Chapter 14: MATLAB Loops

Figure 14.1: A little different but perhaps equally as terrifying of a loop!

There are over 2.38 billion active Facebook users every month. Somehow, Facebook can send a "Happy Birthday" message to every single person that has Facebook. When you think about it, that is a remarkable feat!

Assuming an even distribution of Facebook users means that there are over 6,520,547 birthdays everyday!

$$\frac{2{,}380{,}000{,}000\ users}{365\ days/year} = 6,520,547.95\ birthdays/day$$

For the sake of argument let's say that Facebook messages everyone saying "Happy Birthday! I just wanted to tell you Happy Birthday because I am a corporation that relies on you thinking of me as a human being!" That is 26 words.

The world record for typing speed is 216 words per minute. It would take the fastest person in the world 0.12 minutes or 7.22 seconds to type that message. Not bad!

$$\frac{26\ words/message}{216 words/minute} = 0.12\ minutes\ /\ message\ *\ (60 sec/min) = 7.22\ sec$$

So if Facebook hired the *fastest typist in the world*, and locked them in a room and had them type that message to everyone in the world it would take them 544 days to write a short message to every person who had a birthday everyday.

$$7.22\ sec/message * 6,520,547\ messages = 47,078,349\ seconds * (1\ hour/3600\ seconds) = 13,077\ hours\ *\ (1\ day/24\ hours)\ =\ 544\ days$$

Clearly, this isn't how Facebook does it. Not only is it extremely tedious, it is impossible. What they probably do, is have a single automated message and then loop through their database and send everybody that message who is having a birthday.

Whenever you run into a task that sounds tedious, it is likely a candidate for a computer! And you will probably need loops! So let's learn about them.

Remember our mantra and guiding process for writing complex programs is: **think, sketch, code, test, repeat**.

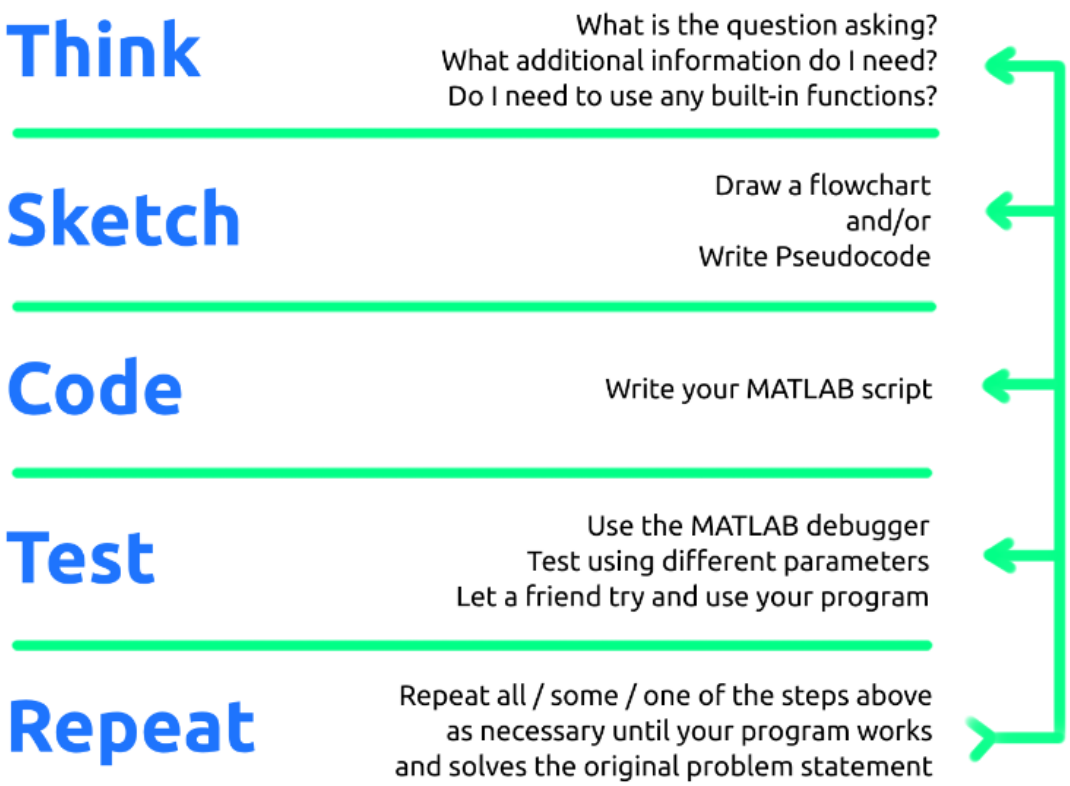

Figure 14.2: Our mantra. Breathe it in. LIVE it.

LEARNING GOALS

In this chapter we will learn how to program loops in MATLAB. That means learning:

- What a MATLAB `for` loop is and how it works.
- How to represent a `for` loop in a flowchart.
- The only case when it is appropriate to use a `while` loop and why it is dangerous.

Maclaurin Expansion of sin(x) - A Tedious Example

Let's start this discussion by looking at a real world example (this one comes from the world of numerical methods, a crucial mechanical engineering course you will soon take!). We will be using this one particular example for a considerable percentage of this chapter.

If you are unfamiliar with maclaurin expansion (it is a special form of a taylor series expansion) that is OK! For now you should just understand that **it is a way to approximate a function as a polynomial with an infinite number of terms**. Khan academy has a great explanation if you would like to understand more (or if you need a quick refresher on infinite sums). If you are shaky on what this paragraph is saying, be sure to take a minute to check out those links and get up to speed.

For our purposes we will just understand that the following is true:

$$sin(x) = \sum_{k=0}^{\infty} \frac{-1^k x^{2k+1}}{(2k+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!} + ...$$

Recall, this is saying that we can represent $sin(x)$ as an infinite polynomial and it will equal $sin(x)$ *exactly* as long as we have an ∞ number of terms.

Let's consider the case where we need to approximate $sin(x)$ with MATLAB and we decide to use the Maclaurin expansion to do so. Clearly, we can't do an ∞ number of terms, that would take an ∞ amount of time! Instead, let's truncate the expansion to 3 terms and see if it is good enough.

$$sin(x) \approx \sum_{k=0}^{k=2} \frac{-1^k x^{2k+1}}{(2k+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!}$$

We can now use this approximation to estimate something like $sin(2.618)$. Note, the angle in this case, 2.618, is in radians.

Before we move on...

Question 14.1: Reference check Show Correct Answer Show Responses

In order to check our approximation of $\sin(2.618)$ you first need to calculate the analytical value (i.e. true value) of $\sin(2.618)$. What is the $\sin(2.618)$ equal to?

Now we know the *correct answer*. We are ready to calculate our *approximation* using a subset of the maclaurin expansion.

$$sin(2.618) \approx 2.618 - \frac{2.618^3}{3!} + \frac{2.618^5}{5!} = 0.6523$$

Ok, that isn't a terrible approximation but it isn't very good. What happens if we increase up to seven terms?

$$sin(x) \approx \sum_{k=0}^{k=7} \frac{-1^k x^{2k+1}}{(2k+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \frac{x^{11}}{11!} + \frac{x^{13}}{13!}$$

Well, we can be sure that it will be more accurate but it is clear that the more terms that we add, the more tedious this process becomes. Imagine increasing our estimate to 20 terms! Whenever a process is tedious and repetitive, it is a perfect candidate to have a computer do it. In order for a computer to do it though, **we need to transfer this process into an algorithm that can solve this problem for an arbitrary n number of terms.**

Remember our mantra, **think, sketch, code, test, repeat.**

THINK - You need to stop and **think**! You should be asking yourself these types of questions (and more!):

- Do you understand the expansion?
- Do you understand our goal?
- Do you have an idea for a way to calculate this in MATLAB?
- What does an n number of terms mean?

SKETCH - I will trust that you spent a minute thinking about the problem and making sure that you understand all the moving pieces. You wouldn't skip a brain workout would you? The next step, is to **sketch**.

TRY IT!

Before you move on and look at my flowchart, can you create pseudocode or draw a flowchart that can can algorithmically solve this problem for an arbitrary n number of terms? It might be good to review what we learned in the thinking algorithmically chapter if you have not done so already.

Keep in mind that it is OK if you are confused or stuck **as long as you gave it an honest try!** Here is my flowchart for this problem:

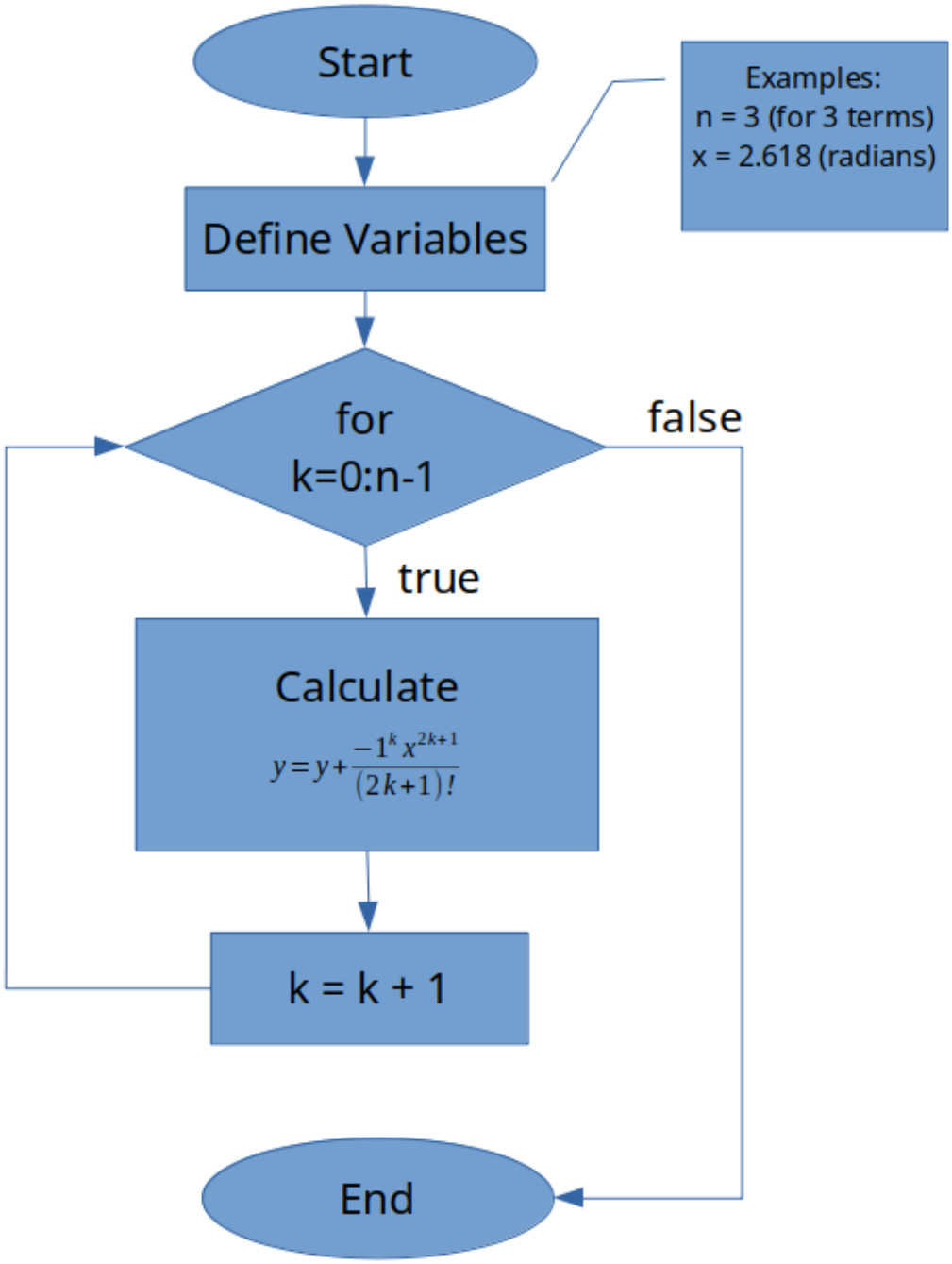

Figure 14.3: My flowchart for the maclaurin series expansion of sin(x)

Now is a good time to go back to the **think** step. Look at my flowchart in figure 14.3 above and see if you can answer the following questions.

- Does the flowchart make sense to you?
- Do you understand how it works? Can you follow it along?
- Do you understand what the true and false text indicators are referring to?
- Do you understand why I have the little extra explanation box giving *examples* of variables?

We have now successfully completed the **think** and **sketch** stages of our process. In order to continue on with the next step, **code**, we need to learn how to program these types of structures into MATLAB. Lets take a break from this particular problem and investigate how to program the necessary functionality into MATLAB.

MATLAB `For-End` Loops

From our flowchart in figure 14.3 above, we can see that we are going to use a `for` loop. In the thinking algorithmically chapter we learned how and when to use a `for` loop. Now we just need to see how to program this type of loop into MATLAB.

Recall that `for` loops execute a pre-determined number of times. Generically, in MATLAB, they are programmed as follows:

```
for index_variable = first_pass : increment  : last_pass

        < code to be executed during the loop >

end
```

`for-end` Loop Important Details:

- `for` loops *require* 2 numbers to be defined: `first_pass` and `last_pass`.
- The number `increment` is optional. If it is omitted, the increment will default to 1.
- MATLAB will modify the `index_variable` by the `increment` amount specified at the *start* of each pass through the loop.
- The loop is completed when the `index_variable` value *is greater than or equal to* the `last_pass` number. In the flowchart below this is what is meant by the true/false condition on the `for` loop diamond box.
- The `index_variable` persists in the workspace after the loop is completed. That means you can check it to make sure that it worked the way that you expected it to.
- The `index_variable` can either be used in calculations inside of the loop if necessary or desired.
- Similarly to decision structures, the `for` and `end` lines of code should not end in a `;`
- The `first_pass`, `increment`, and `last_pass` variables can actually contain functions or mathematical statements as long as they evaluate to a number.
    - For example, consider an array named `gamma` is defined in the workspace:
      `for kappa = 1:length(gamma)`
      is valid because when the function `length(gamma)` is evaluated by MATLAB, an integer is returned.

In this case the increment is equal to 1 because a specific increment was omitted. So if `length(gamma) = 5`, this loop would run 5 times.

The flowchart for the generic for loop is shown below in figure 14.4.

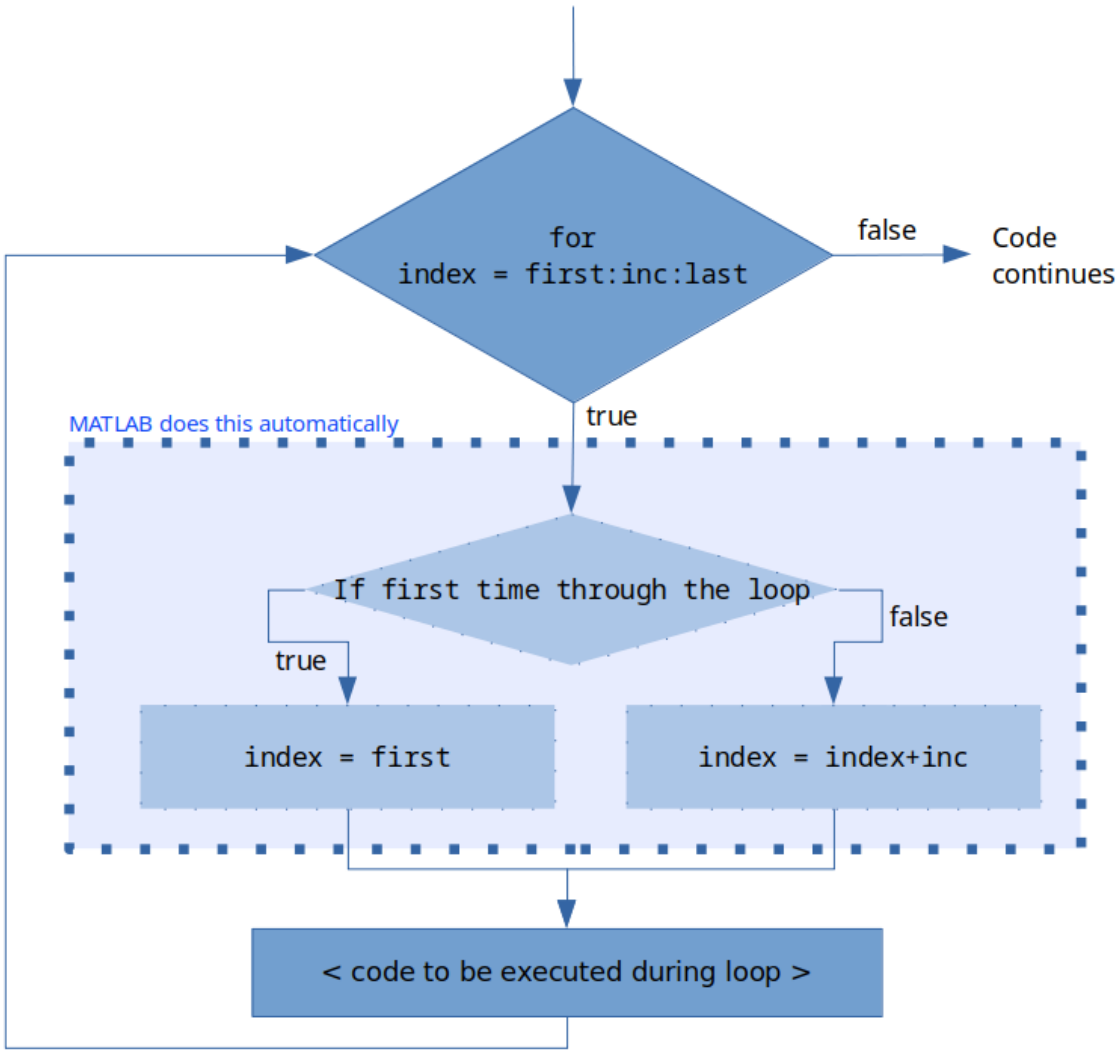

Figure 14.4: Generic MATLAB `for` loop flowchart. The variable names are a little different than in the generic code example above (e.g. `index_variable` was shortened to `index`, etc) to preserve space but the logic is the same.

As an example, consider the example code below in figure 14.5. This may look complicated but in practice it *looks* more complicated than it really is. See if you can follow the logic presented in the figure. It would also help to try and "follow along" on your own on a sheet of paper. See if you can follow the `value` variable throughout the looping process.

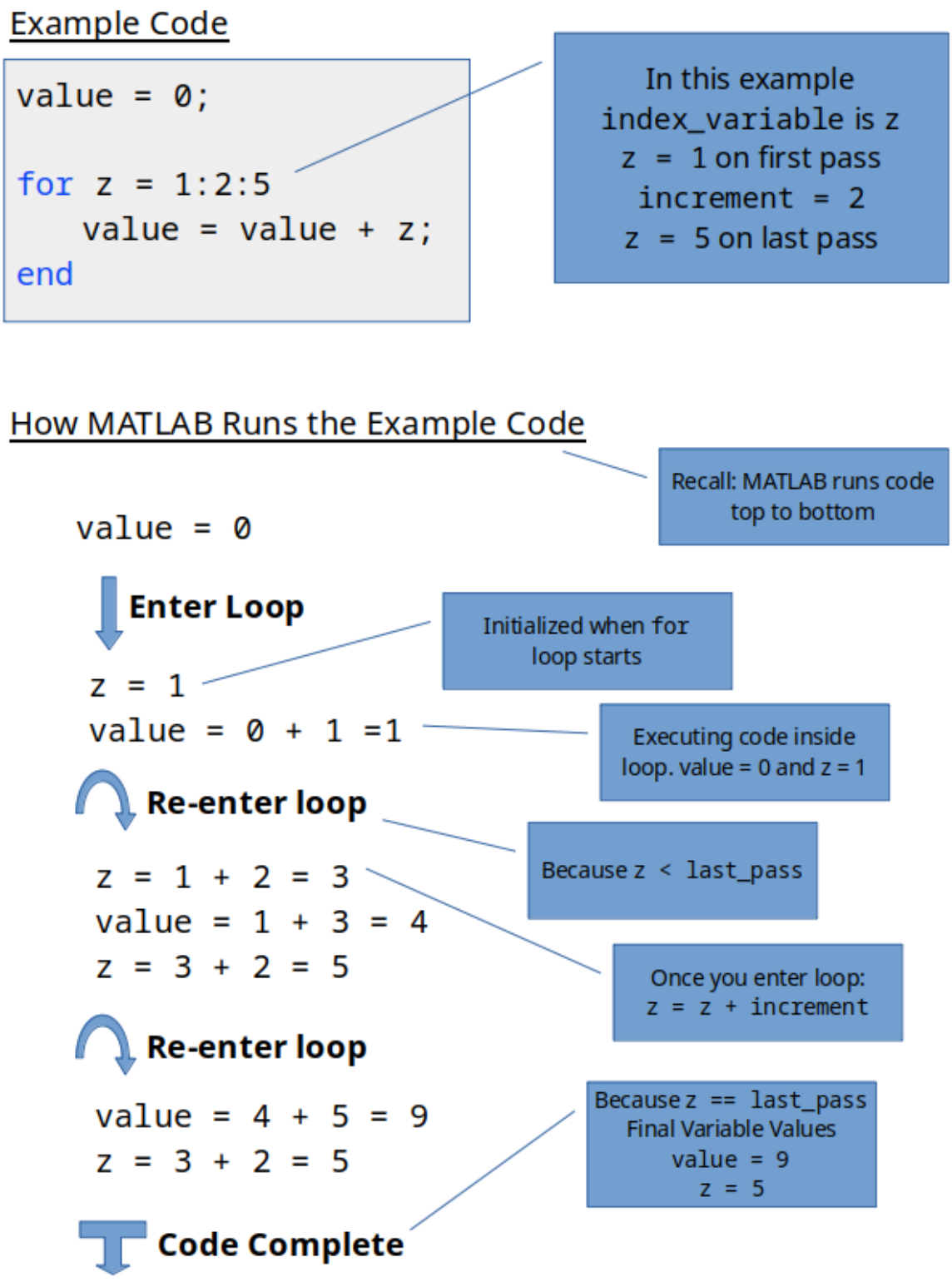

Figure 14.5: An example for loop execution process. You can confirm this by executing the sample code in MATLAB on your own!

STOP & THINK

You need to spend some time reviewing figures 14.4 and 14.5 to really understand how these loops work. If this explanation is confusing to you, try taking a break from this text, and looking at someone else's explanation. The important thing is that you understand what `for` loops are and how to use them.

If you search for "for loops" in youtube you can find hundreds of videos of people explaining these looping structures. Here is one such video.

`for-end` Key Concept #1 - Loop Executed a Specified Number of Times

In the example loop: `for z = 1:2:5` we can see that this loop will only run for a total of three times. The first time through the loop `z = 1`, the second time through, `z = 3`, and the last time through, `z = 5`. The key to remember is that the for-end loop will only run a specified number of times that is pre-determined based off the first line of the loop.

Figure 14.6: The loop will only run a specified number of times. Unlike the roll of the dice, this is predetermined by the programmer.

When designing your programs, it is important to think about how to define your `index_variable` and what values to specify for it so that you can control the number of passes through the loop.

Question 14.2: An Example For Loop Show Correct Answer Show Responses

Consider the code:

```
sneetches = 0;
for stars = 0:5:25
    sneetches = sneetches + stars;
end
```

How many times will the code inside this for-end loop be executed?

`for-end` Key Concept #2 - Follow the Variables

In addition to predicting the number of times loop code will execute you should also be able to "follow the variables" through the execution of the code. Following the variables is important so that you can troubleshoot errors in your loops and predict what values should be generated by your loop. An example of following the variables was shown in figure 14.5 above. The idea is that you can **sketch** what should happen when the code has been run.

Question 14.3: An Example For Loop Show Correct Answer Show Responses

Consider the code:

```
sneetches = 0;
for stars = 0:5:25
    sneetches = sneetches + stars;
end
```

What is the value of the

`sneetches`

variable when this code is run?

`for-end` Key Concept #3 - Analyze Arrays

The final important thing to consider when learning about `for-end` loops is that they can be used to analyze and build arrays. For example, lets consider that we have an array defined in the workspace as follows:

```
man_bear_pig = [1, 1, 2, 1, 4, 2, 4, 5];
```

As an example, lets say that we want to add all the values in this array (you can use the built-in `sum()` function for this but lets consider how to do this using loops as an example). We can use a for loop to cycle through all

of the values in the array! This code will accomplish this:

```
sum = 0;

man_bear_pig = [1, 1, 2, 1, 4, 2, 4, 5];

for index = 1:length(man_bear_pig)

        sum = sum + man_bear_pig(index);

end
```

The key here is to notice how the `for` loop was used to loop through every value of the array `man_bear_pig`. Since the increment is 1, we can use the `index` variable to access the data within the array. This is a common application of for loops and one that you need to have in your tool belt to be a successful programmer.

Question 14.4: Analyze the array Show Correct Answer Show Responses

For this problem, consider the code:

```
sum = 0;
man_bear_pig = [1, 6, 2, 1, 4, 2, 9, 5, 3, 9, 2, 1, 4, 2, 7, 3];
for index = 1:2:length(man_bear_pig)
        sum = sum + man_bear_pig(index);
end
```

Consider the 3rd pass through this loop. What is the value stored in

`man_bear_pig(index)`

on the 3rd pass through this loop?

Finishing the Maclaurin Example

Hopefully at this point you can at least understand how a `for-end` loop is supposed to work and what it might be used for. You should also start to understand what the 3 key concepts you should learn about these loops are. It is OK if you are a little shaky in your knowledge, but you should at least know what you need to learn.

TRY IT!

Returning to our Maclaurin series example, we are *now* ready for the **code** phase. Based off what you have learned about `for` loops, can you write the MATLAB code that accomplishes our programming goal. To remind you, the goal was:

Create a MATLAB script that can return the value of the Maclaurin series expansion of $sin(x)$ for angle x in radians, for an arbitrary n number of terms.

- Recall the Maclaurin series expansion of $sin(x)$ is: $sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!} + ...$
- Remember that although only 4 terms are shown in the bullet point above, that this is an ∞ sum
- When we say an arbitrary n number of terms, your script should allow the user to change a variable `n` to change how many terms are used in the estimation.
- For example, if the user specifies `n = 3`, and wants to estimate $sin(1.15)$ by specifying `x = 1.15`, the program should calculate:

  (this is actually a pretty good estimate by the way)

Give it your best shot! Try and see if you can come up with this code on your own! I believe in you!

Figure 7: I know this is difficult but no pain no gain! Give it your best shot!

My solution is shown below in figure X:

```
1    % sin_maclaurin.m
2    % Created on: 1 June 2019
3    % By: Samuel Bechara, PhD
4    % Description: calculate sin(x) by using Maclaurin series
5
6    % angle of interest in radians
7    x=1.15;
8
9    % minimum of 2 terms
10   num_terms=2;
11
12   % initialize output
13   y=0;
14
15   % loop through terms adding each time
16   for index=0:num_terms-1
17       y=y+(-1)^index*x^(2*index+1)/factorial(2*index+1);
18   end
19
20
```

Figure 8: My solution for the Maclaurin expansion of sin(x)

If you didn't get this exactly it is OK as long as you tried. Spend some time with the script above in figure X until you understand how it works. Keep practicing, reading, and watching YouTube video tutorials until this is starting to click!

The Dangerous Loop

This chapter would not be complete without a mention of `while-end` loops. They will be described below, however, I encourage you to *only use `while-end` loops in cases where it is **impossible** to use `for-end` loops.*

Figure 9: Warning: `while` loops are DANGEROUS

The reason I am warning about while-end loops before discussing them is that they are the only thing that we will learn in this book that can create ∞ loops.

What is an ∞ loop? Well why don't you go ahead and try one and see!

How to Kill (and Create) an Infinite Loop

Before I show you how to create an infinite loop, I will first tell you how to kill one.

To stop an infinite loop, select the MATLAB command window and hit ctrl+c on your keyboard. OK, ready to create an infinite loop? Create the following script, save is as to_infinity_and_beyond.m and click run. Once you get bored of seeing the numbers fly by, hit Ctrl+c on your keyboard to stop it.

```
% to_infinity_and_beyond.m
% creates an infinite loop, remember Ctrl+c stops it!

z = 0;

while z >=0
    z = z + 1 % leave out the ; so we can see numbers counting up
end

```

Figure 10: To infinity, AND BEYOND! I hope Disney doesn't sue me...

What was going on there? The reason that `while` loops are dangerous is because they *do not have a predetermined number of times that they run.* Instead, while loops run until the condition they were initialized with is false. The reason this loop keeps running forever is that the initialized condition, `z >= 0` was never made false. Since `z` started equal to `0`, and all we did in the loop was keep adding positive values to it, `z >= 0` was *always* true.

MATLAB `while-end` Loops

So what are `while` loops and what do they look like? The generic `while-end` loop is show below:

```
while < conditional expression is true >

        < execute code >

        < * YOU BETTER MAKE THAT CONDITION FALSE! * >

end
```

`while` Loop Important Details:

- Similar to `for` loops, `while` loops start with the keyword `while` and must be completed with the `end` keyword.
- It is vitally important that your `while` loop conditional expression is made false at some point during the execution of the loop. If it is not made false, you get an infinite loop.

You can think of `while` loops as over eager dogs. They will run themselves to death unless you tell them to stop. The way that you tell a while loop to stop is to make sure that at some point in the loop, the initial conditional expression becomes false.

Figure 11: The `while` loop is the overenthusiastic sled dog of the loop world. Sled dogs do actually run themselves to death.

The flowchart for the generic while loop is shown below in figure X:

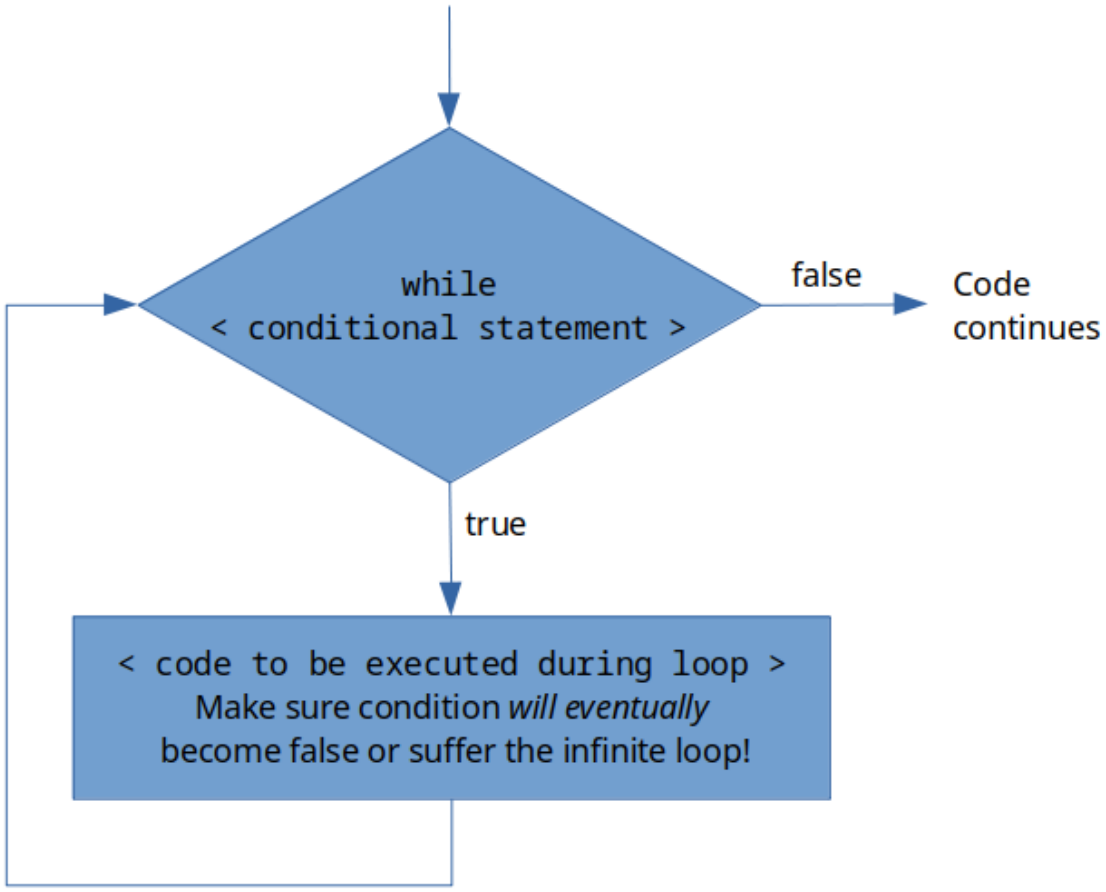

Figure 12: A generic while loop flowchart

The beauty of `while` loops is that you can ensure that your code repeats itself until a specific condition is met. However, `while` loops should only be used in instances where you do not already know how many times the loop needs to be run.

Maclaurin Series Example - `while` Loop Edition

To see an instance where a `while` loop is necessary, let's consider the case of the Maclaurin series expansion of $sin(x)$ one last time. In this case, lets flip the question a little bit. Lets say that the new question is, **you want to know how many terms of the Maclaurin series are necessary to get an estimation of sin(x) that is 99% accurate.**

We know from the Maclaurin series that if we keep adding terms, the estimation will get more and more accurate. We also know that we need an infinite number of terms to make the answer infinitely accurate. But how many terms are necessary to get our estimation of $sin(x)$ 99% accurate? It will depend on the x that the user specifies but it actually might not take as many terms as you might think it does.

Here are a few tips to get you started:

- It is actually easier to use the error rather than accuracy. The formula for percent error is So 99% accurate would be less than 1% error.
- Use the built-in MATLAB function `sin()` to get a "true" value for $sin(x)$. Even though it isn't 100% accurate, it is close enough for our purposes.
- Since you do not know how many terms it will take, you will need to use a `while` loop.

TRY IT!

I am not going to lie, I think that this is a pretty difficult problem. But we are here to workout our brains and difficult problems are our opportunity to show off our strength!

Try to complete the entire process: **think**, **sketch**, **code**, **test**, and **repeat**.

- **Think** - what is this problem asking for? Do you have anything already completed that can help you complete this?
- **Sketch** - Write out a flowchart or pseudocode for this problem and how you would solve it.
- **Code** - Give it your best shot! Make sure that you **repeat** the **think** and **sketch** phases as much as you need to!
- **Test** - Compare your MATLAB solution to your solution by hand (it won't take as many terms as you think!). Try with different x values and make sure that it is robust and it works.
- **Repeat** - repeat any steps necessary until you can get a working script.

I know this is difficult! You can do it!

Discussion 14.1: What is your flowchart?

This post is not anonymous. The professor and participants can see the responses and the author.
For this discussion, upload a picture of your flowchart or write out your pseudocode that you came up with to solve the Maclaurin series "number of terms" problem described directly above. The coding part might be difficult but I have faith in your ability to come up with the algorithm! Take your time and think through it!

Responses

Reply

Ordered by Newest Responses

The code solution to the problem is shown below in figure X. Make sure that you give it an honest try before looking at the solution though! Do not skip your brain workouts, you will only cheat yourself.

```
% sin_maclaurin_num_terms.m
% Created on: 3 June 2019
% By: Samuel Bechara, PhD
% Description: estimate the number of terms (var, num_terms) it takes
% to get the Maclaurin series expansion for sin(x) within 99% accurate
% of MATLAB's built in sin(x) function.

% angle of interest in radians
x=2;

% since the first term is just x, start there
est = x;
index=0;

% assume MATLAB's sin() function is perfect, it isn't, but its close
actual = sin(x);

% loop through, checking error percentage each time
while abs((est-actual)/actual)*100 > 1
    index = index + 1;
    est=est+(-1)^index*x^(2*index+1)/factorial(2*index+1);
end

num_terms = index + 1;
```

Figure 13: The solution. The `num_terms` variable will be an integer that is equal to the number of terms necessary to get better than 99% accuracy.

Question 14.5: How many terms? Show Correct Answer Show Responses

Using your script (or the solution shown in figure X above) how many terms are necessary to get better than 99% accuracy when using the Maclaurin series expansion for $\sin(x)$ when estimating $x = 1.874$ radians?

A Final Note on Loops

Look, I get it. Programming is hard. Loops are hard. But remember that we are treating this as a brain workout. No one walks into a gym and expects to bench 500lbs right away! You need to workout for years before you get to that level. The good thing about programming, you do not need to practice for years to understand loops! You *will need to practice though.*

The other thing to remember is that not everyone is equal. Just like working out, some of your classmates or peers may get this faster than you. It doesn't matter. The only thing that matters is YOU. You need to take the time necessary to understand these concepts. I believe in you!

Challenge Question - Return to the Student Grade Analyzer

Speaking of practice, here is one last challenge question to end the chapter with. Remember last chapter when we used decisions to assign a letter grade to a numeric grade? Now consider the case where instead of a single numeric grade, you instead have an array of grades. Lets say that we have an array called `grades` that is a column vector that contains the final grades for everyone in a class. When you are working on this, only consider 10 or so grades but understand that your algorithm should work for any number of grades (my classes can often be 200+ students!).

Remember, the final grades are calculated as follows:

Grade Range	Letter Grade
`grade > 90`	A
`80 < grade < 90`	B
`80 < grade < 70`	C
`grade < 70`	F

Table 1: How letter grades are calculated from numeric grades

Can you create a MATLAB script that:

- Accepts a 1D row vector called `grade` that contains numeric grades.
- `grade` can have *any number* of grade entries.

- Upon completion, your script creates a corresponding 1D row vector called `letter_grade` array. The index of the arrays matches, so if `grades(2) = 92`, `letter_grades(2) = 'A'`

Good luck practicing! Remember our mantra and guiding process for writing complex programs is: **think, sketch, code, test, repeat**.

End of Chapter Items

Personal Reflection - Chapter 14

This is a completely anonymous submission. The professor will be able to see the responses but the responses will not be attributed to an author. Your participation is required.
What do you think about the content of this chapter? Did you at least attempt the challenge question? Why or why not? Do you think the content in this chapter makes sense? Do you see the logical progression and can you anticipate what we are going to learn next? Do some personal reflection about your learning.

Request for Feedback - Chapter 14

This is a completely anonymous submission. The professor will be able to see the responses but the responses will not be attributed to an author. Your participation is required.
What did you think of this chapter? Anything stand out as exceptionally good? Anything that you would like to see differently? Any feedback is appreciated.

Image Citations:

Image 1 courtesy of Pixabay, under Pixabay Licence.

Image 2 courtesy of Samuel Bechara, used with personal permission.

Image 3 courtesy of Samuel Bechara, used with personal permission.

Image 4 courtesy of Samuel Bechara, used with personal permission.

Image 5 courtesy of Samuel Bechara, used with personal permission.

Image 6 courtesy of Pixabay, under Pixabay Licence.

Image 7 courtesy of Pixabay, under Pixabay Licence

Image 8 courtesy of Pixabay, under Pixabay Licence.

Image 9 courtesy of Samuel Bechara, used with personal permission.

Image 10 courtesy of Pixabay, under Pixabay Licence.

Image 11 courtesy of Samuel Bechara, used with personal permission.

Image 12 courtesy of Samuel Bechara, used with personal permission.

Image 13 .courtesy of Samuel Bechara, used with personal permission.