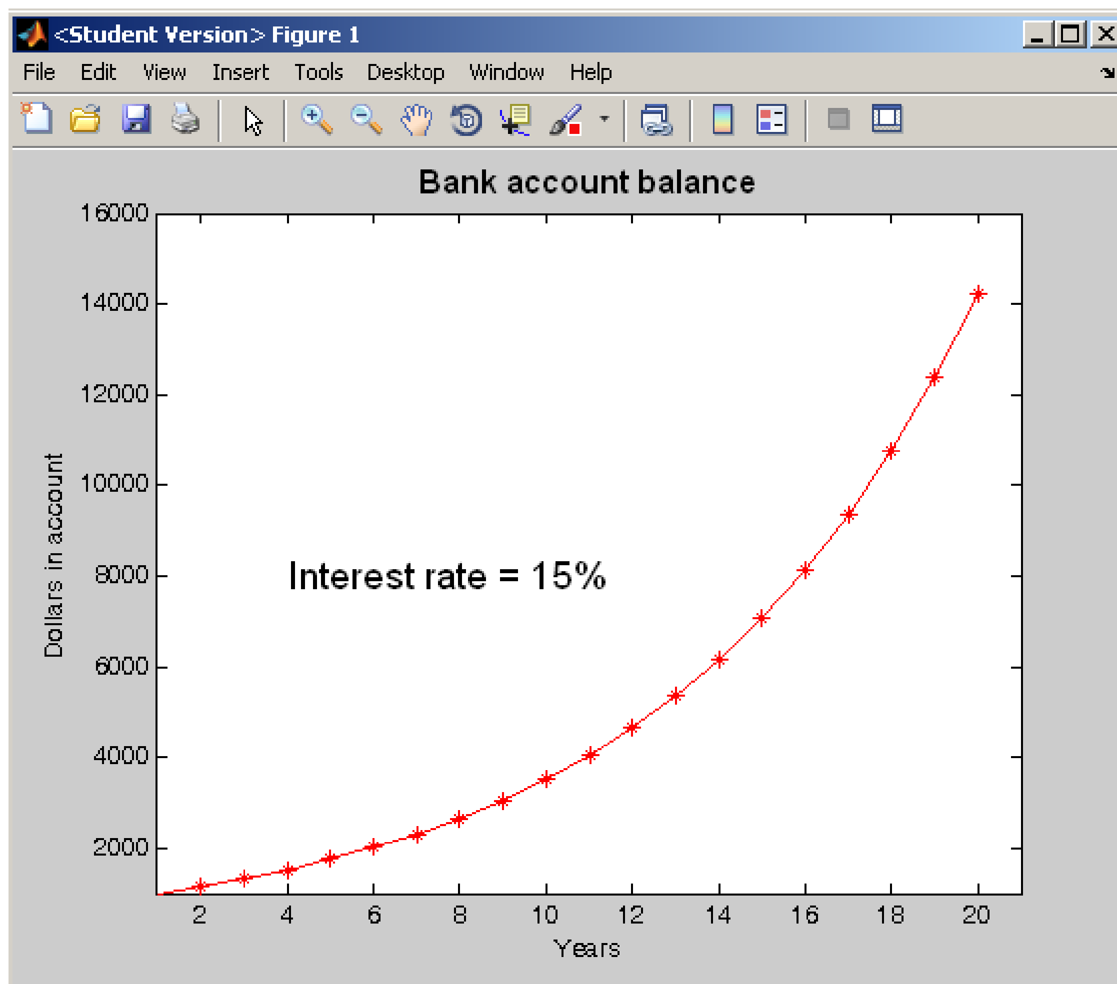
BASIC GRAPHS and PLOTS

You'll be taken back to the figure. It will flash white for an instant:



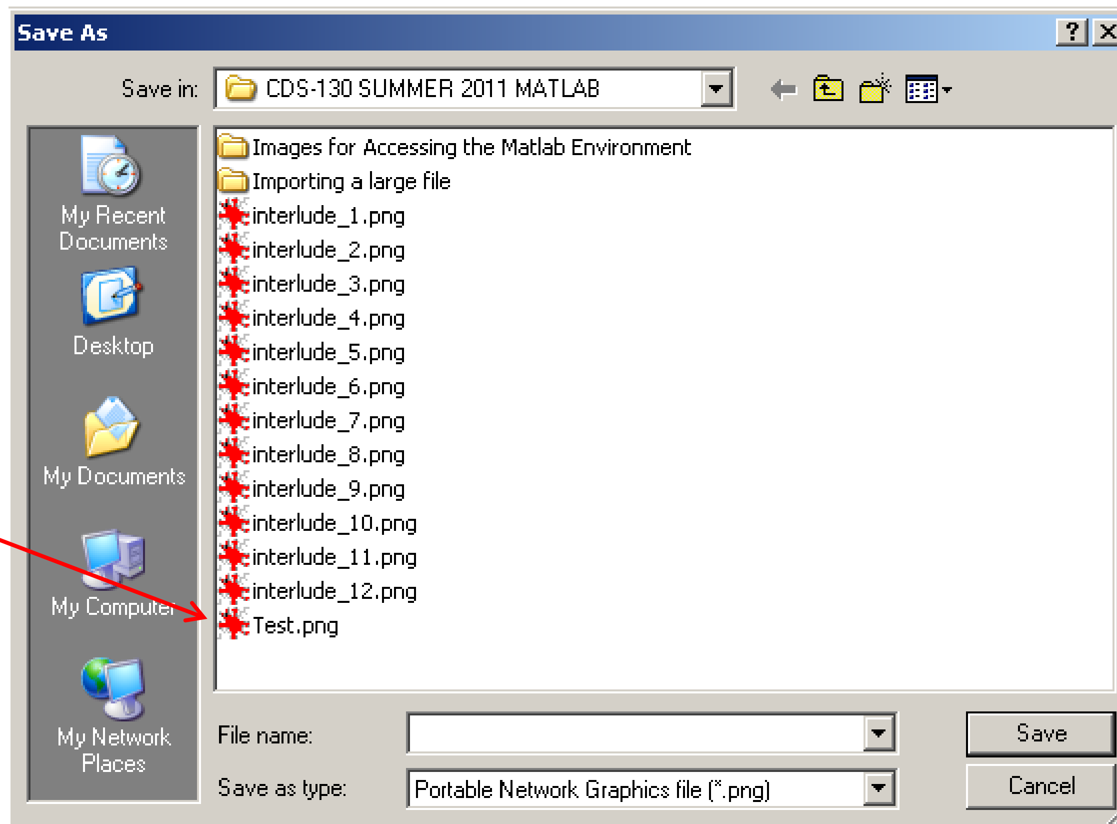
BASIC GRAPHS and PLOTS

And then it will return to its “normal self”, with a grey background. The image has been saved . . .



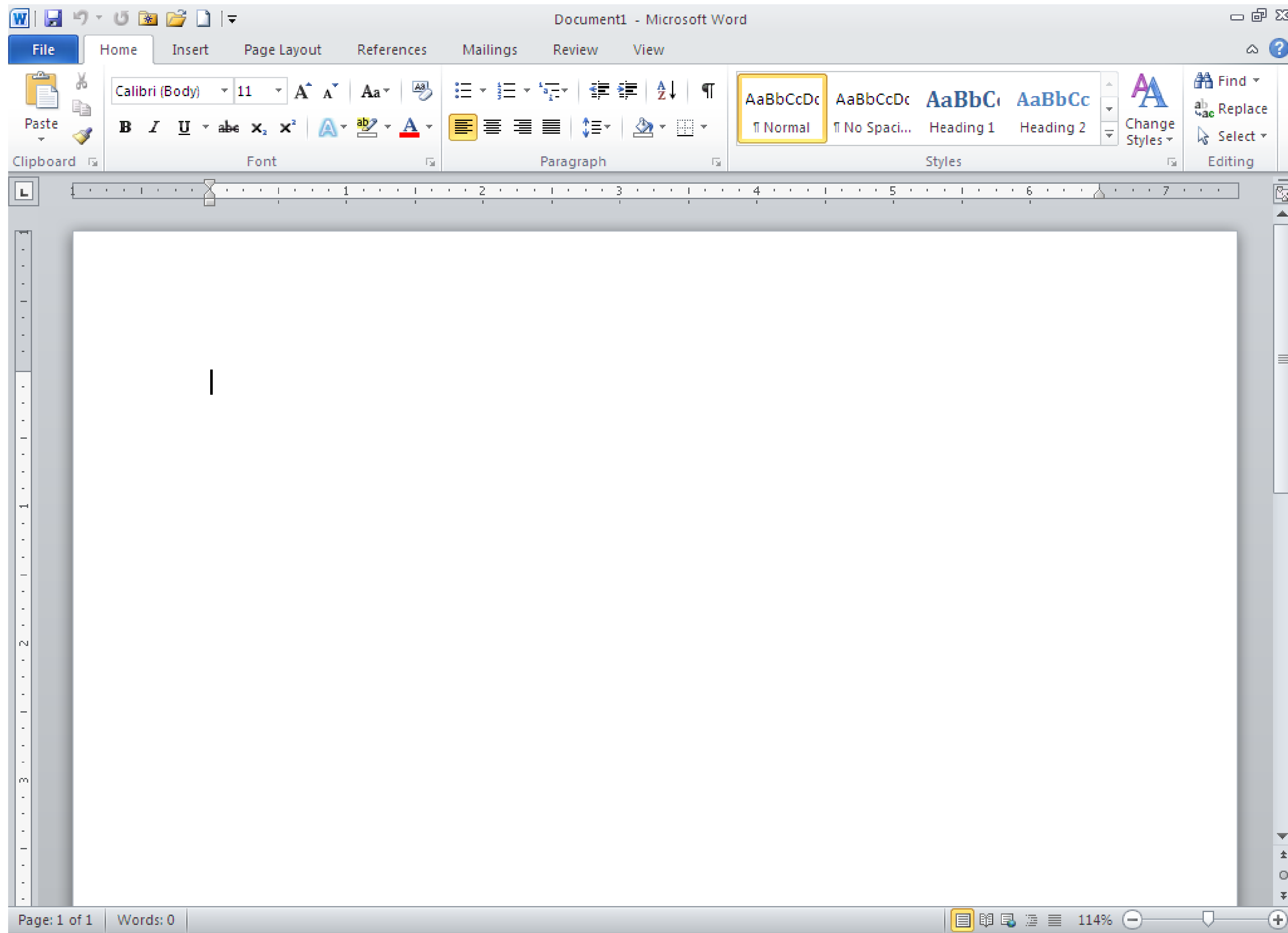
BASIC GRAPHS and PLOTS

Which you can now verify by inspecting the directory structure:



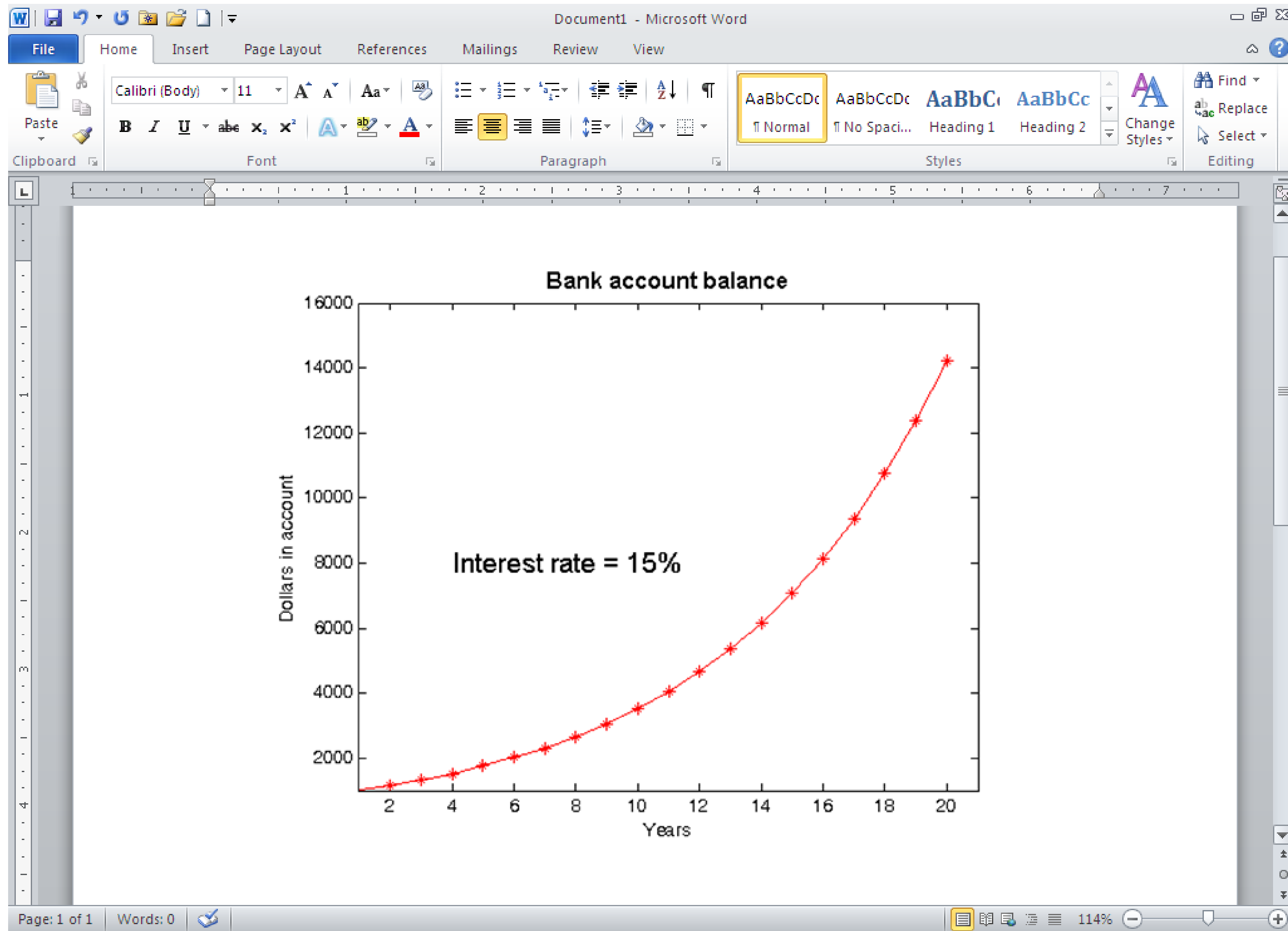
BASIC GRAPHS and PLOTS

Now you can open Microsoft Word:



BASIC GRAPHS and PLOTS

And paste your .PNG graph right onto the page:



GEORGE MASON UNIVERSITY

CHAPTER 8

Write a Matlab Program

Matlab Tutorial Video

1. Online demo video - Writing a Matlab Program (04:57)
   - http://www.mathworks.com/videos/writing-a-matlab-program-69023.html

Links are also available at class website resource page:
http://spaceweather.gmu.edu/jzhang/teaching/2013_CDS301_Spring/resource.html

April 04, 2013

GEORGE MASON UNIVERSITY

CHAPTER 9

ITERATION II: DOUBLE-NESTED FOR LOOPS (DNFL)

GEORGE MASON UNIVERSITY

ITERATION: Double Nested FOR Loops (DNFL)

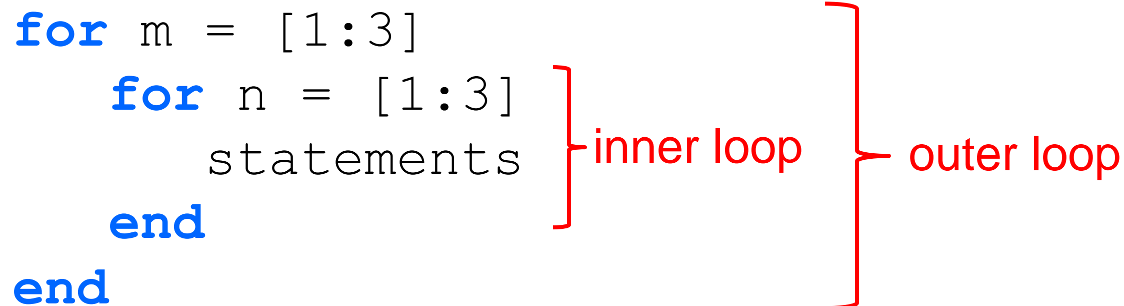
Now we come to a very important syntactic structure, the ***Double Nested FOR Loop*** (aka, “DNFL”)

This is important precisely because double nested FOR loops enable us to “walk through” a two dimensional matrix, element by element.

This is the basis for ***image processing algorithms***.

ITERATION: DNFL

- **Syntax**: ***As shown, and always the same*** (except for the end limit of the indexes `m` and `n`, which could be any number other than 3—and frequently will be!)



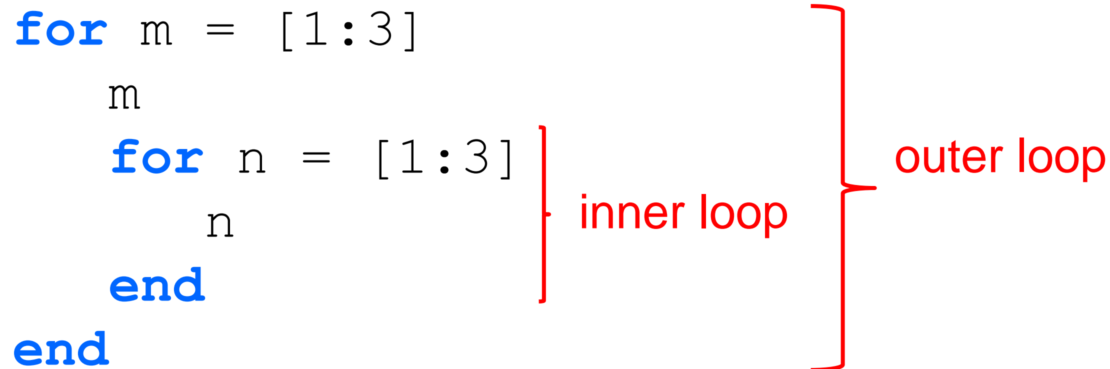
ITERATION: DNFL

```
for m = [1:3]
    for n = [1:3]
        statements
    end
end
```


What's happening:

- `m` is assigned the value 1
- `n` is assigned the value 1
- the `statements` are executed
- the first **end** is encountered, sending execution to the top of the inner loop . More to do? If not, then . . .
- the second **end** is encountered, sending execution to the top of the outer loop. More to do? If not, then EXIT the outer loop and program execution continues onward, immediately following the second end statement.

ITERATION: DNFL



Simple example: When the above code is run, we get the following output . . .

```
m = 1        m = 2        m = 3
n = 1        n = 1        n = 1
n = 2        n = 2        n = 2
n = 3        n = 3        n = 3
```

First iteration outer; inner iterates 3 times

Second iteration outer; inner iterates 3 times

Third iteration outer; inner iterates 3 times

ITERATION: DNFL

It is essential that each of you understand ***WHY*** each of the values of `m` and `n` printed out the way that they did.

Any questions so far?

ITERATION: DNFL – YOUR TURN!

Instructions:

For the next several examples, please try to work out the answers without running the code in Matlab. This is essential, as it will enable you to develop your "Matlab intuition" and also to visualize the sequence of a computation (thus developing your ability to think algorithmically). Furthermore, you will not be allowed to use Matlab software on exams or quizzes and so it's better to get the practice now rather than wait until later! You may, however, use scratch paper to work out answers.

ITERATION: DNFL – YOUR TURN!

Example 21:

```
a = 1;
b = 1;
for m = [1:3]
    for n = [1:3]
        a
        b
    end
end
```

What is printed out by the above Double Nested FOR loop?

ITERATION: DNFL – YOUR TURN!

Example 21:

```
a = 1;
b = 1;
for m = [1:3]
    for n = [1:3]
        a
        b
    end
end
```

What is printed out by the above Double Nested FOR loop?

```
a = 1
b = 1
 (repeated nine times)
```

ITERATION: DNFL – YOUR TURN!

Example 22:

```
a = 1;
b = 1;
for m = [1:3]
    for n = [1:3]
        a = a + 1;
        b = b + a
    end
end
```

What is printed out by the above Double Nested FOR loop?
(You can use a calculator)

ITERATION: DNFL – YOUR TURN!

Example 22:

```
a = 1;
b = 1;
for m = [1:3]
    for n = [1:3]
        a = a + 1;
        b = b + a
    end
end
```

b = 3	b = 15	b = 36
b = 6	b = 21	b = 45
b = 10	b = 28	b = 55
When m = 1	When m = 2	When m = 3

ITERATION: DNFL – YOUR TURN!

Example 23:

```
a = 1;
b = 1;
for m = [1:3]
    for n = [1:3]
        a = a + b;
        b = b + a
    end
end
```

BIG NUMBERS

What is printed out by the above Double Nested FOR loop?
(You can use a calculator)

ITERATION: DNFL – YOUR TURN!

Example 23:

```
a = 1;
b = 1;
for m = [1:3]
    for n = [1:3]
        a = a + b;
        b = b + a
    end
end
```

BIG NUMBERS

```
b =   3        b =   55        b =   987
b =   8        b =   144       b =   2584
b =   21       b =   377       b =   6765

When m = 1     When m = 2      When m = 3
```

ITERATION: DNFL – YOUR TURN!

Example 24:

```
a = 2;
b = 3;
for m = [1:3]
    for n = [1:3]
        a = a + b;
        b = b + 1
    end
end
```

TINY BIT TRICKY!!

What is printed out by the above Double Nested FOR loop?
(You can use a calculator—HINT: IF you really, really need one!)

ITERATION: DNFL – YOUR TURN!

Example 24:

```
a = 2;
b = 3;
for m = [1:3]
    for n = [1:3]
        a = a + b;
        b = b + 1
    end
end
```

TINY BIT TRICKY!!

b = 4	b = 7	b = 10
b = 5	b = 8	b = 11
b = 6	b = 9	b = 12
When m = 1	When m = 2	When m = 3

ITERATION: DNFL – YOUR TURN!

Example 25:

```
A(3,3) = 0;
for m = [1:3]
   for n = [1:3]
      A(m,n) = m;
   end
end
A
```

What is printed out by the above Double Nested FOR loop?

ITERATION: DNFL – YOUR TURN!

Example 25:

```
A(3,3) = 0;
for m = [1:3]
    for n = [1:3]
        A(m,n) = m;
    end
end
A
```

```
A =

     1     1     1
     2     2     2
     3     3     3
```

ITERATION: DNFL – YOUR TURN!

Example 26:

```
A(3,3) = 0;
for m = [1:3]
    for n = [1:3]
        A(m,n) = n;
    end
end
A
```

What is printed out by the above Double Nested FOR loop?

ITERATION: DNFL – YOUR TURN!

Example 26:

```
A(3,3) = 0;
for m = [1:3]
   for n = [1:3]
      A(m,n) = n;
   end
end
A
```

```
A =

     1     2     3
     1     2     3
     1     2     3
```

April 09, 2013

ITERATION: DNFL – YOUR TURN!

Example 27:

```
A = [1, 2, 3; 4, 5, 6; 7, 8, 9];
for m = [1:3]
    for n = [1:3]
        A(m,n) = m - n;
    end
end
A
```

What is printed out by the above Double Nested FOR loop?

ITERATION: DNFL – YOUR TURN!

Example 27:

```
A = [1, 2, 3; 4, 5, 6; 7, 8, 9];
for m = [1:3]
    for n = [1:3]
        A(m,n) = m - n;
    end
end
A
```

```
A =

     0    -1    -2
     1     0    -1
     2     1     0
```

ITERATION: DNFL – YOUR TURN!

Example 28:

```
A = [1, 2, 3; 4, 5, 6; 7, 8, 9];
for m = [1:3]
    for n = [1:3]
        A(m,n) = m*n + 1;
    end
end
A
```

What is printed out by the above Double Nested FOR loop?

ITERATION: DNFL – YOUR TURN!

Example 28:

```
A = [1, 2, 3; 4, 5, 6; 7, 8, 9];
for m = [1:3]
    for n = [1:3]
        A(m,n) = m*n + 1;
    end
end
A
```

```
A =

     2     3     4
     3     5     7
     4     7    10
```

ITERATION: DNFL – YOUR TURN!

Example 29:

```
A = [1, 2, 3; 4, 5, 6; 7, 8, 9];
B = [1 -2  2; 2 -2 3; 3 -3 4];
C(3,3) = 0;
for m = [1:3]
    for n = [1:3]
        C(m,n) = A(m,n) - B(m,n);
        C(m,n) = C(m,n)*(-1);
    end
end
C
```

What is printed out by the above Double Nested FOR loop?

ITERATION: DNFL – YOUR TURN!

Example 29:

```
A = [1, 2, 3; 4, 5, 6; 7, 8, 9];
B = [1 -2  2; 2 -2 3; 3 -3 4];
C(3,3) = 0;
for m = [1:3]
    for n = [1:3]
        C(m,n) = A(m,n) - B(m,n);
        C(m,n) = C(m,n)*(-1);
    end
end
C
```

```
C =

     0    -4    -1
    -2    -7    -3
    -4   -11    -5
```

GEORGE MASON UNIVERSITY

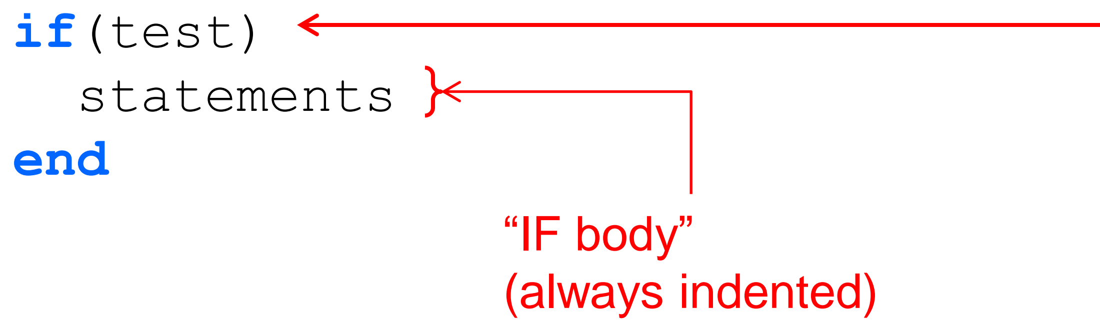
CHAPTER 10

CONDITIONALS: IF STATEMENTS

IF STATEMENTS (Conditionals)

- **Syntax #1**: ***As shown, <u>for first variant</u>***.

NOTE: The keywords `if` and `end` come in a pair—***never one without the other***.



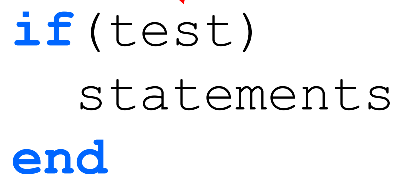
What's happening in the first variant:

- IF checks `test` to see if `test` is **TRUE**
- if `test` is **TRUE**, then `statements` are executed
- if `test` is **FALSE**, then nothing happens, and program execution continues immediately after the `end` statement.

SIDE NOTE: A LOGICAL TEST

IF STATEMENTS (Conditionals)

- **What is this thing, "`test`", all about?**



- `test` is what's called a "logical test", or, "conditional". It's like asking the question, "Is this statement true?"
- Logical tests evaluate to a single truth value: either TRUE, or FALSE (never both!)
- The formal name for a `test` that evaluates to either TRUE or to FALSE, is a **PREDICATE**.

GEORGE MASON UNIVERSITY

IF STATEMENTS (Conditionals)

- **What is this thing, "`test`", all about?**

```
if(test)
   statements
end
```

- The predicate is (usually) composed of a LHS and a RHS featuring "logical operators" and "relational operators":

(logical connective) && means "AND"
(logical connective) || means "OR"
(logical connective) ~ means "NOT"
(relational operator) > means "Greater than"
(relational operator) < means "Less than"
(relational operator) >= means "Greater-than-or-equal-to"
(relational operator) <+ means "Less-than-or-equal-to"
(relational operator) == means "Equal-to"
(relational operator) ~= means "NOT Equal-to"

GEORGE MASON UNIVERSITY

IF STATEMENTS (Conditionals)

- **What is this thing, "`test`", all about?**

```
if(test)
   statements
end
```

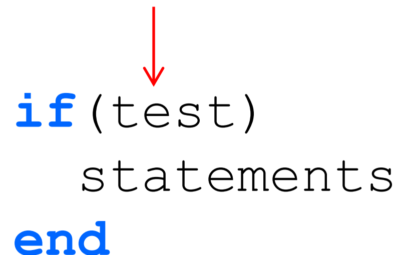
- So a complete `test` (or predicate) usually has a left hand side, one or more logical connectives and/or relational operators, and a right hand side.

- **The complete `test` asks a question that is answered only TRUE or FALSE (or if you prefer, "yes" or "no") – BUT NEVER BOTH (called, the "Law of the Excluded Middle")**

- Example predicate: The `test (x < y)` asks the question, "**Is x less than y?**" There is ONLY one answer to this question: YES (true) or NO (false).

IF STATEMENTS (Conditionals)

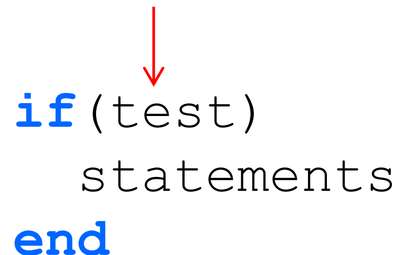
- **What is this thing, “`test`”, all about?**



Predicate Example	Relational Operator	Equivalent Question(s)
`(x > y)`	`>`	`Is x greater than y?`
`(x <= y)`	`<=`	`Is x less-than-or-equal-to y?`
`(x >= y)`	`>=`	`Is x greater-than-or-equal-to y?`
`(x == y)`	`==`	`Is x equal to y?`

IF STATEMENTS (Conditionals)

- **What is this thing, “`test`”, all about?**



```
if(test)
   statements
end
```

Predicate Example	Relational / Logical / Relational Sequence	Equivalent Question(s)
`(x == y)&&(a == b)`	`==, &&, ==`	Is x equal to y AND is a equal to b?
`(x == y)||(a < b)`	`==, ||, <`	Is x equal to y OR is a less than b?
`(x ~= y)&&(a >= 0)`	`~=, &&, >=`	Is x NOT equal to y AND is a greater-than-or-equal-to 0?

How we write “NOT”

GEORGE MASON UNIVERSITY

IF STATEMENTS (Conditionals)—Your Turn !

Instructions:

For the next several examples, please try to work out the answers without running the code in Matlab. This is essential, as it will enable you to develop your "Matlab intuition" and also to visualize the sequence of a computation (thus developing your ability to think algorithmically). Furthermore, you will not be allowed to use Matlab software on exams or quizzes and so it's better to get the practice now rather than wait until later! You may, however, use scratch paper to work out answers.

IF STATEMENTS (Conditionals)—Your Turn !

Example 31:

```
c = 1;
a = 1;
b = 2;
if(a + b < 3)
   c = c + 1;
end
```

Question: Will `c` be incremented by 1? Why or why not?

IF STATEMENTS (Conditionals)—Your Turn !

Example 31:

```
c = 1;
a = 1;
b = 2;
if(a + b < 3)
   c = c + 1;
end
```

Question: Will `c` be incremented by 1? Why or why not?

No, it will not: a + b is equal to 3 and not less than 3!

IF STATEMENTS (Conditionals)—Your Turn !

Example 32:

```
c = 1;
a = pi;
b = 2;
if(cos(a*b) == 1)
  c = c + 1;
end
```

Question: Will `c` be incremented by 1? Why or why not?

IF STATEMENTS (Conditionals)—Your Turn !

Example 32:

```
c = 1;
a = pi;
b = 2;
if(cos(a*b) == 1)
   c = c + 1;
end
```

Question: Will `c` be incremented by 1? Why or why not?

Yes: cosine(2*π) is equal to 1

IF STATEMENTS (Conditionals)—Your Turn !

Example 33:

```
c = 1;
a = pi;
b = 2;
if(cos(a*b) == 1 && a < b)
   c = c + 1;
end
```

Question: Will `c` be incremented by 1? Why or why not?

GEORGE MASON UNIVERSITY

IF STATEMENTS (Conditionals)—Your Turn !

Example 33:

```
c = 1;
a = pi;
b = 2;
if(cos(a*b) == 1 && a < b)
   c = c + 1;
end
```

Question: Will `c` be incremented by 1? Why or why not?

No: Although cosine(2*π) is equal to 1,
a is <u>NOT</u> less than b (rather: π > 2),
and so <u>BOTH CONDITIONS ARE NOT TRUE TOGETHER</u>,
which is what is required by AND,
and so the statements bounded
by IF and END will not execute!

IF STATEMENTS (Conditionals)—Your Turn !

Example 34:

```
c = 1;
a = pi;
b = 2;
if(cos(a*b) == 0 || a > b)
   c = c + 1;
end
```

Question: Will `c` be incremented by 1? Why or why not?

IF STATEMENTS (Conditionals)—Your Turn !

Example 34:

```
c = 1;
a = pi;
b = 2;
if(cos(a*b) == 0 || a > b)
  c = c + 1;
end
```

Question: Will `c` be incremented by 1? Why or why not?

Yes: Although cosine(2*π) is NOT equal to 0, in this case, we're dealing with an OR, which means either the first condition OR the second condition can be true, and then the entire test is considered true. That happens here.

IF STATEMENTS (Conditionals)—Your Turn !

Example 35:

```
c = 1;
for m = 1:3
   if(m <= m^2)
      c = c + 1;
   end
end
c
```

Question: What final value of `c` is printed out?

GEORGE MASON UNIVERSITY

IF STATEMENTS (Conditionals)—Your Turn !

Example 35:

```
c = 1;
for m = 1:3
    if(m <= m^2)
        c = c + 1;
    end
end
c
```

Question: What final value of `c` is printed out?

c = 4

IF STATEMENTS (Conditionals)—Your Turn !

Example 36:

```
a = 0;
c = 1;
for m = 1:3
    for n = 1:3
        if(m > n)
            c = c + 1;
            a = c;
        end
    end
end
a
```

Question: What final value of `a` is printed out?

IF STATEMENTS (Conditionals)—Your Turn !

Example 36:

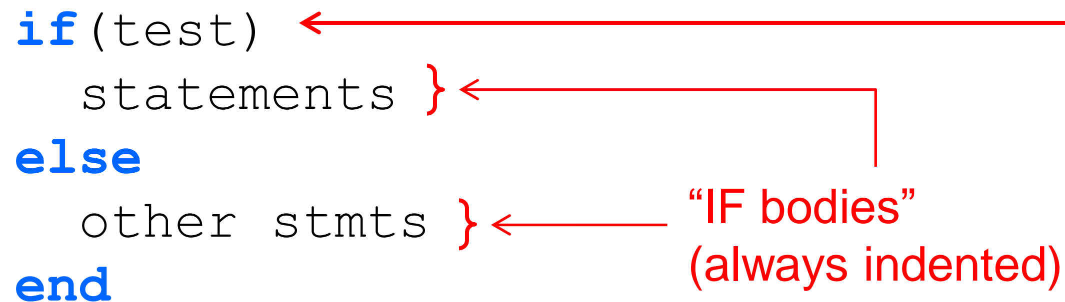
```
a = 0;
c = 1;
for m = 1:3
    for n = 1:3
        if(m > n)
            c = c + 1;
            a = c;
        end
    end
end
a
```

Question: What final value of `a` is printed out?

a = 4

IF STATEMENTS (Conditionals)

• **IF STATEMENT Syntax #2**: ***As shown, <u>for second variant</u>***. **NOTE**: The keywords **`if`** and **`end`** come in a pair—***never one without the other***.



SIDE NOTE: A LOGICAL TEST

What's happening in the second variant:

- IF checks `test` to see if `test` is **TRUE**
- if `test` is **TRUE**, then `statements` are executed
- if `test` is **FALSE**, then `other stmts` are executed, and overall program execution continues immediately after the **`end`** statement.

GEORGE MASON UNIVERSITY

IF STATEMENT Syntax #2: A way to select between TRUE and FALSE:

```
(assume a and b were
 previously assigned)

if(cos(a*b) == 0)
  c = c + 1;
else
  c = c - 1;
end
```

What happens now depends upon whether `cos(a*b) == 0` evaluates to TRUE or to FALSE: If to TRUE, then the statement `c = c + 1;` is executed but if FALSE, then the statement `c = c - 1;` is executed instead.

IF STATEMENT Syntax #2: ILLUSTRATION

```
(assume a and b were
 previously assigned)

A(5,5) = 0.0;
A = (A + 2)/2.0;
if(cos(a*b) == 0)
   c = c + 1;
else
   for m = [1:5]
       for n = [1:5]
           A(m,n) = A(m,n)/2.0
       end
   end
end
```

Here, an **entire** double nested FOR loop is executed if `cos(a*b) == 0` is false! If not, the variable `c` is incremented instead.

IF STATEMENTS (Conditionals)

- **IF STATEMENT Syntax #3**: ***As shown, <u>for third variant</u>***. **NOTE**: The keywords **`if`** and **`end`** come in a pair—***never one without the other***. **`elseif`** doesn't need an **`end`** statement:

```
if(first test)
   first set of statements
elseif(second test)
   second set of statements
elseif(third test)
   third set of statements
       .
       .
       .
end
```

Multiple tests!

<u>Multiple tests</u> MEANS <u>multiple outcomes</u> are possible!

IF STATEMENTS (Conditionals)

IF STATEMENT Syntax #3: ILLUSTRATION 1

(assume a, b and c were previously assigned)

```
A(5,5) = 0.0;
A = (A + 2)/2.0;
if(cos(a*b) == 0)
   c = c + 1;
elseif(cos(a*b) < 0)
   for m = [1:5]
      for n = [1:5]
         A(m,n) = A(m,n)/2.0
      end
   end
elseif(cos(a*b) > 0)
   c = c - 1;
end
```

GEORGE MASON UNIVERSITY

IF STATEMENT Syntax #3: ILLUSTRATION 2

(assume matrix A and variables a, b and c were all previously assigned)

```
if(A(1,1) == 0 && A(1,2) == 0 && A(1,3) == 0)
    c = c + 1;
elseif(A(1,1) == 0 && A(1,2) == 0 && A(1,3) == 1)
    c = c - 1;
elseif(A(1,1) == 0 && A(1,2) == 1 && A(1,3) == 0)
    c = a*b;
elseif(A(1,1) == 0 && A(1,2) == 1 && A(1,3) == 1)
    c = a/b
elseif(A(1,1) == 1 && A(1,2) == 0 && A(1,3) == 0)
    c = cos(a)*b
elseif(A(1,1) == 1 && A(1,2) == 0 && A(1,3) == 1)
    c = a*cos(b)
elseif(A(1,1) == 1 && A(1,2) == 1 && A(1,3) == 0)
    c = log(a*b)
elseif(A(1,1) == 1 && A(1,2) == 1 && A(1,3) == 1)
    c = a^b
end
```

WOW!! A Multi-Branch IF that covers eight possibilities!

IF STATEMENT Syntax #3 ILLUSTRATIONS 1 & 2

The purpose of ILLUSTRATIONS 1 & 2 was to demonstrate to you that by using Syntax #3 (and also Syntax #1 and #2), it is possible to build up some very complex program capability! The ability of a program to evaluate conditionals and then select from a (possibly) wide range of available processing options – based on the outcome of the conditional evaluation – is what's enabled by the IF statement and its variations.

IF STATEMENTS (Conditionals)

Example 37:

```
a = 0;
c = 0;
for m = 1:3
    for n = 1:3
        if(m <= n)
            c = c + 1;
        end
        if(c > a)
            a = a + 1;
        end
    end
end
a
```

TRICKY!!

Question: What final value of `a` is printed out?

GEORGE MASON UNIVERSITY

IF STATEMENTS (Conditionals)

Example 37:

```
a = 0;
c = 0;
for m = 1:3
    for n = 1:3
        if(m <= n)
            c = c + 1;
        end
        if(c > a)
            a = a + 1;
        end
    end
end
a
```

TRICKY!!

Question: What final value of `a` is printed out?

a = 6

IF STATEMENTS (Conditionals)

Example 38:

```
a = 0;
c = 1;
for m = 1:3
    for n = 1:3
        if(m <= n && c ~= a)
            c = c + 1;
        end
        if(c > a^2)
            a = a + 1;
        end
    end
end
a
```

VERY TRICKY!!

Question: What final value of `a` is printed out?

a = 6

IF STATEMENTS (Conditionals)

Example 38:

```
a = 0;
c = 1;
for m = 1:3
    for n = 1:3
        if(m <= n && c ~= a)
            c = c + 1;
        end
        if(c > a^2)
            a = a + 1;
        end
    end
end
a
```

VERY TRICKY!!

Question: What final value of `a` is printed out?

a = 3

The End

GEORGE MASON UNIVERSITY