

The background is a dark blue gradient with a subtle pattern of small white dots. Overlaid on this are several faint, light blue circular elements. On the left side, there is a large circular scale with tick marks and numerical labels ranging from 150 to 260. To the right of the scale, there are several concentric circles, some of which have arrows indicating a clockwise direction. These elements suggest a technical or engineering theme, possibly related to control systems or mechanical design.

# ENGR 101 – Chapter 10

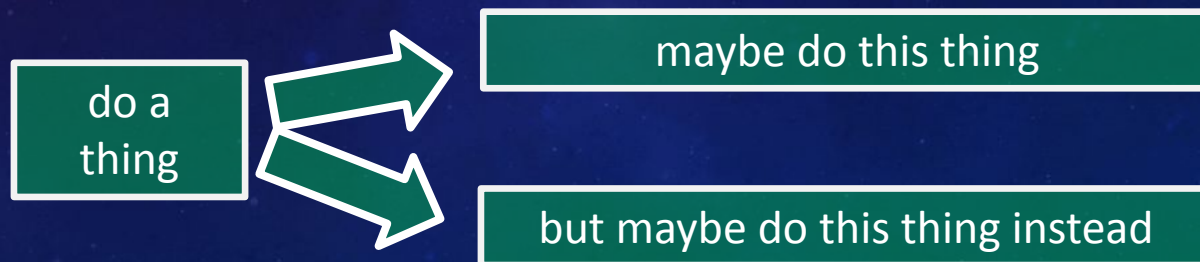
Intro to Control Flow

# Decision-Making in Programs

- So far, we've written programs whose algorithms are pretty much a list of steps and each step is executed by the program.



- But a lot of engineering problems will require the program to only execute some steps for certain situations and execute other steps in different situations. In other words, we need to design our program to have some decision-making ability.



# Program Design

- Recall that an **algorithm** is a well-ordered set of steps to solve a problem.
- **Program design** is the process by which we plan out the algorithm before we start coding.
- The more complex the problem, the more complex the algorithm.
- As we transition to the second half of the semester, we'll start to tackle these more complex engineering problems, so program design will become more important.

# Program Design Concepts

- We have a lot of programming concepts to help us handle this complexity. Some concepts we will use today:
  - Bottom-up design – an example follows
  - Top-down design – we will exercise this approach today
  - Abstraction (functions)
  - Control Flow:
    - Iteration (loops)
    - Branching (if/else)
- Other than abstraction (functions) these concepts will NOT be covered on the MATLAB exam







# A Word-Guessing Game

- Today we'll implement a word-guessing game in MATLAB.
- The game uses a set of 1000 common words.
  - These are stored in `words.txt`.
  - We've provided a function to load these words into a cell array of strings.
- The game picks a word at random and scrambles it. Then the user is asked to guess the original word.

# Example: Scrambling a Word

- Let's start by identifying a specific feature we think we'll need for the program and focus on that...
  - This is called **bottom-up design**.
- We need a way to randomly mix the characters in a word to create the anagram given to the player.
  - The `randperm` function might be useful. Type `help randperm`
  - Let's try it out interactively in MATLAB...





## Example: Scrambling a Word

- After figuring out what we want to do, we create a function that serves as an abstraction for our approach.
- We can then use the function to solve this part of the problem without having to worry about the details!

```
function [ scrambledWord ] = scramble( word )  
% scramble Puts the characters of a word in random order  
  
    scrambledWord = word(randperm(length(word)));  
  
end
```

# Bottom-Up Design

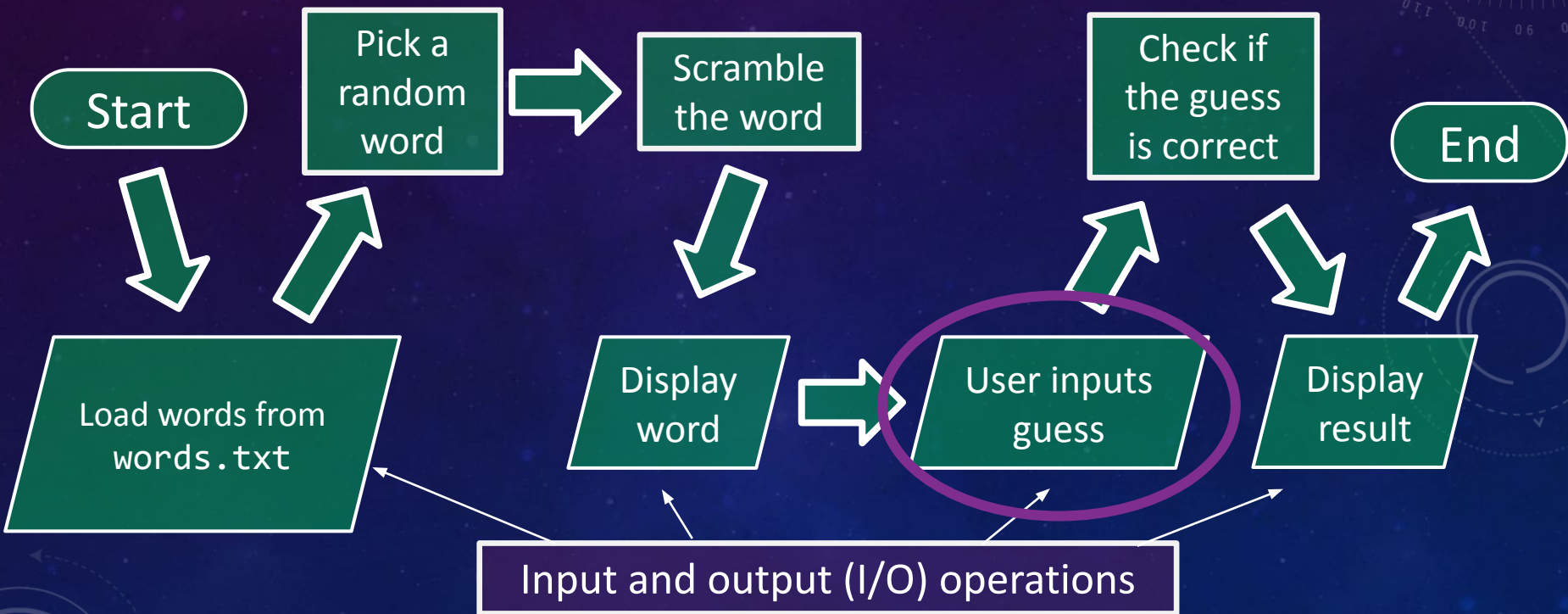
- In **bottom-up design**, we first start with specific features that might be useful and implement them.
- In MATLAB, this involves trying out different approaches interactively to see how they work.
- Once we've settled on an approach, we package the individual pieces of functionality into abstractions.
  - In most programming languages, this amounts to writing functions.
  - A key part of creating an abstraction is deciding on the interface.

# Top-Down Design

- In **top-down design**, we start by thinking about the big picture of what our program needs to do.
- We make use of abstraction in order to avoid having to think about all the implementation details.
  - e.g. Using the abstractions from a moment ago, we can treat scrambling a word and checking for anagrams as basic operations.
- It's often helpful to use **flowcharts** or other diagrams to map out the high-level design of our program.



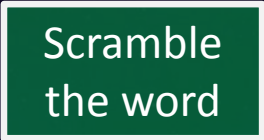
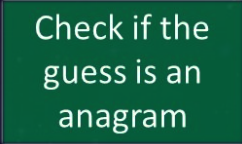
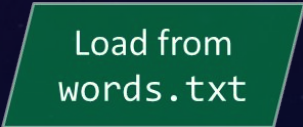
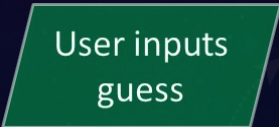
# Flowchart: A Simplified Version of the Game

- Here is a simplified version of the game that has the player guess a single word.





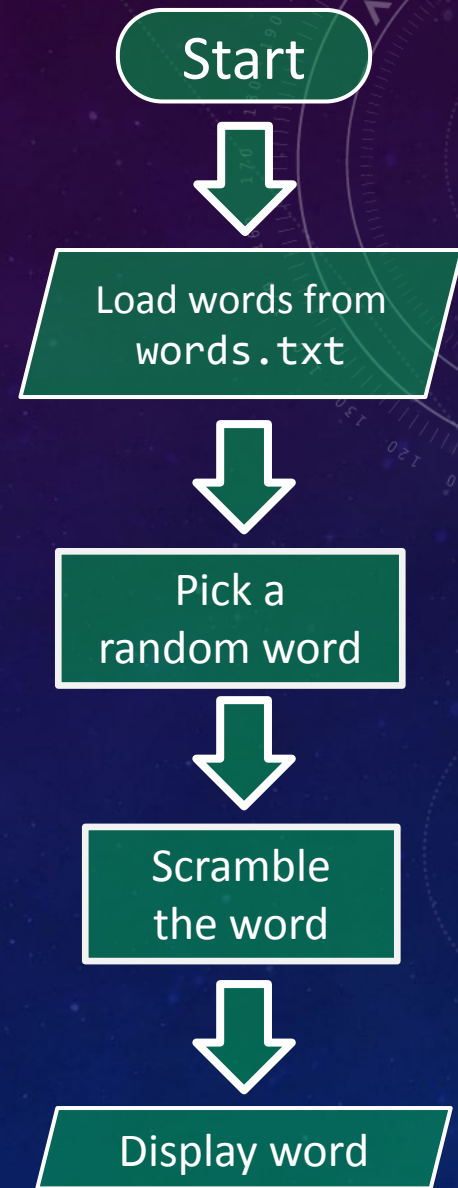
# Flowchart Components

	Purpose	Example
Terminal	Indicate the start and end of the program.	 
Process/Task	Perform a computation or call a function as an abstraction.	 
Input/Output	Read or write from/to an external source such as a file or the terminal.	 
...		
...		



# Word Game: Let's Start

- Open the **WordGame.m** file
- Follow along as we convert this part of the flowchart to MATLAB code...



# Fill in the blank lines from code on next page

```
% A word guessing game. The user is given an anagram of a  
% common word and they must guess the correct version.  
  
% Load the words file (we provide the function loadWords.m)  
;  
  
% Pick a single word randomly  
% Use content indexing to get the word itself (and not the cell)  
;  
  
% get the scrambled version  
;  
  
% display the scrambled word to the user  
;  
;  
  
% prompt the user for a guess - How is this accomplished?
```



```
scrambledWord = scramble(word);  
words = loadWords('words.txt');  
  
disp('Unscramble this word:');  
disp(scrambledWord);  
  
word = words{randi(length(words))};
```

# MATLAB script: A top-down version of the game

```
% A word guessing game. The user is given an anagram of a  
% common word and they must guess the correct version.  
  
% Load the words file (we provide the function loadWords.m)  
words = loadWords('words.txt');  
  
% Pick a single word randomly  
% Use content indexing to get the word itself (and not the cell)  
word = words{randi(length(words))}; % pick from various code  
  
% get the scrambled version  
scrambledWord = scramble(word); % from slide 9  
  
% display the scrambled word to the user  
disp('Unscramble this word:');  
disp(scrambledWord);  
  
% prompt the user for a guess - How is this accomplished?
```

# User Input in MATLAB

- Use the **input** function to get input from the user.
- For example:

```
x = input('Please enter a number: ');
```

The result  
will be  
stored in x.

The prompt that is displayed  
to request input from the user.

An extra space to  
make it look nice.

- Whatever the user enters will be **evaluated** as an expression and returned from `input`. (e.g. if one enters `2 + 3`, the value 5 is returned.)
  - If an invalid expression is entered, MATLAB shows the error and lets the user try again.

# User Input in MATLAB for Strings

- To get a **string** as input, add the 's' parameter to input.

- For example:

```
x = input('Please enter a string: ', 's');
```

- For string input, whatever the user types is simply returned as a vector of characters (it is not evaluated).

- This is what we want for our word game program.





# Code: A Simplified Version of the Game

```
% A word guessing game. The user is given an anagram of a  
% common word and they must guess the correct version.  
  
% Load the words file  
words = loadWords('words.txt');  
  
% Pick a single word randomly  
% Use content indexing to get the word itself (and not the cell)  
word = words{randi(length(words))};  
  
% get the scrambled version  
scrambledWord = scramble(word);  
  
% display the scrambled word to the user  
disp('Unscramble this word:');  
disp(scrambledWord);  
  
% prompt the user for a guess  
guess = input('Enter your guess: ', 's');  
  
% check if the guess and the word are the same  
disp(isequal(guess, word));
```

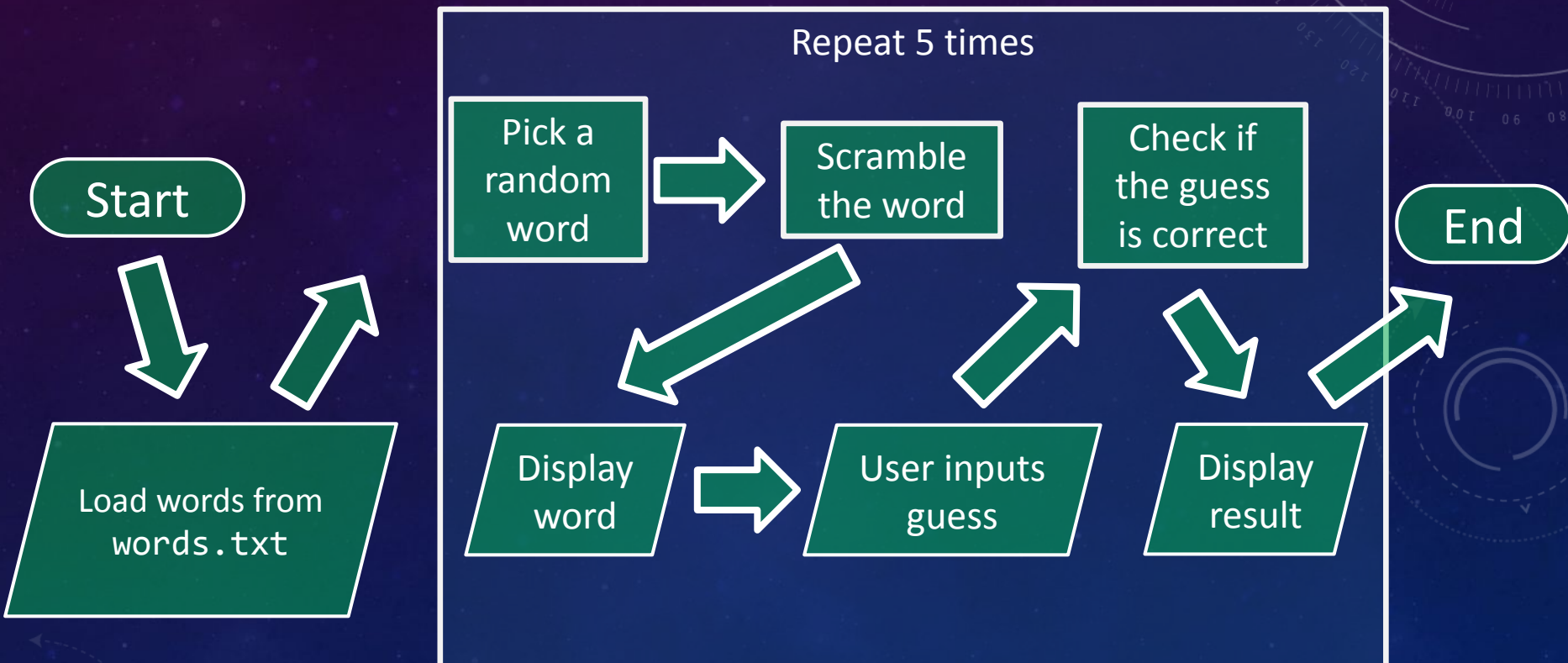
# Code: A Simplified Version of the Game

```
% A word guessing game. The user is given an anagram of a  
% common word and they must guess the correct version.  
  
% Load the words file  
;  
  
% Pick a single word randomly  
% Use content indexing to get the word itself (and not the cell)  
;  
  
% get the scrambled version  
;  
  
% display the scrambled word to the user  
;  
;  
  
% prompt the user for a guess  
guess = input('Enter your guess: ', 's');  
  
% check if the guess and the word are the same  
disp(isequal(guess, word));
```



# Adding Repetition

- Let's modify our program to give the user 5 words...





# Iteration

- Most programming languages provide a mechanism to repeat several lines of code over and over.
  - These are called **loops**.
  - The process of repetition in programs is called **iteration**.
- We're only going to take a very brief look at loops and iteration in MATLAB.
  - We'll focus on this much more in C++.
  - Do NOT use loops on the MATLAB exam!

# for Loops in MATLAB

□ Here's an example:

```
% Print out the numbers 1 through 5
```

*i* is called the **index variable**.

```
for i = 1:5
```

We specify a sequence of values for *i*. In this case, the loop repeats 5 times with the values 1, 2, ..., 5 used in turn for *i*.

```
disp(i);
```

Code that we want to repeat goes inside the for loop.

```
end
```

# for Loops in MATLAB

□ Another example:

```
% Compute the sum of numbers in a vector
% Assume there is a vector named x
total = 0;
for i = 1:length(x)
    total = total + x(i);
end
```

We use each value of *i* as an index to get each element from *x*.

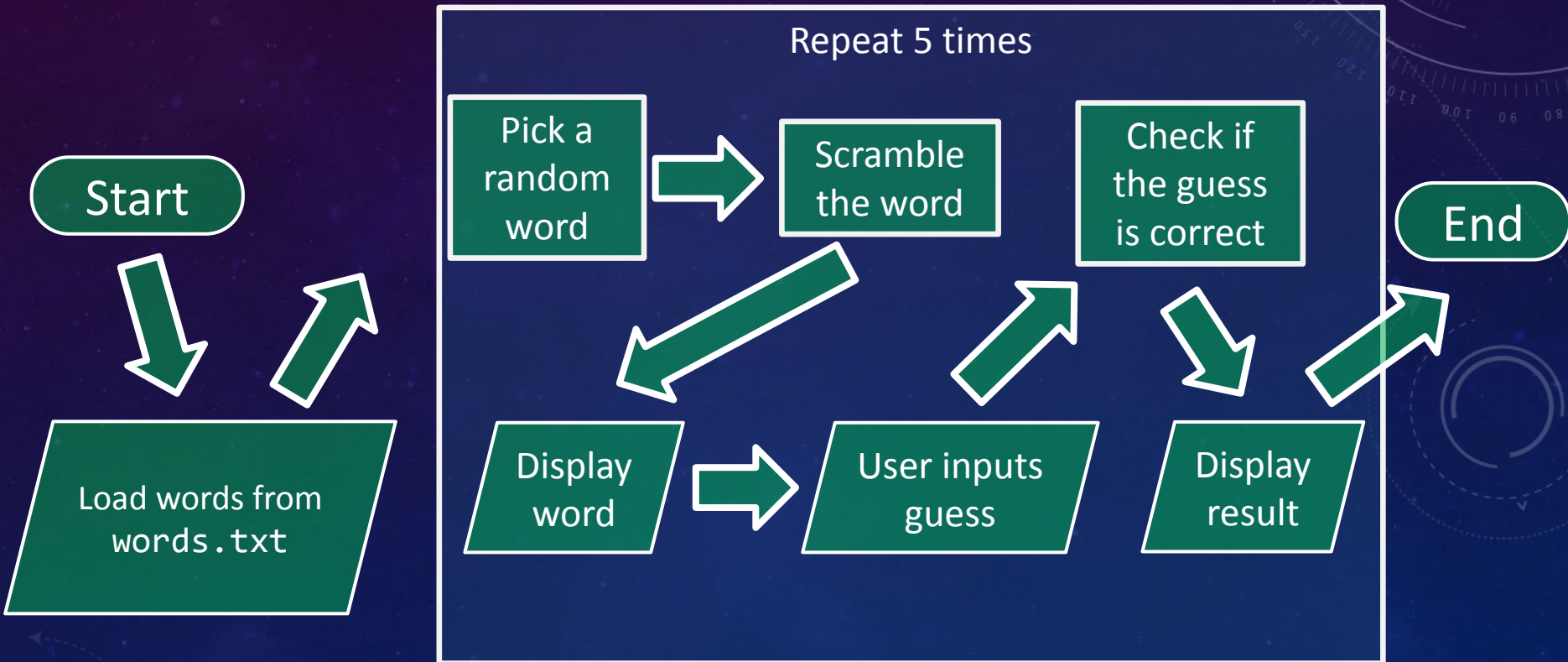
*total* is updated on each iteration of the loop.

STOP

In MATLAB, you would want to just use `sum(x)` instead!

# Word Game: Playing 5 Rounds

- We use iteration to add a new capability to our program. At “Display result”, repeat if index  $\sim$  5





# Code: Playing 5 Rounds

```
% A word guessing game. The user is given an anagram of a  
% common word and they must guess the correct version.
```

```
% Load the words file
```

```
words = loadWords('words.txt');
```

```
for i = 1:5
```

```
    % Pick a single word randomly
```

```
    % Use content indexing to get the word itself (and not the cell)
```

```
    word = words{randi(length(words))};
```

```
    % get the scrambled version
```

```
    scrambledWord = scramble(word);
```

```
    % display the scrambled word to the user
```

```
    disp('Unscramble this word:');
```

```
    disp(scrambledWord);
```

```
    % prompt the user for a guess
```

```
    guess = input('Enter your guess: ', 's');
```

```
    % check if the guess and the word are the same
```

```
    disp(isequal(guess, word));
```

```
end
```

We've wrapped up all the code to repeat in a for loop.

There's a lot going on in this script. How can we make it more readable?



# Abstraction!

- Let's create a function that abstracts playing one round.



# An Abstraction for Playing a Round

- Before implementing a `playRound` function, we need to decide on the interface (i.e. input parameters and returned output).
- This is entirely based on our top-down design and what would be useful for our program as a whole.

The rest of the program needs to know if the user guessed the word.

We need to pass the list of words to the function.

```
function [ wonRound ] = playRound( words )  
% playRound Plays one round of the word-guessing game  
%         using the provided words and returns a  
%         logical representing whether the user won.
```

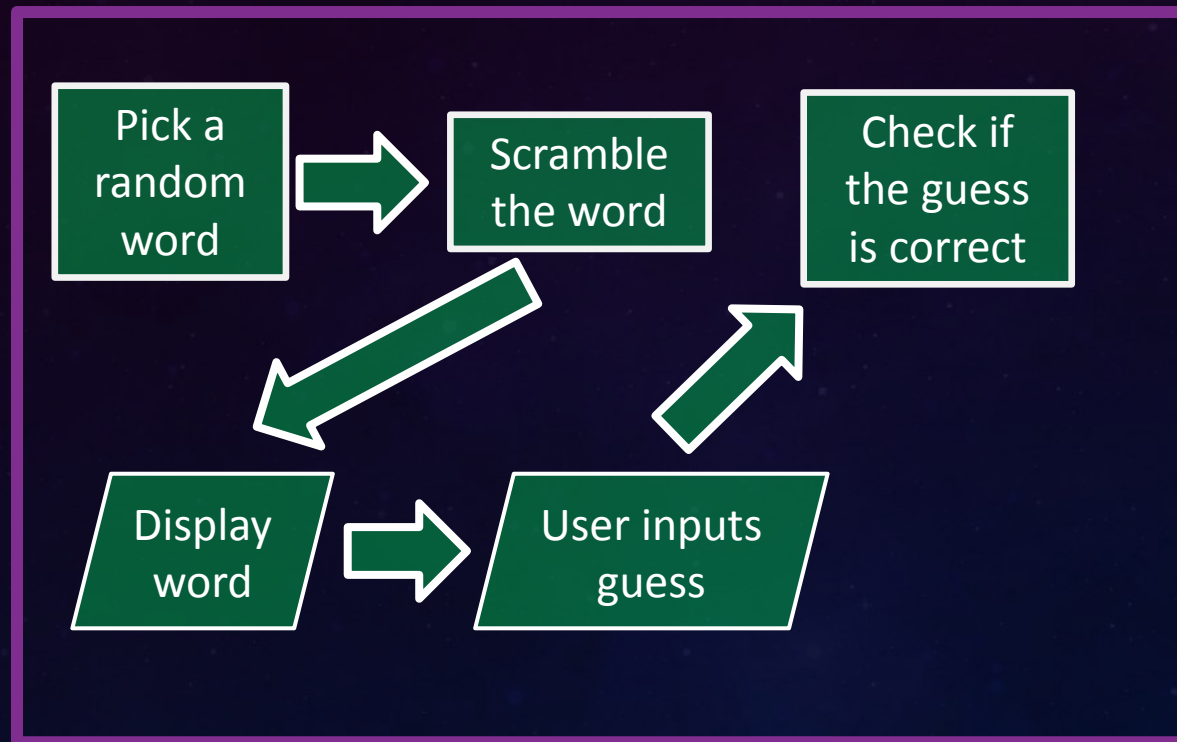
end

The comment describes what the abstraction does.



# Implementing playRound

```
function [ wonRound ] = playRound( words )
```



```
end
```

This is currently in the WordGame script, so let's move it to `playRound`.

We'll have to tweak the code to work as a function.

# Implementing playRound

```
function [ wonRound ] = playRound( words )
% playRound Plays one round of the word-guessing game
%           using the provided words and returns a
%           logical representing whether the user won.

% Pick a single word randomly
% Use content indexing to get the word itself (and not the cell)
word = words{randi(length(words))};

% get the scrambled version
scrambledWord = scramble(word);

% display the scrambled word to the user
disp('Unscramble this word:');
disp(scrambledWord);

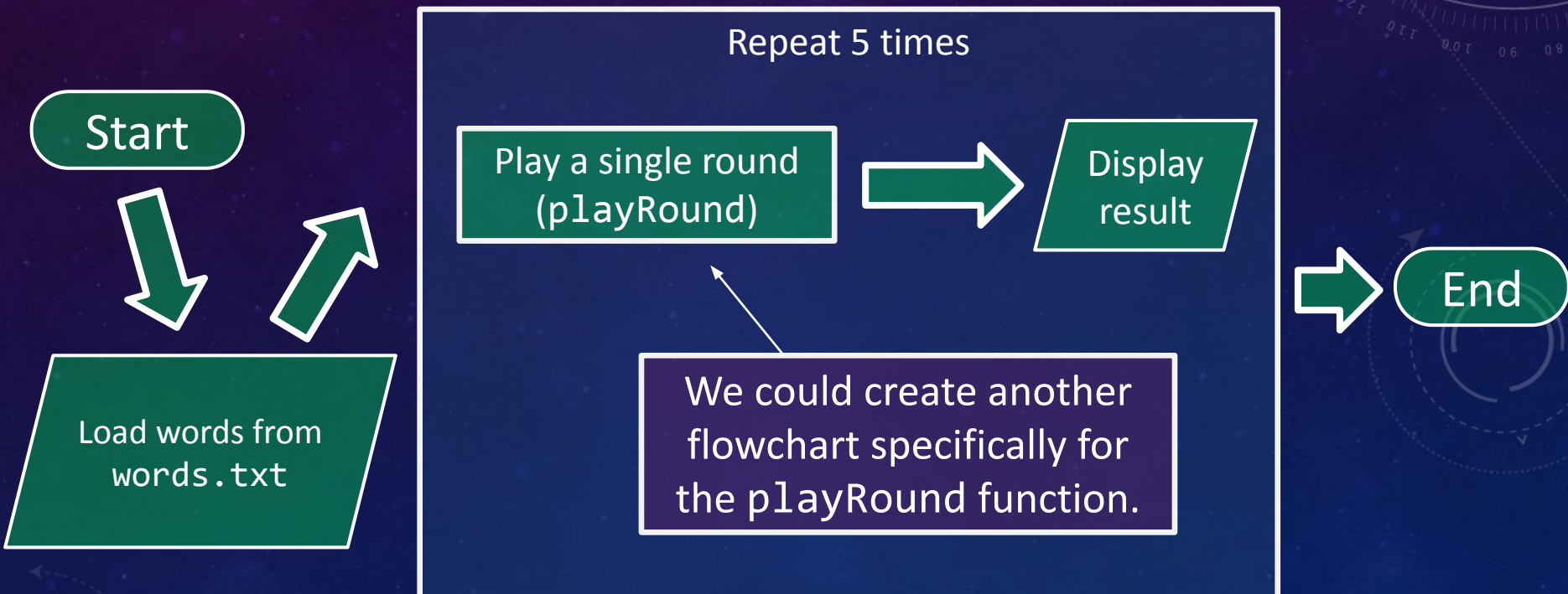
% prompt the user for a guess
guess = input('Enter your guess: ', 's');

% check if the guess and the word are the same
wonRound = isequal(guess, word); ←
end
```

Instead of  
displaying it,  
we return the  
result instead.

# Using the playRound Abstraction

- Now our top-level flowchart is much simpler. Let's update WordGame.m



# Using the playRound Abstraction

- Now that we've written playRound, we can rewrite the main program to be much more concise and easier to understand!

```
% A word guessing game. The user is given an anagram of a  
% common word and they must guess the correct version.
```

```
% Load the words file
```

```
words = loadWords('words.txt');
```

```
for i = 1:5
```

```
    % Play a single round and store the result
```

```
    wonRound = playRound(words);
```

```
    % display whether they won
```

```
    disp(['Result of round ', num2str(i)]);
```

```
    disp(wonRound);
```

```
end
```

Use the index variable to  
create a different  
message for each round.





# Branching

- In code, we use **if** and **elseif/else** statements to branch.
- For example:

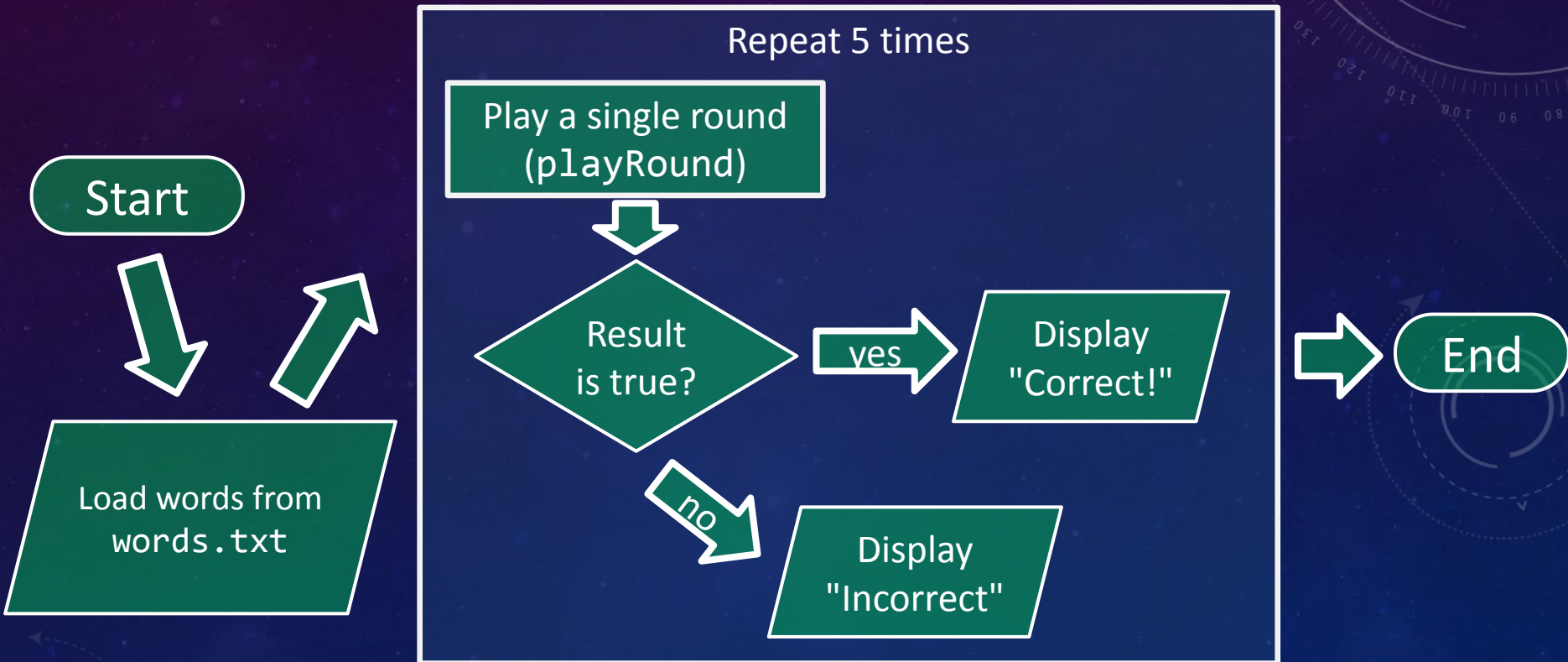
```
% Print a message about the temperature:  
  
temp = input('What temperature is it? ');  
  
if temp > 90  
    disp('That is very hot!');  
  
elseif temp > 50  
    disp('Go take a walk outside!');  
  
else  
    disp('It is pretty cold.');
```

end

When the program executes an if structure, only one of the branches is evaluated. The code then moves to the first line beyond the if structure.

# Branching

- When we need our program to make a decision and take one of several paths through the flowchart, we use a **branch**.



# Branching

```
% A word guessing game. The user is given an anagram of a  
% common word and the guess is evaluated true/false by playRound.
```

```
% Load the words file
```

```
words = loadWords('words.txt');
```

```
for i = 1:5
```

```
    if playRound(words)
```

```
        disp(['Round ', num2str(i), ' correct!']);
```

```
    else
```

```
        disp(['Round ', num2str(i), ' incorrect.']);
```

```
    end
```

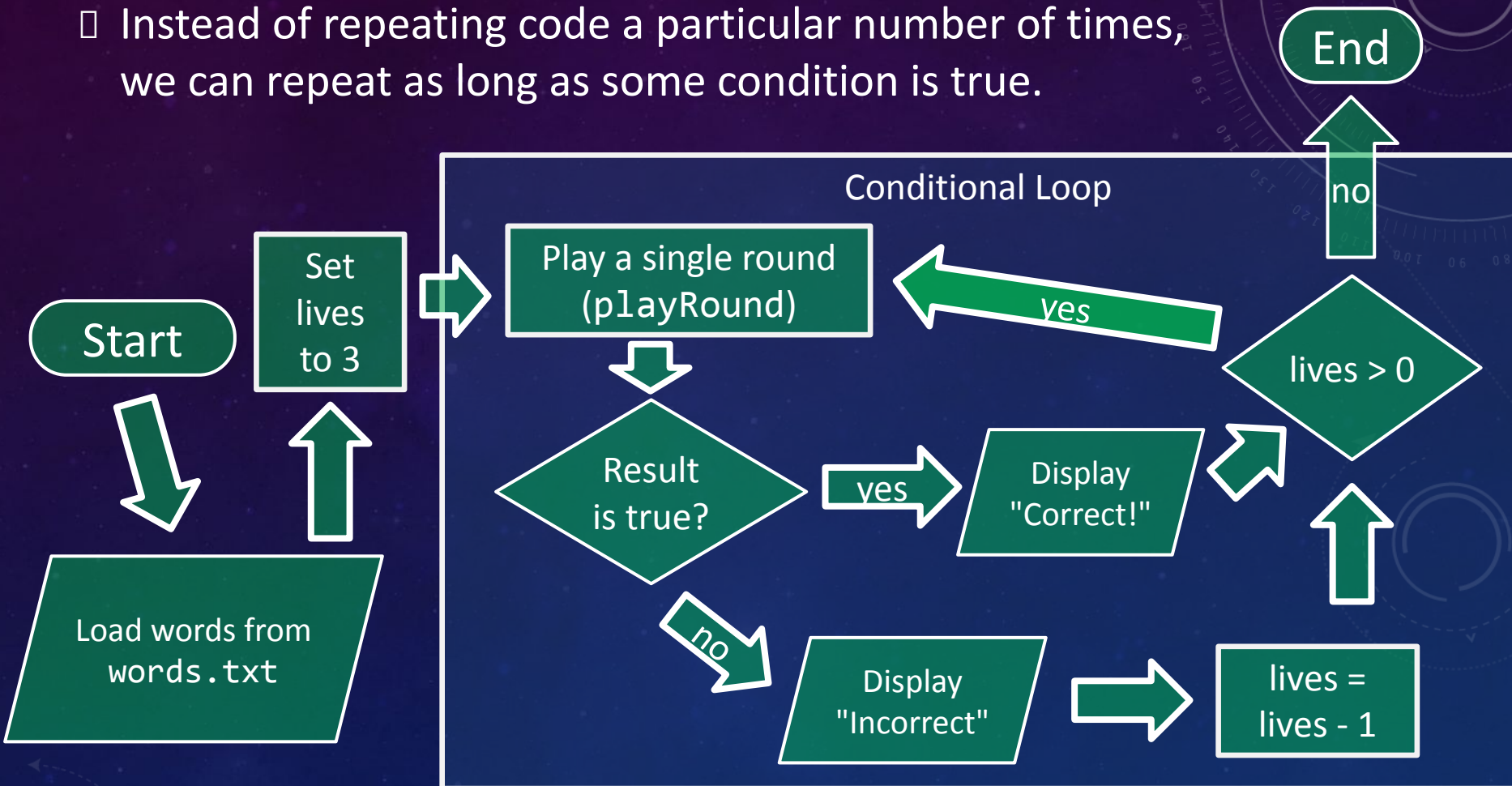
```
end
```

Use the logical result of the  
playRound as the if condition.



# Conditional Loops

- Instead of repeating code a particular number of times, we can repeat as long as some condition is true.



# while Loops in MATLAB

## □ while loop example:

```
% Force the user to enter a positive number  
n = -1;  
while n < 0  
    n = input('Please enter a positive number: ');  
end
```

This expression is the **loop condition**.

Execute while loop code inside the **body** as long as the condition remains TRUE.

# Conditional Loops

```
% The game now repeats until you miss 3 times and keeps score.
```

```
% Load the words file
```

```
words = loadWords('words.txt');
```

```
lives = 3; ← Initialize lives and score.
```

```
score = 0;
```

```
while lives > 0 ← Keep going as long as lives > 0
```

```
    if playRound(words)
```

```
        disp('Correct!');
```

```
        score = score + 1; ← Update loop variables inside
```

```
    else
```

```
        disp('Incorrect.');
```

```
        lives = lives - 1; ← separate branches.
```

```
        disp(['You have ', num2str(lives), ' lives remaining.']);
```

```
    end
```

```
    disp(['Score: ', num2str(score)]); ← The score is printed on each loop
```



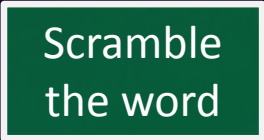
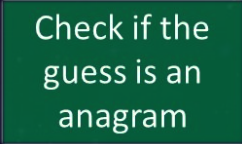
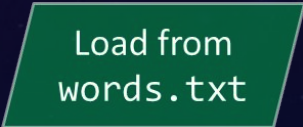
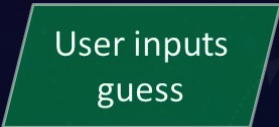
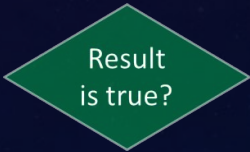

```
end
```

```
disp(['Game over. You got ', num2str(score), ' points.']);
```





# Flowchart Components

	Purpose	Example
Terminal	Indicate the start and end of the program.	 
Process/Task	Perform a computation or call a function as an abstraction.	 
Input/Output	Read or write from/to an external source such as a file or the terminal.	 
Branching	Take one of several "branches" through the program depending on a condition.	 
Iteration	Repeat part of the program a certain number of times or while a condition holds.	

# Control Flow

- Branching and iteration are techniques for managing **control flow** in our programs.
  - The line of code that is currently executing is said to have "control".
- In particular, flowcharts are an effective tool for mapping out the control flow of our program design.
- Mechanisms like `if`, `for`, and `while` allow us to structure our code to follow the desired control flow.

# Control Flow: Why did we wait so long?

- We've taken a unique approach to introducing programming concepts in ENGR 101.
  - In particular, we didn't show control flow mechanisms until now.
- This is because the problems you solve in MATLAB *rarely* need the use of control flow constructs like `if` and `for`.
  - Many engineering problems naturally involve processing information in a linear fashion...
  - ...and when you do need repetition or conditional behavior, vectorization and logical indexing are almost always better!

# Prefer Vectorized Array Operations to Loops:

- Let's say you wanted to plot the function  $x^2 + 2x$ :
- Using a loop:

```
x = linspace(0, 5, 100);  
for i = 1:100  
    y(i) = x(i)^2 + 2*x(i);  
end  
plot(x,y);
```

This code runs slower  
than the vectorized  
version!

- Using array operations:

```
x = linspace(0, 5, 100);  
plot(x, x.^2 + 2.*x);
```

The array operations work element-by-element,  
so they do the same thing as the loop.



# Prefer Logical Indexing to Loops

- Let's say you wanted to replace all negative elements with 0:
- Using a loop:

```
% assume we have a vector x
for i = 1:length(x)
    if x(i) < 0
        x(i) = 0;
    end
end
```

This code runs slower  
than the vectorized  
version!

- Using logical indexing:

```
% assume we have a vector x
x(x < 0) = 0;
```

# Prefer Ranges to Loops

- Let's say you wanted to create a vector of the numbers 1-100:
- Using a loop:

```
% assume we have a vector x  
for i = 1:100  
    x(i) = i;  
end
```

You're using a range to create a range. That's irony or something.

- Using a range:

```
x = 1:100;
```

Literally just use the range.

# When to use loops in MATLAB?

- Hint: If you ever find yourself using a loop to do anything with elements of an array, you're probably doing it wrong.
  - Use array operations, logical indexing, and ranges instead!
  - Use built-in MATLAB functions. Most are vectorized.
- If you need repetition for something else...
  - Are you sure you can't do it with vectorization?
  - Are you really sure?
  - All right *fine*, there are some cases where it's okay, like repeating a series of operations until a condition is met and that condition is not known ahead of time. But seriously: think hard before using loops in MATLAB.