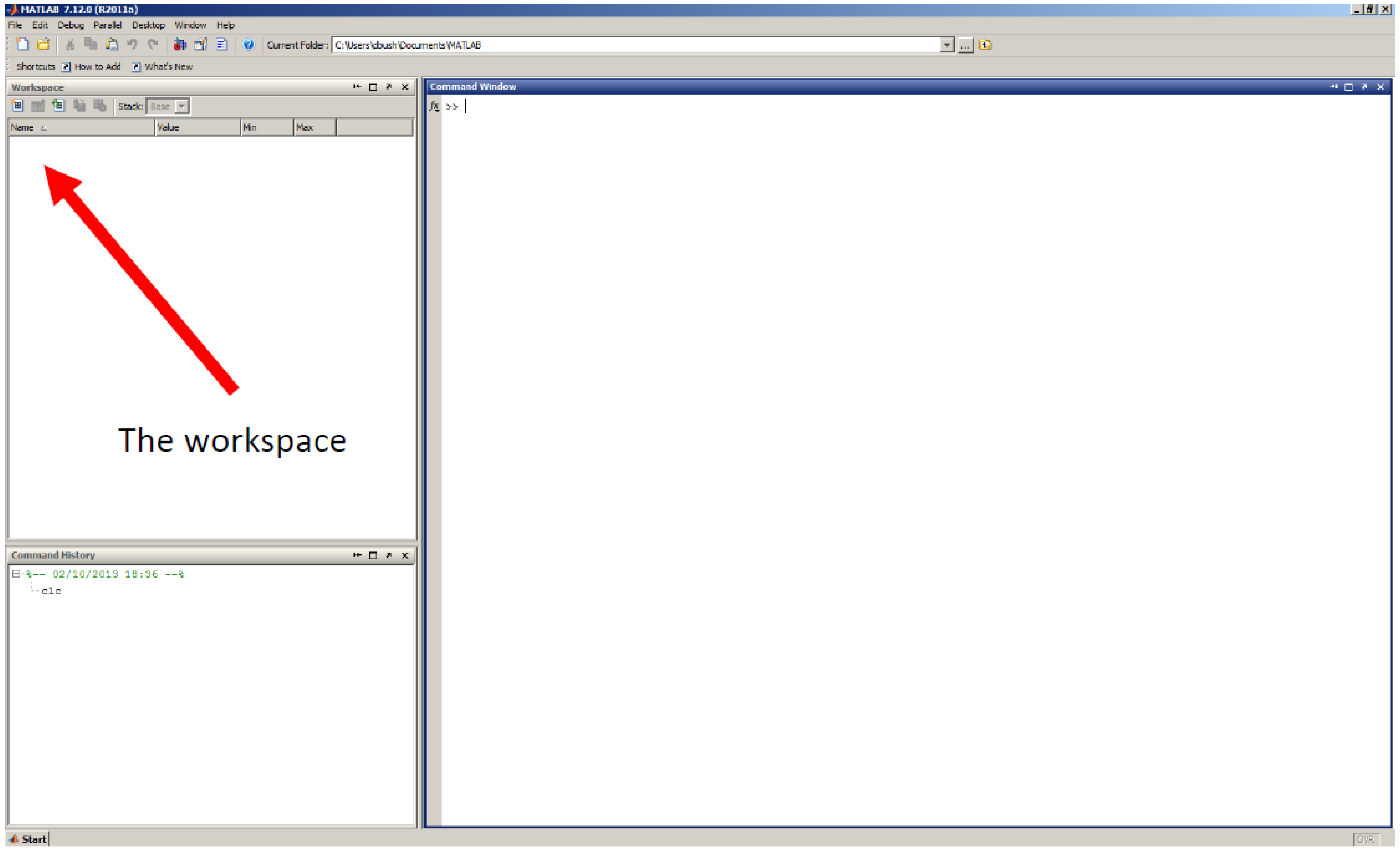
Introduction to Biomedical Engineering

Section 4: Basics of High-level programming: Matlab

Lecture 4.1 Basics of coding in Matlab

POLYTECH
Peter the Great
St.Petersburg Polytechnic
University

OPEN
POLYTECH



Current Folder: C:\Users\dbush\Documents\MATLAB

```
%-- 02/10/2013 18:36 --%
clc
```

```
>>
```

The exact layout can differ from machine to machine, but the windows are always labelled!

Variables

- MATLAB **does not care** about **spaces** in expressions
- In MATLAB, you can also assign values to **variables**
- **Mathematical operations** can then be performed on those **variables** in the **same way**
- All variables in the current ‘**stack**’ (i.e. **in memory**) appear in the **workspace**
- **Unassigned** output is automatically placed in the variable ‘**ans**’ (i.e. answer)
- **Output** to the **command window** can be **suppressed** with a **semi-colon** ‘;’…
- …but the value of the **variable** in the **workspace** will still be **updated**
- You can **display** the **value** of any variable by **typing its name** and pressing return
- You can **clear all variables** from the workspace by typing ***clear***
- You can clear **individual variables** by typing ***clear variable_name***

Vectors and matrices

- MATLAB is specifically designed to perform operations on **matrices** or **vectors**
- Matrices and vectors are assigned with **square brackets**
- Rows are **separated by semi-colons** within square brackets
- **Numerical sequences** can be assigned with **colons** (i.e. *start_value* : *finish_value*)
- The **step size** can also be **defined** (i.e. *start_value* : *step_size* : *finish_value*)

Vectors and Matrices

- To perform '**element-wise**' operations on a **matrix**, you must use the '**.**' prefix
- This means the **operation** is performed on **each element individually**
- To perform **matrix operations** you do not need this prefix
- **Elements** of a matrix can be **indexed** using *matrix_name(n)* or *matrix_name(row, col)*
- The **row index** always **comes first,** then **column** (a handy mnemonic: **Roman Catholic**!)
- **One entire row** of a matrix can be **accessed** using *matrix_name(row , :)*
- Similarly, **one entire column** of a matrix can be **accessed** using *matrix_name(: , col)*
- Matrices can be **collapsed** into vectors using *matrix_name(:)*

Functions

- MATLAB has a **huge** number of **built in functions**
- These range from **very simple, general** functions like '**mean**'…
- …to **very specific, complex** functions like '**bsxfun**'
- The real trick to MATLAB is **learning** which **functions exist** and **how to use them**
- If you want to perform some **operation**, just **Google it** – a **function will exist**!
- The **standard syntax** for all functions is:

```
[output1 output2 …] = function(input1, input2, …)
```

- Again, if you **do not assign the output** to a particular **variable**, it will go into '**ans**'

Functions

- ‘**mean**’: compute the mean of a set of numbers
- ‘**std**’: compute the standard deviation of a set of numbers
- ‘**min**’ and ‘**max**’: extract the minimum and maximum values of a vector or matrix
- ‘**rand**’, ‘**randn**’: generate a uniform or normally distributed random number
- ‘**size**’: output the size of a matrix
- ‘**randperm**’: randomly permute a set of numbers
- ‘**sqrt**’: compute the square root of a set of numbers
- ‘**find**’: find any value or inequality within a vector or matrix

Data Handling

- The **variables in any workspace** can be **saved to disk** as a ***.mat file**
- This can be achieved by typing **save *filename***
- This will **overwrite any existing files** with the **same name, without warning**!
- This can also be achieved through the **toolbar** (**File->Save Workspace As**)
- Files will **automatically be saved** to the **location** in the **Current Folder**
- Make sure you **keep track** of where your **files are saved**!
- Files can be **loaded** in the same way – by typing **load *filename*** or using **File->Open**
- Loading a *.mat file will **overwrite** any **existing variables** with the **same name**
- MATLAB also has a '**path**' of **locations** that it will **search for files or functions**
- You can edit this path using **File-> Set Path**

Matlab Help Is Very Good

- F1

Basics of Programming in Matlab

Why program in MATLAB?

- Allows you to **keep a record** of the commands you have executed
- **Saves time** if you are running **multiple** lines of code **more than once**
- Allows you to **write your own functions** for use by others
- Does **not** require **constant attention**!
- Excellent **built-in debugging** and **straightforward syntax**

But First… Strings, Cells and Structures

- MATLAB deals with **many different types of variable**
- So far we have only considered **single numbers**, **arrays** and **matrices**
- MATLAB can also operate on **strings**, which are just **text variables**
- Strings are entered using **single quotation marks**
- Strings can be **treated much the same** as **numeric variables**
- They can be **concatenated**, but follow **standard rules**
- The functions ‘**int2str**’ and ‘**num2str**’ convert **numbers to strings**, for display reasons
- The function ‘**disp**’ **displays a string** in the MATLAB command window

But First… Strings, Cells and Structures

- A **cell array** can contain **variables of different sizes** in each element
- Cell arrays are **created** and **accessed** with **curly brackets { }**
- Elements **within cell arrays** are accessed with **curly** and then **normal brackets**
- Cell arrays are **useful** for holding **strings of different length**
- Variables of **all types** can also be **subsumed** as **fields** within a **structure**
- This allows you to **group related data** of **different types** within **one overall variable**
- Structures are **useful** for storing **all participant data** (i.e. in SPM)
- i.e. they can **hold** both **name** (as a **string**) and **test data** (as a **numeric matrix**)
- Structures can also contain **multiple sub-structures**, accessed with normal brackets

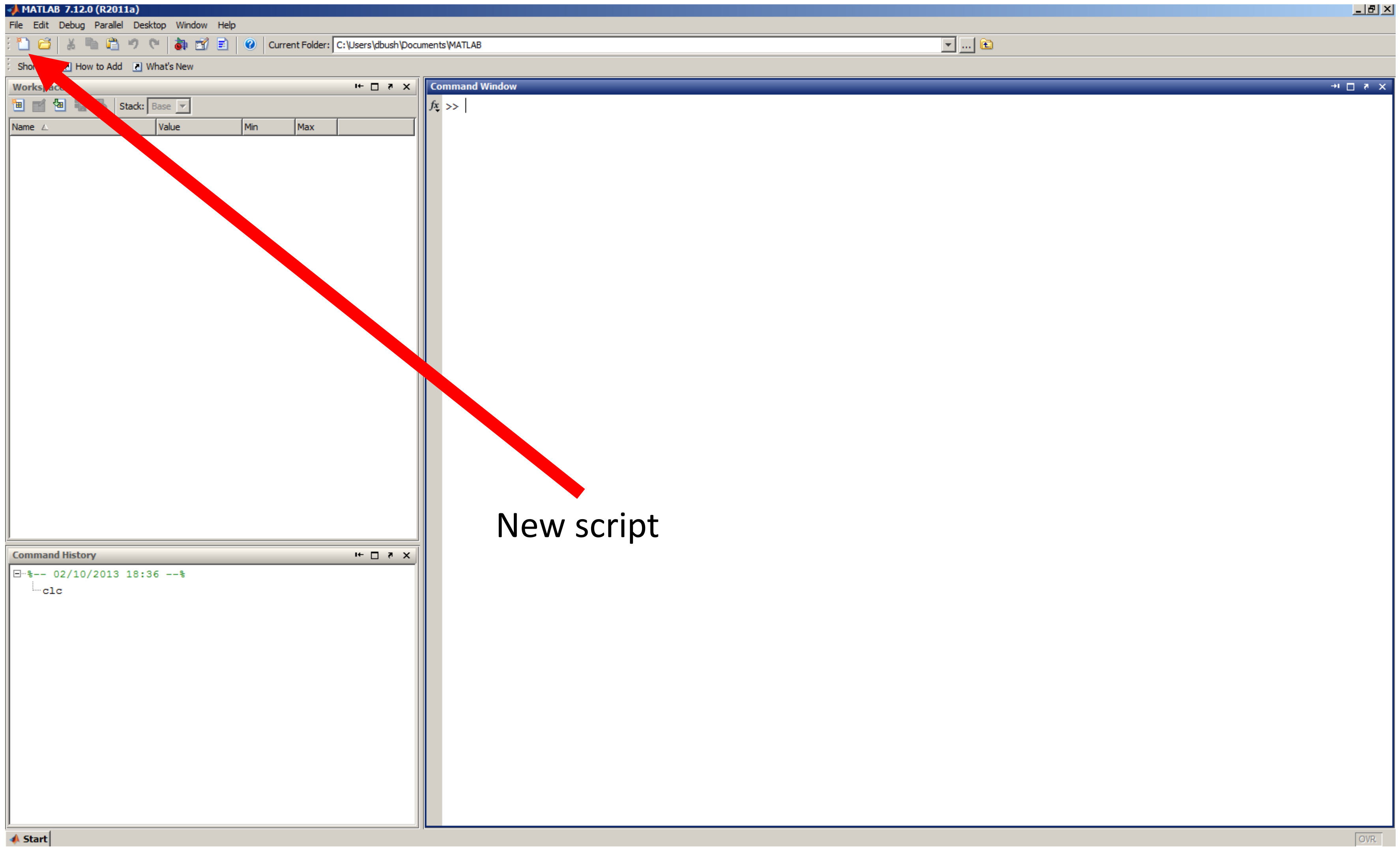
Creating a script

- **Start a new script** by clicking on “**New script**”, typing **ctrl+N**, or **File -> New -> Script**
- This opens the **scripting** / **Editor screen**
- You can now **start to program!**
- A MATLAB script is essentially **a stored list of commands** to be executed
- These scripts are stored as ***.m files**, which are essentially **text files**

Creating a script

- In the **simplest case**, we can just type out **a series of commands** and save them
- The whole m-file can then be **executed** by clicking on “**Save and run**” or **pressing F5**
- **Note**: you can only **run scripts** from your **current folder**
- You can execute **individual parts** of the code by **highlighting** them and **pressing F9**
- Scripts can use variables that **currently exist in the workspace**
- Any variables **created within the script** are **output to the workspace**

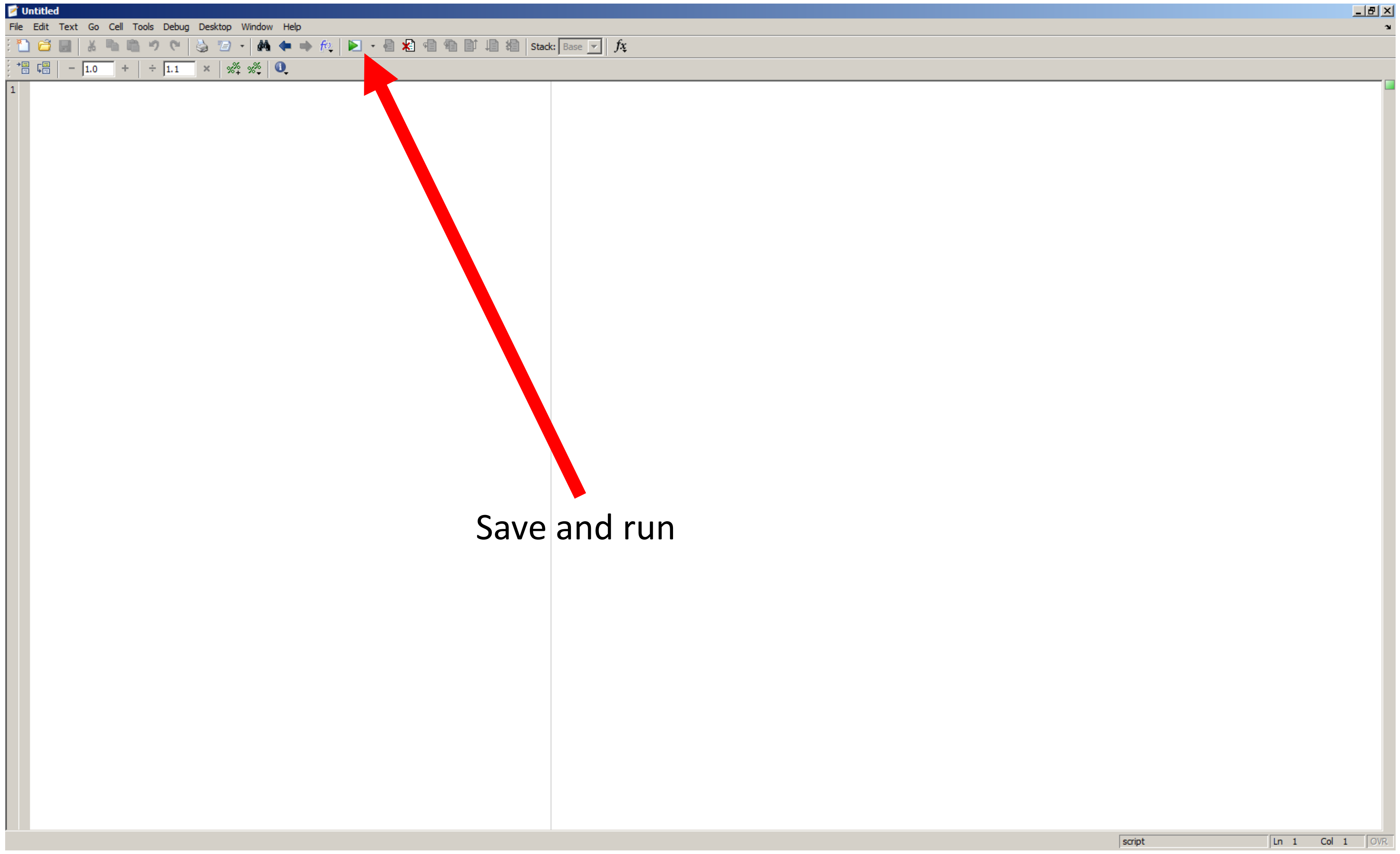
Creating a script



```
>>
```

```
%-- 02/10/2013 18:36 --%
clc
```

Creating a script



Save and run

Annotating Code

- It is good practice to **annotate** or **comment on your code**
- This helps you to **follow** what your **code is doing**, step by step
- This also helps **other people** to **understand your code**
- **Comments** can be entered by **prefacing with %**
- **Multiple lines** of commenting can be entered on **separate lines** between **%{** and **%}**
- Code can also be **divided into ‘cells’** by **prefacing with %%**
- **Individual code ‘cells’** can be **executed** using **ctrl+enter**
- If you are entering **very long statements**, you can use ‘**…**’ to **continue on a new line**

More Advanced Programming

- More advanced programming makes use of **loops** and **conditional arguments**
- **Loops** are used to execute the **same piece of code multiple times**
- e.g. if you wish to run the **same piece of analysis** on **each participant's data**
- **Conditional arguments** use **relational / logical statements** to select **what code to run**
- e.g. if you wish to run **different analyses** on **data from different groups**
- The **most important shortcut** in MATLAB: **ctrl+c**
- This **terminates** any **ongoing loop**

Quick Review of MATLAB Operators

- There are **several** commonly used **relational** and **logical operators** in MATLAB:

==	‘**is equal to**’
<	‘**less than**’
>	‘**more than**’
<=	‘**less than or equal to**’
>=	‘**more than or equal to**’
~=	‘**not equal to**’
&&	‘**and**’
||	‘**or**’

- Note that **element-wise logical operators** (**&** and **\|**) also exist for **arrays** or **matrices**

‘If’ Statements

- To make use of **conditional arguments**, use an ‘**if**’ **statement**
- i.e. there are **parts of your code** that you **only want to access IF something is true**
- ‘**if**’ statements **must** always **be terminated** with an ‘**end**’ statement
- You can then introduce **alternative outcomes** with an ‘**else**’ or ‘**elseif**’ statement

```
if condition1
        …
elseif condition2
        …
else
        …
end
```

- Be **wary** of the **difference** between ‘**elseif**’ and ‘**else if**’
- ‘**else if**’ enters a **new** or ‘**nested**’ **loop** of ‘if’ statements

‘Switch/Case’ Statements

- **Alternative outcomes** can also be selected with a ‘**switch / case**’ statement
- ‘**switch/case**’ statements can only be used to **evaluate a single variable**
- ‘**switch/case**’ statements are (almost) **equivalent** to **nested** ‘**elseif**’ statements
- ‘**switch/case**’ statements often used to **evaluate strings**
- An alternative is to use the function ‘**strcmp**’
- ‘**switch/case**’ statements must always be **terminated** with an ‘**end**’ statement

```
switch variable
        case option1
                …
        case option2
                …
end
```

‘For’ Loops

- If you want to **repeat an operation multiple times**, use a ‘**for**’ loop
- The loop is executed ‘**for**’ **each of the entries** in a **counting array**
- In **some cases**, the **counting array** is **superfluous** to the code within the loop
- In **other cases**, the **code is executed** using **each value** in the **counting array**
- ‘**for**’ loops **must** always be **terminated** with an ‘**end**’ statement

```
for count = 1 : n
        output(count) = command(count);
end
clear count
```

- It is also generally **good practice** to ‘**clear**’ your **counting array**
- ‘**for**’ loops can also be **nested**

‘While’ Loops

- If you want to **repeat an operation until a condition is satisfied**, use a ‘**while**’ loop
- The loop is executed ‘**while**’ waiting for the condition to be satisfied
- Hence, the conditional variable **must be updated within the loop**
- ‘**while**’ loops **must** always be **terminated** with an ‘**end**’ statement

```
while condition(variable)
      update variable
end
```

- It is **easy** to get **stuck** in an **infinite while loop** (remember **ctrl+c**!)

Writing Functions

- **Any script** can be **converted to a function** using the following **title line**:

```
function[output1 output2 …] = function_name(input1, input2, …)
```

- Unlike scripts, **functions** use their own ‘**private**’ **workspace**
- Any **required input must** be **passed directly** to the function
- Only **assigned output** will be **delivered** to the (base) **workspace**
- **Help information** can be entered in **comments at the top** of the function script

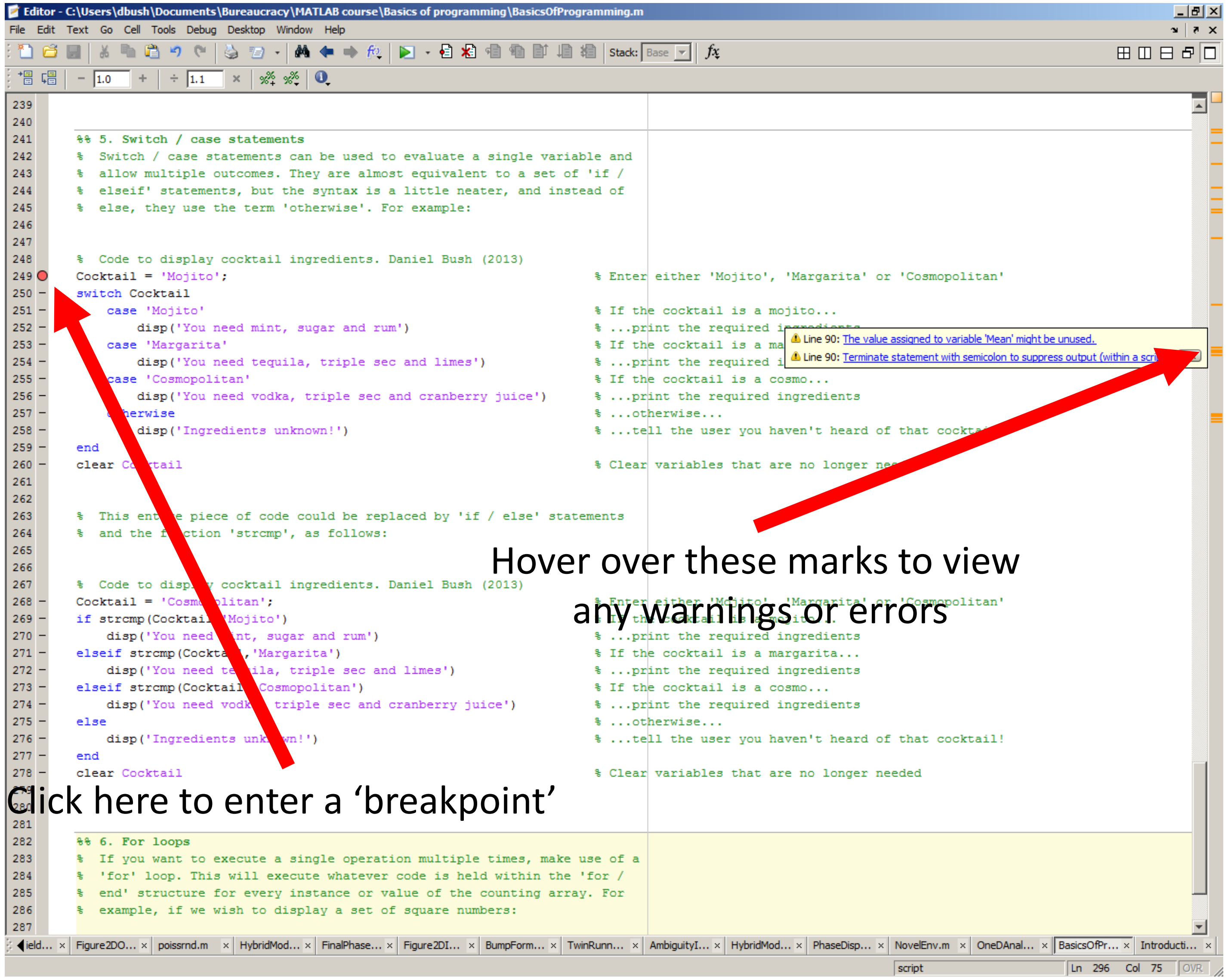
Debugging

- MATLAB has **very powerful built-in debugging** for scripts and functions
- **Potential errors** are **underlined in red**
- They are also **highlighted** by **orange or red marks** in the **warnings bar**
- MATLAB will **suggest solutions** to these **potential errors**
- These solutions are **not always appropriate** or **correct**!
- MATLAB **cannot detect all potential errors**

Debugging

- You can make use of the **specific debugging mode**
- This is **initiated** by **clicking** in the **left hand bar** (by the line numbers)
- Your code will then only **run up to that line** (where a **red dot** will appear)
- This is called a **breakpoint**
- **Clicking** on the **breakpoint** again will **remove it**
- A **small green arrow** will indicate **current position** within the code being executed
- You can subsequently '**step through**' your code **one line at a time**
- This allows you to **identify the location and source of errors**

Debugging


Editor - C:\Users\dbush\Documents\Bureaucracy\MATLAB course\Basics of programming\BasicsOfProgramming.m
File Edit Text Go Cell Tools Debug Desktop Window Help
Stack: Base
1.0
1.1
ield... Figure2DO... poissrnd.m HybridMod... FinalPhase... Figure2DI... BumpForm... TwinRunn... AmbiguityI... HybridMod... PhaseDisp... NovelEnv.m OneDAnal... BasicsOfPr... Introducti...
script Ln 296 Col 75 OVR

```
%% 5. Switch / case statements
% Switch / case statements can be used to evaluate a single variable and
% allow multiple outcomes. They are almost equivalent to a set of 'if /
% elseif' statements, but the syntax is a little neater, and instead of
% else, they use the term 'otherwise'. For example:


% Code to display cocktail ingredients. Daniel Bush (2013)
Cocktail = 'Mojito';                                       % Enter either 'Mojito', 'Margarita' or 'Cosmopolitan'
switch Cocktail
    case 'Mojito'                                          % If the cocktail is a mojito...
        disp('You need mint, sugar and rum')               % ...print the required ingredients
    case 'Margarita'                                       % If the cocktail is a margarita...
        disp('You need tequila, triple sec and limes')     % ...print the required ingredients
    case 'Cosmopolitan'                                    % If the cocktail is a cosmo...
        disp('You need vodka, triple sec and cranberry juice')  % ...print the required ingredients
    otherwise                                              % ...otherwise...
        disp('Ingredients unknown!')                       % ...tell the user you haven't heard of that cocktail!
end
clear Cocktail                                             % Clear variables that are no longer needed


% This entire piece of code could be replaced by 'if / else' statements
% and the function 'strcmp', as follows:


% Code to display cocktail ingredients. Daniel Bush (2013)
Cocktail = 'Cosmopolitan';                                 % Enter either 'Mojito', 'Margarita' or 'Cosmopolitan'
if strcmp(Cocktail,'Mojito')                               % If the cocktail is a mojito...
    disp('You need mint, sugar and rum')                   % ...print the required ingredients
elseif strcmp(Cocktail,'Margarita')                        % If the cocktail is a margarita...
    disp('You need tequila, triple sec and limes')         % ...print the required ingredients
elseif strcmp(Cocktail,'Cosmopolitan')                     % If the cocktail is a cosmo...
    disp('You need vodka, triple sec and cranberry juice') % ...print the required ingredients
else                                                       % ...otherwise...
    disp('Ingredients unknown!')                           % ...tell the user you haven't heard of that cocktail!
end
clear Cocktail                                             % Clear variables that are no longer needed


%% 6. For loops
% If you want to execute a single operation multiple times, make use of a
% 'for' loop. This will execute whatever code is held within the 'for /
% end' structure for every instance or value of the counting array. For
% example, if we wish to display a set of square numbers:
```

Line 90: The value assigned to variable 'Mean' might be unused.
Line 90: Terminate statement with semicolon to suppress output (within a scri

Hover over these marks to view any warnings or errors

Click here to enter a 'breakpoint'

Thank you for your attention!

POLYTECH
Peter the Great
St.Petersburg Polytechnic
University

OPEN
POLYTECH