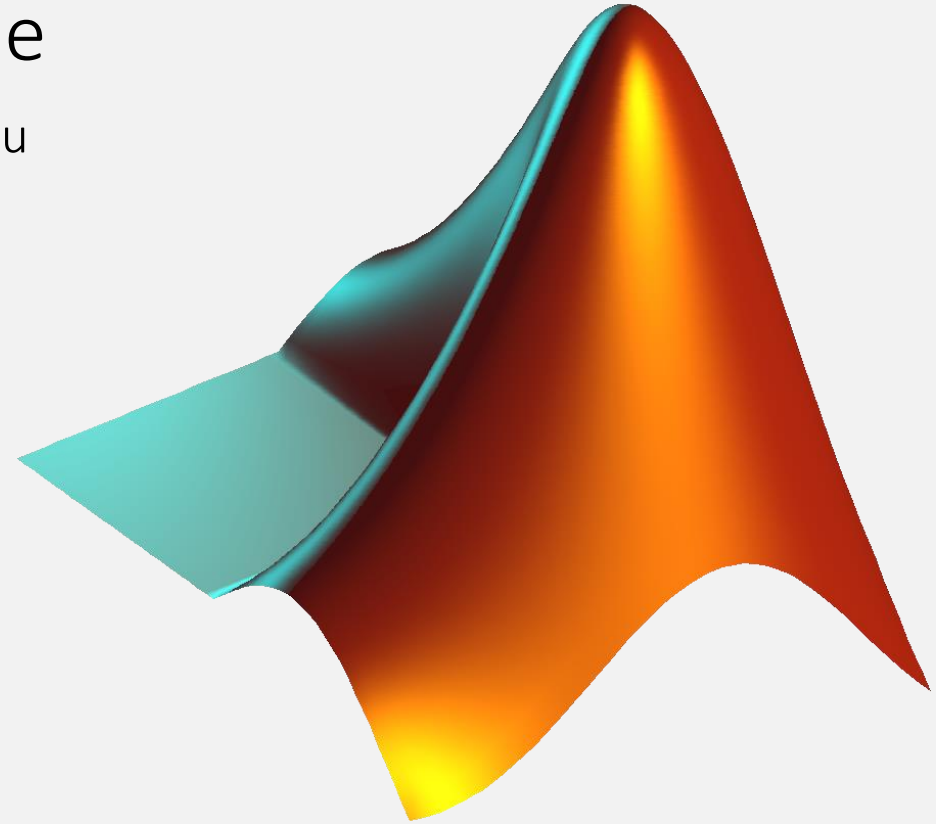
Winter 2022

Intro to MATLAB

Part 2/3

Amandine Gamble

amandinegamble@ucla.edu



UCLA

QCBio Collaboratory

Outline of the workshop

Day 1

- Interface
- Command lines and basic syntax
- Variables and operations
- Scripts
- `if` statements

Day 2

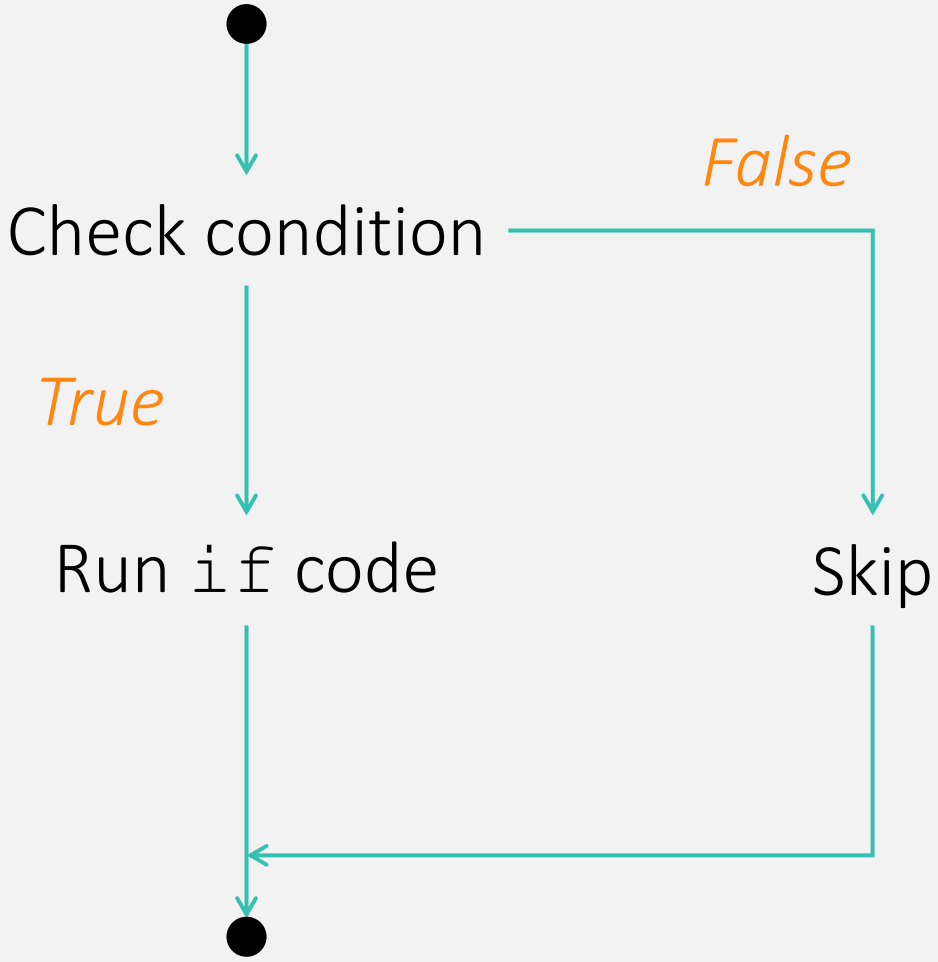
- **`for`** and **`while`** loops
- More matrices
- Functions
- Files

Day 3

- Plotting
- Introduction to dynamical systems: ODEs

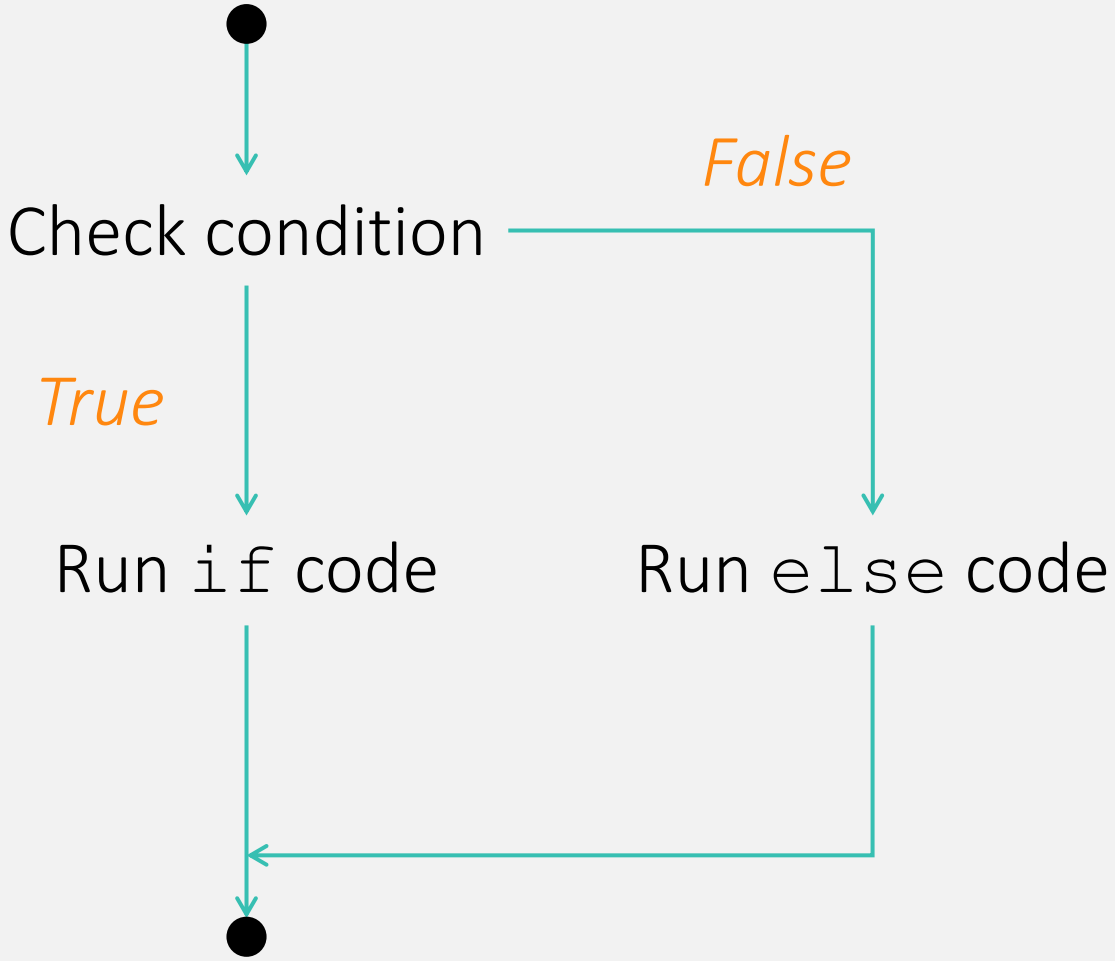
`if` statements (reminder)

`if` statements




```
exp1 = 400;
exp2 = 500;
if (exp1 < exp2)
    min = exp1
end
```

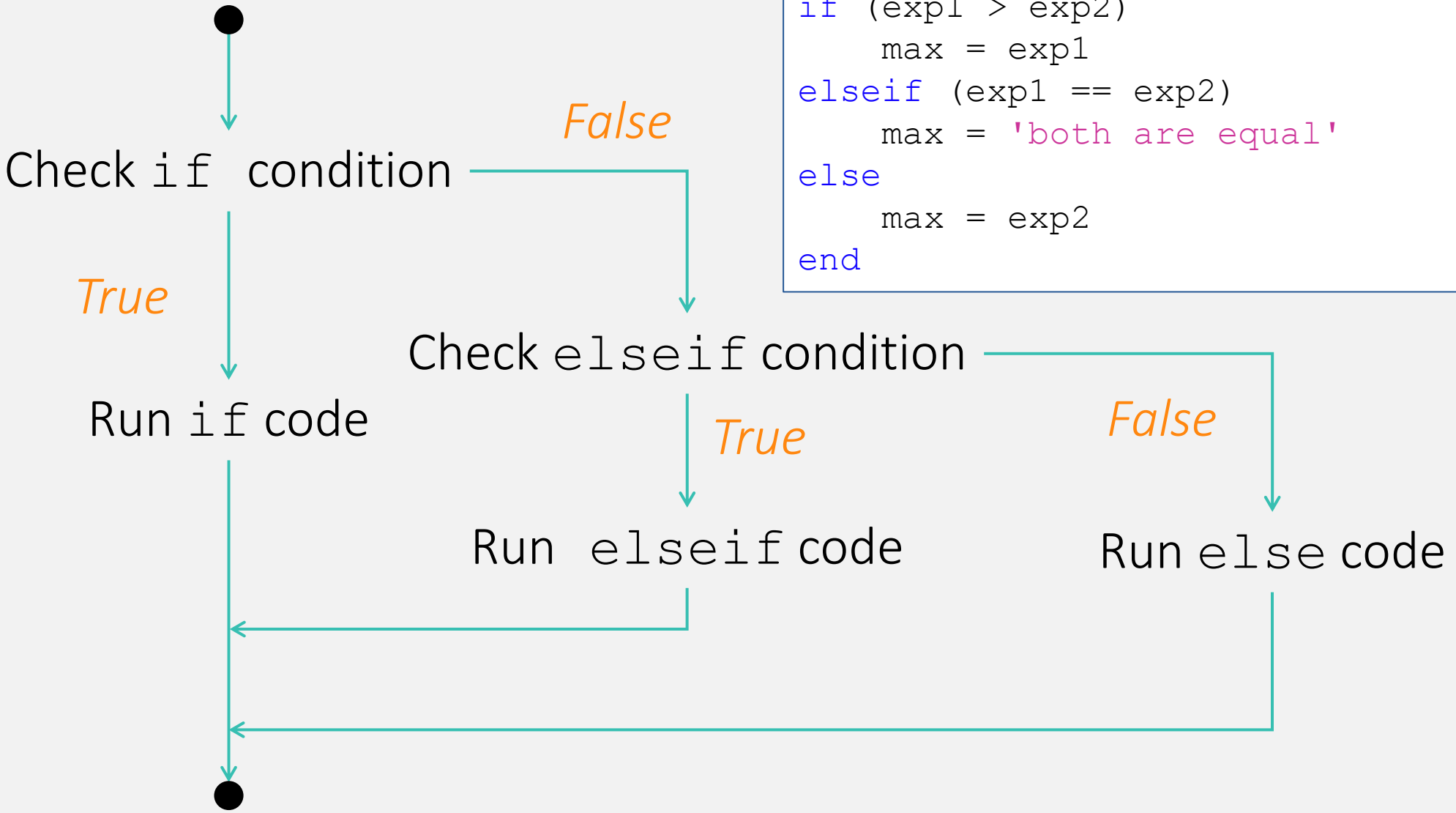
`if` statements




```
exp1 = 400;
exp2 = 500;
if (exp1 >= exp2)
    max = exp1
else
    max = exp2
end
```

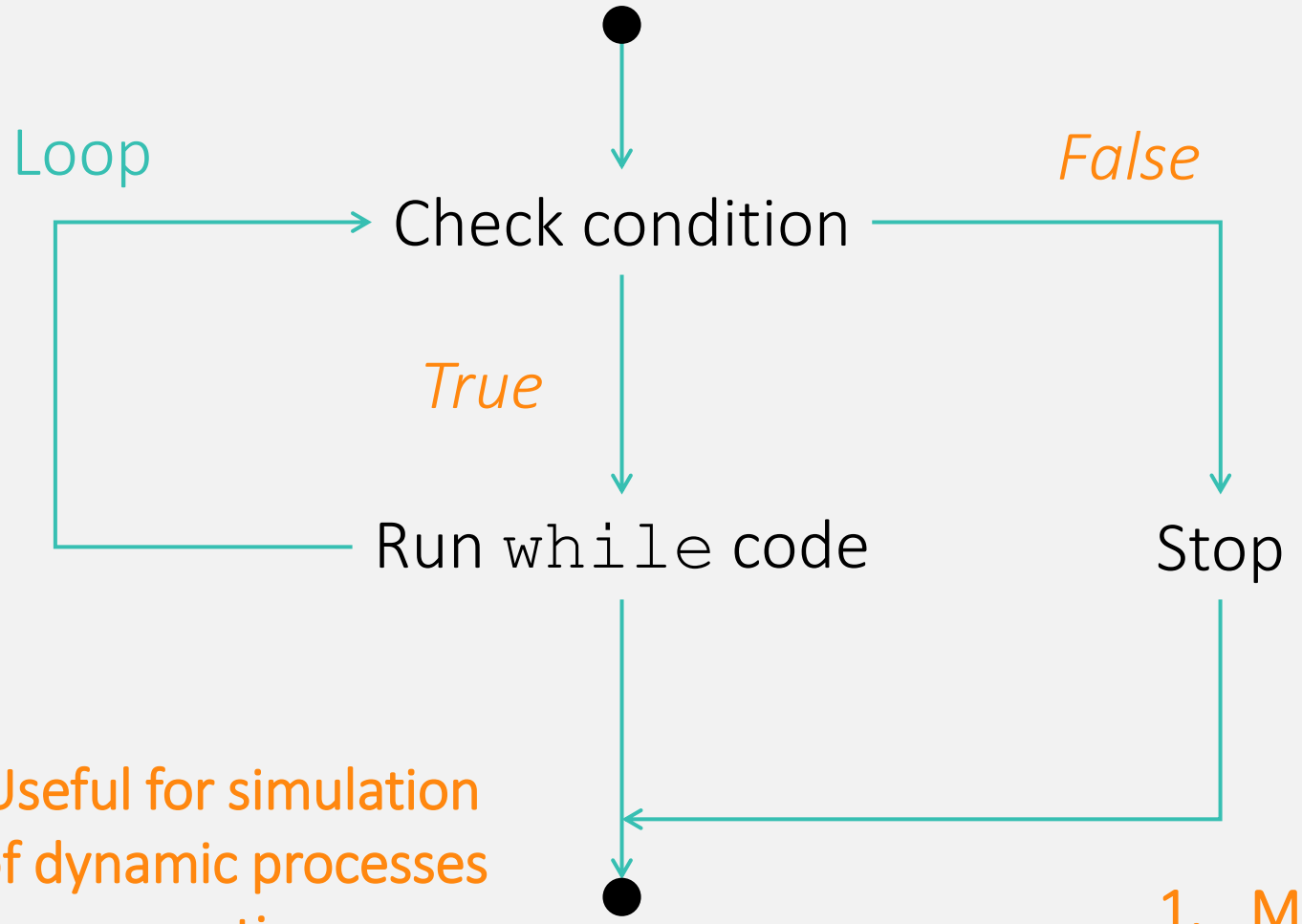
`if` statements

```
exp1 = 400;
exp2 = 500;
if (exp1 > exp2)
    max = exp1
elseif (exp1 == exp2)
    max = 'both are equal'
else
    max = exp2
end
```



`for` and `while` loops

`for` and `while` loops



Useful for simulation of dynamic processes over time

Need to declare the index

```
aa = 10;
% while loop execution
while (aa < 20)
    disp(aa)
    aa = aa + 1;
end
```

Incrementation

```
% for loop execution
for (bb = 10:19)
    disp(bb)
end
```

Index declaration and incrementation happen in **`for`**

1. Make it print 20
2. Make it display a sentence indicating the value of a (e.g., 'value of aa: ')

`for` and `while` loops

- A longer algorithm….

```
fprintf('New simulation\n');
aa = 10;
bb = 10;
% while loop execution
while (aa <= 20)
    % e.g., aa is a time step
    fprintf('value of aa: %d\n', aa);
    aa = aa + 1;
    % e.g., bb is a variable that doubles at every time step
    fprintf('value of bb: %d\n', bb);
    bb = bb*2;
end
```

`for` and `while` loops

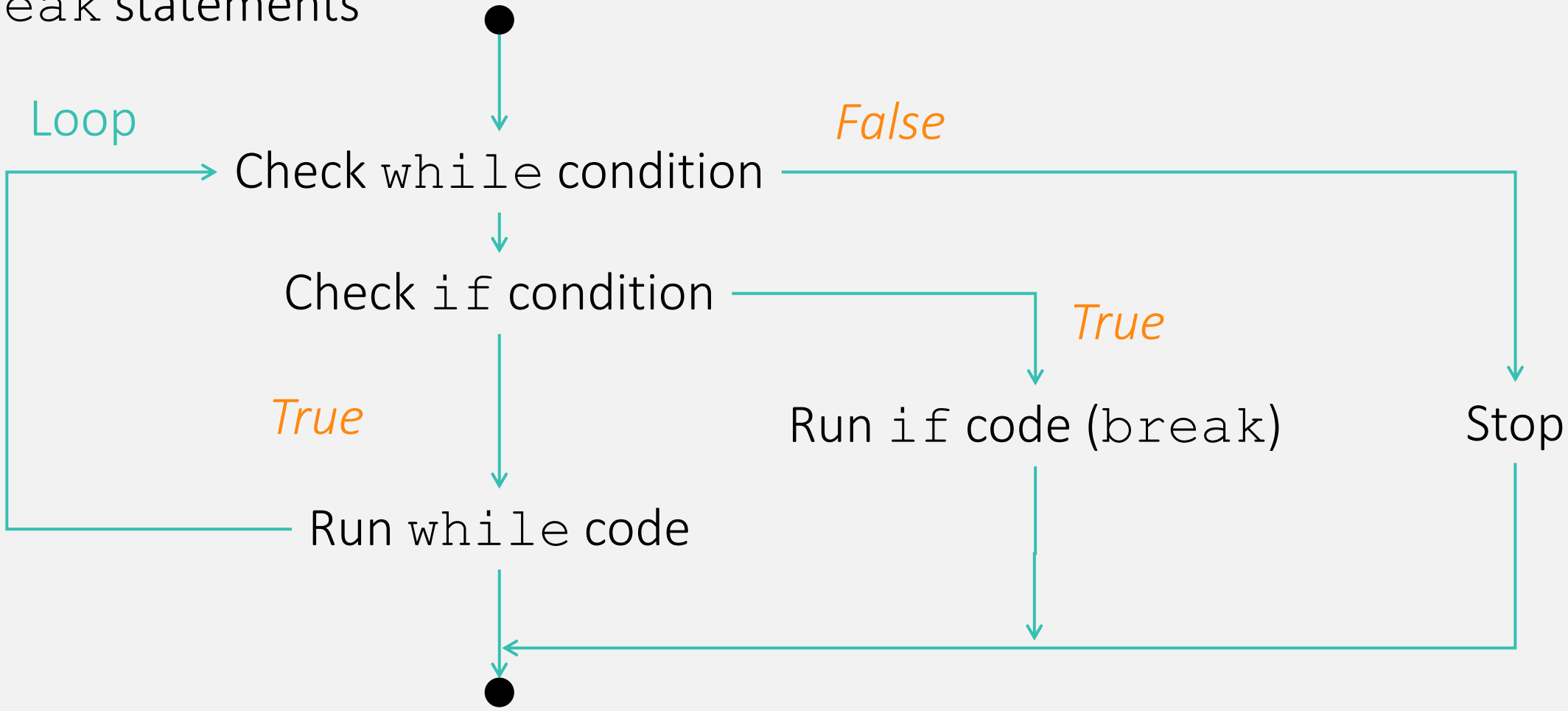
Print aa and bb in the same command

- `break` statements

```
fprintf('New simulation\n');
aa = 10;
bb = 10;
% while loop execution
while (aa <= 20)
    % e.g., aa is a time step
    fprintf('value of aa: %d\n', aa);
    aa = aa + 1;
    % e.g., bb is a variable that doubles at every time step
    fprintf('value of bb: %d\n', bb);
    bb = bb*2;
    if (bb > 5000)
        % terminate the loop
        break;
    end
end
```

`for` and `while` loops

- `break` statements



`for` and `while` loops

- Nested loops

```
fprintf('New simulation\n');
% for loop execution
% e.g., define a grid
for (aa = 1:10)
    for (bb = 1:10)
        fprintf('aa is %d and bb is %d\n' , aa, bb);
    end
end
```

More matrices

Calling a i^{th} position in a matrix

- With a vector

```
>> myVector = [4, 5, 6, 7, 9];
>> myVector(1)

ans =

     4

>> myVector(4)

ans =

     7
```

```
>> myVector(:)

ans =

     4
     5
     6
     7
     9

>> myVector(2:4)

ans =

     5     6     7

>> myVector(2:length(myVector))

ans =

     5     6     7     9
```

Calling a i^{th} position in a matrix

- With a matrix

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

myMatrix =

     1     2     3
     4     5     6
     7     8     9

>> myMatrix(1, 2)

ans =

     2

>> myMatrix(2, 1)

ans =

     4
```

Calling a i^{th} position in a matrix

- With a matrix

```
>> myMatrix(1, :)

ans =

     1     2     3

>> myMatrix(:, 1)

ans =

     1
     4
     7
```

Creating special matrices

- Matrices of 0s and 1s with the functions `zeros` and `ones`

```
>> zeros(3)

ans =

     0     0     0
     0     0     0
     0     0     0

>> ones(4)

ans =

     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1
```

```
>> ones(1, 4)

ans =

     1     1     1     1

>> ones(4, 5)

ans =

     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
```

High-dimensional matrices

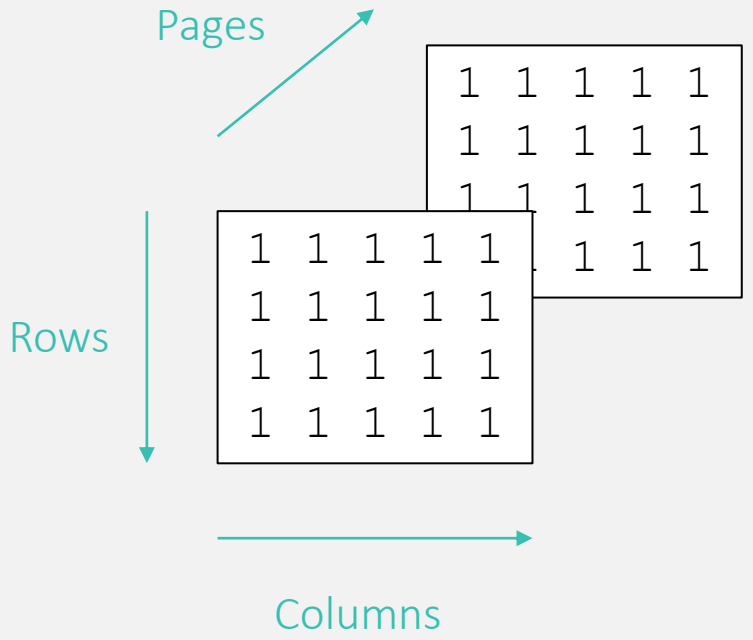
```
>> ones(4, 5, 2)

ans(:,:,1) =

     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1


ans(:,:,2) =

     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
```



Edit elements in a matrix

```
>> myMatrix = ones(4, 5, 2);

>> myMatrix(1, 1, 1)

ans =

     1
```

```
>> myMatrix(1, 1, 1) = 2

myMatrix(:,:,1) =

     2     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1


myMatrix(:,:,2) =

     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
```

Sorting

- Sort elements with the `sort` function

```
>> myVector = [5, 12, 8, 0];
>> sort(myVector)

ans =

     0     5     8    12
```

```
>> myMatrix = [12, 8, 5; 1, 22, 4];
>> sort(myMatrix, 1)

ans =

     1     8     4
    12    22     5

>> sort(myMatrix, 2)

ans =

     5     8    12
     1     4    22
```

Cell arrays

- Like matrices but different
  - Defined with { } instead of [ ]
  - Handle strings betters
    Avoid issues with string dimensions

```
>> geneNames = {'1L1A', 'NFKBIA', 'BCL2'}

geneNames =

  1×3 cell array

    {'1L1A'}    {'NFKBIA'}    {'BCL2'}
```

```
>> geneNames = ['1L1A', 'NFKBIA', 'BCL2']

geneNames =

    '1L1ANFKBIABCL2'

>> geneNames = ['1L1A'; 'NFKBIA'; 'BCL2']
Error using vertcat
Dimensions of arrays being concatenated are
not consistent.
```

Cell arrays

- Generating cell arrays with the function `cell`

```
>> myCells = cell(2, 3);
>> myCells(1, :) = {'1L1A', 'NFKBIA', 'BCL2'}

myCells =

  2×3 cell array

    {'1L1A'    }    {'NFKBIA'  }    {'BCL2'    }
    {0×0 double}    {0×0 double}    {0×0 double}

>> myCells(2, :) = {120, 446, 653}

myCells =

  2×3 cell array

    {'1L1A'}    {'NFKBIA'}    {'BCL2'}
    {[ 120]}    {[   446]}    {[ 653]}
```

Cell arrays

- ( ) refers to a set of cells
- { } refers to the data within the cells

```
>> class(myCells(2, 1))

ans =

    'cell'

>> class(myCells{2, 1})

ans =

    'double'
```

```
>> myCells(1, :)

ans =

  1×3 cell array

    {'1L1A'}    {'NFKBIA'}    {'BCL2'}

>> myCells{1, :}

ans =

    '1L1A'


ans =

    'NFKBIA'


ans =

    'BCL2'
```

Cell arrays, `for` loops and `if` statements

- Example: I have a set of genes and there levels of expression; I want to know the names and levels of expression of the genes expressed above a set threshold

Pseudocode

1. Look at each gene one by one
2. Compare its level of expression to the threshold
3. Display the ones that have higher levels of expression

`for` loop
`if` statement
`fprintf` function

```
exp1Genes = {'1L1A', 'NFKBIA', 'BCL2'};
exp1Results = [120, 446, 653];
threshold = 200;
for (ii = 1:length(exp1Results))
    if (exp1Results(ii) > threshold)
        fprintf('Genes %s expression %d\n', exp1Genes{ii}, exp1Results(ii))
    end
end
```

String comparisons

- Compare strings with the `strcomp` function

```
>> 'BCL2' == 'BCL2'

ans =

  1×4 logical array

   1   1   1   1
```

```
exp1Genes = {'1L1A', 'NFKBIA', 'BCL2'};
exp1Results = [120, 446, 653];
geneOfInterest = 'BCL2';
for (ii = 1:length(exp1Results))
    if (strcmp(exp1Genes{ii}, geneOfInterest))
        fprintf('Gene %s expression %d\n', exp1Genes{ii}, exp1Results(ii))
    end
end
```

String comparisons

- Compare strings with the `strcomp` function

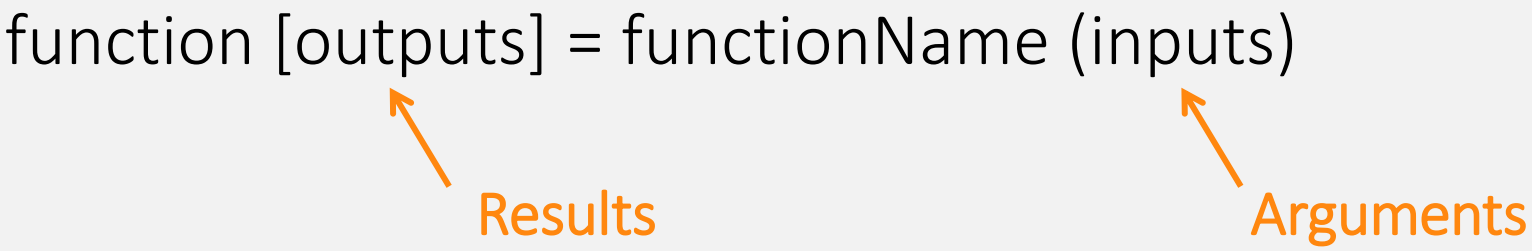
Keep those scripts saved or open, we will reuse them

```
exp1Genes = {'1L1A', 'NFKBIA', 'BCL2'};
exp1Results = [120, 446, 653];
geneOfInterest = 'BCL2';
for (ii = 1:length(exp1Results))
    if (strcmp(exp1Genes{ii}, geneOfInterest) == 1)
        fprintf('Gene %s expression %d\n', exp1Genes{ii}, exp1Results(ii))
    end
end
```

```
exp1Genes = {'1L1A', 'NFKBIA', 'BCL2'};
exp1Results = [120, 446, 653];
geneOfInterest = 'BCL2';
for (ii = 1:length(exp1Results))
    if (strcmp(exp1Genes{ii}, geneOfInterest) == 0)
        fprintf('Gene %s expression %d\n', exp1Genes{ii}, exp1Results(ii))
    end
end
```

Functions

Function syntax

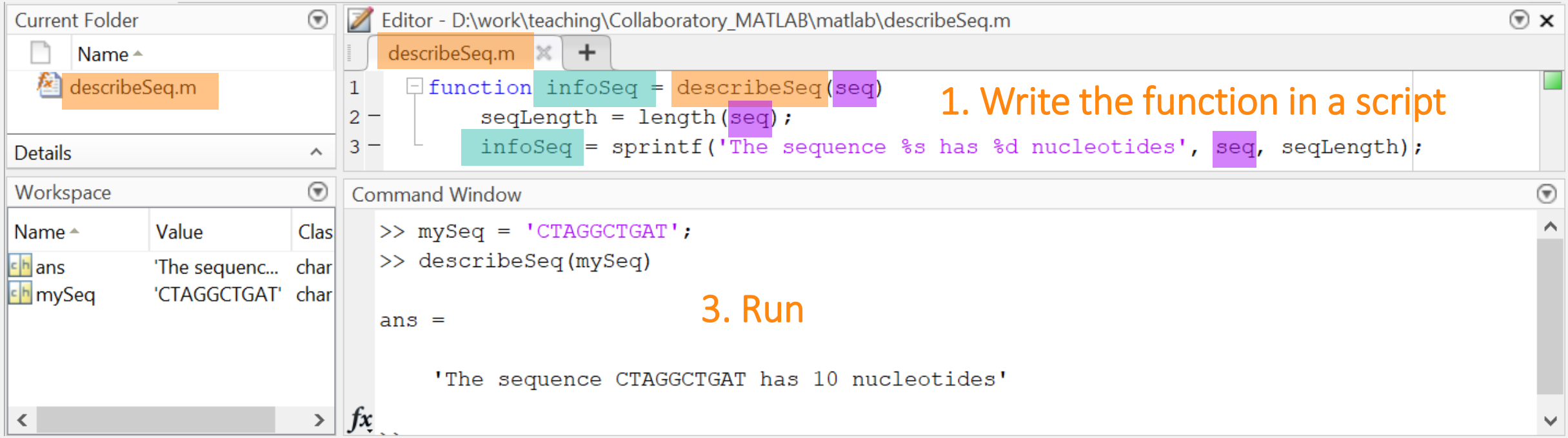


- In MATLAB, functions have to be in their own script file functionName.m

Our first function

```
function infoSeq = describeSeq(seq)
    seqLength = length(seq);
    infoSeq = sprintf('The sequence %s has %d nucleotides', seq, seqLength);
```

2. Save in the working directory with fileName = functionName



A more complicated function

Transform this into a function

Add function definition: name, output, input

```
function out = printGeneExpression(exp1Genes, exp1Results, geneOfInterest)
```

```
exp1Genes = {'IL1A', 'NFKBIA', 'BCL2'};
exp1Results = [120, 446, 653];
geneOfInterest = 'BCL2'
for (ii = 1:length(exp1Results))
    if (strcmp(exp1Genes{ii}, geneOfInterest))
        fprintf('Gene %s expression %d\n', exp1Genes{ii}, exp1Results(ii));
    end
end
```

```
out = exp1Results(ii);
```

Remove input definition (happens outside of the function)

Attribute a value to the output

A more complicated function

1. Save
2. Define your input variables
3. Exectute the function

```
function out = printGeneExpression(exp1Genes, exp1Results, geneOfInterest)
    for (ii = 1:length(exp1Results))
        if (strcmp(exp1Genes{ii}, geneOfInterest))
            fprintf('Gene %s expression %d\n', exp1Genes{ii}, exp1Results(ii));
            out = exp1Results(ii);
        end
    end
```

```
>> exp1Genes = {'1L1A', 'NFKBIA', 'BCL2'};
exp1Results = [120, 446, 653];
geneOfInterest = 'BCL2'

>> printGeneExpression(exp1Genes, exp1Results, geneOfInterest)
Gene BCL2 expression 653

ans =

   653
```

A more complicated function

- Always expect the unexpected

```
>> printGeneExpression({'1L1A', 'NFKBIA', 'BCL2'}, [120, 446, 653], 'STAT5A')
```

```
function out = printGeneExpression(exp1Genes, exp1Results, geneOfInterest)
    out = 'No gene found';
    for (ii = 1:length(exp1Results))
        if (strcmp(exp1Genes{ii}, geneOfInterest))
            fprintf('Gene %s expression %d\n', exp1Genes{ii}, exp1Results(ii));
            out = exp1Results(ii);
        end
    end
```

A more complicated function

- Comment!

```
function out = printGeneExpression(exp1Genes, exp1Results, geneOfInterest)
    % Function returns the expression valye from a gene of interest
    out = 'No gene found';
    for (ii = 1:length(exp1Results))
        if (strcmp(exp1Genes{ii}, geneOfInterest))
            fprintf('Gene %s expression %d\n', exp1Genes{ii}, exp1Results(ii));
            out = exp1Results(ii);
        end
    end
```

```
>> help printGeneExpression
  Function returns the expression valye from a gene of interest
```

```
>> help max
```

Working with files

- Read a file

```
>> url = 'https://tinyurl.com/yyd24pm8';
>> imageFile = 'example.jpg';
>> urlwrite(url, imageFile);
>> A = imread(imageFile);
>> image(A);
```

Working with files

- Read a file in a function

Download this file
https://www.signalingsystems.ucla.edu/users/Simon/experiment1.txt

```
function out = printGeneExpressionFromFile(inputFile, geneOfInterest)
    % Function returns the expression valye from a gene of interest
    delimiter = '\t'; % tab delimited
    headerLines = 1; % column headers are on line 1

    A = importdata(inputFile, delimiter, headerLines);
    exp1Genes = A.textdata(2:end, 1);
    exp1Results = A.data;

    fprintf('% s loaded. %d genes\n', inputFile, length(A.data));

    % call function to print gene information
    out = printGeneExpression(exp1Genes, exp1Results, geneOfInterest);
```

Working with files

- Write a file
  - `fieldIdentifier = fopen('fileName', 'w');`
  - `fprintf(fileIdentifier, 'someText');`
  - `fclose(fieldIdentifier);`

```
function out = printGeneExpression(exp1Genes, exp1Results, geneOfInterest,
outputFile)
    % Function returns the expression valye from a gene of interest
    fid = fopen(outputFile, 'w');
    out = 'No gene found';
    for (ii = 1:length(exp1Results))
        if (strcmp(exp1Genes{ii}, geneOfInterest))
            fprintf(fid, 'Gene %s expression %d\n', exp1Genes{ii}, exp1Results(ii));
            out = exp1Results(ii);
        end
    end
    fclose(fid);
```

Working with files

- Write a file

```
function out = printGeneExpressionFromFile(inputFile, outputFile, geneOfInterest)
    % Function returns the expression valye from a gene of interest
    delimiter = '\t'; % tab delimited
    headerLines = 1; % column headers are on line 1

    A = importdata(inputFile, delimiter, headerLines);
    exp1Genes = A.textdata(2:end, 1);
    exp1Results = A.data;

    fprintf('% s loaded. %d genes\n', inputFile, length(A.data));

    % call function to print gene information
    out = printGeneExpression(exp1Genes, exp1Results, geneOfInterest, outputFile);
```

Take home, questions?

Outline of the workshop

Day 1

- Interface
- Command lines and basic syntax
- Variables and operations
- Scripts
- `if` statements

Day 2

- `for` and `while` loops
- More matrices
- Functions
- Files

Day 3

- **Plotting**
- **Introduction to dynamical systems: ODEs**