Learn algebra by code*

*(not pencil and paper)

$<x^2/>$

Codebymath.com

What is this book?

This book teaches basic algebra by having the student write short, simple computer programs (or “code”), to investigate topics typically found in an introductory algebra class. Topics include symbolic manipulation, factoring, problem solving, graphing, equation solving, and quadratic equations, all presented with the backdrop of writing simple code.

The programming environment used is the site at `www.codebymath.com`, which is a website dedicated to the parallel learning of coding and mathematics. The website has a straightforward design for coding: a “run” button executes code typed into a text-window on the left side of the screen. Any output generated, will appear in second window on the right side of the screen. Many programming extensions are available, for text, graphics, professional charts, sound, and maps.

The student will be pleased with the motivation and insights coding brings to learning algebra.

Inspiration for this book came from an essay called “A Mathematician’s Lament,” by Paul Lockhart (2002), Dan Meyer’s TED talk called “Dan Meyer: Math class needs a makeover” (2010), and the book “An Elementary Introduction to the Wolfram Language,” by Stephen Wolfram (2015).

Contents

Chapter 1

Introducing coding and math

1.1 Views on math teaching

Perhaps we are not doing the best we can by continuing to teach algebra with paper and a book, as a handwritten experience. This mode is too antiquated for the modern world, and too boring for most students. It only appeals to a remarkable few who can visualize complex relationships and logic in their minds, for subsequent presentation on paper. This is the way Albert Einstein worked (and his intellect was rare). So from the first written math worksheets given our 2nd graders, we are holding them to the mindset of Einstein. Why do we keep doing this?

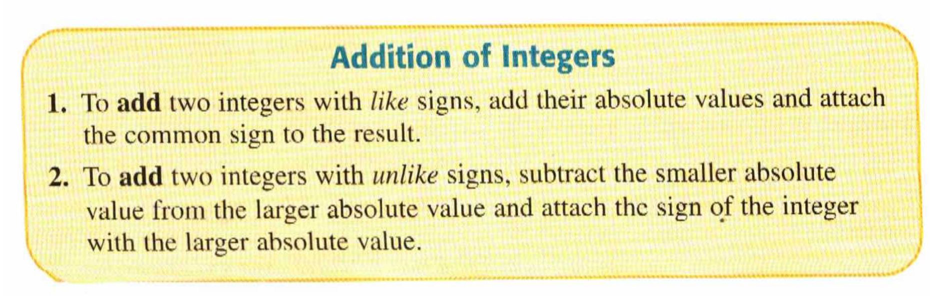
Beyond that, working with pencils and paper to solve complex technical problems is a skill the world simply doesn't need anymore. Other than a "back of the envelope" feasibility study in a coffee shop, technical problems are best solved by using technology, that allows one to try, test, collaborate, and share potential solutions. Problems are solved by continually to trying "what if" situations, failing then fixing what is wrong, while we push on toward a solution. All of our work, including schoolwork should use a "share" button at some point along the way.

We don't understand why K12 math education insists on minimizing the usefulness of technology. And no, we don't consider the use of graphing calculators as "technology." These also fall into the realm of another skill the world doesn't need anymore (small ugly screens, awkward keyboard and software, limited connectivity, way too expensive, and only exist inside the walls of a school). Teachers using "smartboards" don't count either.

1.2 Coding and math

So what does coding have to do with it? Well mathematics is really the original form of computer programming. Both are born by starting with some simple facts, that build upon themselves, until something new is discovered or some need is filled. While an app for the iPhone starts by opening a "view controller," a mathematical problem might begin by "assuming x is a real number." From there, "Flappy Bird" eventually comes, or Fermat's Last Theorem is eventually solved. Both proceed with rules, logic, and (re)use of proven ideas. This is how progress is made. It's just that for the first 4,000 years of mathematics, there were no computers. But now we have them, even on our wrists and in our pockets, so we should use for learning mathematics.

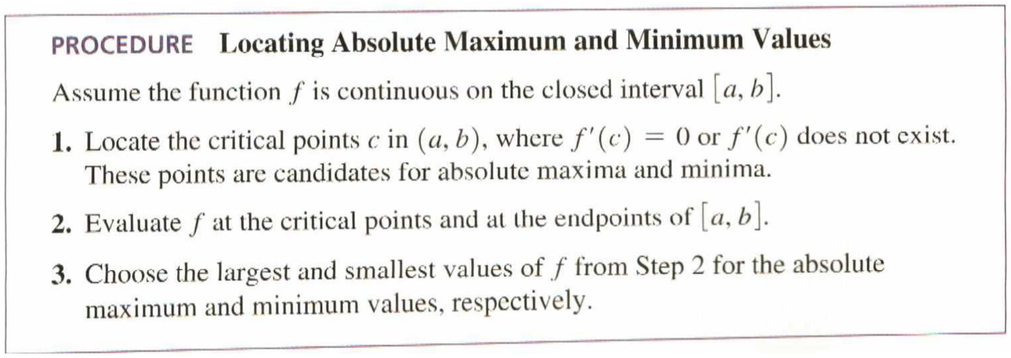
There's a funny trend in the modern math books anyway: they are becoming very algorithmic, with lessons that simply tell the students the exact steps to take in order to solve a problem. Here's a table from an Algebra book by Larson (5th ed) on adding integers:



Addition of Integers

1. To **add** two integers with *like* signs, add their absolute values and attach the common sign to the result.
2. To **add** two integers with *unlike* signs, subtract the smaller absolute value from the larger absolute value and attach thc sign of the integer with the larger absolute value.

and another from a Calculus book by Briggs


PROCEDURE Locating Absolute Maximum and Minimum Values

Assume the function f is continuous on the closed interval $[a, b]$.

1. Locate the critical points c in (a, b), where $f'(c) = 0$ or $f'(c)$ does not exist. These points are candidates for absolute maxima and minima.
2. Evaluate f at the critical points and at the endpoints of $[a, b]$.
3. Choose the largest and smallest values of f from Step 2 for the absolute maximum and minimum values, respectively.

As you can see, some lessons in these books boil down to giving students a recipe to follow. Almost as if the textbook author is "programming" the student. This is happening more and more in the latest edition math and science books.

Should teach kids to code then? Yes, we think so. The match between mathematics and programming is very natural. We think learning to code is as important as learning to read and write, and should be started as early as possible. Perhaps even "the three Rs" (Reading, wRiting, aRithmetic) should be four: Reading, wRiting, aRithmetic, and pRogramming." But really its the aRithmetic that needs to change. It should be "aRithmetic/pRogramming."

The idea of fully integrating computers and mathematics enhances both fields. The computer weans us of pencil and paper mathematics (i.e. the "Einstein-mode") and mathematics gives learning to code a badly needed context.

1.3 Why a new website and book?

The motivation for this book and the site `http://www.codebymath.com`, came from a father infinitely frustrated with the topics his son went through, during his years of math in junior high and high school in the United States. As we went through lesson after lesson, we just didn't understand why most of them weren't being done on a computer. Also, we "get math," and like it too. We use it daily at a rather high level (all have a Ph.D.s in physics), and understand, for example, what things like $\frac{d^2r}{dt^2} = -kr$ mean.

By way of motivation for writing this book, we have compiled a list here of the most frustrating topics in algebra. The frustration was not from the hardworking teachers, or our childrens' grasp of the material, or for the spent time helping them each evening. No, our frustration is from the utter uselessness of these topics, and watching everyone involved (including the *teachers*) struggle with these topics (and form their opinions about math with them in mind). We have two categories. The first is for algebra topics that should only be taught using the computer. The second are topics that (sorry) should not be taught anymore.

1.3.1 Do only on the computer

After all of those years of compulsory math lessons "just because," we believe the following topics, should not be taught again with the pencil-and-paper method. These topics should all be taught on the computer, by having the students write simple computer programs.

- Finding roots of high-degree polynomials by counting sign changes.
- Making a graph by hand.
- Finding factors of integers by hand.
- Reducing fractions to lowest terms.
- Hand factoring quadratic equations, and trinomials.
- Making some any distinction between integers, whole numbers, real numbers, fractions, natural numbers, etc. (hint: they're all *just numbers*).
- Divisibility rules.
- Testing if a number is prime, for any number above 20.
- Finding the GCD or LCF of numbers over 10.

- Percent problem that ask anything other than finding a percentage of a given number. Let's stop problems like "25.2 is 140% of what number?"
- Any operation (add, subtract, multiply, divide, exponent) with polynomials.

1.3.2 Don't do anymore

To go on, the topics listed below are so egregious that should be removed from our math books, and never be taught again, using computer or not:

- Dividing polynomials
- Rationalizing a denominator (all that work for readability?)
- Word problems involving mixtures of things like peanuts and cashews, or the cost of so much of "Brand A" and "Brand B" coffee.
- Word problems that mix solutions, like "How may liters of pure alcohol are there in 2L of a 30% solution of alcohol?"
- Adding, subtracting, or multiplying more than two polynomials.
- Factoring a polynomial with any power greater than 2.
- Factoring polynomials whose "leading coefficient is not 1."
- Factoring by grouping.
- Finding mixed numbers from improper fractions. (What is a "mixed number" anyway?)

The only outcome of such problems is in making students loath algebra (and math and science in general), and cause them (and their parents) to have miserable evenings doing homework. But, we are all for preserving culture and remembering our accomplishments (including those of pencil-and-paper mathematics), so by all means let's archive these lessons into books on the "history of mathematics," or put them into a "museum of mathematics."

We have written this book, that presents a first course in algebra, as it could be learned in terms of writing simple computer programs. It follows topics that are generally ordered the way a typical algebra book would present them. You'll be surprised at the insights writing simple code for algebra problems brings.

1.4 For the experts (keep the emails coming...)

- Yes...we know this book has a lot of brute force searching for solutions using for-loops and random numbers. We know this isn't a good use of computational power and is inefficient. But computers are very powerful and typical algebra problems are so contrived, over such a limited and unimaginative domain, why not just solve it with brute force? We still get an answer and the student did some coding. This is far better than an algebraic answer written on paper, with a box around it.
- Yes...we know our lessons have a lot of print statements, dumping text to the screen for the student to analyze in assessing a given topic. We believe these are first steps in coding.

  ```
  a = np.linspace(0,10,100)
  b = np.exp(-a)
  plt.plot(a,b)
  plt.show()
  ```

  will come later.
- Yes...we know about the books "Numerical Recipes" and TAOCP. We know them very well.
- Yes...we know about Mathematica, Maple, Matlab, Sagemath, and SymPy. We also know we're not sitting an introductory algebra student in front of these.

Chapter 2

Warm-up

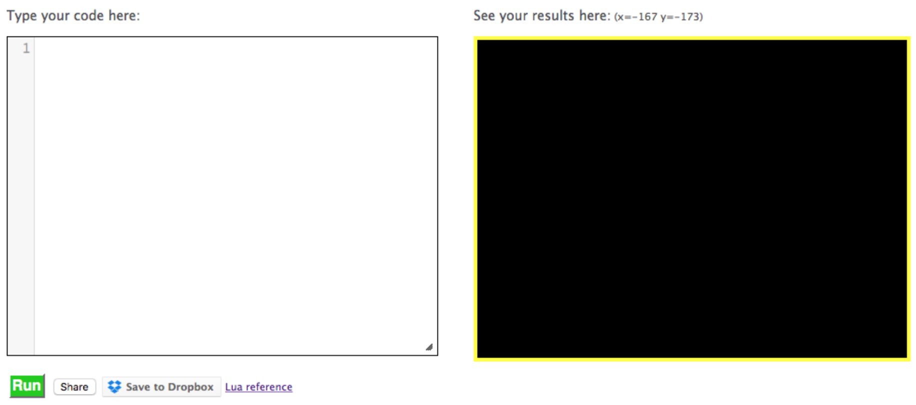
The intent of this chapter is to teach you some basic programming skills, that will take you through this whole book. If you can get through this chapter, understanding variables, conditionals, the for-loop and the if-then statement, then you can actually start learning some math through coding.

2.1 Where do I code?

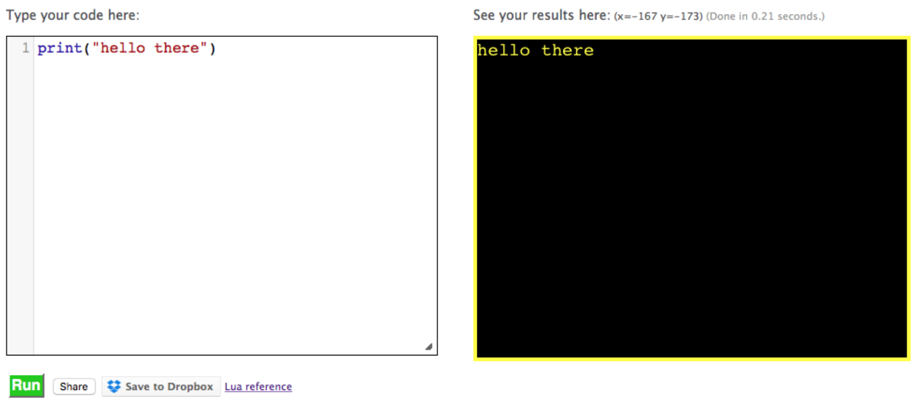
To starting coding according the the lessons this book, go to this page:

http://www.codebymath.com/index.php/welcome/sandbox

The page should look like this



where you type your code in the left box and click "Run." Any output will be seen in the right box. All of this is web-based. You don't need to buy or install any software. All you need is a web-browser and a computer connected to the Internet. Here's an example, where your code is run when you click the green "Run" button.

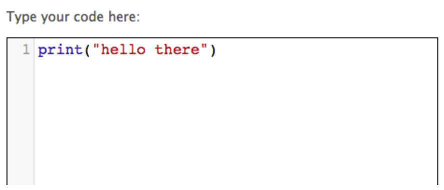

Type your code here:
See your results here: (x=-167 y=-173) (Done in 0.21 seconds.)
Run
Share
Save to Dropbox
Lua reference

```
print("hello there")
```

```
hello there
```

Your code here will be short, so there's no reason to worry about saving it, although you can save things to your Dropbox account. You can also copy and paste code out of the editor window and into some file on your local computer.

Throughout the book, we'll represent the left code window, that looks like this:



```
print("hello there")
```

with a simpler box that looks like this:

```
print("Hello there")
```

We'll assume you can type the code into the code-editor and click the green "Run" button to see your results.

2.2 Basic output and the print statement

Computers typically have some output area on their screen, where they can show you bits of information. For most of the work in this book, we'll use the `print` statement, that prints or displays information you give it to the output window. Try these.

1. Here is a simple message.

```
print("Hello there")
```

2. Assuming your name is "Steve" try this, or substitute in your own name.

```
print("Hello, Steve")
print("How are you?")
```

Try printing any other message you may want.

2.3 The computer as a calculator

Computers are very good at doing calculations. If you just put your formula into a print statement, the computer will happily crunch any formula for you. Try these.

```
print(2+2)
```

```
print(6*9)
```

```
print(400/20)
```

```
print((2+3)*100-67/2^2-(4+14)/100)
```

Note the key operators:

- + means add, as in `print(3+3)` would give 6.
- - means subtract, as in `print(10-3)` would give 7.
- * means multiply, as in `print(5*9)` would give 45.
- / means divide, as in `print(15/5)` would give 3.
- % means to find the remainder of the division, as in `print(10 % 3)` would give 1.
- ^ means exponent, as in `print(5^2)` would give 25.
- (and) are grouping symbols, as in `print( (3+5)*2)` which would give 16, since the (and) force the 3+5 do be done before the 5*2.

Do you remember how to evaluate something like $3+4\times 2$. Is the answer 14, or is it 11? What about $5-3\times 10$. Is the answer 20 or -25? You can check these on the computer like this, but first know on a computer that "times" is done with a `*` and "divide" is done with a `/`. So $3+4\times 2$ can be checked with

```
print(3+4*2)
```

The answer is 11, since division (and multiplication) is done before addition (and subtraction). You can check $5 - 3 \times 10$ with

```
print(5-3*10)
```

which is -25, for the same reason above.

You can also work on those integer addition and subtraction problems that have different signs on the numbers, like these:

1. -18+(-62)

   ```
   print(-18+(-62))
   ```

2. 22+(-17)

   ```
   print(22+(-17))
   ```

3. -84+14

   ```
   print(-84+14)
   ```

4. -13-7+11-(-4)

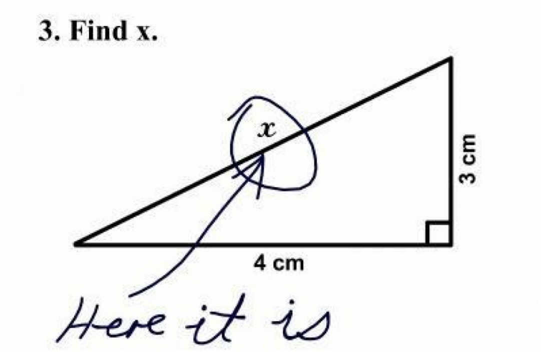
   ```
   print(-13-7+11-(-4))
   ```

As you can see, the computer takes in mathematical expressions in an almost identical form to your book.

2.4 Variables

As you study algebra, you'll find that often ideas of math will be presented, not in terms of numbers, but in terms of letters that stand for number. These letters are called variables in algebra. Why do we do this?

Well suppose you have 4+3, which of course is equal to 7. But suppose you wanted to be more general and needed to study what happens to any number when 3 is added to it. You could write x+3, where x is the variable. Using x as a variable is very common. In fact, x is probably the most famous variable in algebra, even the butt of jokes, like this one.



(Hint: when asked to “Find x” in math, it usually means to use math to find what x should be, or to ”solve” for x. In this case, we’d use something called the “Pythagorean Theorem” to compute x, which would be or x=5.)

Now, back to our example of adding 3 to a number. If you think of x as holding a 4, youd get 4+3 or 7..nothing changes there. But using “x” instead of 7, allows you to form a mathematical statement about adding 3 to any number you wish. Why is this done? Mostly to be more general about a fact of math. In this case, 4 isnt the only number you can add 3 to.

We can even start working with code on this. Go to the code page discussed above and type in the following:

```
x=4
print(x+3)
```

When you click the “Run” button, a 7 appears in the output area. Now go back and change the x to a 3.What do you get now when you click Run? Change the x to any number you wish, and click “Run” again. Try this a few times. Is the result correct? In this example, using the variable x allowed us to be very general is presenting how we add 3 to a number.

Here’s another experiment concerning the general nature of variables. Suppose you had a bus that seats 50 people, and you needed a real time counter that tells you how many seats are still available, given the number of people may already be on the bus. If there are 27 people on the bus, you could write some code like this

```
print("For your 50 seat bus,")
print("there are",50-27, seats left.")
```

This would print

```
For your 50 seat bus,
There are 23 seats left.
```

But what if 29 people were on the bus? You'd have to go in and change the 27 to a 29. Also, what if you got a bigger bus that could hold 75 people? Youd now have to change the 50 (in two places!). It would be easier to just variables, like this.

Lets have the variable `n` be the number of seats on the bus, and `x` be the number of people on the bus. With this, at any time, `n-x` would be the number of seats available, so your code would look like this:

```
n=50
x=27
print("For your",n,"seat bus, ")
print("there are",n-x,"seats left.")
```

Now, for any bus size, or any number of people, you just have to change the assignments to n and x in the first two lines. In the second print statement, the "rule" `n-x` is the generalized expression for how many seats are left on the bus, given that the bus can hold n people, and x people are already on board.

This is the epitome of algebra. Using variables, and being very general about math rules to express what we want about math.

Exercises

1. The area of a square is base time height. Suppose you have a square whose height is 5.5 and base is a variable x. Write some code to define x, then print the area of the square.
2. Extending #1 above, by trial and error in your assignment of x, find what x should be to give an area of 52.47 square units.

2.5 Doing calculations with variables

Understanding a bit about variables, now allows you to do some calculations with them, and also see how good computers are at crunching numbers. But first a bit of review.

You can also calculate with variables, like this

```
x=3
print(x+3*x)
```

which should give 12, but feel free to change the x=3 line and put whatever formula you wish inside of the parenthesis of the print statement. How about this?

```
a=10
b=16
x=2
print(a+a-b*x+a*b-663)
```

So in summary, in algebra and on the computer, variables and numbers are to be mixed in crunching numbers, as if they all are actually numbers in the end.

Exercises

1. Write some code that will assign x=some-number. Print 2x, 3x, and 4x.
2. Define 3 variables whose names you choose. Invent a formula that uses all of the variables to compute a number, whose result is the day of the month on which you were born.
3. Suppose you divide two numbers. When you divide one by the other, can the remainder ever be greater than the divisor? Use the `%` operator to find out.

2.6 Input from the keyboard

Just as the `print` statement allows you to send information (text) to the screen as *output* from your program, the `input` statement allows you to send information into your program as *input*. It works like this:

```
print("What is your name?")
x=input()
print("Hello there,",x)
```

Here the `input()` function pauses your program and waits for you to type a line of text, which is subsequently assigns to variable x, for use in your program. You can probably guess what the program above does.

What's useful about this is that it can make your programs, that use variables, *interactive*. Take the program above, that told you how many seats are left on a 50 person bus.

```
n=50
x=27
print("For your",n,"seat bus, ")
print("there are",n-x,"seats left.")
```

Suppose you replace the `x=27` with `x=input()` (and a text prompt) like this:

```
n=50
print("How many people are on the bus?")
x=input()
print("For your",n,"seat bus, ")
print("there are",n-x,"seats left.")
```

Now the program *will ask you* how many people are already on the bus and compute the number of empty seats.

Using the `input()` function is a handy way of getting different values into variable, without having to edit the code itself.

2.7 Counting

Counting can be fun, particularly if you can reach some very large number, by ones, without stopping or leaving any out. But machines are better counters than we are.

If you have a calculator, try this. Key in “1” then a “+” then a “1” again (to add 1 to 1), and lastly hit the “=” key. Now press “=” over and over again. The calculator should count up by ones. How high can you go? Counting is more convenient when aided by a machine, like your calculator, because although keeping track of the numbers can be boring, the actual numbers themselves can be interesting.

Computers are very natural counters, using something called a “for-loop.” Let’s count from 1 to 10 like this:

```
for x=1,10 do
 print(x)
end
```

The word “for” here sort of means “which” or “for which” as in “give me the numbers which are 1 to 10” or “generate numbers for which are in the range from 1 to 10." The word "do" tells the computer what you want it to do with each number it find in the range. The word "end" tells the computer the end of what you want it to do with each number.

You can see how easy it is to count using a computer, and you don't always have to start at 1 and end at 10 either. Try some others ranges, like

```
for x=1,100 do
 print(x)
end
```

or

```
for x=57,113 do
 print(x)
end
```

or

```
for x=350000,350500 do
 print(x)
end
```

When was the last time you counted up near 350,000? For-loops don't just have to count by ones either. You can supply the increment by putting another comma and number after the second number, like this:

```
for x=1,20,2 do
 print(x)
end
```

to count to 20 by twos (1, 3, 5, 7..). What about

```
for x=5,50,5 do
 print(x)
end
```

to count from 5 to 50 by fives. What about counting down? Well when counting down, the increment has to be negative, so you can try something like

```
for x=10,1,-1 do
 print(x)
end
```

or

```
for x=99,1,-1 do
 print(x,"bottles of beer on the wall")
 print(x,"bottles of beer...")
 print("Take one down, pass it around..")
 print(x-1,"bottles of beer on the wall.")
 print()
end
```

Exercises

1. Write some code using a for-loop that would produce multiplication tables for 3s, like this: $1 \times 3 = 3$, $2 \times 3 = 6 ... 20 \times 3 = 60$.

2. Do you think 1,000,000 is a large number for a computer to count to by ones (starting at 1)? Try it.

2.8 More than just counting

Remember we were talking about using variables above, as a general way of talking about math? As you've seen, the for-loop allows you to run a given variable through a whole range of numbers. What's nice about this is that for each number the for-loop visits, you can do something (like a calculation) with the number itself. "Do something" means you can use your for-loop variable in a larger mathematical expression.

How about instead of just printing x going from 1 to 10 (as we did above), we print "two times x," like this:

```
for x=1,20 do
 print(2*x)
end
```

Or, let's make a table of numbers from 1 to 20, that prints the number, its square, and its cube, like this:

```
for x=1,20 do
 print(x)
 print(x^2)
 print(x^3)
end
```

You can also use intermediate variables, as in s for the square, and c for the cube, like this:

```
for x=1,20 do
 s = x^2
 c = x^3
 print(x,s,c)
end
```

Exercises

1. Write some code that will print all integers from 1..100, including their squares, and square roots. Use math.sqrt() to compute the square root of a number, so print(math.sqrt(25)) would give 5.

2. Print the area of all squares that have a height of 3.5 and all widths from 5 to 95 in steps of 5.

3. Use a for-loop to show that 45*89 is the same as adding 45 to itself 89 times.

2.9 Comparing numbers

Computers are really good at comparing numbers. The symbols < (less than), > (greater than), <= (less than or equal to), and >= (greater than or equal to) are known by the computer. Two more are == (two equal signs) to check if two numbers are equal, and ∼= to check if two numbers are not equal. These operations check the numbers, just as you would yourself, and return a "true" or "false," depending on how it all works out. (True and false are sort of like you thinking "yes" or "no" to yourself, when comparing two numbers.) Here are some examples.

To check if 5 is greater than 2, you'd do

```
print(5 > 2)
```

which would result in a "true" being displayed, meaning "yes, 5 is greater than 2." It of course works for all numbers, including negative numbers. Try these:

```
print(-5 > 2)
```

which results in false, as in "no, -5 is not greater than 2." What about

```
print(-10043 < -773)
```

which results in true. Next we try

```
print(5 >= 5)
```

which also results in true. For equality, we have

```
print(273 == 273)
```

(that's two equal signs in there, or **==**), which results in true, and

```
print(273 ~= 273)
```

which results in false since 273 does not equal 273. Lastly, we have

```
print(273 ~= 5)
```

which results in true, as in “yes, 273 is not equal to 5.”

These comparison operators also work with fractions. If you interpret the fraction bar to be “divide,” then fractions can be put into the computer almost as you’d write them down, with $\frac{1}{2}$ going in as `1/2` and $\frac{2}{3}$ going in as `2/3`. Fractions can be hard to visualize when comparing them, so let’s see how the computer can help. Try this:

```
print(1/2 > 2/3)
```

to see if $\frac{1}{2}$ is greater than $\frac{2}{3}$. Here is another example involving $\frac{1}{3}$ and $\frac{1}{5}$.

```
print(1/3 > 1/5)
print(1/3 < 1/5)
print(1/3 ~= 1/5)
```

Decimals work here too. Try this to see which comparisons of -3.1 and 2.8 come out to true or false.

```
print(-3.1 > 2.8)
print(-3.1 < 2.8)
print(-3.1 ~= 2.8)
```

Exercises

1. Test if 2+4*3 is equal to (2+4)*3 or 2+(4*3).
2. Are all negative numbers you can think of always less than zero?
3. Is 55 is larger than 25. Test this. Is -55 larger than -25?
4. Use some simple code and the comparison operators to demonstrate how these numbers compare.
   (a) $|-84|$ and 84

(b) $|-10|$ and $|4|$.

(c) $|\frac{5}{2}|$ and $|\frac{8}{9}|$

(d) $-|-1.8|$ and $|5.7|$

(e) $|2.3|$ and $-|2.3|$.

2.10 Making Decisions

When dealing with numbers, sometimes you want to make a decision for a course of action, depending on the value of a variable. Decisions on a computer are made in this regard, at the “true” or “false” level. For example, if some variable x, that is holding some temperature value is greater than 95, you might want to print “it’s hot in here.” So if “$x > 95$” is true, you’d want to print your message.

Decisions on a computer are made with the `if-then` statement, and it work like this. You come up with a condition to test, like $x > 95$, and insert it into an `if-then` statement like this

```
x=100
if x > 95 then
 print("x is greater than 95")
end
```

These read in words pretty much the way they read in the code here: “if x is greater than 95, then....do what’s between the ’then’ and the ’end’ words.” In this case the text “x is greater than 95” will be printed.

There’s a slight extension to the `if-then` statement, which is the `if-then-else` statement, which works like this

```
x=100
if x > 95 then
 print("it's hot in here")
else
```

```
  print("it's not that hot.")
end
```

If the condition between the `if` and `then` is false, then the statements between the `then` and `else` will be skipped and instead, the statements between the `else` and `end` will be done. For the code above, “it's hot in here” will be printed. But suppose you changed the `x=100` to `x=85`. Then “it's not that hot.” will be printed. This is because the $x > 95$ is now false, so the `print("it's hot in here")` will be skipped and the `print("it's not that hot.")` will be done instead.

Now that you know how to mix variables and decisions, let's do some math with it all.

2.10.1 The absolute value

The absolute value of a number means to take a number and remove its minus sign, if it is negative. The symbols for absolute value are | and |, so $|-5|$ is 5, $|-200| = 200$, and $|67| = 67$ (nothing changes for a positive number).

But how is the minus sign dropped? How about by finding the negative of the negative number. As an example, what is the “negative” of -5? It would be $-(-5)$ or $+5$. So $|-5| = -(-5) = 5$, just like $|-113| = -(-113) = 113$.

Can we write some code to handle this? That is, print the absolute value of a number? What are the rules for this? How about “if a number is negative, print the negative of it, otherwise just print the number.” Think carefully now, how do you know if a number is negative? Try this:

```
x=??
if x < 0 then
  print(-x)
else
  print(x)
end
```

Try filling in different test numbers into the `x=??` line, both positive and negative. Does this code always show the absolute value of a number?

2.10.2 Ordering numbers

Suppose you had two numbers in variables a and b, and you always wanted to display them in numerical order, smallest first. How would you do this with an if-then-else statement? How about this:

```
a=??
b=??
if a < b then
 print(a,b)
else
 print(b,a)
end
```

Fill in numbers for the `a=` and `b=` lines. Can you explain how this works? Can you explain what happens if a=b? Does this code work for negative numbers too? What about for very large numbers? Decimals?

2.10.3 Divisibility

Divisibility is about finding out if a number divides evenly into another number. Such a number is called a “divisor,” or ”factor” of the other number. For example, 12 is divisible by 4 because $12 \div 4 = 3$ with no remainder (or a remainder of 0). You can also say that 3 is a divisor of 12. You can also say that 3 is a “factor” of 12. But the numbers can get larger. Is 73 is a factor of 1241? Divisibility is easily handled using the remainder operator (sometimes called the “mod” operator), which is `%` symbol. So,

```
print(12 % 4)
```

would print zero, and

```
print(1241 % 73)
```

would print zero too. This is because 12 is evenly divisible by 4 and 1241 is evenly divisible by 73. You can make this more general, with code like this

```
a = 1241
b = 73
if a % b == 0 then
 print("Yes,",b,",evenly divides into",a)
else
 print("No",b,"does not divide evenly into",a)
end
```

Here we once again use the general nature of variables to make the code more useful. All you have to do is change the variables, and you can easily test any numbers.

2.10.4 Prime and Composite Numbers

Usually after divisibility, some work is done on prime vs. composite numbers. Prime numbers are only evenly divisible by 1 and themselves. Composite numbers are evenly divisible by 1, themselves, and other numbers as well. So for example, 13 is prime, because only 1 and 13 evenly divide into it, while 14 is not prime, since 1, 2, 7, and 14 all divide evenly into it.

Using a for-loop, the remainder operator (`%`), and an if-then statement, one can have a prime or composite number checker up and running. We can use a for-loop to visit all number between 1 and a given number of interest, using `%` to check if a given number divides evenly into our number of interest. Code like this

```
n = 1241
for i=1,n do
    if n % i == 0 then
     print(i,"divides evenly into",n)
    end
end
```

Where that's two equal signs, or `==` (with no space between them) between the `n % i` and the `0`. Here we note that this code shows us that 1241 has another factor lurking out there other than 1, 1241 and 73. Do you know what it is?

2.11 Multiplication and Division

You may have heard that multiplication is repeated addition and division is repeated subtraction. Take 5×3. This is 5 added to itself 3 times $(5+5+5)$ or 3 added to itself 5 times $(3+3+3+3+3)$. Both are equal to 15. You can test this here using a for-loop and some variables. Try this to "multiply" 5 and 3 to get 15.

```
p = 0
for i=1,3 do
   p = p + 5
end
print(p)
```

Here notice the for-loop isn't used for its driving variable (the `i`) directly. It's only used to loop 3 times, to handle the 3 in 5×3. Notice also how we used a variable called `p` (as in product) to keep track of the repeated addition of 5, as in the line `p=p+5`. These lines can be tricky to understand, but with the single `=` sign, it always does this: 1) compute the right hand side of the `=` sign. In this case `p+5`, then 2) send the result leftward, and save it into the variable on the left hand side of the equal. So, the first time through the for-loop, a 5 will be in variable `p`. Add the line `print(p)` after the `p=p+5` to see this.

This of course can all generalized with variables. Say we want to multiply a number in `a` by a number in `b`, we could write

```
a=5
b=3
p = 0
for i=1,a do
   p = p + b
end
print(p)
```

where all you have to do is set values to `a` and `b` to have them multiplied by the repeated addition.

Division as repeated subtraction is a bit tricker. Essentially, you keep subtracting the divisor from the dividend, until the dividend reaches zero. The quotient is how many times you could do this subtraction. Here's the code

```
a=75
b=25
q = 0
while a > 0 do
   a=a-b
   q=q+1
end
print(q)
```

This uses another type of loop, called the while-loop. A while loop repeats what's between its `do` and `end` lines *while* the condition between the `while` and `do` words is true (hence the name "while-loop"–keep doing something *while* a condition is true). In this case, 25 can be subtracted from 75 a total of 3 times before the 75 becomes zero.

2.12 Fractions and the greatest common divisor

In Section 2.10.3, you saw that a divisor is a number that divides evenly into another number. The greatest common *divisor* (G.C.D.) is the largest number that can divide evenly into two numbers. Get it? Greatest...common.....divisor. So, the GCD of 10 and is 5. The GCD of 100 and 30 is 10. You can find the GCD of any two numbers using a built in function called `gcd`. It works like this, for example, to find the GCD of 100 and 30:

```
print( gcd(100,30) )
```

Did you know that the GCD is how you reduce fractions to lowest terms? That's right, if you take a fraction and divide both the numerator and denominator by the GCD of the numerator and denominator, you'll have reduced the fraction to lowest terms.

Let's test this on the fraction $\frac{18}{24}$. What's the GCD of 18 and 24? We can find out like this

```
print( gcd(18,24) )
```

in which case we'll get 6. If we now divide 18 by 6 (=3) and 24 by 6 (=4), the fraction will become $\frac{3}{4}$, which is $\frac{18}{24}$ in lowest terms. Here's some code you can try on any fraction, by setting `n` to the numerator, and `d` to the denominator:

```
n=18
d=24
print(n,"/",d," in lowest terms is: ")
g=gcd(n,d)
n = n / g
d = d / g
print(n,"/",d)
```

Note we hold the GCD of `n` and `d` in a third variable called `g`, to make it convenient to use in dividing it into both the numerator and denominator.

2.13 Adding fractions and the least common multiple

A multiple of a number is what you get when you take a number and multiply it by some integer. Take 3. Multiplying 3 by 2 gives 6. So 6 is a multiple of 3. Multiplying 3 by 445 gives 1,335, so 1,335 i a multiple of 3. Here are the first 50 multiples of 7:

```
for n=1,50 do
 print(7*n)
end
```

The "least common multiple" or LCM is the smallest number that is a multiple of two other numbers. So, for 4 and 6, the LCM would be 12. We get 12 by multiplying 4 by the integer 3, and 12 by multiplying 6 by the integer 2. Furthermore, 12 is the smallest number that is a multiple of both 4 and 6. You can see how this happens here, where we'll print the first 20 multiples of both 4 and 6:

```
for n=1,20 do
 print("n is ",n,"4n=",4*n,"6n=",6*n)
end
```

where you can see that 12 is the smallest multiple that pops up for both numbers.

Once again, computers are good at finding LCMs. Here, you can use the built in function `lcm`. So to find the LCM of 4 and 6 you would write

```
print( lcm(4,6) )
```

which would result in 12.

Did you know that the LCM is used in adding fractions, when their denominators are not the same? In fact the common denominator of a fraction is the LCM of both denominators. It works like this. Let's add $\frac{4}{5}$ and $\frac{11}{15}$. The LCM of 5 and 15 is

```
print( lcm(5,15) )
```

which would result in 15. So we want the denominator of the $\frac{4}{5}$ to be a 15. This means multiplying both the top and bottom by 3, as in $\frac{4 \cdot 3}{5 \cdot 3} = \frac{12}{15}$. But see what this did? It gave us a version of $\frac{4}{5}$ that has a denominator of 15, just like the other fraction (the $\frac{11}{15}$). If you don't believe $\frac{4}{5}$ is the same as $\frac{12}{15}$, do this

```
print( 4/5 == 12/15)
```

So now the problem becomes adding $\frac{12}{15} + \frac{11}{15}$ which is $\frac{23}{15}$. Here's some code that will add any two fractions, $\frac{a}{b} + \frac{c}{d}$, in this case we'll add $\frac{4}{5}$ and $\frac{11}{15}$.

```
a=4
b=5
c=11
d=15
L=lcm(b,d)
f=L/b
print(a,"/",b,"becomes")
a=a*f
b=b*f
print(a,"/",b)

f=L/d
print(c,"/",d,"becomes")
c=c*f
d=d*f
print(c,"/",d)

print("So our answer is")
print(a+c,"/",b)
```

Exercises

1. Can you extend the fraction adder to also reduce the final answer to lowest terms?

2.14 Fractions and Decimals

Fractions may be converted into decimals quite easily, since this is how they're represented on a computer anyway. Try these

```
print( 1/4 )
```

and

```
print( 1/6 )
```

where you can see that 1/4 terminates, but 1/6 does not. You can see the repeating pattern in 8/33 by

```
print( 8/33 )
```

There is of course the famous approximation for π or 22/7

```
print( 22/7)
```

But 335/113 is even better and 57843/18412 is even better still. Notice anything about 800/81?

2.15 Exponents

A lesson above describes how repeated addition is what multiplication is. *Exponents* are what you get when you repeatedly multiply the same number by itself. As an example, 7×7 is 49 and this 7^2, and $5 \times 5 \times 5 = 125$ is $5^3 = 125$. You can write some code that uses a for-loop to test this. This code will raise the number in variable `n` (for "number") to the power in variable `e` (for "exponent").

```
n=5
e=3
p = 1
for i=1,e do
 p = p * n
end
print(n,"^",e," is",p)
```

Note how the for loop is used the moderate the sequence of multiplications. Here we start variable `p` (for "product") off at 1, and with each iteration of `e` times through the for-loop, `p` gets multiplied by `n`. This effectively multiplies `p` by `n`, `e` times, which is what an exponent is.

Of course on the computer the ^ is used for exponent, so

```
print(5^3)
```

will print 125, and

```
n=5
e=3
p=n^e
print(n,"^",e," is",p)
```

is the same as the first listing here, but without the for-loop.

Chapter 3

Fundamentals of Algebra

3.1 Coefficients

Hopefully you're familiar with variables now, and understand what code like this does.

```
x = 5
print(x^2)
```

which would result in a 25 being displayed, since this code computes x^2. It turns out that if you put a number in front of the x^2, like this $3x^2$ or

```
x = 5
print(3*x^2)
```

the 3 is called the "coefficient" of x. A coefficient is a number that is *multiplied* by a variable. Coefficients are helpful, because they allow you to scale the result of something like your x^2 calculation. Here's an example.

Suppose you needed to compute the area of cloth needed to cover a square table. If the side length of the table was in variable x, then the area would be base$\times$width or $A = x \cdot x$ or $A = x^2$. For a table whose side is 2 (feet) in length, the code to compute this would be

```
x = 2
print(x^2)
```

But suppose we needed 4 tablecloths. We'd put the coefficient of 4 in front of the x^2 as in $4x^2$, to get our result, like this

```
x = 2
print(4*x^2)
```

which would display the area of 4, 2×2 tablecloths.

But expressions aren't always so simple, we can have $x^5+x^4+x^3+x^2+x+1$, which you can see is a pretty "rigid" equation. Once we plug in a value for x, as in

```
x = 2
print(x^5+x^4+x^3+x^2+x+1)
```

we see that all of the powers of x count equally the same. We have "one unit" of x^2, "one unit" of x^4, erc.

Adding coefficients allows us to get almost any value out of it, as in

```
x = 2
print(3*x^5-2*x^4+0.5*x^3-x^2+x/5+1)
```

Here you can see the x^5, x^4, x^3, x^2, and the x, but they're all "dressed up" now with the coefficients of x^5 is 3, that of x^4 is -2, 0.5 for x^3, -1 for x^2, 1/5 for x, and 1 for the x^0 term. You'll get quite a different answer when evaluated at $x=2$.

In the big picture, polynomials with their coefficients don't really have much use, other than one: in basic motion. If you have an object moving with speed v (v for velocity), with acceleration a, its position at some point t later in time is given by $x = vt + \frac{1}{2}at^2$. This is a polynomial not in x, but in time, t. You'll note that the object's speed v is the coefficient of t, and half of its acceleration, $\frac{1}{2}a$, (don't ask–it's just the way it is) is the coefficient of t^2.

So if an object is moving at 10 m/s, with an acceleration of 3 m/s^2, its position (x) can be found from $x = 10t + \frac{3}{2}t^2$, where you can see the roles that the 10 and $\frac{3}{2}$ play.

There is a lot of coding possibilities with this. Here's code that will compute the object's position in 5 seconds:

```
t=5
v=10
a=3
print(v*t+0.5*a*t^2)
```

Here is its position in 2 second intervals for 50 seconds:

```
t=5
v=10
a=3
for t=0,50,2 do
 print(v*t+0.5*a*t^2)
end
```

where again note the v (speed that you choose) is the coefficient of t and half of a (the acceleration you choose) as the coefficient of t^2.

3.2 Like terms and combining them

A coefficient and a variable, such as the 3 and the "x" in $3x$ are collectively called a "term." There are certain rules of algebra that dictate how these terms are combined. Consider this code:

```
x=5
print(3*x)
```

that prints the result of $3x$ when x is 5 (you'll get 15). Suppose we extended the code to print $3x + 3x$. What would we get?

```
x=5
print(3*x+3*x)
```

You'll see a 30 in your output window. So what really happened? In this case, $3x$ gave 15, and if we add a $3x$ plus another $3x$ we should get $15 + 15$ or 30, which we did. But is there some algebra in here? Try this code:

```
x=5
print("Does",3*x+3*x,"=",6*x,"?")
```

where you should see "Does 30 = 30 ?" on the screen, confirming that for at least $x = 5$, $3x + 3x$ is indeed $6x$. You could test it for more numbers with a for-loop like this:

```
for x=1,50 do
  print("Does",3*x+3*x,"=",6*x,"?")
end
```

where it also seems to hold for all numbers from 1 to 50.

As you can see, for terms that are added, given that they have variables to the exact same powers, 1 in this case, as $x = x^1$, you can combine them into a single term by just adding their coefficients. Try this

```
x=5
print("Does",183*x+7*x,"=",190*x,"?")
```

and

```
x=5
print("Does",7*x-2*x,"=",5*x,"?")
```

What about $3x + 5x^2$? Is there some way of combining these? $8x$? $8xx^2 = 8x^3$? It turns out there is not. These cannot be combined because they are not like terms, since the 3 is attached to an x, and the 5 is attached to the x^2.

3.3 Properties of Algebra

There are few properties (or rules) of algebra that always hold when combining terms. They have the names below. We don't think these names should be memorized, but you should simply know that the properties they exhibit are "rules of algebra," and be able to recognize when they are at work in a given algebraic expression.

We'll need a basic code structure to test these rule, by running one form on one side of an equal sign, and another form on the other side, to see if they're true (that is, if the left-side of the equal sign always equals the right). We'll run our test range over the first 20 integers, and use the two variable x and y as our test variables. Feel free to choose your own range. Here's the code basic structure, where you'll note that we have a for-loop inside of another for-loop, which is a perfectly OK thing to do:

```
for x=1,20 do
 for y=1,20 do
  print("Does",x,"=",y,"?")
 end
end
```

Note the so called "nested" for loops work like this: the first value of the outer loop (the x-for-loop) is set, in this case starting at 1. With $x = 1$, the y for-loop is run from 1 to 20, with all values of y being visited while $x = 1$. When this inner for-loop is done (the y for-loop), the outer one takes over again, advancing x to 2, at which time the inner one will run from 1 to 20 again, but now with $x = 2$. This repeat until the x for-loop runs its course up to 20.

Note, this code doesn't actually do anything. It's just the core structure we'll use to test the rules of algebra below. We need a bunch of combinations of two variables to do this, hence the two for loops above.

3.3.1 Commutative Property of Addition

This one states that two numbers (x and y) can be added in any order and still get the same answer, or $x + y = y + x$. You can test this in code via

```
for x=1,20 do
 for y=1,20 do
  print("Does",x+y,"=",y+x,"?")
 end
end
```

3.3.2 Commutative Property of Multiplication

This one states that any two numbers (x and y) can be multiplied in any order and still get the same answer, or $x \cdot y = y \cdot x$. This can be tested via

```
for x=1,20 do
 for y=1,20 do
  print("Does",x*y"=",y*x,"?")
 end
end
```

3.3.3 Associative Property of Addition

This one states that when adding any three numbers, it doesn't matter which two you add first, as in $(x+y)+z = x+(y+z)$. We'll put in a third for-loop to run values over z too, as in:

```
for x=1,20 do
 for y=1,20 do
  for z=1,20 do
   print("Does",(x+y)+z,"=",x+(y+z),"?")
  end
 end
```

```
end
```

3.3.4 Associate Property of Multiplication

This one states that when multiplying three numbers, it doesn't matter which three you multiply first, as in $(xy)z = x(yz)$. Here's some code, again with the third for-loop over z to test this:

```
for x=1,20 do
 for y=1,20 do
  for z=1,20 do
   print("Does",(x*y)*z,"=",x*(y*z),"?")
  end
 end
end
```

3.3.5 Distributive Property

This one states that $x(y + z) = xy + xz$, where you an see on the left hand side how the x gets "distributed" in along the y and z. Here's the test code:

```
for x=1,20 do
 for y=1,20 do
  for z=1,20 do
   print("Does",x*(y+z),"=",x*y+x*z,"?")
  end
 end
end
```

3.3.6 Additive Identity Property

This one simply says that $x + 0 = 0 + x = x$, or if you add zero to a number, the number doesn't change. The code to test this is rather quick with:

```
for x=1,20 do
    print("Does",x+0,"=",x,"?")
end
```

3.3.7 Multiplicative Identity Property

This one is similar to the "additive identify property" above, but instead of adding 0 to a number, it refers to multiplying a number by 1, which always gives the same number. It basically says that $1 \times x = x$. The code to test this is rather quick with:

```
for x=1,20 do
    print("Does",1*x,"=",x,"?")
end
```

3.3.8 Additive Inverse Property

This one says that if you take a number and add *its negative* to it, you'll get zero, as in $x + (-x) = 0$. Here's some test code:

```
for x=1,20 do
    print("Does",x+(-x),"=0?")
end
```

3.3.9 Multiplicative Inverse Property

This one is similar to the "additive inverse," but works with multiplication. It says if you multiply a number x, by its inverse, or $\frac{1}{x}$, you'll get 1, as in

$x \cdot \frac{1}{x} = 1$. Here is some test code:

```
for x=1,20 do
    print("Does",x*1/x,"=1?")
end
```

3.3.10 Testing algebraic properties with random numbers

In the last section, a variety of algebraic properties were tested using for-loops. The loops were useful in iterating over many numbers to test if the properties held up. In many cases, you might be skeptical (or at least unimpressed) with the test range, since a rather small range of numbers was tested (just from 1 to 20 in many cases). If you want to expand the test ranges, you could change the ranges of the for-loops, but the programs would start to take a long time to run, and would generate a lot of output. If, for example, you tested the "distributive property" from 1 to 100, it would result in $100 \times 100 \times 100$ or $1,000,000$ lines to read! What if we could visit a broader range of number combinations, but perhaps not so detailed and inclusive to test the algebraic properties? We can, with *random numbers*.

As the name implies, instead of testing the cases with sequential groups of numbers, say 20 numbers from 1 to 20, why not test them with again 20 numbers, but allow the numbers to range from 1 to 1,000,000? In this case, we'd still have only 20 lines to inspect, but we'd see a *wide range* of test numbers.

We can draw a random number using a function called `math.random(low,high)`, which will return a random number between `low` and `high`. Try running this code a few times:

```
x=math.random(1,100)
print(x)
```

This code will display a different (i.e. random) number between 1 and 100 each time the code is run. Armed with this idea, here's another test for the "Multiplicative Identity Property" (above):

```
for i=1,20 do
    x=math.random(1,1000000)
    print("Does",1*x,"=",x,"?")
end
```

and another for the "multiplicative inverse property:"

```
for i=1,20 do
    x = math.random(1,1000000)
    print("Testing:",x,"Does",x*1/x,"=1?")
end
```

In these listings, note the for-loop in i is only meant to run the programs 20 times. The actual x value to use in the tests is chosen using the `x=math.random(1,1000000)`, which chooses a random number between 1 and $1,000,000$.

Here's a random number test for the "distributive property:"

```
for i=1,20 do
   x=math.random(1,1000000)
   y=math.random(1,1000000)
   z=math.random(1,1000000)
 print("Does",x*(y+z),"=",x*y+x*z,"?")
end
```

In all of these cases, within 20 easy-to-inspect lines, you'll see a richly diverse and interesting test-set of numbers imposed on the algebraic property. Perhaps you'll even be more convinced on the truth of each property now.

3.4 Simplifying Algebraic Expressions

The reason for presenting the rules in Section 3.3, is because sometimes you'll encounter an expression like this

$$5x + (x - 7)2 \tag{3.1}$$

and be asked to "simplify it." In this case, you'd recognize the $(x - 7)2$ as ripe for the distributive property, where it'll become $2x - 14$. Putting this back into the original expression will give:

$$5x + 2x - 14. \tag{3.2}$$

Now the $5x$ and $2x$ are "like terms," so they'll combine to give $7x$, so all told the original expression it equal to $7x - 14$. In the spirit of testing such things with random numbers as outlined in Section 3.3.10, you can test that this is true like this:

```
for i=1,20 do
  x=math.random(1,1000000)
  print("Does",5*x+(x-7)*2,"=",7*x-14,"?")
end
```

You should always know that a "simplified" expression is mathematically identical in function to its "unsimplified" form, and code like this should prove it.

In pencil and paper algebra, you will look at a multitude of randomly constructed expressions, like $(2x-10)-(2x-10)$, $2(x-2)+4$, or $8m-(3m-7)$, and be asked to "simplify" them. It turns out though, that computers are *really good* at simplifying such expressions. Here we offer the function called `algebra()` that can do such work for you. It works like this.

Suppose you want to simplify the expression above $5x + (x - 7)2$. Code to do this would be:

```
result=algebra("5x+(x-7)2")
print(result)
```

Here you'll see that within the parentheses of the function `algebra`, is the expression you want to work on (in double quotes). The function `algebra` will apply the algebraic rules above and return the result, in this case which will be put into a variable called `result` (in computer programming, variables can be whole words, not just single variables, like x or y). Next, we simply print the result to the screen, which in this case will be $-14 + 7x$.

Here are a couple more examples. This one will simplify $8m - (3m - 7)$:

```
result=algebra("8m-(3m-7)")
print(result)
```

This one will simplify $z(3z - 7) + 4z^2$):

```
result=algebra("z(3z-7)+4z^2)")
print(result)
```

We encourage you then to use the `algebra` function to simplify your mathematical expressions, and use such a tool to check your work should you be doing the simplification by hand. If needed use some randomly selected numerical values for the involved variables to test that the simplified version yields the same result as the original.

3.5 Introduction to Equations

Equations are at the heart of algebra, and indeed most of mathematics. In form they consist of an equal sign, with something to the left of it (the "left hand side" or LHS), and to the right of it (the "right hand side" or RHS). An example might be $x = 10$, with x on the LHS and 10 on the RHS. Another is $2x+5 = 14z-8+x$ with $2x+5$ on the LHS, and $14z-8+x$ on the RHS. Equations can signify one of two things:

1. In a form like $x = 10$, it means that one should think of "10" wherever they see an x. Another form of this would be $x = 2y+9$, which means $2y + 9$ can be substituted in for x, wherever x appears.
2. In a form like $2(x + 5) = 14(-2x - 1)$, it means there's a delicate balance, meaning that x can only take on the special value that will

make $2(x+5)$ numerically equal to $14(-2x-1)$. (Sometimes, there can even be more than one value of x that will make this happen.)

In traditional algebra, a lot of time will now be spent solving such equations. This will involve a lot of use of the properties outlined in Section [?]. Let's do this here. Let's solve $2(x+5) = 14(-2x-1)$.

1. First, we'll distribute the 2 on the LHS and the 14 on the RHS, like this: $2x+10 = -28x-14$.
2. Next, we'll add $28x$ to both sides like this $2x+10+28x = -28x-14+28x$. We can always add or subtract the exact same amount from both sides of an equation. Now we can combine the $2x$ and $28x$ on the LHS to get $30x$, and the $-28x$ and $28x$ on the RHS to get 0.
3. We now have $30x+10 = -14$.
4. Now we'll subtract 10 from both sides, like this: $30x+10-10 = -14-10$.
5. This will give us $30x = -24$.
6. Now, we'll divide both sides by 30 to get: $x = -24/30$.

So this is our answer. A value of x of $\frac{-24}{30}$ or -0.8 will make $2(x+5)$ exactly equal to $14(-2x-1)$. Let's test it by putting -0.8 for x into both the LHS and RHS of our equation.

- $2(x+5)$ becomes $2(-0.8+5)$, $2(4.2)$ or 8.4.
- $14(-2x-1)$ becomes $14(-2(-0.8)-1)$ or $14(1.6-1)$ or $14(0.6)$. Now, 14×0.6 is 8.4

So, in solving the equation $2(x+5) = 14(-2x-1)$, we got that the special value of $x = -0.8$ is what makes the equation true.

This "hand method" of solving equations will dominate much of algebra, and indeed any math you take all the way into college. The trouble is that all of this effort will promptly become useless around your junior year in college. Why? Well, any sort of mathematics you do for your senior project in college, or any sort of analysis in graduate school (or in a job someday) will always result in an equation that you cannot solve by hand. “Real world” equations are just too complicated to solve using the techniques you learned from grade 8 to your college degree. In these years, you go through many years of specially “cooked up equations” that can be solved by hand, for the sake of letting you practice this skill you’ll never use again.

The solutions to equations are still important, but we think you should just solve them on a computer, get the answer, then work with the answer, in what it means, and how it will help you to advance some larger goal of yours. Once again, computers are very good at solving equations.

Here we have a function called `solve` that takes an equation, and a variable to solve for, and will solve the equation for you. Here’s code to solve the one above

```
result=solve("2(x+5)=14(-2x-1)","x")
print("The result is:");
print(result)
```

where you’ll see the answer of $-4/5$ displayed, which is -0.8. You can test this with code like this:

```
x=-0.8
print("Is",2*(x+5),"equal to",14*(-2*x-1),"?")
```

3.5.1 Remarks on solving equations

So we are a bit torn. We think equations should be solved on a computer, not by hand. The emphasis of lessons on equations should be in *forming an equation* that represents some problem you are working on. Once the equation is formed, you should get a quick computer solution. Your effort should then resume on *interpreting and using* the solution. We think too much time is spend on the *mechanics of solving equations* and more effort should be put into *using the solutions to equations*.

The problem here is that if we don't spend time solving equations, a typical year of math would be much shorter, since so many lessons are dedicated to the mechanics of solving equations. We'd have to think of others things to cover in our lessons.

3.5.2 Searching for solutions to equations

Let's take a look then at how an equation is solved by the computer. Let's take the equation

$$5x + 2 = 3(x + 10) \tag{3.3}$$

and try to solve it using the computer. In other words, what value of x would make $5x + 2$ equal to $3(x + 10)$?

To begin, let's guess that the solution is somewhere between $x = -100$ and $x = 100$. We are not basing this on anything other than a hunch. The numbers in the equation are sort of "normal" to us: a 4, a 3 and a couple of 2s, so we can't think of a reason why a solution would be out there in the thousands or millions.

So let's do this then: let's *search* for a solution in the range between $x = -100$ and $x = 100$. We can use a for-loop to do this, perhaps something like `for x=-100,100 do`. For each value of x that the for-loop delivers, we'll see if it possibly makes the LHS equal to the RHS. This is how we'll test if it solves the equation, because this is what an equation means: the LHS has to be equal to the RHS. So for each x, let's just evaluate the LHS and the RHS and see if they're equal. How about some code like this?

```
for x=-100,100 do
 lhs = 5*x+2
 rhs = 3*(x+10)
 print("x=",x," is",lhs,"=",rhs,"?")
end
```

If we run this code, about 200 lines will be displayed that we need to inspect. If we do so, low and behold we see that at $x = 14$, the LHS indeed equals and RHS! So we have found a solution to the equation. Let's check it by putting $x = 14$ into $5x + 2$ and into $3(x + 10)$:

- When $x = 14$, $5x + 2$ is $5(14 + 2)$ or $5 \cdot 16$ or 72.
- When $x = 14$, $3(x + 10)$ is $3(14 + 10)$ or $3 \cdot 24$ or 72.

So indeed $x = 14$ is the solution to the equation $5x + 2 = 3(x + 10)$. Do we really need to inspect so many lines each time? No, because we can use an `if` statement to check if the LHS equals the RHS, and only print the line where the LHS equals the RHS like this:

```
for x=-100,100 do
 lhs = 5*x+2
 rhs = 3*(x+10)
 if lhs == rhs then
  print("x=",x," is",lhs,"=",rhs,"?")
 end
end
```

Let's try another equation, like $x(x - 7) = -12$:

```
for x=-100,100 do
 lhs = x*(x-7)
 rhs = -12
 if lhs == rhs then
  print("x=",x," is",lhs,"=",rhs,"?")
 end
end
```

This code reveals TWO solutions, at $x = 3$ and $x = 4$. Indeed putting $x = 3$ into $x(x - 7)$ gives $3(3 - 7)$ or -12, as does $x = 4$, with $4(-4 - 7)$ which is also -12. So, there are two special values of x that makes $x(x - 7) = -12$ hold.

Let's try another. How about $4(2 - x) = 3(2 + x)$, with this code:

```
for x=-100,100 do
```

```
 lhs = 4*(2-x)
 rhs = 3*(2+x)
 if lhs == rhs then
  print("x=",x," is",lhs,"=",rhs,"?")
 end
end
```

When this is run, the output screen come up blank. Does this mean there's no solution? Or maybe the solution is outside of -100 to 100? If we expand the range of the for-loop to -1000 to 1000, still no solution appears. What could the problem be? Let's remove the `if` and go back to inspecting rows of numbers, like this:

```
for x=-100,100 do
 lhs = 4*(2-x)
 rhs = 3*(2+x)
 print("x=",x," is",lhs,"=",rhs,"?")
end
```

About the closest we get is for x at -1, 0 and 1. At $x = -1$, the LHS is 12 and the RHS is 3. At $x = 0$ they are 8 and 6 and at $x = 1$ they are 4 and 9. Everything else seems larger, which is our clue how to proceed. The "largeness" of the LHS vs. the RHS. Why don't we tweak our code to print *the difference* between the LHS and RHS. If the equation is to be solved, the difference should be 0 (meaning the LHS is equal to the RHS). Let's use this code:

```
for x=-100,100 do
 lhs = 4*(2-x)
 rhs = 3*(2+x)
 d = lhs - rhs
 print("x=",x,"d=",d)
end
```

In out output, remember you are looking for a d of 0. It doesn't exist, but indeed the smallest values of d occur at $x = -1, 0$, and 1 as we noticed before. Still no $d = 0$ though.

If we look carefully we see that d is 2 when $x = 0$ and -5 when $x = 1$. So the difference went from -5 (meaning the RHS was 5 greater than the LHS) to 1 (meaning the LHS was 1 greater than the RHS) between $x = 0$ and 1. So between $x = 0$ and 1 The RHS goes from being 5 greater than the LHS, to 1 less than the LHS. Perhaps this means that in transitioning between being larger the being smaller than the LHS, the RHS was equal to the LHS somewhere in between?

Indeed this is our issue. The solutions to equation do not have to be integers. Recall our for-loop is only looking at numbers like $-100, -99, -98, ... -1, 0, 1, 2...98, 99, 100$. In this case, a solution looks like it exists between $x = 0$ and $x = 1$, for our for-loop is skipping it! How can we fix this? Let's first make our search space smaller, perhaps from $x = -5$ to 5. Then, let's have the for-loop count in smaller increments, perhaps in an increment of 0.1. Take a look at this code:

```
for x=-5,5,0.1 do
 lhs = 4*(2-x)
 rhs = 3*(2+x)
 d = lhs - rhs
 print("x=",x,"d=",d)
end
```

Inspecting the output, we see that at $x = 0.3$, d, or the difference between the LHS and RHS has become small, at -0.09. Let's further reduce our search space, to go from -2 to 2 and go in steps of 0.01, like this

```
for x=-2,2,0.01 do
 lhs = 4*(2-x)
 rhs = 3*(2+x)
 d = lhs - rhs
 print("x=",x,"d=",d)
end
```

Here we find that at $x = 0.29$, d drops to -0.03. This means the RHS is only 0.03 larger than the LHS. Let's see:

- When $x = 0.29$, $4(2 - x)$ is 6.84.
- When $x = 0.29$, $3(2 + x)$ is 6.87.

Not bad! We can sort of conclude that the solution to this equation is $x = 0.29$. The actual answer is $x = 2/7$ which is 0.285714, so 0.29 is pretty close. One could hone in on this solution by continually decreasing the search space (to minimize the number of output lines we have to inspect), and by decreasing the step size. If we go to 0.001, we get $d = -0.002$ at $x = 0.285$.

Can we automate this some more? Yes, by specifying the precision of the solution we desire. Suppose we wanted a solution to within 0.01, let's set $p = 0.01$ and use this code:

```
p=0.001
step=p/10
for x=-2,2,step do
 lhs = 4*(2-x)
 rhs = 3*(2+x)
 d = math.abs(lhs - rhs)
 if d < p then
  print("x=",x,"d=",d)
 end
end
```

Here we have a few modifications. First, we set $p = 0.01$ our desired precision. Second, we figure if we want to achieve a precision of p in our solution, we should probably search in steps of something smaller than p, here we choose a 10th of p as our step size, as in `step=p/10`. The for-loop proceeds as normal, and we have a change in how we compute d. Here we find not d as in `lhs-rhs` as before but we use `d=math.abs(lhs-rhs)`, which computes the difference between the LHS and RHS, but now gives us the *absolute value* of this difference. Why? In looking for our solution, to the precision we want, we don't care if the LHS is larger than the RHS, or the RHS is larger than the LHS. We just want the two to agree to within the precision we want. If we run the above code, we'll see that $x = 0.28569999999978$, with d being 0.0001000.

So we've developed a bit of code that seems to solve equations for us, given that we can at least guess the range in which the solution exists (-2..2 currently), and the precision of the answer we want. We'll note that the code here is horribly inefficient. A so called "exhaustive search" (that is, looking at all numbers from -2 to 2 in steps of 0.001) is really a waste of the computers power. Why? Because there are more intelligent ways of proceeding with such a solution search and more clues to exploit that a solution is nearby, but we won't worry about this here. (For those interested, we'll defer to "Newton's Bisection Method" as the next step.)

The equations you'll encounter for many years to come are easy for the computer to deal with, and your computer is incredibly powerful. We'll sacrifice some computational power in our brute force search, as we enlighten ourselves on the issues of solving equations using a computer, and not by hand.

Chapter 4

Algebra and problem solving

In Chapter 3, *forming an equation* and *using its solutions* were thought to be more important than the mechanics (or steps and techniques) of actually solving the equation. Let's do some of this "equation forming" now, and see how we can use the computer to get quick solutions. In this chapter, we'll work on forming equations, as translated from short descriptions. Then, we'll take it all in a different direction: we'll also translate them into code, for a deeper understanding.

In order to get going, you have to be able to read facts about a system (in plain language), and then translate these facts into a mathematical expression. Then, the expression should be put into your code for testing and observations.

4.1 Warm-ups

The goal here to read the relatively plain statements, form an expression, and get some code running that reflects on the expression.

Let's start with "The sum of a and b." Mathematically, this means $a + b$. In code it would be

```
a=5
b=10
sum = a + b
print(sum)
```

where you can verify $a + b$ is the sum of a and b. Adapt the code for "The product of L and W."

Now to move on, how about "21 decreased by C?" In math this would be $21 - C$, where you can see that you start with 21, and it get decreased (or reduced, or lowered, etc.) by what's in variable C. Here's some code:

```
C=3
result = 21 - C
print(result)
```

which will print 18, which is "21 decreased by C" or "21 decreased by 3." This all means $21 - 3$ of course.

You can do more testing of this idea with a for-loop, as in:

```
for C=1,10 do
 result = 21-C
 print(C,result)
end
```

which will print two columns of numbers: a bunch of values for C in the first column, followed by the result of 21 lowered by the C in the second column. Do the result makes sense? Do you see a bunch of C's in the first column, then 21 reduced by C in the second column?

Here are some more examples:

- "Two less than x?" Here's some code:

```
for x=1000,1020 do
 twoless = x-2
 print(x,twoless)
```

```
end
```

Is the second column two less than the first column? See the equation? Here `twoless = x -2` where we've used a descriptive name like "twoless" to hold the result.

- "Three times the square of t, plus 10."

```
for t=1,10 do
 x = 3*t^2+10
 print(t,x)
end
```

When you square a number in the first column, multiply the result by 3, then add 10, do you get the number in the second column? Note our result is in variable z.

- "32 added to 1.8 times a number."

```
for x=20,35 do
 f = 1.8*x + 32
 print("x=",x,"f=",f)
end
```

Take a number from the first column, and multiply it by 1.8, then add 32 to the result. Do you get the number in the corresponding second column?

You might not think much about the purpose of these examples, but you are using algebra. You took a simple need (the statement), translated it into an equation, then used code to explore a wide range of results the equation generated. Note each left-column was just a straight and sequential number, counted to via a for-loop. These are simple to produce. The transformation of the simple numbers into something else via an equation is at the heart of algebra (maybe even all of mathematics).

The algebra generated a new result from these numbers: adding together two numbers that might be dollar amount in a financial transaction, finding the amount of gallons in a 21 gallon tank of water, after so many gallons have been used, finding how many people might be left in a crowded concert hall when 2 people leave, finding the position of an object with an acceleration of 6 ($3 = \frac{1}{2} \times 6$) with an initial position of 10, or lastly converting Celsius into Fahrenheit.

Remember also with the algebra and the code, you can *explore* the relationships. Try changing the range of the for-loops, or the constants, as in make the "-2" into a "-50" or the "21-C" into a "1000-5.8*C." What do you get, and what does it mean?

4.2 More "real sounding" situations

Salary Calculator

Let's use algebra and write some code that would read in a value for a person's salary, then print twice the salary. Let s=the salary, then $2s$ would be twice the salary. Something like this would do it

```
print("Salary amount:")
s = input()
twice = 2 * s
print(twice)
```

Cost of stamps

If you know the cost of one stamp, how much would 5 stamps cost? 6? 10? $1,745$? Here's some code, assuming one stamp costs $c = 0.50$, here's code to compute the cost of n stamps, which is $n \times c$.

```
c = 0.50
n = 20
cost = n * c
print(cost)
```

The code can be extended to allow one to type in the cost of a stamp, then how many they'd like to buy.

```
print("Cost of one stamp?")
c = input()
print("How many stamps?")
n = input()
cost = n * one
print(cost)
```

Ages of two people

Suppose Mary is 10 years older than Nancy. If you know Nancy's age, compute how old Mary is. If M is Mary's age, and N is Nancy's, the equation would be $M = N + 10$. You can run this for many possible ages of Nancy with a for-loop, as in:

```
for N=20,50 do
 M = N + 10
  print("Mary=",M,"Nancy=",N)
end
```

Sum of two numbers

The sum of two number is 10. If the numbers are x and y, then $x + y = 10$. There are a few ways of exploring this. Here's a way with a for-loop, knowing that is one number is x, the other number must be $10 - x$. You can test this in your mind. Let $x = 4$ so $10 - x = 6$, and $4 + 6 = 10$. Try some others.

```
for x=0,10 do
  y = 10 - x
  print(x,y)
end
```

Look at the two columns. Do they always add to 10? You can also run it with random numbers discussed in the previous chapter, examining many more numbers

```
for i=1,100 do
 x = math.random(0,10)
 y = 10 - x
 print(x,y)
end
```

You can even throw decimals into the mix like this

```
for i=1,100 do
 x = 10*math.random()
 y = 10 - x
 print(x,y)
end
```

The outputs looks very messy now with all of the decimals, but do you see how the numbers in each row add to 10?

4.3 Investigating numbers

Sum of consecutive integers

How about the sum of "three consecutive integers?" If an integer is in variable n, what would the next integer be? How about $n + 1$? What about the next integer after that? How about $n + 1$ but $+1$ again, or $n + 1 + 1$ or $n + 2$? So, if the three consecutive integers are a, b, and c, we'd set $a = n$, $b = n + 1$ and $c = n + 2$, the sum would be $s = a + b + c$, here are some sums

```
for n=1,20 do
 a = n
 b = n +1
 c = n + 2
 s = a+b+c
 print(a,"+",b,"+",c,"=",s)
end
```

where you'll see the sum of many combinations of "three consecutive integers." You can really explore a bunch of consecutive integers with random numbers, like this:

```
for i=1,20 do
 n = math.random(1,10000)
 a = n
 b = n +1
 c = n + 2
 s = a+b+c
 print(a,"+",b,"+",c,"=",s)
end
```

Sum of consecutive even integers

How about finding the sum of "three consecutive even integers?" If an integer is in variable n, how can you alway be sure its even? What if you multiply it by 2? Try this

```
for n=1,20 do
 print(n,2*n)
end
```

where the first column is n and the second $2n$. Note that no matter if n is even or odd, the second column (or $2n$) is always even. Here's another test with random numbers:

```
for i=1,20 do
 n = math.random(1,100)
 print(n,2*n)
end
```

where again, you'll note that the second column always shows an even number. So we know how to generate even numbers—just multiply *any* integer by 2. What would the next even number be? Well after every even number, is an odd number, right? Try this code,

```
for i=1,20 do
 n = math.random(1,100)
 print(n,2*n,2*n+1)
end
```

where you'll note the numbers in the third column are always odd. What about this?

```
for i=1,20 do
 n = math.random(1,100)
 print(n,2*n,2*n+1,2*n+2)
end
```

Here you'll see that the second column ($2n$) and the fourth column ($2n + 2$) are consecutive even numbers. Three consecutive even numbers can be found via

```
for i=1,20 do
 n = math.random(1,100)
 print(2*n,2*n+2,2*n+4)
end
```

So, three consecutive even integers, a, b, and c would be: $a = 2n$, $b = 2n + 2$ and $c = 2n + 4$, given that we started with n as *any* number at all. Summing them would work something like this

```
for n=1,20 do
 a = 2*n
 b = a + 2
 c = a + 4
 s = a+b+c
 print(a,b,c,s)
```

```
end
```

You should try this with n as a random number too.

Sum of consecutive odd integers

How about the sum of "three consecutive odd integers?" In the last section, we saw that even integers were found by taking any integer, n, and multiplying it by 2. Where are the odd integers? Well, suppose we have an even integer, like 12. From 12, aren't 11 and 13 odd? So if we're at an even integer we can either go one lower or one higher to find an odd integer.

Would you agree then that odd integers can be found via $2n-1$ or $2n+1$? These mean to go to an even integer ($2n$), then look one lower as in $2n-1$ or one higher as in $2n+1$. These will always be odd, as you can verify here:

```
for n=1,20 do
 a=2*n+1
 print(n,a)
end
```

or here

```
for n=1,20 do
 a=2*n-1
 print(n,a)
end
```

where you'll note that the right column is always odd. Here is the one with random numbers

```
for i=1,20 do
 n = math.random(1,1000)
 a = 2*n+1
 print(a)
end
```

where you'll note a list of odd numbers. Now, where are the consecutive odd numbers? Two away again. For example, if you're at 13, the next two odd integers are at 15 and 17. Or lower, 11 and 9. So here's some code that will print three consecutive odd numbers:

```
for i=1,20 do
 n = math.random(1,1000)
 a = 2*n+1
 b = a + 2
 c = b + 2
 print(a,b,c)
end
```

We'll leave it to you know to figure out how compute and print the sum of a, b, and c, and finish up finding the "sum of 3 consecutive odd integers."

4.4 Changing Words into Equations

We hope you get the general idea of how statements (like "the sum odd integers") can lead to equations and then to code. The code you wrote was then run and allowed you to study the output (the results) the equation generated. This is a very powerful process, hard to do with pencil and paper only. With the output, you could then look for patterns or trends in numbers generated by the equations, and come to some conclusions. You investigated the various scenarios using your code, and we hope this gave you a feeling for the how math generated by the statements, drove your code, which brought it all to life.

In this section, we'll do more of the same thing, but we'll form a specific equation that we'll solve to get a unique solution.

Fifteen plus twice an unknown number is 37

Let's call the unknown number x. Twice the unknown is $2x$ and adding 15 to it gives $2x + 15$. The statement says this should all equal to 37, as in $2x + 15 = 37$. If you remember the example from the last section, that given a number x, $2x$ is even, this might be saying "what even number plus 15 gives 37?" Coding can help you decipher this if needed. Here's a bunch of numbers from 1 to 30 in the left column, then doubled (or twice the number) in the right column:

```
for x=1,30 do
  print(x,2*x)
end
```

Now you can add 15 to each with a quick edit to your code, as in

```
for x=1,30 do
  print(x,2*x+15)
end
```

You can see in the left column, that when x is 11, the right column (or $2x + 15$) is 37, which is what the problem is asking for ($x = 11$ is the answer to this problem).

But don't forget the power of an equation, like $2x + 15 = 37$. You don't have to search through numbers with a for-loop. You can solve it by hand (pencil and paper), or you can let the computer solve it like this:

```
result = solve("2x+15=37","x")
print(result)
```

which confirms that $x = 11$ solves the equation. Manually, you can see that $2 \cdot 11 + 15 = 22 + 15 = 37$, so it all works out.

Twice the sum of 6 and an unknown number is equal to 20

Here “the sum of 6 and an unknown number” is $6+x$. Twice this is $2\cdot(6+x)$. Similar to the last example, let’s run some numbers in a for-loop through the equation, as in

```
for x=1,30 do
   print(x,2*(6+x))
end
```

where you can see when x is 4 (right column of the output), $2(6+x)$ is 20. So indeed $x=4$ is the answer to this problem. You can also find this from

```
result = solve("2(6+x)=20","x")
print(result)
```

The sum of four consecutive integers is 106

What are the integers? If n is the first integer, $n+1$, $n+2$, and $n+3$ are the other three consecutive ones. Here’s some code to find some sums

```
for n=0,100 do
 a=n
 b=n+1
 c=n+2
 d=n+3
 s = a+b+c+d
 print(a,b,c,d,s)
end
```

where you can see the 106 in the 5th column after the 25, 26, 27, and 28, which are the four consecutive integers that add to 106. The full equation is $n+(n+1)+(n+2)+(n+3)=106$, which can be solved for directly with

```
result = solve("n+(n+1)+(n+2)+(n+3)=106","n")
print(result)
```

You need 730 points in your math class to get an A

The final exam is worth 200 points, and you have 545 points going into the final. What range of scores will give you an A in the class?

No matter what, you'll earn between 0 and 200 points on the final, and your final grade will be $545 + x$, where x is your grade on the final exam. Here's some code to find your final grade, given the points possible on the final:

```
for x=0,200 do
 grade = 545 + x
 print(x,grade)
end
```

You can inspect the output and see that you'll get at least 730 points in the class if you score 185 points on the final. You can save yourself some work inspecting the numbers with an `if` statement to check if your grade is greater than or equal to 730 like this

```
for x=0,200 do
 grade = 545 + x
 if grade >= 730 then
  print("You'll need ",x,"points on the final")
 end
end
```

You spent $2.36 for 22 pieces of candy

If you bought only $0.10 candy and $0.12 candy, how many of each kind did you buy? Let's suppose you bought x number of the $0.10 candy. Since you bought 22 total pieces, you must also have purchased $22 - x$ of the $0.12 candy. So let's try a full-out *computational approach* to this problem. Let's pick the number of $0.10 candy at random, so we'll do `x=math.random(0,22)` assuming we'll choose between 0 and 22 pieces of candy. The the number of

\$0.12 candy will be $y = 22 - x$. Some code to tell us the number of possible pieces will be

```
for i=1,200 do
 x = math.random(1,22)
 y = 22 - x
 print(x,y)
 end
```

Now what about the cost? Well, for the \$0.10 candy, we'll spend $0.1x$ and for the \$0.12 candy, we'll spend $0.12y$. This makes the total of $t = 0.1x + 0.12y$. Here's some code that will compute and show us the total cost too:

```
for i=1,200 do
 x = math.random(1,22)
 y = 22 - x
 t = 0.1*x + 0.12*y
 print(x,y,t)
 end
```

If you scan your columns, you'll likely see a total cost of \$2.36. We say *likely* because the computer is choosing at random—maybe it didn't choose the right combination. If not, run it again, or make it choose more samples (increase the 200 in the for-loop).

Can we use an `if` statement to check for our answer? Sort of. Try this (where the `T` stands for "total."

```
for i=1,200 do
 x = math.random(1,22)
 y = 22 - x
 T = 0.1*x + 0.12*y
 if T== 2.36 then
  print(x,y,T)
 end
end
```

The output will be blank. Why? Does the computer continually *not find* the right combination? No, in this case, it has to do with how the computer handles decimals. The amount computed in `T` won't always match 2.36 exactly. It might come out to be 2.359 or 2.361, or something *really close* to 2.36, but with `==`, it has to be exact, and the close numbers like 2.359 or 2.361 aren't exactly 2.36, so the `if` statement never triggers.

So what do we do? We look at *how close T* comes to 2.36, which means we look at the difference between T and 2.36, or T-2.36. But since this can be positive or negative, depending on if T is larger or smaller than 2.36, let's look at $|T - 2.36|$ instead. If this "closeness" is less than some small number (like 0.001), then we'll assume we found our answer. Try this, and know that `math.abs()` is how you take absolute values

```
for i=1,200 do
 x = math.random(1,22)
 y = 22 - x
 T = 0.1*x + 0.12*y
 if math.abs(T- 2.36) < 0.001 then
  print(x,y,T)
 end
end
```

where you'll see the computer indeed finds our answer every time.

If you don't like the random number approach, a simple for-loop visiting all possible counts for x and y would work too, like this

```
for x=0,22 do
 y = 22 - x
 T = 0.1*x + 0.12*y
 if math.abs(T- 2.36) < 0.001 then
  print(x,y,T)
 end
end
```

You have \$4.80 in nickels, dimes and quarters

If you have 3 more dimes than quarters, and 3 times as many nickels as quarters, how many of each coin do you have? Let's call d=times, q=quarters, and n=nickels. Translating the text:

- "3 more dimes than quarters" would be $d = q + 3$.
- "3 times as many nickels as quarters" would be $n = 3q$.
- "\$4.80 in nickels, dimes, and quarters" would be how you sum the amount of money you have. To find this, you multiply the dollar value of each coin by the number of coins you have. For examples, if $q = 3$, you have $3 \cdot \$.25 = \$0.75$ in quarters. So the dollar amount of nickels would be $0.05n$. That of dimes would be $0.10d$ and that for quarters, $0.25q$. The \$4.80 comes from adding all of this to get $0.05n + 0.10d + 0.25q = 4.80$.

So how do we find how many of each coin we have? Search for it with a for-loop. Notice that the number of dimes and nickels are all based around the number of quarters, so if we search based on q, we should be able to find our answer. We'll sort of guess with common sense how many coins would be in our pocket. We'll say up to 10 quarters at most.

```
for q=0,10 do
 n=3*q
 d = q + 3
 T = 0.05*n + 0.10*d + 0.25*q
 print(T)
 if math.abs(T-4.80) < 0.001 then
  print("Answer: q=",q,"n=",n,"d=",d)
 end
end
```

Would random numbers work here? Let's try looking through 30 combinations of quarter numbers from 0 to 20:

```
for i=1,30 do
```

```
    q = math.random(0,20)
    n=3*q
    d = q + 3
    T = 0.05*n + 0.10*d + 0.25*q
    print(T)
    if math.abs(T-4.80) < 0.01 then
        print("Answer: q=",q,"n=",n,"d=",d)
    end
end
```

where you'll see your answer will come up here as well.

You can try something too, relaxing the requirement that $d = q + 3$ and $n = 3q$, to effectively look for *any combination* of nickels, dimes, and quarters, that will add up to \$4.80, like this

```
for i=1,100 do
    q = math.random(0,20)
    n = math.random(0,20)
    d  = math.random(0,20)
    T = 0.05*n + 0.10*d + 0.25*q
    print(T)
    if math.abs(T-4.80) < 0.01 then
        print("Answer: q=",q,"n=",n,"d=",d)
    end
end
```

Here you can see that indeed there are different combinations of nickels, dimes, and quarters, that will yield \$4.80. We find it amusing that if you choose an amount like \$4.40, you'll never get an answer, since this amount is impossible to reach, with just nickels, dimes, and quarters.

An item cost $175 and is discounted by 15%

What is the cost of item? Note that a percentage is never a number to be used in a calculation. Being on a scale from 0 to 100 just makes it easier for humans to reconcile in a conversation. You always have to divide a percentage by 100 first. To test this, you probably know that 50% of 1000 is 500, right? You wouldn't get this by multiplying 50% $\times$ 1000, which equals 50,000. No, you'd first divide 50 by 100 to get 0.5, and $0.5 \cdot 1000 = 500$, which is what we thought.

So, to start, find 15% of $175, which is $0.15 \cdot 175 = 26.25$. So the item will be sold for $175 – $26.25 = $148.75. Code to handle this would look like

```
c = 175
p = 15
off = p/100 * c
sell = c - off
print("It'll sell for",sell)
```

You can investigate all kinds of discount percentages with a for-loop like this:

```
c = 175
for p=0,100,10 do
    print("Percent off:",p)
    off = p/100 * c
    sell = c - off
    print("It'll sell for",sell)
end
```

where note the `p/100` term to make the percent into a number you can calculate with. The "killer app" here is to make it all automated like this:

```
print("Item cost:")
c = input()
for p=0,100,10 do
    print("Percent off:",p)
    off = p/100 * c
    sell = c - off
    print("It'll sell for",sell)
end
```

where you'll see how nice a 100% discount is!

Marking up an item by some percentage can be handled by adding the percentage of the original cost, instead of subtracting it, as in

```
c = 175
p = 15
up = p/100 * c
sell = c + up
print("It'll sell for",sell)
```

Investigating a range of markups is possible too, like this

```
print("Item cost:")
c = input()
for p=0,100,10 do
    print("Markup:",p)
    up = p/100 * c
    sell = c + up
    print("It'll sell for",sell)
end
```

4.5 Where to go from here?

Here, like most algebra books, one can only present so many worked examples of word problems. Study guides at your local bookstore are filled with problems like those above. Indeed, translating words into equations is hard, and is at core of most STEM classes, through college and into graduate school. You shouldn't expect to "get how to do them" right away. And of course, we are obliged now to say "practice more of these problems, and you'll get better at them."

This issue of translating text based scenarios into equations is likely the most important skill you will develop as you learn algebra. But is also the most difficult, and also a skill that we think is less and less useful in the 21st century. Translating them into computer solutions might not be as hard, and is more useful in the long run.

In terms of paper-and-pencil algebra, let's reconsider the coin problem above. It would be solved like this:

- Let q=quarters, n=nickels, d=dimes
- Reading the text and translating, we get $n = 3q$ (this is hard step)
- Reading the text and translating, we get $d = q + 3$ (this is hard step)
- Reading the text and translating, we get $4.80 = 0.05n + 0.10d + 0.25q$ (this is hard step)
- Substituting in for n and d we get $4.80 = 0.05(3q) + 0.1(q+3) + 0.25q$. This is one equation that can be solved for q to get $q = 9$. (This is a big aspect of traditional algebra we think should be transitioned into code, not pencil and paper work.)
- Now that we know q (the core variable), we can get $n = 3q$ or 27, and $d = q + 3$ or 12.
- You can also get the number of coins without doing the substitution, by just putting the equations into the computer and telling it to solve them, like this:

```
result=solve("n=3*q,d=q+3,0.05*n+0.1*d+0.25*q=4.80","n,q,d")
print("The result is:");
print(result)
```

  We prefer this approach, if now promise to study the solution a bit more and be sure you know what it all means.

Suppose you couldn't figure out the text-to-equation translation ($n = 3q$ and $d = q + 3$) steps. Armed with code and a little creativity, you could still make some progress on solving this problem. How? Using the random number approach to this problem:

```
for i=1,100 do
     q = math.random(0,20)
     n = math.random(0,20)
     d  = math.random(0,20)
     T = 0.05*n + 0.10*d + 0.25*q
     print(T)
     if math.abs(T-4.80) < 0.01 then
          print("Answer: q=",q,"n=",n,"d=",d)
     end
end
```

This code will find combinations of quarters, nickels, and dimes and total in value of $4.80. You could then sift through these combinations, until you find one that also has "three times as many nickels as quarters," and "3 more dimes than quarters." Then you'd have your solution. You could also do a more exhaustive approach by iterating for-loops through many combinations of coins like this

```
for q=0,20 do
   for n=0,20 do
      for d=0,20 do
         T = 0.05*n + 0.10*d + 0.25*q
         print(T)
         if math.abs(T-4.80) < 0.01 then
              print("Answer: q=",q,"n=",n,"d=",d)
         end
      end
   end
end
```

Since computers are so powerful and prevalent, such attacks on solving problems are a "computational way" of thinking. After school, you won't be able to form equations anyway (the problems you'll work on are too complicated). Like what? How about these: Write some equations that model the stock market. Write some equations that you can solve for where a hurricane will hit. Form and solve an equation that will land a rocket on a barge in the middle of the ocean. Find x where x is the number of homeless you can get off of the streets for every y percent rise in a local tax.

So you might look at some more word problems in your algebra book. Thinking computationally, you might follow these steps:

1. Define variables needed in the problem.
2. Form some equations based on the information in the problem.
3. Code up a `for-loop` that iterates over core variable in the problem (for starting algebra, there will always be only one).
4. Use `print` to display the core variable, and any other variables that it might drive.
5. Carefully inspect the output and see if it makes sense, or if you see a potential solution lurking in the columns of numbers.
6. See how you might insert a `if` statement into the mix to stop the code, or highlight a solution.

So, see if you can find solutions to word problems computationally. Try to minimize your grief with equations and solving and maximize your trial and error on the computer. Use for-loops, use variables, use if-statements, dump numbers to the screen and sift through them. If you solve any this way, let us know. We'll happily put your work into a future edition of this book.

Welcome to computation with a computer!

Chapter 5

Polynomials

A polynomial is a mixture of constants (coefficients) multiplied by various powers of some variable, like x. Examples are $4x^2 + 3x$, $7y + 7y^2$, or $x^3 + 3x^2 + 3x + 1$. Algebra is loaded with lessons on learning how to manipulate polynomials, which are all ripe for a strong connection with programming, since computers are *very good* at working with polynomials.

In the years to come, the only real use of polynomials will be in physics, when you deal with accelerated motion, in which case $x = x_0 + v_0 t + \frac{1}{2}at^2$ will be a key polynomial. This tells you the position of an object (x), give that it started at some point x_0, has some speed (v), and some acceleration (a), at some time t in the future. There almost no other real use of polynomials, other than using them as exercises for other lessons in mathematical topics. Polynomials occasionally come up in curve fitting of data, but mostly the higher-order fit parameters have no real-world meaning.

5.1 Putting polynomials into the computer

With polynomials, there are a lot of numbers and exponents. On the computer, the exponent is represented as the caret symbol (^), so x^2 would be put in as `x^2`, and $x^3 + 3x^2 + 3x + 1$ would be put in as `x^3+3x^2+3x+1`.

Note also it is easy to display polynomials on the computer in a *very ugly* manner. Doing `x^3+3x^2+3x+1`, for example, is ugly. Codebymath.com has a mathematical typesetter built right in. So if you do `print("x^2+5x")` you'll get the ugly polynomials, where doing `print_math("x^2+5x")` will result in a proper looking polynomial. Try this

```
print("x^2+5x")
print_math("x^2+5x")
```

where you'll note that the lower polynomial in the output screen looks much better.

5.2 Adding and subtracting polynomials

For doing calculations with polynomials, use a function called `algebra()`. It knows how to do many operations with polynomials.

Suppose you needed to subtract $(3x^2 + 5x - 4)$ from $(2x + 5)$. You could write some code like this (where `r` stands for "result").

```
r = algebra("(3x^2+5x-4)-(2x+5)")
print_math(r)
```

You could also try $(5x^3y^2 - 3x^2y^2 + 4xy^3) + (4x^2y^2 - 2xy^2) + (-7x^3y^2 + 6xy^2 - 3xy^3)$. This could be done with:

```
r = algebra("(5x^3y^2-3x^2y^2+4xy^3)+(4x^2y^2-2xy^2)+(-7x^3y^2+6xy^2-3xy^3)")
print_math(r)
```

Here's another example. Your instructions are to "simplify:" $(2m^2 - m + 4) + (3m^2 + m - 5)$. As code this would be done with

```
r = algebra("(2m^2-m+4)+(3m^2+m-5)")
print_math(r)
```

We are sorry we don't have any more instructions for you here other than:

1. Learn how to put polynomials into a computer, with the coefficients and exponents, and
2. learn how to typeset them in some "non-ugly" manner.

Try coding up some examples from your algebra book, even if just from worked examples.. It's fun to see the computer spit out the answer that matches the one in your book.

5.2.1 Testing the results

With the computer taking a lot of the work off of you by adding polynomials for you, we can investigate the results a bit (to verify them). In the case of doing $(3x^2 + 5x + 4) - (2x + 5)$, we got $-1 + 3x + 3x^2$. Using a for-loop, we can test if these are indeed equal, like this:

```
for x = 0,20 do
 p1 = (3*x^2+5*x+4)-(2*x+5)
 p2 = -1 + 3*x + 3*x^2
 d = p1-p2
 print(p1,p2,d)
end
```

where we evaluate each polynomial at a given x and place the results into `p1` and `p2`. (Here the "p" stands for "polynomial," and yes, it's OK to have numbers in your variable names, just as long as the variable names don't *start* with a number.) Then we *subtract* `p1` from `p2`. If they are the same, we should always get zero for their difference. You can also try dividing `p1` by `p2`. Being careful that `p2` is not zero, what should the quotient always be if the two polynomials are equal?

5.3 Multiplying Polynomials

Multiplying polynomials on the computer is similar to that with addition and subtraction—you just put the problem into the computer. Typically you'll start with "monomials" or multiplying single groups of constants and variables, like $(3x^3y)(2xy)(-5xy^2)$. These are exercises in multiplying all constants, then combining similar variables by adding their exponents, as can be see here:

```
r = algebra("(3x^3y)(2xy)(-5xy^2)")
print_math(r)
```

You can also make use of the distributive property in problems like this one: $(3x^3y)(2xy-5xy^2)$ which is $3x^3y \cdot 2xy - 3x^3y \cdot 5xy^2$, which can be found from:

```
r = algebra("(3x^3y)(2xy-5xy^2)")
print_math(r)
```

5.3.1 A project on polynomial multiplication

The classic multiplication problem is that which describes the F.O.I.L. method, in doing a problem like: $(x+1)(x-2)$, where you multiply the **F**irst terms $(x \cdot x)$, then to this add the product of the **O**uter terms $(1 \cdot -2)$. To these, add the product of the inner terms or $(1 \cdot x)$. Lastly, add the **L**ast terms or $1 \cdot -2$. Collecting all of this, we'll get $x^2 - 2 + x - 2$, or $x^2 - x - 2$, which is what the computer would give you from this:

```
r = algebra("(x+1)(x-2)")
print_math(r)
```

Your ability to code can lead to a fun project in how to multiply polynomials: let's write our own multiplication tool, that even explains the steps to us. What you might have realized, is that to multiply polynomials, you multiply all terms in the first polynomial by all of the terms in the second polynomial, then add all of the resulting products together. You then simplify the addition, and you're done. Let's try to do this here.

First, we'll represent each polynomial as an array. An array in programming is just a comma separated list of values. Since a polynomial looks just like a list (of terms added together), this is a good fit. So if our first polynomial is $x+1$, we'll represent it like this: `p1 = {"x",1}`. We'll call the first polynomial `p1`. As for the second polynomial, we'll call is `p2` as in `p2 = {"x",-2}`. You need the double quotes around the "x" because you literally mean "x" and don't want the computer trying to calculate anything for it at this point. Now, let's use a for-loop to inspect our arrays, like this:

```
p1 = {"x",1}
for i=1,#p1 do
 print(p1[i])
end
```

Here the `#` in front of the `p1` finds the number of elements in the array (`p1` here, which in this case has two elements), so we can instruct the for-loop to count from 1..2 (or count over each element in the array).

As for the other polynomial, we'd have:

```
p2 = {"x",-2}
for i=1,#p2 do
 print(p2[i])
end
```

Each of these programs should show all of the terms of each polynomial. We can put this together into a single program like this:

```
p1 = {"x",1}
p2 = {"x",-2}

for i=1,#p1 do
   for j=1,#p2 do
```

```
    print("multiply",p1[i],"by",p2[j])
  end
end
```

We know multiplying polynomials involves multiplying each term in one, by each term in the other, then adding all of the products together. To visit all elements in each polynomial once, we have a for-loop inside of another for-loop. These are called “nested” for-loops, and they work like this.

Suppose is $i = 1$ from the outer for-loop. The inner one starts at $j = 1$, then $j = 2$, then it stops (because `#p2` is 2). The code then goes back to the outer for-loop, which causes i to go from 1 to 2. This once again starts up the j for-loop to visit $j = 1$ and $j = 2$ again. So you see, we had $i = 1$ and $j = 1$ and 2. Then we had $i = 2$ and $j = 1$ and 2. These are the needed combinations to visit each term in each array (or polynomial) once. Adding the `print` statement in the inner part of the inner loops gives us the directions we need to multiply to two polynomials.

We can take it a step further. If we construct the actual product to be taken, we can feed this to the `algebra` function so it can actually carry out the work. Consider this code:

```
p1 = {"x",1}
p2 = {"x",-2}

for i=1,#p1 do
  for j=1,#p2 do
    prod = p1[i] .. "*" .. p2[j]
    print(prod)
  end
end
```

Here we construct a string called `prod` which is the concatenation of the term from `p1` a “*” (for multiply) and the term from `p2`. Here “..” means concatenate, which itself means to tack letters on to the end of a string (so “he” .. “llo” would result in “hello”). You can see the products needed printed one by one. To actually carry out the products, we can feed `prod` to the `algebra()` function, like this:

```
p1 = {"x",1}
p2 = {"x",-2}

for i=1,#p1 do
   for j=1,#p2 do
      prod = p1[i] .. "*" .. p2[j]
      r = algebra(prod)
      print_math(r)
      print()
   end
end
```

This code will print out each term you need to add together, each on a line by itself.

We can take it to its final form. Let's collect all terms (with + signs in between), into a single string called `final`, like this

```
p1 = {"x",1}
p2 = {"x",-2}

final = ""
for i=1,#p1 do
   for j=1,#p2 do
      final = final .. " + "
      prod = p1[i] .. "*" .. p2[j]
      r = algebra(prod)
      final = final .. r
   end
end

print(final)
r = algebra(final)
```

```
print_math(r)
```

Here we can see the final collection of terms, which we can even run through the `algebra()` function for simplification.

So we've done it. We've written a polynomial multiplier, that shows us step by step, how to multiply two polynomials.

5.3.2 Use your code now

The neat part of coding is that you can actually use it now for other jobs. In this case, you can put in any two polynomials, by just changing the arrays for `p1` and `p2` and letting the code crunch through the multiplication process. Here's the same code above, that will multiply $x^2 + x + 5$ by $x^4 + 2x + 15$:

```
p1 = {"x^2","x",5}
p2 = {"x^4","2*x",15}

final = ""
for i=1,#p1 do
  for j=1,#p2 do
    final = final .. " + "
    prod = p1[i] .. "*" .. p2[j]
    r = algebra(prod)
    final = final .. r
  end
end

print(final)
r = algebra(final)
print_math(r)
```

5.4 Special Products

We feel obligated to tell you about some "special products," although they're really nothing different than what you've been doing above. It's just more multiplication of polynomials.

The first one is what you get when you multiply $(a+b)(a-b)$, in other words two binomials that have the same term in their first position (the "a" in this case), and the same term in the second position (the "b" in this case), and one of the b's is negative. If you run this

```
r=algebra("(a+b)(a-b)")
print_math(r)
```

you'll see that you get a^2-b^2. Run the code with $(x+1)(x-1)$, $(z+17)(z-17)$, $(3y+14)(3y-14)$ to see that you always get the first term squared minus the second term squared.

The second special product is what you get when you multiply $(a+b)(a+b)$, in other words two binomials that have the same term in their first position (the "a" in this case), and the same term in the second position (the "b" in this case), and both are positive. If you run this

```
r=algebra("(a+b)(a+b)")
print_math(r)
```

you'll see that you get the a^2 and b^2 like the above, but will also get the "cross term," $2ab$. Try this with $(x+1)(x+1)$, $(z+5)(z+5)$, and $(2y+8)(2y+8)$. You'll always get *three* terms in your answer: the first term squared, the second term squared, and that cross term that is twice the first term $\times$ the second term.

Again, in the computer mode, there is nothing special going on here. These are all just polynomials being multiplied, that give a product as a result.

5.5 Exponents

As you likely know, raising a polynomial to an exponent is the same as multiplying it by itself the number of times in the exponent. So doing something like $(x+1)^2$ is the same as the "special product" $(x+1)(x+1)$ from above. As for computation we like seeing how large polynomials grow when raising them to a large power, which is something you just wouldn't want to do by hand. Trying something like $(x+5)^{20}$ could be done with

```
r=algebra("(x+5)^20")
print(r)
```

where at this time you have to use `print` as `print_math` has some trouble displaying long results (depending on your browser—we're working on a fix for it). For fun, what's the coefficient of x^{34} in $(x-1)^{50}$?

Chapter 6

Factoring

6.1 Introduction

Factoring is a pivotal concept for a book (like this one) whose theme is about getting away from pencil and paper algebra. This is because factoring consumes a very large amount of time in a typical algebra class, and fuels many subsequent lessons.

It turns out that the job of factoring doesn't even have a well prescribed plan; there are no rules or strict methodology for successful factoring. So factoring really pushes the "Einstein mode" of work, as we discussed in the preface to this book. You just have to "see" how to factor somehow.

What is factoring? In concept, factoring involves making a simpler and equivalent version of a given number or expression. So it's a lot of work that results in producing something that is mathematically identical to what one started with in the first place.

With numbers, factoring is a plan (or lack thereof) to find numbers that when multiplied, give the original number itself. So if we "factor" 10, we'd write $1 \cdot 2 \cdot 5$, or declare that $1, 2$ and 5 are the factors of 10 (since $1 \times 2 \times 5 = 10$). Factoring 30 would give $1 \cdot 2 \cdot 3 \cdot 5$ (since $1 \times 2 \times 3 \times 5 = 30$). Unfortunately, in today's algebra, you also have to be careful when you declare factors, that each is prime, meaning each factor cannot itself have factors, or you'll "get it wrong." So you'd be "wrong" to write that $1, 3$, and 10 are factors of 30 even though you are right, but wrong according to your teacher, since your 10 can be written as $2 \cdot 5$.

Lastly, we'll say that there's no real *use* of factoring in classroom mathematics, other than to simplify mathematical expressions, and make boxed in, final answers as simple as possible. Factoring is used to solve equations, but there are some "we now have computers" issues with this, as you'll see in the next chapter.

As you saw in a past chapter, reducing fractions to lowest terms makes use of factoring. So if you had 10/30, you could write each number in terms of its factors or $\frac{1\cdot2\cdot5}{1\cdot2\cdot3\cdot5}$. Canceling all of the common factors gives $\frac{1}{3}$. Mostly in math, you'd be wrong to ever write 10/30 as a final answer anywhere since it can be reduced to 1/3, even though 10/30 is equal to 1/3. See how silly this is?

The issue with the computer is that the following two `print` statements print the same thing:

```
print(10/30)
print(1/3)
```

because $10/30 = 1/3$, so the computer isn't bothered by different forms of the same thing. So the computer seems to have it right!

6.2 With polynomials

How does this work with polynomials? Well, take $15x^3 + 9x$. It turns out if you stare at this for a while (and this is the basic plan for factoring, believe it or not), you may notice that you can divide an x out of both terms, like this: $x(15x^2 + 9)$. In other words, since both $15x^2$ and $9x$ have at least one x in them, you can divide (or factor) an x out of both. So, $15x^3 + 9x$ can be written as $x \cdot (5x^2 + 9)$. Now, if you keep staring at it, you might notice that 15 and 9 both have a 3 “in them” as factors. In other words, 15 and 9 both have 3 as one of their factors, so again, you can divide out a 3 as in $3x(5x^2 + 3)$, which means $3x \cdot (5x^2 + 3)$. So, just as $2 \times 3 = 6$, $3x \times (5x^2 + 3) = 15x^3 + 9x$. Thus you can conclude that $3x$ and $5x^2 + 3$ are factors of $15x^3 + 9x$. You can do the same thing for $12x + 8$, which is $4(3x + 2)$ and $2a^2b + 4ab^2$ which is $2ab(a + 2b)$. It goes on. You might get some dog like $-14x^8y^9 + 42x^5y^4 - 28xy^3$, which is $-14xy^3(x^7y^6 - 3x^4y + 2)$, where this final, factored form doesn't even look much simpler!

You can do some tests on these, which shows that an expression and its factored version are the same thing. Try this on the $12x + 8$ and $4(3x + 2)$ example,

```
for x=0,10,0.1 do
 print(12*x+8,4*(3*x+2))
end
```

where you'll get two columns of identical numbers, showing that indeed that $12x + 8$ is $4(3x + 2)$.

6.3 Computers and Factoring

Once again, computers are very good at factoring polynomials. We have a function called `factor()` that can handle just about any polynomial you might encounter in your algebra class. You can try these codes on the examples discussed above.

- Factor: $15x^3 + 9x$

```
r = factor("15x^3+9x")
print_math(r)
```

- Factor: $2a^2b + 4ab^2$

```
r = factor("2a^2b+4ab^2")
print_math(r)
```

- Factor: $-14x^8y^9 + 42x^5y^4 - 28xy^3$

```
r = factor("14x^8y^9+42x^5y^4-28xy^3")
print_math(r)
```

We find the speed at which the computer can spit out factors rather amusing. We can write a bit of code to make a quick "factoring app" for ourselves like this:

```
s = input()
r = factor(s)
print(s,"factored is")
print_math(r)
```

First off, sometimes the computer puts out answers that don't look right to us. Terms may be ordered or grouped differently than we'd do it by hand, but this is OK. Be flexible with this incredibly powerful mathematics tool (the computer).

Next, with this fast factoring, let's look at a few results. Remember in the last chapter we discussed some "special forms," like $(a + b)^2$ and $(a - b)^2$—the first one has the cross term, and the second one doesn't. Run the factoring app above on $a^2 - b^2$, and you'll see the two binomials this comes from (one with the $+b$ and the other with the $-b$). Next, try it with $a^2 + b^2$, where you'll see that the computer just spits out $a^2 + b^2$, because this cannot be factored. But, if you put in $a^2 + 2ab + b^2$, you'll see that it can be factored.

6.4 Factoring experiments on the computer

The last section was short as usual. Once you figure out how the computer can compute math results for you, in this case factoring, you can just *use* the computer to obtain such results, and get on with things. There's really not much else to say. That said, since we are using the computer, let's do two experiments with factoring, that we hope will give you some fresh and valuable insight into what factoring actually means. With the computer and some simple code, let's first see if we can write our own simple "factoring engine." Next, let's explore about the only real *use* of factoring that we can think of: information security.

6.4.1 A factoring engine

Suppose we run the following code,

```
r = factor("x^2 + 7*x + 12")
print_math(r)
```

where the computer happily spits out $(4 + x)(3 + x)$ as the two factors. How does it do this? We honestly don't know (if you're curious, look up "heuristic algorithms), but let's try to write our own code that can factors such polynomials.

To start, let's assume that our factoring engine will try to factor a polynomial with 3 terms in it. These are called a "trinomial." We'll further stipulate that it'll have a quadratic term, a linear term, and a constant term, and we'll assume it can be factored into two binomials. So we talking about taking in an expression like $x^2 + 6x + 8$. Note here that this meets our model's requirements in that it has a quadratic term, the x^2, a linear term, the $6x$ (linear in x, or x to the first power), and a constant term, the 8.

The factors of this are $(x + 4)$ and $(x + 2)$. We know for such polynomials like x^2+6x+8, that the factor are always "x plus-or-minus something" times "x plus-or-minus something else." In this case, the "something" is $+4$ and the "something else" is $+2$. So factoring is really just an issue of finding the "something" and the "something else." So let's do this. Let's set up a big unknown, which is our model of the factors we seek: `(x+A)` and `(x+B)`. If we can just find `A` and `B`, we'd be all set.

So the question is really then, how do we know if the `A` and `B` that we choose are correct? Well, remember how factoring works, in that for any x, the two factors multiplied together must equal to the original polynomial evaluated at x. So let's do this. Let's find any `A` and `B`, and at some x (any number), compute $(x + A)(x + B)$, keeping in mind that we must allow `A` and `B` to be both positive and negative. We'll compare the result of $(x + A)(x + B)$ with the original polynomial evaluated at the same value of x. If they're the same, then we know the `A` and `B` we chose must form the correct factors.

As we proceed, know that factorable polynomials are rare and hard to find. The legions of those in your algebra book are all "cooked up" to be factorable and doable by you, on paper, so their limits are quite tight. All As and Bs we'd expect will likely be integers (i.e. no decimals), and be between -100 and 100. So if we did an exhaustive search over -10..10 for A and -10..10 for B, we'd have to search 20×20 or 400 combinations of A and B at worst. This is very easy for a computer to do.

In this code, we choose a random x between -100 and 100, then start iterating over possible values for `A` and `B`. We'll compute `orig`, which comes from simply evaluating the polynomial at the chosen x. Then, we'll compute `fact`, which is the presumed form of the factors, or $(x + A)(x + B)$. Then we'll use an `if` statement to see if `orig` and `fact` are equal. If so, this must mean the original polynomial is equal to the product of its factors, so we'll print out the factors, with the `A` and `B` shown.

This code is set to work on $x^2 + 7x + 12$:

```
x = math.random(-100,100)
orig = x^2 + 7*x + 12
for A=-10,10 do
   for B=-10,10 do
      fact = (x+A)*(x+B)
      if orig == fact then
         print("(x+",A,")(x+",B,")")
      end
   end
end
```

and displays $(x + 3)(x + 4)$ and $(x + 4)(x + 3)$, which is telling us that our factors are $(x + 3)$ and $(x + 4)$. Using the "foil method," we get that $(x+3)(x+4) = x^2 + 3x + 4x + 12$, which is $x^2 + 7x + 12$. So we've done it! We've written some cod that can factor a trinomial.

In the most general case, factors of polynomials of the form of our trinomials here are: $(Ax+B)(Cx+D)$, so a more general factor app would be something that would iterate over A, B, C and D, as in

```
x = math.random(-100,100)
orig = 5*x^2+13*x+6
for A=-10,10 do
  for B=-10,10 do
   for C=-10,10 do
    for D=-10,10 do
      fact = (A*x+B)*(C*x+D)
      if orig == fact then
        print("(",A,"x+",B,")(",C,"x+",D,")")
      end
     end
    end
   end
end
```

which will reveal a weakness in our plan. This code will tell us about a lot of factors that one might interpret as factors of the original polynomial, but they are not. They are simply other factors of other polynomials that equal to each other at a given x. As an example if x happened to be zero, then $5x^2 + 13x + 6$ and $x^2 + 5x + 6$ would be equal to each other, and the code would report that $(5x+3)(x+2)$ and $(x+2)(x+3)$ are both pairs of factors of $5x^2 + 13x + 6$. All this means is that the logic behind our `if` statement isn't robust enough. Let's try to strengthen it some.

We know the form of the factors we are looking for is $(Ax + B)(Cx + D)$. We can multiply this out using the computer code here

```
r = algebra("(A*x+B)*(C*x+D)")
print_math(r)
```

to get that $(Ax+B)(Cx+D) = ACx^2 + (AD+BC)x + BD$. So we see that given our choices for A, B, C, and D, we observe three things:

1. the coefficient of x^2 in our original polynomial must be equal to AC (or $A \times C$).
2. the coefficient of x in our original polynomial must be equal to $(AD + BC)$ or (or $A \times D + B \times C$).
3. the constant in our polynomial must be equal to BD (or $B \times D$).

Thus we will have to tell our code the coefficient of the x^2 term (we'll call this `x2`), the coefficient of the x term (we'll call it `x1`) and the constant term, which we will call `const`.

Thus, our code will start like this for $5x^2 + 13x + 6$:

```
x2=5
x1=13
const=6
```

Next, we'll choose A, B, C, and D somehow, but to check if they seem to be giving a factor of our polynomial, given the 3 needed relationships above, we'll need an if statement that checks if:

1. `x2 == A*C`
2. `x1 == A*D+B*C`
3. `const = 6`

all at the same time. Although we are used to `if` statements asking only a single question, many question can be tied together using the `and` word, like this:

```
if  A*C == x2 and D*A+B*C == x1 and B*D == const then
...
end
```

where the code between the `then` and —end— will only be executed if all three conditions hold, or if AC==x2 **AND** D*A+B*C==x1 **AND** B*D==const.

As usual, we'll use 4 four-loops to iterate through possible values of A, B, C, and D. Limiting them all from -10..10, gives us $20^4 = 160,000$ combinations to sift through (no big deal for a computer). Here is our code for this:

```
x2=5
x1=13
const=6

for A=-10,10 do
 for B=-10,10 do
  for C=-10,10 do
    for D=-10,10 do
     if A*C == x2 and D*A+B*C == x1 and B*D == const then
      print("(",A,"x+",B,")(",C,"x+",D,")")
     end
    end
   end
  end
end
```

where we indeed find our needed factors. To work on $3x^2+16x+5$ we change the first three lines to

```
x2=3
x1=16
const=5
...
...
```

where once again, we'll get our needed factors.

Sometimes, you'll see a polynomial like this $9x^2 - 49$, which is the same as $9x^2+0x-49$. Here, the coefficient of x is zero, so in the code, we'd set `x1=0`. These modifications would find its factors:

```
x2=9
x1=0
const=-49
...
...
```

6.4.2 Factoring, proof of work, and data privacy

It turns out that factoring can be used in security applications. This is because there is no definite plan for finding factors, of either a number of a polynomial. This lack of a plan means finding factors is difficult. Security can exploit this difficulty.

Proof of work

Suppose you were hiring for a job, and only wanted people who really love math and coding. How could you really ensure this of the people you interview? You could tell them that you will hire them if they can tell you the factor of some polynomial. You could, for example, think of some really difficult polynomial to factor, and tell people that you'll hire them "if they can tell you the factors of $3x^2 + 20x + 12$" for example. Even if you type `factor 3x^2+20x+12` into Google, nothing immediately helpful comes up. So if someone can tell you a factor of polynomial, then likely 1) they're really good at math, or 2) they know how to code and found the factor using something like the "factor engine" above. So in some sense, a person who can factor this has "proven their smartness" to you. In actual Internet security, this would be called "proof of work," and literally means proof that someone has done some (hard) work. In this case, factoring is the hard work. It turns out that Bitcoin transactions, with its "blockchain" rely on this "proof of work" methodology.

Remember that you can generate a polynomial for someone to factor from factors themselves (that you can make up)—just remember to keep the factors a secreat. So if $3x^2 + 20x + 12$ seems too "easy," you can multiply out something like $(23432x + 2334)(2128x + 7641)$ which comes to $17834094 + 184010664x + 49863296x^2$. Ask someone to factor that!

A similar thing can be done, but with factoring just a number. You could, for example, only interview people who could tell you a factor of 913. It's likely not really clear to most beginning algebra students (without a computer) how one would find such factors. The following code would do it though:

```
for i=1,100 do
  if 913 % i == 0 then
    print(i,"is a factor")
  end
end
```

by finding a number that divides into 913 with no remainder (recall % is the "remainder" operator).

The larger number you choose, the harder it becomes to find factors of it. You should always choose an odd number (because even numbers are always divisible by 2), and one that isn't prime. To do this, use the Internet to find a list of prime numbers, and choose a large odd number that is *not* in the list. (Hint: how about 7917? It looks prime, but has some factors.) So your proof of work here would be for someone to tell you the factors of 7917.

Data security

Suppose you wish a secret number to a friend of yours. How could you do it, so if the message is somehow intercepted, it cannot be decoded? Let's say the number is 88. How could you "encode" it so that no one except your friend would be able to read it? It turns out that you can use factoring to do it.

To start, choose two prime numbers, in this case we'll choose $p = 17$ and $q = 11$. Multiply them to get $N = 187$. Note that 17 and 11 are factors of 187. Anyone wanting to read your secret will need the 187. But don't give out the 17 or the 11. These are the (hard to find) factors of 187, and are to be kept secret.

Next choose another prime number less than both p and q. Let's call it e and choose $e = 7$. Now, the N and e are you "public keys." You can give them out to anyone, post them on your blog, or whatever.

Next calculate d like this: You want $e \times d$ divided by $(p-1)(q-1)$ to have a remainder of 1. In this case $(p-1)(q-1)$ is 160, and $161 \div 160$ has a remainder of 1, so $e \times d = 161$, or $d = 161/e$ or $d = 23$. Keep d somewhat secret—you shouldn't make it public, but must give it to people who want to read (decrypt) your encrypted messages. Notice that d (in part), came from your secret p and q.

To send your message in secret, don't send the 88, send the remainder you get when 88^e is divided by N (your public key). So your encrypted message is another number, that is the remainder you get when you divide your original message raised to the e power by your public key.

On the computer, you can use the `algebra()` function and inside of it, use the `mod` function to find the remainders. As an example, the remainder you get when 144 is divided by 17 is 8, which you can compute with

```
r = algebra("mod(144,17)")
print(r)
```

In this case, you can use this code to encrypt your message:

```
r = algebra("mod(88^7,187)")
print("send:",r)
```

which will result in an 11. So send your friend the 11. This is the encrypted version of your original message, the 88.

To decrypt your message, you'll use d, which is a secret only to you and your trusted friend. To do so, raise the encrypted message to the d power, or 11^{23}. Now use `mod` again to find the remainder you get when 11^{23} is divided by N, your public key, or

```
r = algebra("mod(11^23,187)")
print("Message was:",r)
```

where you'll see your original message of 88 recovered again.

What does factoring have to do with this? Well, remember how this started. You produced your public key that came from two secret factors. Although the public key can be shared, its factors are hard to find. In practice, the public key would be a 200 digit number. Given that there is “no plan” for factoring, it would take a computer longer than the age of the universe to find the factors of a 200 digit number.

The public key is used both in encrypting and decrypting your message. BUT, the secret factors are used to compute d, which is required to *decrypt* your message. If a person doesn’t have d, they won’t be able to decrypt your message, and since they can’t guess your private factors (p and q), they’re stuck.

This exercise was adapted from:

- https://math.berkeley.edu/ kpmann/encryption.pdf
- https://blogs.msdn.microsoft.com/plankytronixx/2010/10/22/crypto-primer-understanding-encryption-publicprivate-key-signatures-and-certificates/.

Chapter 7

Solving an Equation

The title of this chapter, “Solving an Equation,” is undoubtedly something synonymous in your mind with math or algebra. That’s what you do after all in math, solve equations right? Let’s see what it’s all about.

7.1 An equation

An equation is formed when two algebraic expressions are joined by an equal sign. For example, we can have $2x + 5$, and we can have 13. If these were joined by an equal sign, we’d have

$$2x + 5 = 13. \tag{7.1}$$

In this case $2x + 5 = 13$ is *an equation*. The $2x + 5$ is sometimes called the “left hand side” (or LHS), and the 13 is called the “right hand side” (or RHS). Equations can have a lot of meaning. This one might be for: “You have x gallons of water. If you pour twice that much into a large bucket that already has 5 gallons in it, how many gallons did you have in the first place, if the large bucket ends up with 13 gallons in it?” People often ask for meanings to such equations, or want to know “what does this have to do with the real world?” or “When will I ever use this again?” We say don’t get too hung up on this right now. At this time, you are learning quite a lot of logic and math; just go with it. Everything you do and learn in your life doesn't have to have an immediate money making purpose.

In normal algebra, you'd set out to solve $2x + 5 = 13$ by first subtracting 5 from both sides to get $2x = 8$. Then you'd divide both sides by 2 to get $x = 4$. So what have you done?

You've found the singular number, 4 that makes $2x + 5 = 13$ work. So if we put 4 in for x, we'd get: $2(4) + 5$ which is $8 + 5$ which is 13. So say $x = 4$, the left side of the equation computes to 13, and this is equal to the right side of 13. So we see that the equation holds. So if you have 4 gallons of water, double that to 8, then add it to a bucket already containing 5 gallons, you'll have 13 total gallons.

Let's see what more we can learn from the computer now.

7.2 Solving an equation on the computer

Let's think of the RHS and LHS as separate expressions, like $2x + 5$ and 13. Let's write some code to plot them now. If we hover the mouse over the black output area on the coding screen, we'll see that it extends horizontally from $x = -225$ to $x = +225$ or so. Let's use these in a for-loop to generate some x-axis values, and we'll plot the 13 with like this:

```
pcls(0)
for x=-225,225 do
 pset(x,13)
end
```

Here, you see the command `pset(x,y)` plots a dot at the (x, y) point you give it. When this code is run, we'll see a horizontal line (that's what $y = 13$ is, right?). Anyway, if we hover the mouse over the line, we'll see that the line is at about $y = 13$, as it should be. Next, we'll add some code to plot $2x + 5$, like this:

```
pcls(0)
for x=-225,225 do
 pset(x,13)
 pset(x,2*x+5)
end
```

As you can see, the two lines intersect at a certain point. What do you think it means for two lines to intersect?

It means they have at least one (x, y) point that they have in common, or share. It also means that you should be able to find exactly which x point at which they intersect, and this will be "the solution" to the equation. Or, in other words, the x point where they intersect, will be the value of x that makes $2x + 5$ equal to 13.

Let's do this: hover the mouse over the intersection point and watch the x and y values above the drawing area. The best we can do is find x to be around 5 or 6. Nothing too exact. But if we want to really drill down on the intersection point, let's zoom in using the `zoom()` function. Add one line, `zoom(5)` to your code like this:

```
pcls(0)
zoom(5)
for x=-225,225 do
 pset(x,13)
 pset(x,2*x+5)
end
```

You'll see your plot line now to be more broken up, but this is what a line (which is a collection of pixels), looks like under magnification. Now, we can definitely see (with more mouse hovering), that the intersection point is around $x = 4.2$ or 4 or so. This matches the $x = 4$ we got above.

Here's some code that will work up a solution to $2x - 1 = -3x + 2$.

```
pcls(0)
```

```
for x=-225,225 do
  pset(x,2*x-1)
  pset(x,-3*x+2)
end
```

Where at this level, we get an intersection point of around $x = 0.2$. Let's zoom in a bit, like this:

```
pcls(0)
zoom(10)
for x=-225,225 do
  pset(x,2*x-1)
  pset(x,-3*x+2)
end
```

where now we can find a better intersection point around $x = 0.6$ or so. If we solve $2x - 1 = -3x + 2$, we'll get $5x = 3$, or $x = 3/5$, and $3/5$ is about 0.6. So the idea of two curves intersecting worked again for solving some LHS=RHS equation.

One more time, what does the $x = 0.6$ mean? If you plug it in for x on the LHS, you get $2(0.6) - 1 = 0.2$. Plug it in for x on the RHS, you'll get $-3(0.6) + 2$ which is $-1.8 + 2$ or 0.2. So once again, the $x = 0.6$ makes the LHS equal to the RHS.

If you want, pull some equations out of your algebra book, and see if you can solve them by plotting them as shown here.

7.3 Getting more organized

We hope you see the idea of using graphs, and the intersection of the LHS and RHS on a graph to tell where their solution is. In this section, let's organize this idea some more.

It turns out in math, for the sake of organization, the RHS of equations will be zero (most of the time). Why zero? Well, it works like this. Suppose we re-looked at our first equation above, $2x + 5 = 13$. What if we moved the 13 to the LHS, by subtracting it from both sides? We'd get $2x+5-13 = 13-13$, or $2x-8 = 0$. Now let's do this. Since the RHS is zero, let's plot a horizontal line at $y = 0$ like this:

```
pcls(0)
line(-225,0,225,0)
```

We like the $y = 0$ line (which is the x-axis) because if we're going to organize our equation solving, and always have the RHS=0, then this line will always be our reference. Or, if our equation intersects this $y = 0$ line, we'll use the mouse cursor to find the x value of the intersection point, and that will solve our equation.

So, let's add in the plotting of $2x - 8$ (as above), like this:

```
pcls(0)
zoom(1)
line(-225,0,225,0)
for x=-225,225 do
  pset(x,2*x-8)
end
```

Low and behold, our LHS of $2x - 8$ intersects our $y = 0$ reference line. Hovering with the mouse finds the intersection point at $x = 6$. (We find it easier to locate these intersection points with the horizontal line too.) The $x = 6$ is the number that makes $2x - 8$, or $2(6) - 8$ which is $8 - 8$ equal to zero.

What about organizing $2x - 1 = -3x + 2$? For this we'd get $2x + 3x = -3x + 3x + 2$ or $5x = -2$, or $5x + 2 = 0$. So the code would be

```
pcls(0)
zoom(1)
line(-225,0,225,0)
for x=-225,225 do
```

```
  pset(x,5*x+2)
end
```

Here we do need to zoom in a bit to better locate the intersection point, as per

```
pcls(0)
zoom(5)
line(-225,0,225,0)
for x=-225,225 do
  pset(x,5*x+2)
end
```

And remember, if you want to add more points to your zoomed plot, you can always add a step-size to your `for` loop, like this (note the `0.5` in `for x=-225,225,0.5 do`):

```
pcls(0)
zoom(5)
line(-225,0,225,0)
for x=-225,225,0.5 do
  pset(x,5*x+2)
end
```

With this, we find $x = -0.4$ to be a good read on the intersection point. Now $5(-0.4)+2$ is $-2+2$, which is zero. So we've solved the equation again.

So, here's the summary of this section

> *With "organized equations," that look like LHS=0, the point where the equation crosses the y-axis is the solution to the equation.*

7.4 Better looking plots

If we're going to use plots to solve equations, let's make some better looking plots. For this, let's use the built-in `graph()` function. It works like this (just one line—the computer does all of the work for you):

```
graph("5*x+2",-10,10,0.5)
```

Note! You have to put your equation in double quotes in order for this to work, so you put in `graph("5*x+2",-10,10,0.5)` *and not* `graph(5*x+2,-10,10,0.5)`. Sorry about that!

In this code, the function $5*x+2$ will be plotted from $x = -10$ to 10, in increments of 0.5. This means x will run from $-10, -9.5, -9, -8.5...0...8.5, 9, 9.5, 10$. The plot will appear below your code window (it won't appear in the black output window). The plot looks much nicer, with clearly labeled axes, and grid lines and all. But now with the computer doing most of the work, we can focus on the math more. Here's how.

If you run the code above, you'll see your line. You can hover over it with the mouse and identify (x, y) pairs, just like before. Remember, when thinking of $5x + 2 = 0$, we still want to know when $5x + 2$ will cross the $y = 0$ line. When looking at the initial graph, remember that we don't have to run our plot between $x = -10$ and 10, as we can choose any plot range we wish. A quick check with mouse hovering tells us $5x + 2$ will cross $y = 0$ somewhere between -2 and 1 so, let's try plotting between these two values like this

```
graph("5*x+2",-2,1,.5)
```

This is like "zooming in" before. We can also fill in more data points by say, changing the `0.5` to a `0.01` instead, like this:

```
graph("5*x+2",-2,1,0.01)
```

Wow! So even with zooming in, we can have a nice solid looking line. Using the crosshairs and mouse hovering, we can clearly see that $5x + 2$ crosses

$y = 0$ at $x = -0.4$, the exact solution to this equation! You can zoom in still further if you want, say between -1 and 0.5, like this

```
graph("5*x+2",-1,0.5,0.01)
```

where the $x = -0.4$ is even easier to identify.

7.5 More complicated functions

7.5.1 Equations with powers of x larger than 1

We hope you see the same theme here: all we're doing is trying to identify where curves we plot cross the $y = 0$ line, or cross the x-axis.

In algebra you'll have more complicated equations than the ones we've been using, but the idea is still the same: 1) organize it so it's in the form of LHS=0, then 2) plot it and 3) try to find (with hovering and zooming) where it crosses the $y = 0$ line.

Let's try solving $4x^2 - 6 = 0$. That is, what value of x makes $4x^2 - 6$ equal to zero? Here's our code

```
graph("4*x^2-6",-5,5,0.1)
```

Notice anything funny about the graph? It crosses the $y = 0$ line in *TWO PLACES*. What does this mean? It means the equation has *more than one solution*! It looks like all $y = 0$ crossings happen between -2 and 2, so let's recast our plot in this region:

```
graph("4*x^2-6",-2,2,0.1)
```

Next, we'll fill in some more data points

```
graph("4*x^2-6",-2,2,0.01)
```

The $y = 0$ crossings appear to happen at $x = -1.22$ and $x = +1.22$. What gives? Well, let's look. Does $4x^2 - 6$ equal to zero when $x = -1.22$? Here's our check: $4(-1.22)^2 - 6$ is $5.95 - 6$ which is almost zero! The same goes for $x = +1.22$ as per $4(1.22)^2 - 6$ is $5.95 - 6$. We can zoom in even more like this:

```
graph("4*x^2-6",-1,1,0.01)
```

Here we might find -1.23 and 1.23 as solutions, making it even better than 1.22. What would we get by hand? It works like this $4x^2 - 6 = 0$, or $4x^2 = 6$, or $x^2 = 6/4$ or $x = \sqrt{6/4}$. This gives $x = 1.225$, pretty close to our graphical results. The two solutions are found here because of the x^2. Turns out that 1.23^2 and -1.23^2 are both 1.51.

Want to try something? How about solving $x^3 - 4x - 2 = 0$? We started with

```
graph("x^3-4*x-2",-3,3,0.01)
```

After some zooming in, we found $y = 0$ crossings at $x = -1.67$, -0.54 and 2.2 using the techniques shown here.

7.5.2 The "power" of an equation

Did you notice something? You probably learned that the highest power of a variable in an equation is called the "power" of the equation. For $5x + 2$, the highest power of x is 1. For $4x^2 - 6$, the highest power is 2 and for $x^3 - 4x - 2$, the highest power is 3. Notice that $5x + 2$ had one solution, $4x^2 - 6$ had two solutions and $x^3 - 4x - 2$ had 3 solutions! Yes! The power of an equation is also the maximum number of solutions it will have!

Want a final challenge? Find the solutions to $x^5 - 5x^4 + 5x^3 + 5x^2 - 6x + 1$ between $x = -2$ and $x = 3$. (Hint: there's 5 of them!)

7.6 Factoring and solving equations

Remember the discussion on factoring from the last chapter? Let's take a look at it one more time, with the polynomial $5x^2 - 34x - 7$. If we factor this, we'll get $(x-7)(5x+1)$. Now, suppose we wanted to solve $5x^2 - 34x - 7 = 0$.

As usual, since it's in the form of LHS=0, we can graph it and look for places the plot crosses the $y = 0$ line, using this code:

```
graph("5*x^2-34*x-7",-10,10,0.1)
```

We two crossings. A quick scan with some mouse hovering show the crossings between about $x = -1$ and about 8, so we'll zoom in some and plot more points with:

```
graph("5*x^2-34*x-7",-1,8,0.1)
```

We can identify the solutions to be at $x = -0.19$ and $x = 7$. Plugging these back in to $5x^2 - 34x - 7$, we get $5(-.19)^2 - 34(-.19) - 7$, which gives -0.35 (which is close to zero), and $5(7^2) - 34(7) - 7$ which is equal to zero, so we're on the right track (our visual inspection of the -0.19 is obviously a bit off). What does this have to do with factoring?

Well, our equation is $5x^2 - 34x - 7 = 0$. If we write this in factored form, we'd get

$$(x-7)(5x+1) = 0. \tag{7.2}$$

So what? The factors $(x - 7)$ and $(5x + 1)$ are multiplied in the equation. This means if either is zero, the whole LHS will be zero, and this would make the equation hold. This is because $0 \cdot (5x+1)$ is zero, as is $(x-7) \cdot 0$. So, we can also solve the equation by setting its factors, one by one, equal to zero like this:

$$x - 7 = 0, \tag{7.3}$$

or

$$5x + 1 = 0. \tag{7.4}$$

With the first factor, we'd get $x - 7 = 0$ or $x = 7$, which is the same as the $x = 7$ we got using our mouse hovering above! With the second factor, we'd get $5x + 1 = 0$, or $x = -1/5$, which is about $x = -0.2$, which is almost equal to the -0.19 we also got with our mouse hovering.

So do you see how the $y = 0$ crossing and factoring are related? If you put your equation in LHS=0 form, and factor the LHS, the little "mini-equations" made by setting each factor equal to zero, gives solutions to the original equation. Thus, you can solve equations by *factoring them*. Not all equations are factorable, so this won't always work.

7.6.1 Factoring by graphing

No matter what equation you have, you can always plot it and look for its $y = 0$ crossings. Since these $y = 0$ crossing are solutions to the equation, you can use them to construct the factors too. Let's take a look at $10x^2+25x-15$. If we graph this with get $y = 0$ crossings at $x = -3$ and 0.5. This means we can immediately write factors of this equation to be $(x + 3)$ and $(x - 0.5)$. Does this mean that $(x+3)(x-0.5)$ is equal to $10x^2 + 25x - 15$? Not quite, because $(x + 3)(x - 0.5)$ is $x^2 + 2.5x - 1.5$. Hmmm.

If we set this up as a LHS=0 equation (which is what makes our $y = 0$ crossing technique work), we'd have

$$(x + 3)(x - 0.5) = 0 \tag{7.5}$$

or

$$x^2 + 2.5x - 1.5 = 0. \tag{7.6}$$

If we multiply the 2nd equation through by 10, we'll get

$$10x^2 + 25x - 15 = 0, \tag{7.7}$$

which is the original equation. It tells us that factors can be found using our $y = 0$ crossing technique, but they might be off by some multiplicative factor, in this case 10. Why is this? It's because of the RHS being zero. We can multiply the LHS and RHS by any number we want, as still get zero on the RHS. This means our factors found in this way can be off by some multiplicative factor. In this case, 10 is missing, so the complete factors would be

$$10(x + 3)(x - 0.5). \tag{7.8}$$

In this case, the 10 could be combine with the $(x - 0.5)$ to get $(10x - 5)$, so the factors wold be

$$(x + 3)(10x - 5), \tag{7.9}$$

which when multiplied out do give us $10x^2 + 25x - 15$.

The point of this section is that factoring can be hard, as discussed in the last chapter. The $y = 0$ technique at least guides us in forming a likely pattern of the factors. When these are multiplied out, the result can be compared to the original, to see what the remaining ambiguous multiplicative number might be.

7.7 What solution?

As a final look at this $y = 0$ crossing technique, let's work with one more equation, $5x^2 + 10 = 0$. As usual, we can make a graph of it using the following code:

```
graph("5x^2+10",-3,3,0.01)
```

Now, if you hover over the graph with the mouse, you might notice something: *it never crosses $y = 0$!* What does this mean? It means the equation doesn't have any solutions (or at least solutions that are real numbers). If you look at it, it's not hard to see why. The term $5x^2$ is always going to be a positive number, for any value of x you can think of. Next, if you add 10 to it, you get an even larger positive number. Now, if you think carefully, how can you get zero from $5x^2 + 10 = 0$ by adding together two positive numbers? You can't. Thus $5x^2 + 10 = 0$ doesn't have any solutions that are real numbers,[1] and the graphical method very quickly reveals this.

The lines we looked at earlier, like $5x + 2$ always have a solution because the $5x$ can be positive and negative (depending on your choice of x), so $5x$ could conceivably be -2, making the equation work, as per $-2 + 2 = 0$.

7.8 Computer search for equation solutions

Did you notice something about all of the graphs you made in this chapter? In particular around a $y = 0$ crossing? Let's look again at $4x^2 - 6$.

```
graph("4*x^2-6",-2.0,2.0,0.01)
```

Take a look at the $y = 0$ crossing on the left by hovering the mouse near $x = -1.21$. Look at the curve in this location. If you go further to the left, the curve is positive, since it has $y > 0$ (it's above the x-axis). If you look to the right of $x = -1.21$, the curve is negative (it's below the x-axis). You can see a similar thing around the right $y = 0$ crossing at $x = 1.22$. Left of it, the curve is negative. Right of it, the curve is positive. This is a pattern on solving LHS=0 type equations:

The equation switches signs at a $y = 0$ crossing.

This might make sense. Afterall, what is zero? It's the number that separates positive and negative numbers. So here, $y = 0$ crossings separate positive

[1]To be 100% complete, it does have the two imaginary solutions of $+i\sqrt{2}$ and $-i\sqrt{2}$, but don't worry about "imaginary numbers" for now.

and negative sections of the equation. For every $y = 0$ crossing, you'll see the equation switches its position (or sign) relative to the y-axis. If above it on one side of the $y = 0$ crossing, it'll be below it on the other side. If below it on one side, it'll be above it on the other. We can use this pattern to automatically have the computer solve equations for us. We'll do it like this.

Let's give the computer a guess as to what range we think a solution might be found. We'll think of these as low and high x values. These value will set a range that we're sure contains the solution to an equation. This range isn't that hard to figure out, and doesn't have to be anything exact. Say for $4x^2 - 6$, we'll choose -10 and 10. In other words, we're guessing a solution exists between -10 and 10. For the sake of this conversation, let's say $a = -10$ and $b = 10$. We can think of a as being the left bound, and b being the right bound for our solution.

Now, our best guess at a solution will be right in them middle of this range, or at $(a+b)/2$. In this case, it'll be at (-10+10)/2=0. What we'll do next is this: we'll look at the sign of the equation at $x = a$ and the sign of the equation at x itself. If the equation has the *same sign* at both of these locations, we can assume there's no solution between a and x. Why? Because as we discussed above, an equation changes sign as it crosses a solution. In the case of the equation having the same sign at a and at x, we'll think of a as being too far to the left, and bring it in to x. In other words, we'll set a to be x.

What if the equation at a and x has opposite signs? We can then assume a solution exists between a and x, and that b is too far over to the right. Thus, let's bring b in, by setting $b = x$.

Now, with a or b adjusted, we can compute a new midpoint again via $(a+b)/2$ and go again.

For ease, we'll put out LHS=0 equation into its own function like this

```
function f(x)
  return 4*x^2-6
end
```

Thus, "the equation" we discussed above all just becomes $f(x)$. The logic with checking the sign of the equation and all can be implemented with an `if` statement like this

```
if f(a)*f(x) > 0 then
  a = x
else
  b = x
end
```

Where you can see that `f(a)*f(x)` is how we check the sign of the equation at a and at x. If the product of the two is greater than zero, they must have the same sign. (Right? $3 \times 3 > 0$ and so is $-3 \times -3 > 0$. The 3s have the same sign, so their products are always greater than zero.)

Here we go then. This is all called the "bisection method" for finding solutions (or roots) to equations. Wikipedia has a nice description of it at `https://en.wikipedia.org/wiki/Bisection_method`. Our complete code to work this out is here:

```
function f(x)
  return 4*x^2-6
end

a=-10
b=10
x = (a+b)/2

for i=1,20 do
  if f(a)*f(x) > 0 then
    a = x
  else
    b = x
  end
  x = (a+b)/2
  print(x,f(x))
```

```
end
```

Here, we use a `for` loop to run through the procedure 20 times. With each try, we print `x` and the equation evaluated at `x`. Remember that we want to find `x` that will make `f(x)` as close to zero, as possible. Thus watch the numbers. The left column will be the computer's current value of `x`, and the right column `f(x)`. If everything is working, you'll see `f(x)` become very small, meaning it is approaching zero, and you can become more and more confident that the `x` value is the solution to the equation.

The code above puts out

```
-5 94
-2.5 19
-1.25 0.25
-0.625 -4.4375
-0.9375 -2.484375
-1.09375 -1.21484375
-1.171875 -0.5068359375
-1.2109375 -0.134521484375
-1.23046875 0.05621337890625
-1.220703125 -0.039535522460938
-1.2255859375 0.0082435607910156
-1.22314453125 -0.015669822692871
-1.224365234375 -0.0037190914154053
-1.2249755859375 0.0022607445716858
```

where you can see after 10 runs through the bisections, `f(x)` has become `0.00226...`, which is small and close to zero. Thus we can think of `-1.22` as a solution to $4x^2-6$. Which is it, right? Let's see $4(-1.22)^2-6$ is 0.0022, which is pretty close to saying LHS=0, or LHS=0.0022 in this case.

We note that this program and technique works very well for just about any LHS=0 type equation you'll encounter in an algebra class. Try some other equations, from your algebra book and see what your get. Here are some examples:

- To solve $5x-3=-13$ change `f(x)` function to (get it in LHS=0 form):

  ```
  function f(x)
    return 5*x+10
  end
  ```

- To solve $2x^2-4x+7=0$ change `f(x)` function to

  ```
  function f(x)
    return 2*x^2-4*x+7
  end
  ```

- To solve $6x^2=5-7x$ change `f(x)` function to

  ```
  function f(x)
    return 6*x^2+7x-5
  end
  ```

  Likely, the initial guesses of $a=-10$ and $b=10$ will be fine for these.

Chapter 8

Quadratic Equations

Quadratic equations are perhaps the defining work of an algebra class. We think there are millions of former algebra students walking around out there mumbling something like “minus b plus or minus the square root of b squared...” Let’s see what this all means.

8.1 Introduction

A quadratic equation is an equation that (typically) has both an x^2, x, and a constant term in it. Examples might be $x^2 - 2x - 8 - 0$ and $5 - 2x^2 = 3x$. The “defining work” we speak of is in solving these types of equations, which means of course, finding the values of x (there are typically 2), that make the LHS (left hand side) of the equation equal to zero. For $x^2 - 2x - 8 - 0$, the value of x are -2 and 4. These two numbers mean two important things for the equation:

1. $x = -2$ and $x = 4$ make the LHS=0. You can check this via $(-2)^2 - 2(-2) - 8 = 0$ which is $4 + 4 - 8$ or $8 - 8$ which is zero. With the 4, this is $4^2 - 2(4) - 8 = 0$ or $16 - 8 - 8 = 0$ or $16 - 16 = 0$. So they both work in making the LHS=0.

2. -2 and 4 help to form the factors of $x^2-2x-8-0$, in that $(x+2)(x-4)$ is equal to $x^2-2x-8-0$.

In this chapter, we'll study two aspects of quadratic equations and their use on the computer. The first is in what the quadratic formula tells us, and the second is in implementing the quadratic formula in some code, and seeing a deeper understanding of it.

8.2 The quadratic formula

The so called "quadratic formula" is what we were referring to in the opening of this chapter. It is

$$x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}, \tag{8.1}$$

and tells us this. Suppose you had a quadratic equation, and were able to organize it into the form

$$ax^2+bx+c=0. \tag{8.2}$$

This means something like $x^2-2x-8=0$, for which $a=1$, $b=-2$ and $c=-8$. Or, $5-2x^2=3x$. If this is rearranged, we'll get $5-2x^2-3x=0$ or $-2x^2-3x+5=0$, meaning $a=-2$, $b=-3$, and $c=5$. In either case, a is the coefficient of the x^2 term, b that for the x term, and c is the constant. For any equation with a, b, and c known, you can use the quadratic formula to find solutions to our equation.

In the case of $x^2-2x-8=0$, with $a=1$, $b=-2$ and $c=-8$, we'd get

$$x=\frac{-(-2)\pm\sqrt{(-2)^2-4(1)(-8)}}{2(1)}, \tag{8.3}$$

which is

$$x = \frac{2 \pm \sqrt{4+32}}{2} \tag{8.4}$$

which is

$$x = \frac{2 \pm 6}{2}. \tag{8.5}$$

This means that x could be $\frac{2+6}{2}$ or $\frac{2-6}{2}$, or x could be 4 or -2. The final result is that both $x = 4$ and $x = -2$ are solutions to the equation $x^2 - 2x - 8 = 0$. Checking this, we get $(4)^2 - 8 - 8$ or $16 - 8 - 8$ or 0, which checks. The other check is $(-2)^2 - 2(-2) - 8$ or $4 + 4 - 8$ or 0, which also checks.

Now, using the graphing techniques we learned in the last chapter, we can do something like

```
graph("x^2-2*x-8",-3,5,0.1)
```

and easily identify -2 and 4 as the $y = 0$ crossing points. Thus, the quadratic formula gives us the value of x where the equation makes its $y = 0$ crossings. They are the points that make the equation zero as in LHS=0.

8.3 Coding and the quadratic formula

In this section, we'll develop some code that produces solutions to quadratic equations, so long as you can tell the computer what a, b, and c are. The discussion drills down into the quadratic formula, giving you good insight into actually what's going on with the various quadratic presented.

8.3.1 Quadratic solver version 1.0

So let's write a quadratic equation solver then. As input, let's have the code ask us the values of a, b, and c. From there, let's compute the solutions to the quadratic equation using the quadratic formula and display the results.

To read in a, b, and c, we can use the `input()` function like this

```
print("What is a?")
a = input()
print("What is b?")
b = input()
print("What is c?")
c = input()

print(a,b,c)
```

where the `print` statement verifies that a, b, and c are coming in properly.

To find the solutions, we'll program in the quadratic formula as shown by Eq. 8.1 above. We'll call the solutions `x1` and `x2`, since one involves the plus sign and the other the minus sign. So we'll have code that looks like this:

```
print("What is a?")
a = input()
print("What is b?")
b = input()
print("What is c?")
c = input()

x1 = (-b+sqrt(b^2-4*a*c))/(2*a)
x2 = (-b-sqrt(b^2-4*a*c))/(2*a)

print(x1,x2)
```

Note that `sqrt()` is a built in function that gives us the square root of its number, so `print(sqrt(16))` would print 4. If we run this to solve $x^2 - 2x - 8 = 0$ we'd type in -1 for a, -2 for b and -8 for c. With these the code will print `4` and `-2`. So we've done it! We've written a quadratic formula solver. Try it with some quadratic equations in your book. It's quite fun! But there's more. Let's take a look.

8.3.2 Quadratic solver version 2.0

Let's use the code above to solve $x^2 + 2x + 6 = 0$. Here $a = 1$, $b = 2$ and $c = 6$. If we type these in to our program above, the computer will output `nan` and `nan`. What does "nan" mean?[1] There is something strange going on here. The computer will put out `nan` if we trying something like `print(1/0)` too (because you can't divide by 0).

If we graph the equation using

```
graph("x^2+2*x+6",-5,5,0.1)
```

we'll see something we alluded to in Section 7.7: the equation is entirely above the x-axis! It other words, it doesn't have any $y = 0$ crossing points. There's nothing wrong with the equation. It's just that it doesn't cross the x-axis (or $y = 0$ anywhere). What gives?

Let's look closely at the quadratic formula, in particular, the part under the square root, or $\sqrt{b^2 - 4ac}$. This called the "discriminant." For the equation above, $a = 1$, $b = 2$ and $c = 6$. If we put these into the discriminant, we'll get

$$\sqrt{2^2 - 4(1)(6)} \tag{8.6}$$

or

$$\sqrt{4 - 24} \tag{8.7}$$

giving

$$\sqrt{-20}. \tag{8.8}$$

Now wait a minute! The square root of -20? That's impossible! What real number can you multiply by itself that will give -20? There isn't one. So

[1] "nan" means "not a number."

the computer putting out `nan` is itself stumbling on this issue. There is no real number that is the square root of -20.

As an aside, you actually *can* find $\sqrt{-20}$. It involves imaginary numbers, using the symbol "i" which is defined to be $i = \sqrt{-1}$, so $\sqrt{20}$ can be written as $\sqrt{20}i$, which is $2\sqrt{5}i$. But in your first algebra class, you are only concerned with *real numbers* as answers to your equations, so all of this "imaginary number" nonsense just means "no solution," or "no real numbered solution."

This is where the coding helps you to look deeper. You see that the equation's graph never crosses the $y = 0$ axis, and now you see the discriminant wants the square root of a negative number. With this latter fact, we can make our code a bit more graceful. How about after having a, b, and c, we'll first compute the stuff under the radical sign, or just $b^2 - 4ac$, like this:

```
print("What is a?")
a = input()
print("What is b?")
b = input()
print("What is c?")
c = input()

d = b^2-4*a*c
print(d)
```

In this case, the value of $b^2 - 4ac$ is displayed, which we put into the variable `d` (for discriminant). You can see that it is negative for $a = 1$, $b = 2$ and $c = 6$. So what now? Let's use an `if` statement to check if d is negative (< 0). If so, we can 1) tell the student that there are no real roots, and stop the display of the confusing "nans," and 2) demonstrate our insights into the quadratic formula. How about something like this?

```
print("What is a?")
a = input()
print("What is b?")
b = input()
print("What is c?")
```

```
c = input()

d = b^2-4*a*c

if d < 0 then
 print("There are no real roots to your equation.")
 return
end
```

Here the `return` statement causes the code to stop right away. We can finish the code by computing the two solutions, and printing them. And unlike last time, let's use `d`, since it's already calculated, and it'll make our formulas simpler. Here's our final result.

```
print("What is a?")
a = input()
print("What is b?")
b = input()
print("What is c?")
c = input()

d = b^2 - 4*a*c

if d < 0 then
 print("There are no real roots to your equation.")
 return
end

x1 = (-b+sqrt(d)) / (2*a)
x2 = (-b-sqrt(d)) / (2*a)

print(x1,x2)
```

8.3.3 Quadratic solver version 3.0

There's one more item to address with the discriminant. Let's use our solver to find solutions to $x^2 - 4x + 4$. Here $a = 1$, $b = -4$ and $c = 4$. If we put these into the solver code above (Version 2.0) it'll put out `2 2` as solutions. Two twos? That's right. If we plot this equation, with

```
graph("x^2-4*x+4",-3,5,0.1)
```

you'll see that the equation touches the $y = 0$ line in just one spot. It doesn't really cross it, but touching it is close enough—it still means the equation equals zero. In this case, it touches the axis at $x = 2$. Given that the power of a quadratic is 2, due to the x^2 term, the 2 and 2 *are* the two solutions. What is going on with the discriminant? With $a = 1$, $b = -4$ and $c = 4$, we get

$$\sqrt{b^2 - 4ac} \tag{8.9}$$

which becomes

$$\sqrt{(-4)^2 - 4(1)(4)} \tag{8.10}$$

or

$$\sqrt{16 - 16} \tag{8.11}$$

giving

$$\sqrt{0} \tag{8.12}$$

or just zero! The discriminant is zero! What does this mean?

Well, in the quadratic formula, we have

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}. \tag{8.13}$$

But with $d =$ the discriminant, we have

$$x = \frac{-b \pm \sqrt{d}}{2a}. \tag{8.14}$$

Now, if $d = 0$ then this becomes

$$x = \frac{-b \pm 0}{2a}, \tag{8.15}$$

or just

$$x = \frac{-b}{2a}. \tag{8.16}$$

Thus when $d = 0$ there is nothing to add or subtract from $-b$ in the famous quadratic song "negative b plus or minus..." This means that when the discriminant is zero, the quadratic equation only has one unique solution (or two identical solutions), given by $x = -b/(2a)$. That's right, c is ignored. If $a = 1$, $b = -4$ and $c = 4$, we get $x = -(-4)/(2(1))$ or $x = 2$, the same x value where the equation touches the x-axis.

So, as a last modification to our code, instead of printing two identical solutions if d is zero, let's print a friendly message. Something like this should do.

```
print("What is a?")
a = input()
print("What is b?")
b = input()
print("What is c?")
c = input()

d = b^2 - 4*a*c

if d < 0 then
 print("There are no real roots to your equation.")
```

```
  return
end


x1 = (-b+sqrt(d)) / (2*a)
x2 = (-b-sqrt(d)) / (2*a)

if d == 0 then
   print("The only unique solution is ",x1)
else
  print(x1,x2)
end
```

Now, here in version 3.0, we have a very user-friendly quadratic equation solver, and shows off our deep insights into quadratic equations, and the quadratic formula.

This book teaches basic algebra by having the student write short, simple computer programs (or ``code"), to investigate topics typically found in an introductory algebra class.

Topics include symbolic manipulation, factoring, problem solving, graphing, equation solving, and quadratic equations, all presented with the backdrop of writing simple code.

The programming environment used is the site at www.codebymath.com, which is a website dedicated to the parallel learning of coding and mathematics. The website has a straightforward design for coding: a ``run" button executes code typed into a text-window on the left side of the screen. Any output generated, will appear in second window on the right side of the screen. Many programming extensions are available, for text, graphics, professional charts, sound, and even Google Maps.

The student will be pleased with the motivation and insights coding brings to learning algebra.

Codebymath.com