Task: The following code produces several errors. Debug the code to correct the errors.

```matlab
w = 0:2;
t = linspace(0,2*pi);
a = (.5 .75 1);
for i = w
    x = a.*sin(w(i)*t);
    plot(t,x),hold on
    plot(t,a)
end
```
Fixed Code:

```matlab
% plot sin(wt) for t =0 to t =2pi for varying % frequencies and varying amplitudes. Also plot % lines for the amplitude
w = 0:2;
t = linspace(0,2*pi);
a = [.5 .75 1];
for i = w
    x = repmat(a',1,100).*...
        repmat(sin(omega(i+1)*t),3,1);
    plot(t,x),hold on
    plot([0;2*pi],repmat(a,2,1))
end
```
Debugging

Error Reports

??? Error: File: errtest.m Line: 5 Column: 17
Unexpected MATLAB expression.

MATLAB’s error reports appear in the command window. The error report contains a location and a description.

Error reports for files will contain a filename, a line number and a column number. Error reports for command line statements will point at the error.
Debugging

Error Reports

- Learning the descriptions can take a bit of experience
  - Unexpected MATLAB expression
    An expression is not recognized by MATLAB likely caused by a typo
  - Undefined function ‘---’ for input arguments of type ‘---’
    Incorrect function name or incorrect inputs
  - Attempted to access w(0); index must be a positive integer or logical.
    Index values for arrays start at 1. Logical indexing is a separate consideration we will discuss later.
Debugging
Error Reports

- Error using `.*` Matrix dimensions must agree. Element-by-element operations need to be applied to objects with the same dimensions.
- Error using `plot`. Vectors must be the same lengths.
  Points are plotted by specifying the $x$ and $y$ coordinates. Not all points are specified if the vectors have different lengths.
Task: The following code tries to compute $\sum_{k=0}^{\infty} r^k$ to a specific level of accuracy. Debug the code.

Code:

```
r = .2;
s = 0; k = 0; % initialize sum and counter
tol = 1e-10; % set tolerance
FLAG = true; % set flag to start while loop
% "Short Circuit" terminal condition
while FLAG  err > tol
    s = s + r^k; % update sum
    err = r^k/s; % compute relative change
    k = k+1; % update k
end
```
Task: The following code tries to compute \( \sum_{k=0}^{\infty} r^k \) to a specific level of accuracy. Debug the code.

Fixed Code:

```matlab
r = .2;
s = 0; k = 0; % initialize sum and counter
tol = 1e-10; % set tolerance
FLAG = true; % set flag to start while loop
% "Short Circuit" terminal condition
while FLAG err > tol
    s = s + r^k; % update sum
    err = r^k/s; % compute relative change
    k = k+1; % update k
    FLAG = false; % update FLAG
end
```
Debugging
The Debugger

- If a program is running long, possibly due to an infinite loop, use Ctrl + C to terminate operation
- Since MATLAB is an interpreter language, the interpreter can be stopped in the middle of running the code. One way to achieve this is through breakpoints, which are placed at lines in the code
- MATLAB’s debugger can also freeze a program when an error occurs. This is especially useful for errors in functions so you can see the state of the program when the error occurred.
Debugging

The Debugger

► When the debugger has initiated, the command window prompt changes from >>> to $k>>>$. With the program frozen, commands can be used here to check, experiment, or otherwise troubleshoot the program.

► The debugger allows step-by-step evaluation of the program for a user to observe the program operation.

► The workspace and variable editor can be used during debugging to observe the state of the program and experiment with changes.
File I/O: Beyond .mat

- .mat files are proprietary to MATLAB
- Useful for staying in MATLAB but what if you want or have another file type?
- Let’s look at how we can handle csv, els, and general text
File I/O: CSV

- CSV's are super easy in MATLAB:

  ```matlab
  >> X = csvread('myCSV.csv');
  >> X(1) = 5;
  >> csvwrite('mynewCSV.csv',X);
  ```

- Only handles numerical data
- Empty fields in the CSV return zeros in X
File I/O: XLS

- XLS works the same way

```matlab
>> X = xlsread('myCSV.csv');
>> X(1) = 5;
>> xlswrite('mynewCSV.csv',X);
```

- Basic usage is just like for CSVs
- There is also greater functionality with text data
File I/O: XLS

- XLS works the same way

```matlab
>> X = xlsread('myCSV.csv');
>> X(1) = 5;
>> xlswrite('mynewCSV.csv',X);
```

- Basic usage is just like for CSVs
More Data Types: Introduction

Vectors and Matrices store values of the same type only. What if we have to store mixed data types as in this Excel Sheet?
What happens when you do the following:

```
EDU>> x = ['hello' 55 ; 'world' 77]
```
What happens when you do the following:

EDU>> x = ['hello' 55 ; 'world' 77]

x =
hello7
worldM

In MATLAB you cannot create an array that is a direct analog to an Excel range which can contain mixed data types.
More Data Structures: Introduction

For an array of strings, can we do:

```python
strings = [ 'Introduction', 'to', 'Matlab']
```
More Data Structures: Introduction

For an array of strings, can we do:
strings = [ 'Introduction', 'to', 'Matlab']

EDU>> strings = [ 'Introduction', 'to', 'Matlab']
strings =
Introduction to Matlab
More Data Structures: Introduction

- Data structures are variables that store more than one value, e.g. vectors, matrices
- Different values should be logically related so that having them in a data structure makes sense
- Array: Store numerical values of a single type, e.g. doubles
- Cell Array: Stores values of arbitrary type, including strings
Creating Cell Arrays

- Cell arrays in MATLAB are arrays that can store different values.
- We use the curly braces \{\} instead of square braces [],

```matlab
>> strings = {'Introduction', 'to', 'Matlab'}
strings =
    'Introduction'    'to'    'Matlab'
```
Creating Cell Arrays

- Creating a $1 \times 4$ cell array.

```
>> cellRowVec = { 23, 'a', 1:2:9, 'hello'}
```
```
cellRowVec =
    [23]   'a'   [1x5 double]   'hello'
```

- Creating a $4 \times 1$ cell array.

```
>> cellColVec = { 23 ; 'a' ; 1:2:9 ; 'hello'}
```
```
cellColVec =
    [23]
    'a'
    [1x5 double]
    'hello'
```
Reference for Cell Arrays

>> cellRowVec = { 23, 'a', 1:2:9, 'hello'}
cellRowVec =
  [23]    'a'    [1x5 double]    'hello'

- Reference individual elements using curly braces

>> cellRowVec{3}
ans =
    1   3   5   7   9
>> cellRowVec{2}
ans =
a
Reference for Cell Arrays

```matlab
>> cellRowVec = { 23, 'a', 1:2:9, 'hello'}
cellRowVec =
    [23]    'a'    [1x5 double]    'hello'

- Assigning a new value:

```matlab
>> cellRowVec{2} = 'changed this'
cellRowVec =
    [23]    'changed this'    [1x5 double]    'hello'
```
Reference for Cell Arrays

```matlab
>> cellRowVec = { 23, 'a', 1:2:9, 'hello'}
cellRowVec =
  [23]    'a'    [1x5 double]    'hello'

- Reference using : produces content for another cell array:

>> newCellVec = {cellRowVec{1:2}}
newCellVec =
  [23]    'changed this'
```
Example: Storing and Printing Strings

One good application of a cell array is to store strings of different lengths.

- Create a cell array of names.
Example: Storing and Printing Strings

One good application of a cell array is to store strings of different lengths.

▶ Create a cell array of names.

```matlab
>> names = {'Alex', 'Milinda', 'Sang'}
names =
    'Alex'    'Milinda'    'Sang'
```

▶ Loop through the element of the array.
Example: Storing and Printing Strings

One good application of a cell array is to store strings of different lengths.

- Create a cell array of names.

  ```matlab
  >> names = {'Alex', 'Milinda', 'Sang'}
  names =
    'Alex'    'Milinda'    'Sang'
  ```

- Loop through the element of the array.

  ```matlab
  >> for i = 1:length(names)
      disp(names{i})
    end
  Alex
  Milinda
  Sang
  ```
Example: Multiple Legend Entries

```matlab
x = linspace(0, 2*pi, 101)';
y1 = sin(x);
y2 = cos(x/2);
y3 = sin(x) - cos(2*x);
plot(x, [y1 y2 y3]);
legend('sin(x)', 'cos(x)', 'sin(x) - cos(2x)')
```
Example: Multiple Legend Entries

What if we want to re-use the same legend for multiple plots?

... plot(x, [y1 y2 y3]);
fnames = {'sin(x)', 'cos(x)', 'sin(x)−cos(2x')};
legend(fnames)
Recall: Cell Arrays and `switch`

The expression following the `case` can be a cell array.

```matlab
% prompts the user to enter a number and displays % if its finite, infinite or zero
x = input('Enter a real number: '); % prompts the user to enter a number and displays % if its finite, infinite or zero
switch x
    case {inf, -inf}
        disp('plus or minus infinity');
    case 0
        disp('zero');
    otherwise
        disp('nonzero and finite');
end
```
Summary: Cell Arrays

- A more general structure
- Cells can contain whole cell arrays
- Manipulation works very similar to numerical arrays
- For more information, check out the documentation.
Structs

- Structs group together using fields
- Fields are named variables to the struct
- Structs are NOT arrays, but you can create arrays of structs
- Initializing a struct:

  ```
  s = struct('field1', values1, 'field2', values2)
  ```
Initializing a struct: Create a structure which stores a students’ name, id and his quiz scores.

```matlab
>> student = struct('name', 'Junot', ...
    'SUID', 05619999, ...
    'midtermScore', 45, 'finalScore', 75, ...
    'assignmentsScore', [75, 38, 83] )
student =
    name: 'Junot'
    SUID: 5619999
    midtermScore: 45
    finalScore: 75
    assignmentsScore: [75 38 83]
```
Structs

- Initializing a struct: Create a structure which stores a students’ name, id and his quiz scores.

```matlab
>> student = struct('name', 'Junot', ... 'SUID', 05619999, ... 'midtermScore', 45, 'finalScore', 75, ... 'assignmentsScore', [75, 38, 83] )
student =
    name: 'Junot'
    SUID: 5619999
    midtermScore: 45
    finalScore: 75
    assignmentsScore: [75 38 83]
```
Use the dot . to access fields

```plaintext
>> student.finalScore
ans =
    75

>> student.name
ans =
  Junot
```
Creating arrays of structs: Appending to the end

% Appending another record to the struct to create an array.
>> student(2) = struct('name', 'Robert', ...
'SUID', 05619872, ...
'midtermScore', 75, 'finalScore', 95, ...
'assignmentsScore', [65, 45, 82] )
Structs: Arrays of structs

- Q: How does student look now?
Structs: Arrays of structs

Q: How does student look now?

student =
1x2 struct array with fields:
  name
  SUID
  midtermScore
  finalScore
  assignmentsScore
How do we refer to the final score of the first student?

```plaintext
>> student(1).finalScore
ans =
  75
```
Structs: Arrays of structs

- How do we refer to the final scores of all the students?

```matlab
>> student.finalScore
ans =
   75
ans =
   95
% Store all the values of a field in a vector
>> [student.finalScore]
ans =
  75  95
```
Structs: Arrays of structs

- Print all the names of the students

```matlab
% Using a for loop
for i = 1:length(student)
    fprintf('%s
', student(i).name)
end

% A more efficient method
fprintf('%s
', student.name)
```
Print all the names of the students

% Using a for loop
>> for i = 1:length(student)
    fprintf('%s\n', student(i).name)
end
Junot
Robert
% A more efficient method
>> fprintf('%s\n', student.name)
Junot
Robert
Structs bring together data objects with different types: e.g. students with names, IDs, and scores

Access fields of the structs using .

Bring structs together into arrays