

The Designer's Guided Tour of DragginMath® by Steven T Abell

DragginMath is an iPad app for teaching, learning, and doing algebra. It converts equations into interactive pictures. Then you can solve your equations by literally *dragging math* around on the screen. Sounds simple, right?

Actually, it is. If there is a problem, it is that DragginMath does *so much* math just by dragging it around on the screen.

Can you learn to use DragginMath just by playing around with it? I like to think so, but I created it. Your answer might not be the same as mine. Perhaps you are one of those people who enjoys the experience of discovery, but most people want a more direct path to understanding DragginMath.

So let me help you out.

Who Is Reading This?

One of the challenges of writing a Guided Tour like this is trying to guess who will be reading it. Are you a student who is just getting started in algebra? Are you a parent who is trying to help a child? Or are you a professional math teacher who hopes to use this app in the classroom? Each of these different people might need a different Guided Tour. But for now, there is only this one.

Whoever you are, I will not assume you are a genius, or that you already know a lot of math. But we have to start

somewhere. I will assume you are at least basically sensible and intelligent. You probably are, and that is all you need to operate DragginMath. I will also assume you already know *some* math: basic arithmetic and the language that goes with it.

So let me ask you some questions. Do you know what a *reciprocal* is? A *root*? A *logarithm*? Do you know (or think you know) how *operator precedence* works? Are words like *commutative*, *associative*, or *distributive* familiar to you? Do you know the *rules of exponents*? If you already know what these words mean, that's great. If you don't, let me suggest you view this as an opportunity, not a deficiency. Reading this will help to introduce these concepts to your mind. This document is not a substitute for real math instruction, which DragginMath was designed to *assist*, not to *replace*. So if you encounter unfamiliar words or concepts in this Guided Tour, *learn what you can* from it, which might mean no more than becoming aware that these words and concepts exist. Then seek out deeper understanding from someone who already knows the math. When you come back to the Guided Tour, you will be ready to use DragginMath in ways you could not have done before. On the other hand, if you are a math teacher, you already know the words, the concepts, and the math, although working through it with DragginMath may show you some things that are new to you. The process of designing this app certainly showed me some things that were new to me.

DragginMath Is Not Math On Paper

Another challenge in writing a Guided Tour like this comes from DragginMath's very nature: the words in this document

just sit there on the page, but DragginMath is about math in motion. I can describe what to look for, then what to do, then what to expect, but you still have to see the right things *when they happen*. This is hard to convey convincingly in print. Live demonstrations are better, but you may not have that option. Here on paper, we will do the best we can.

There is a DragginMath channel on YouTube. The videos there can make much of this more clear. But people have asked for a written Guided Tour, so here it is.



Two Aspects of Learning Math

Learning math is actually two things: learning *what math means*, and also learning *how to write it*. These two things are related, but they are not the same. I could talk at length about the difference, and I have done so elsewhere. But that is not what this Guided Tour is about.


DragginMath is mostly about *one way math can be written*. This is not the way math is usually written. But the way math is usually written is less helpful and even troublesome for many people when they first learn algebra. By learning to read and use this *different* way of writing and doing algebra, you will gain a better understanding of what the traditional way of writing algebra is telling you. Then you can move forward with greater ease, assurance, and success. The alternative can be and often is to simply give up. DragginMath helps you, your child, or your students avoid that awful fate. Using this app is not a guarantee. But it is a powerful tool to help you achieve a valuable goal.

Getting Started


When you start DragginMath for the first time, there are two pale grey labels near the top of the screen. After you have used DragginMath even once, you will never see those labels again, so pay special attention to them now.

One label says “Tap Here to Learn How.” There is also an arrow pointing up toward  in the upper left corner of the screen. That  is called an Info button. These are found in many places throughout DragginMath. Perhaps you have seen these, or something like them, in other apps. They exist to explain things to you. Tap that button now.



Did you see the screen change?

Here is a whole list of interesting topics, each with its own . Tap the first  on this screen.

Did you see the screen change again?

Read this short article now on your screen. It is called “How to Read This”. If an  article is short, it may fit on one screen. If it is longer, swipe up and down on it with a finger to see the whole thing. When you finish reading, tap **OK** at the bottom.

Did you see the screen change back?

Whenever you finish reading an  page, tap **OK** at the bottom. This takes you back to whatever you were doing before you asked to read the  page. There are other places in the app where you need to tap **OK** when you are finished with some particular task, so get used to doing this.

At this point, you are back to the screen with the list of articles that describe DragginMath, how to use it, and why it is

the way it is. If you want to read more of these now, go ahead. Or you can come back and read them some other time. That information really is helpful and good to know. But this Guided Tour is probably all you need to think about right now.

Do you need to read this Guided Tour all at once? No. Let me suggest that you read *some* of this, play around with DragginMath a little to get a better understanding of what you just learned, then come back later to read more.

Tap **OK** again to get back to DragginMath's main screen. This is where you will do most of your work. Notice the grey label in the text field at the very top of the screen. It says "Tap Here to Start Working." So do that now. Whenever you are done working on a problem in DragginMath, tap the text field at the top to erase the screen and start working on the next problem.

If you are part way through a problem and then decide to do something else, just switch to another app, or even put your device to sleep. The problem you were working on will still be there when you return, just as you left it. If you *kill the app*, it won't still be there, but most people don't do that very often.

If you decide you want to kill the app, tap twice quickly on your device's home button. See small versions of the screens of all running apps. Find the small DragginMath screen, then swipe up on it. This kills the app. You can restart it any time by tapping the DragginMath icon again on your home screen.



Introducing the Screen Keyboard

When you tap in the text field at the top, a screen keyboard rises up from the bottom. It has all the characters you need to operate DragginMath. If a hardware keyboard is attached to your


iPad, you can type on that, too. A hardware keyboard can be easier to use than the screen keyboard, but it doesn't have all the characters you need. We will return to discuss more features of the screen keyboard from time to time.

Entering Expressions

Let's start with something basic: enter $2+3+4$. As you type, notice things happening elsewhere on the screen. Some kind of picture is forming there. That picture is called an *operator tree*. Most of the work you do with DragginMath involves operator trees. This app exists to let you see and work with them. Later in this Guided Tour, we will talk about how they are made and why that is useful to know.


If you happen to enter something *incorrect*, such as $2++$, the text field **flashes red** and your device (((beeps))). Notice that the incorrect character was not accepted. You may only enter expressions that are *correct*. Being correct doesn't mean you have not made a *mistake*. In this context, a mistake is when you enter a character you didn't intend, but it still follows the rules of math syntax, such as $2+6$ when you wanted $2+3$. If you make a mistake, use the  key to back up. If you decide you want to simply start over, flick your fingertip sideways off the  key to erase the whole line.

You may add or delete characters only at the end of the text field. Cutting or pasting anywhere else is not allowed. The app reads and analyzes what you enter *as you enter it*, so it would become confused if you could type anywhere other than the end.

When you finish entering your expression, tap the  key, which is like the **return** key on a hardware keyboard. This

finishes the construction of the operator tree, which then moves to the middle of the screen.

If your expression has open parentheses that you haven't closed, DragginMath quietly does that for you. But if your expression is otherwise *incomplete*, the text field **flashes red** and your device (((beeps))). Remember that you may only enter expressions that are *correct*, which includes being *complete*.

If you have an incomplete expression and you decide you are no longer interested in it, flick your finger sideways off the  key. Just as before, this erases the whole line. You may then tap the ↵ key with an empty text field, and the screen keyboard politely goes away. If you still have an incomplete expression lying around, it won't.

Talk About Trees

Everything interesting about DragginMath involves the *operator trees* mentioned in the previous section. That means your life will be easier if you learn some basic words we can use to talk about them.

A tree has a *root*, *branches*, and *leaves*. In the usual mental picture of a tree, these grow and spread upward. But tree diagrams can be drawn in other directions, too. They might be drawn growing and spreading from left to right, or from right to left, or from top to bottom. For specific uses, you might prefer one of these directions over the others. But the way a tree is *connected* doesn't change, regardless of which direction it grows, and *it is only this connectedness that matters*.

In DragginMath, trees are drawn so they grow from top to bottom. The place from which the tree grows is called the *root*,

even when it is at the top of the picture. The root is a special case of a tree *node*. A *branch* connects two nodes. In a tree, branches can only spread out: they never come back together. Nodes and branches can lead to other nodes and branches, which can lead to even more nodes and branches. The length of a branch doesn't matter in these pictures. Only the fact that a branch exists matters. So branches can be any length, and we draw them at whatever length makes the picture simple and compact. When a node has a branch coming in, but no branch going out, that node is called a *leaf*. A tree can have many leaves, but only one root.

Other words that are useful are *parent* and *child*. The root node of a tree can have children, but it has no parent. A leaf node has a parent, but it has no children. Nodes that are neither root nor leaf are sometimes called *internal nodes*. Each of these has one parent and one or more children.

Another important word is *subtree*. Any node in a tree is *the root of its own subtree*, which is a tree in its own right.

When discussing algebra in DragginMath, we will often talk about roots, nodes, and leaves, and parents and children, and subtrees, and the roots of subtrees. We need these words because algebraic expressions are in fact trees of operators and operands, even though they are traditionally written as a line of symbols. Understanding this is the start of your success in algebra. It is the reason DragginMath exists.

One of the major issues in teaching or learning algebra is *operator precedence*. This is the hierarchy of operations found in algebraic expressions. Rather than hoping you can look at a line of math symbols and see the hierarchy in it, DragginMath

simply shows you the hierarchy as a tree diagram. Then you can move its nodes around to make algebraic changes.

Introducing the Commutative Property

If you successfully entered $2+3+4$ ↵, you now have a completed operator tree. So let's do some algebra.

Put your finger on the **2**. Notice a red circle around your fingertip. The red circle tells you that DragginMath knows you are touching the **2**. Drag your finger *straight to the left* for a ways, then lift your finger. The circle disappears, and the **2** floats back to where it came from. Other than this, nothing happened. Nothing is wrong here: this is typical DragginMath behavior. If your drag gesture does not identify a legitimate algebraic change, nothing happens, and the parts that moved simply return to wherever they came from.

Put your finger on the **3**. Notice the red circle again. Drag your finger *straight to the right* for a ways, then lift your finger. The **3** floats back where it came from. Nothing happened. Once again, nothing is wrong here: that's what DragginMath does when nothing is supposed to happen.

Put your finger on the **2** again. Notice the red circle. Drag your finger *straight to the right* this time. When your fingertip passes the **3**, the red circle turns purple. DragginMath is now in Purple Mode. Being in a mode (there are four) intentionally limits the actions DragginMath can perform. This is an important part of this app's design. Once that red circle changes color (to purple in this case), it remains that color, and remains in that mode, until you lift your finger.

See that the **3** has moved to where the **2** used to be. Lift your finger and see the **2** move to where the **3** used to be. This algebraic change is now complete. What is the name of this algebraic change? This is the Commutative Property of Addition.

Look at the text field at the top of the screen. Before the change you just caused to happen, it read $2+3+4$. Now it reads $3+2+4$. **Any time DragginMath changes the structure of an operator tree, the text equivalent of its new form appears in the text field at the top.**

Put your finger on the $+$ that joins **3** and **2**. Drag right until the red circle turns purple. Lift your finger. Once again, this change is due to the Commutative Property of Addition. The text field at the top now reads $4+(3+2)$.

So DragginMath can commute simple operands of addition, such as **2**. It can also commute compound operands that are the result of other operations, such as $(3+2)$.

Introducing the Associative Property

In $4+(3+2)$, drag the $+$ that joins **3** and **2**, dropping it on the $+$ at the top. This motion requires you to *drag up*, which puts DragginMath into Blue Mode, where it can do things that Red Mode and Purple Mode cannot. Once the app enters Blue Mode, it stays in that mode until you lift your finger.

You can only drop a tree node onto another tree node when DragginMath says you are *on target*. It does this in four ways, all at once:

- 1) It clicks.
- 2) The entire screen background changes color slightly.

- 3) The drop target is boldly outlined in black.
- 4) The 🎯 icon appears in the upper left corner.

If you are on target and then your fingertip moves away, you are *off target*. DragginMath says you are off target in four ways, all at once:

- 1) It clicks, but at a lower pitch.
- 2) The background changes back to white.
- 3) The black outline around the drop target goes away.
- 4) The 🎯 icon disappears.

If you are dragging and you lift your finger when off target, whatever you were dragging will drift back to wherever it came from. As was mentioned earlier, this is normal DragginMath behavior.

When you dropped the lower + onto the upper +, did you see the change that happened? What is the name of this algebraic change? This is the Associative Property of Addition.

Look at the text field at the top of the screen. Before the change, it read $4+(3+2)$. Now it reads $4+3+2$. More important: the *structure* of the operator tree has changed. When using DragginMath, *it is the operator tree that matters*.


What Algebra Is & What DragginMath Does







Many people never gain a good understanding of what algebra *is*. With this example, we can clear that up. **Algebra is a set of tools for *changing the structure of a mathematical relationship without changing its meaning*.** That is what algebra *is*, and that is what DragginMath *does*.





At first glance, DragginMath appears to be a way to move math symbols around on the screen. But that is not what it does.



DragginMath *restructures algebraic relationships* on the screen, which is something different. It only makes changes in structure if they don't change meaning. The only actions it performs correspond *exactly* to known algebraic properties. When you ask it to change something, it either makes the *entire* change or none at all.


About Those Other Buttons

The first thing you learned about DragginMath was the  button in the upper left corner. It raises the primary info screen, which leads you to several pages that can help you understand DragginMath better. Some of that information overlaps this document, although it is organized differently. Do not assume reading this document makes those others unnecessary.

But  on the main screen isn't always there. That space is sometimes taken by , which can tell you how many moves, changes, and seconds you have used on the current problem. To see  again, tap the text field at the top to raise the screen keyboard and bring back . And that same space is used for the  icon when you drag something *on target*.  goes away again as soon as you either drop or drag *off target*.

 and  are Undo and Redo. Undo reverts to the previous state of your work. It can do this, one step at a time, all the way back to the beginning of the current problem. Don't confuse Undo  with Backspace , which only appears on the screen keyboard. Redo moves forward again, simulating what you do with your finger when you drag math around on the screen. This means you not only see the result of a step, you also see the move that caused it to happen.

 raises a screen that shows the History of your work. Here you can select the problems you have worked on recently, either from their start or from individual steps. You can edit your problem sets, even save them to files and email them. Look for  on the History screen to learn its use in detail.

When graphical user interfaces were first invented back in the 1970s, one of the first screen icons was \equiv . Traditionally, this brings up a list of software settings or configuration switches. Someone thought this symbol looked like a hamburger, so it became known as “the hamburger button” (these days, some apps use a gear icon instead). DragginMath’s hamburger button is in the upper right corner. Tap it to raise a configuration screen. Each item on that screen has a separate . Be sure to read these to understand your choices in what DragginMath can do for you. Some of these issues are simple and obvious, while others are subtle or sophisticated. The default settings make DragginMath read, write, and do math most closely to traditional notation and behavior. As you become more familiar with DragginMath, you may discover good reasons to have it do things differently.

Imagine you are already working on a problem when you discover you need another expression on the screen to help you solve it. That’s what $;$ is about. It raises the screen keyboard so you can enter more text, creating another operator tree alongside whatever you already have on the screen. Notice there is a $;$ on the keyboard, also, so you can enter more than one expression at the outset if you know you’ll need that.

More About the Screen Keyboard

If you want to be sure to make someone unhappy, all you have to do is design a keyboard. Any attempt to arrange all those little buttons into something sensible will meet with someone who disapproves of it. DragginMath is no exception.

The numeric keys are laid out just like those on any ordinary calculator. The basic operators on the side are like those on most calculators. Everything else is different.



DragginMath works on both iPad and iPhone. It does the same algebra the same way on both kinds of device. But the small screen on an iPhone makes a different keyboard layout necessary there, especially when working in the upright orientation.

On both kinds of keyboard, there are columns with colored spacers between the buttons. These columns can be swiped up and down to reveal more symbols. Look for a column of characters with \uparrow at the top (on iPad, there are two such columns). Swipe lightly up and down to see everything that column has to offer, including the special numbers π and e . As you use DragginMath, you will sometimes need to reposition this column to find the symbol you want.

You need to learn to do two things with these special columns: swipe them up and down, and tap the buttons they contain. A little practice may be required to do each of these actions reliably.

The iPad keyboard has a grid of all letters arranged in alphabetical order. The iPhone keyboard has two columns of letters that can be individually swiped up and down. In their initial positions, one column shows **abc**; the other shows **xyz**.

These are the letters used most often in algebra, but you can swipe the columns to find any other letters you might need.

 is the shift key. You will only need this for an advanced feature described later. Be sure to read the  in the lower left.

Trees Into Text and Back Again

Operator trees really do mean the same thing as traditional math notation, but some people aren't sure about that and need persuading. Also, trees take a lot of space on the screen.

You can convert any tree or subtree into linear text by flicking *up* on its root. When you do this, parentheses may be added to the text to preserve operator precedence. Convert it back to a tree by flicking *down* on it. Flick with a light touch.

Once in its text form, a node can be dragged around in that form. Most other interactions with such a node will cause it to turn back into a tree without flicking. What you can't do with a node in text form is drag or target the parts *inside* it.

More About the Associative Property

This app allows you to make experiments safely and freely. **DragginMath converts algebraic expressions into playthings.** So let's play.

In the previous example, we used the Commutative and Associative Properties of Addition to rearrange an operator tree. In this example, we will use the Associative Property of Multiplication, which works the same way as the Associative Property of Addition. In the previous example, we used the

Associative Property in a small way. In this example, we will use the Associative Property in a big way.

Tap the text field at the top of the screen, then enter **abcdefg** ↵. This expression means we could multiply these seven variables together if we knew their values. We don't know their values, but there are other things we *can* do, and writing this expression gives us something concrete to look at, talk about, and work on.

In the operator tree, notice the asterisk * symbol, which means *multiply*. Use of this symbol for this purpose originated in computer programming. DragginMath won't ask you to program computers, but it will ask you to use the asterisk symbol, which is increasingly common in mainstream math. You may use * when you enter an expression, for example, **a*b*c*d*e*f*g**. You may also use implied multiplication in many situations, for example: **abcdefg**. Regardless of how you enter an expression, you will *always* see * in operator trees containing multiplication. It is only in the text field at the top that you might not see it. Under the hamburger button ≡, there is an option setting that gives you some control over this. Read the ⓘ there to learn about it.

Touch the * attached to **f**. Drag it onto the topmost *.

Touch the * attached to **e**. Drag it onto the topmost *.

Touch the * attached to **d**. Drag it onto the topmost *.

Touch the * attached to **c**. Drag it onto the topmost *.

Touch the * attached to **b**. Drag it onto the topmost *.

After all these changes, the operator tree means the same thing it meant when we started. Notice what you didn't have to do: redraw the operator tree after each change. DragginMath does that for you, quickly and correctly.

Touch the * attached to **g**. Drag it onto the topmost *.

Touch the * attached to **a**. Drag it onto the topmost *.

Touch the * attached to **g**. Drag it onto the * attached to **d**.

Touch the * attached to **e**. Drag it onto the topmost *.

Using the Associative Property, you can make complicated operator trees that branch arbitrarily left and right. When you make sweeping association changes across multiple nodes, DragginMath tends to order them in linear chains. Still, you might have to associate more than once to fully straighten out some complicated trees.

More About the Commutative Property

You have already seen how the Commutative Property works for expressions like $2+3+4$. It also works for expressions like $a*b*c$. You expect this because addition and multiplication are commutative.

What happens if you commute $a-b$? You might expect that DragginMath will do nothing because subtraction is not commutative: $a-b \neq b-a$. But DragginMath is an environment in which you can try things to see what happens. So try commuting $a-b$ the same way you would commute $a+b$: drag one of those operands sideways past the other.

Yes, you can commute $a-b$ with a result of $-b+a$. This is sometimes called *turning subtraction into addition*. What happens if you commute this resulting addition? Try it and find out. What happens if you drag the - onto the +? Try it and find out. How do you learn if DragginMath can do something useful when you move tree nodes around in particular relationships? Here is a good general answer: *try it and find out*.

So even though subtraction is not *commutative*, it is still *commutable*: the app knows how to make all the necessary changes, safely and reliably. The word *commutable* is probably not found in any math book, but we need a word like this to talk about DragginMath. It's not that DragginMath has invented some new kind of math. But it does make some aspects of math more explicit. That's what makes it a good learning tool.

What happens if you commute $\mathbf{a} \div \mathbf{b}$? You might have conflicting expectations now. Division is not *commutative*, but can it be *commutable*? What would that look like? Try it and find out.

Yes, DragginMath can turn division into multiplication, just as it can turn subtraction into addition. The $\mathbf{1} \mathbf{b} * \mathbf{a}$ you see on your screen includes a unary division (reciprocate) operator, exactly analogous to the unary subtraction (negate) operator. This appears as $\mathbf{1}$ in the text field and in operator trees. $\mathbf{1}$ is *not the number 1*. It is an *operator*, which means that it *does something*. This unary division operator divides a number into 1, just as the unary subtraction operator subtracts a number from 0.

Perhaps you have never thought about the existence of a unary division operator before. This is one of several things our traditional math notation obscures from us. If you are already experienced with algebra, you have used this idea many times, perhaps without realizing it. In DragginMath, it can't hide any longer. If it weren't there, you would notice that and wonder why it didn't exist. Now it does.

Can you enter this operator from the keyboard? Yes, you can. Think about how you enter a negate operator from the keyboard: DragginMath knows from context whether $-$ is

negate or *subtract*. It also knows from context whether \div is *reciprocate* or *divide*.

Incidentally, if you have a hardware keyboard attached to your iPad, you can use the slash / character for either kind of division. It automatically converts to \div or $\frac{1}{}$ at the right time.

Still More About the Associative Property

You have already seen how the Associative Property works for expressions like $2+3+4$. It also works for expressions like $a*b*c$. You expect this because addition and multiplication are associative.

If what you just read sounds familiar, maybe even repetitive, there is good reason for that.

What happens if you reassociate $2-3-4$? You might expect that DragginMath will do nothing because subtraction is not associative: $2-3-4 \neq 2-(3-4)$. But try reassociating $2-3-4$ the same way you would reassociate $2+3+4$: drag the lower $-$ up into Blue Mode, then drop it on the upper $-$.

Yes, you can reassociate $2-3-4$ with a result of $2-(3+4)$. In the previous discussion of commuting $3-2$, we could talk about *turning subtraction into addition* even though subtraction is not commutative. As far as I know, there is no name or phrase for what we just did with $2-3-4$. People who learn algebra well know this can be done, but we don't have a convenient name or phrase to talk about it.

So even though subtraction is not *associative*, it is still *associable*: the app knows how to make all the necessary changes, safely and reliably, for all combinations of $+$ and $-$. This works even when they appear together in long chains, such

as $2+3-4+5-6+7-8$. This word *associable* is probably not found in any math book, but we need a word like this to talk about DragginMath. Once again, it's not that DragginMath has invented some new kind of math. But it does make some aspects of math more explicit.

What happens if you reassociate $2\div 3\div 4$? Division is not *associative*, but can it be *associable*? What would that look like? Try it and find out.

Yes, DragginMath can do this, too, and with expressions much more complicated than this. After your last several experiences with the app, perhaps this is no longer a surprise.

Introducing the Distributive Property

We started with the associative and commutative behavior of DragginMath because these are the foundations of algebra, which also means they are the foundations of the behavior of the universe. Really.

What's next? Let's look at the Distributive Property.

You know that $3*9 = 27$. Observing this fact from a different angle, this also means that $3*(4+5) = 27$. Drag the + up into Blue Mode, then drop it on the *. DragginMath just told you that $3*(4+5)$ means the same thing as $3*4+3*5$. This works for any numbers: $\mathbf{a(b+c) = ab+ac}$.

The full name for this is the Distributive Property of *Multiplication Over Addition*. It also works for multiplication over subtraction: $\mathbf{a(b-c)}$. Try it to see how that works. It also works for division *under* addition or subtraction: $\mathbf{(a+b)\div c}$, but not for division *over* addition or subtraction: $\mathbf{a\div(b+c)}$. Try both of these to verify that the first one does something, but the

second one does nothing. Is there a Distributive Property of *Addition Over Multiplication*? Enter $\mathbf{a+(b*c)}$, then drag * up onto + to find out.

Distribution can be invoked in two different ways: *general* and *specific*. The general way assumes you want operands distributed as far as they can be. The specific way assumes you want them distributed only to specific places you will indicate.

To see general distribution, enter $\mathbf{a(b+c+(d+ef))}$. Drag the topmost + onto the * attached to **a**. DragginMath looks under that topmost + for any other addition or subtraction operators, all the way down all branches of the tree, and distributes to their operands. If it encounters any other kinds of operators on its way down the branches of the tree, it stops. With general distribution, DragginMath does all of the work, finding *all* distributions that can *correctly* be done in that subtree of operators. General distribution happens when you drag up from the *topmost* + or – under * or ÷.

To see specific distribution, enter $\mathbf{a(b+c+(d+ef))}$. Drag the + attached to **b** (not **b** itself) onto the * attached to **a**. In this case, you have told DragginMath where to *stop* distributing. It will distribute down to that node, *but no farther*. To do this correctly, it may have to distribute to side branches also, but not inside them. For example, notice that the distribution of **a** did not get inside the $\mathbf{(d+ef)}$ branch for this specific distribution, but it did for the general form.

General distribution is usually what you want. But sometimes it is too much. In those cases, specific distribution can do exactly what you want, but it sometimes takes several steps to do it.

DragginMath can do multi-factor distributions. For example, enter $\mathbf{ab(c+d-e)\div f}$. Look carefully at the structure of that tree. Drag $-$ up onto \div . Now look carefully at the structure of the result.

We have been discussing the Distributive Property of Multiplication (or Division) over (or under) Addition (or Subtraction). There are other kinds of distributive operations, but they aren't usually called that. Many go under the general heading of *Rules of Exponents*, which we will encounter soon. Others have no generic name that I know of. Whether they have names or not, DragginMath can do both general and specific distribution for them. The drag-and-drop invocations follow the same patterns. Some can do multi-factor distributions, while others cannot. For those that cannot *now*, some might in the future, while others won't ever because the math doesn't work that way. In any case, if you want to know if DragginMath can do something, think up an example and *try it*.

Introducing Factoring

If you can distribute things in DragginMath, can you also un-distribute them? Yes, you can. Of course, a better name for this is *factoring*. It comes in different forms, depending on what you want to factor, and what you want to factor it out of.

Enter $\mathbf{x+x}$. Now drag either one of the \mathbf{x} s *up* to deliberately enter Blue Mode. If you aren't in Blue Mode, you will probably just commute the \mathbf{x} s, which is not the goal here. But in Blue Mode, drop one \mathbf{x} onto the other \mathbf{x} , and the result is $\mathbf{2x}$.

That was a start. Let's do something more interesting. Enter $\mathbf{ax+ax+ax+ax+ax}$. Drag the bottom-left $\mathbf{*}$ onto the topmost $\mathbf{*}$. In

this example, you don't have to deliberately enter Blue Mode: it will just happen if you make the direct motion. The result is $5ax$.

Inside DragginMath, this is called *factoring by counting*.

What if you have $xa+xb$ and you want to factor out the x ? Inside DragginMath, this is done by a completely different process (*factoring by extraction*), but it is invoked the same way. Once again, drag the bottom-left x up into Blue Mode, then drop it onto the other x . The result is $x(a+b)$.

What about $xa+xb-xc+xd$? Drag the bottom-left x onto the topmost x . The result is $x(a+b-c+d)$. Notice that the xb and xc terms were factored also, even though you didn't actually interact with them. That's because they were *between* the two terms you did interact with. If DragginMath encounters a term that does not contain the factor you asked to extract, the operation fails and the parts go back where they came from.

DragginMath can factor expressions much more complicated than this.

Paying attention to some details here can prevent frustration later on. To invoke factoring, *you must be in Blue Mode when you drop*. That might mean you have to drag up deliberately, if only to enter the mode. And here is another critical point. Most Blue Mode operations are invoked by dragging something up and dropping it onto the *root* of an operator subtree. Factoring is different. To invoke factoring, you must drop something onto an *off-root* node that is *recognizably equal*.

I know the language in the previous paragraph is kind of dense. I don't know a simpler way to say it. Even complicated factoring operations are easy once you know how. But factoring is the most challenging aspect of learning to use DragginMath.

Introducing Cancelling

Some of the things DragginMath does really are obvious. Some are obvious once you've seen them. A few things have to be explained and thought about a little. Factoring is one of these. Cancelling is another. The explanation that follows tells you how to cancel in DragginMath. It also gives you some understanding of *why* it was implemented this way.

Enter $\mathbf{a+b}$. You know how to commute this addition: just drag one operand sideways past the other. You can see the *structure* of the expression has changed: $\mathbf{b+a}$. But you know that the *meaning* of the expression hasn't changed because addition is commutative.

Enter $\mathbf{a+a}$. Commute it. You know that the meaning of the expression hasn't changed. But in this case, you can't even see that its structure changed. As far as you can tell, commuting $\mathbf{a+a}$ *does nothing*, as if it didn't happen.

Earlier, we talked about commuting subtraction. This is not so simple, and DragginMath has to do other things to make subtraction commutable. That is, except for one special case: when subtracting something from itself. We know that $\mathbf{a-b} = \mathbf{-b+a}$. But $\mathbf{a-a} = \mathbf{a-a}$ regardless of what \mathbf{a} is or the order in which we write it. So commuting a subtraction of things that are equal *does nothing*. But we also know that a subtraction of things that are equal *always has the same result: 0*.

Cancelling the subtraction of equals is something that DragginMath needs to do. And there has to be a unique way for you to tell DragginMath to do it. So DragginMath uses this

operation that would otherwise *do nothing to do something special*.

Commuting a subtraction of things that are equal cancels them into 0.

Similarly, commuting a division of things that are equal cancels them into 1.

If the previous paragraphs seem complicated and pointless to you, just remember these last sentences about commuting subtraction and division of things that are equal.

What if the operands of these subtractions and divisions are not simple? What if they are complicated, like **$ab-ab$** ? That works, too. What if they are equal but not written the same, like **$ab-ba$** or **$(ab+c)-(c+ba)$** ? Those work, too. If the Commutative Property is the only thing needed to show these operands are truly equal, DragginMath will figure it out and do what you want. If some other algebraic properties are needed to show the operands are equal, you will have to help DragginMath by doing those operations yourself before cancelling.

Introducing Replication

When thinking about arithmetic, we know that $3*4 = 12$. When thinking about algebra, we know that $3*4 = 3+3+3+3 = 4+4+4$. This is called *replication*. We sometimes need this to solve algebra problems.

Enter **$3a$** . Drag **3** up onto *****. The result is **$2a+a$** . Drag **2** up onto *****. The result is **$a+a+a$** .

Enter **$(a-b)4$** . Drag **4** up onto *****. The result is **$(a-b)+(a-b)3$** . You know what to do next. You can take this as far as needed.

There will be more to say about replication later.

Introducing the Rules of Signs

Enter $- - - - -a$. This picture looks different from the others we have seen. This is less of a tree and more of a stalk, because it is composed of only *unary* operators. Drag **a** onto the topmost operator. The result is $-a$. This is because *two negatives are the same as nothing at all*. In this example, two separate pairs of negate operators cancelled each other out, and only one negate remains.

Enter $\div \div \div \div \div a$. This becomes $\frac{1}{\frac{1}{\frac{1}{\frac{1}{\frac{1}{a}}}}}$. Drag **a** onto the topmost operator. The result is $\frac{1}{a}$. This is because *two reciprocals are the same as nothing at all*. In this example, two separate pairs of reciprocate operators cancelled each other out, and only one reciprocate remains.

Enter $- \frac{1}{- \frac{1}{\frac{1}{- - \frac{1}{a}}}}$. Drag **a** onto the topmost operator. The result is **a**. In this example, all of these *sign operators* cancelled each other out. They don't need to be grouped together by kind in order to do that.

Whenever you ask DragginMath to do *anything* in an operator tree, it *first* tries to simplify the sign operators in that part of the tree. Then it tries to do the thing you wanted.

Enter $- \frac{1}{a}$. Drag $\frac{1}{}$ up onto $-$. The result is $\frac{1}{-a}$. Now drag $-$ up onto $\frac{1}{}$. The result is $-\frac{1}{a}$.

Enter $-(ab)$. Drag **a** onto $-$. The result is $-a*b$.

Enter $-(ab)$ again. Drag **b** onto $-$. The result is $a*-b$.

Here you see that DragginMath can move sign operators into various places you might want them to be. This is critical to

the solution of some equations. DragginMath won't move sign operators into places it knows they don't belong.

Negate - and reciprocate $\frac{1}{}$ are not the only sign operators DragginMath knows about. Another is plus-and-minus \pm . This coalesces with itself and with negate, so $\pm - \pm a$ reduces simply to $\pm a$. You can move these operators around also, but as with the other sign operators, only to the right places. Is this useful? Yes. For example, if you use DragginMath to derive the Quadratic Formula, you will need this to get to the traditional form. If you ever derived $(-b \pm \sqrt{b^2 - 4ac}) \div (2a)$ on paper, you might not have noticed this little thing when you did it.

Introducing Raise, Root, and Log

In traditional notation, *x to the second power*, or *x squared*, is written x^2 . In other words, *there is no visible exponentiation operator symbol*. Instead, there is this *typographic convention*. When you see characters written this way, you are expected to *infer* the existence of an operator.

Working with pen and paper, this is not a bad way of writing math. It might even be a really good way... *with pen and paper*. But when working with a keyboard-driven computer, it is really not good.

Traditional notation becomes even more of a problem when writing roots. For anything but the square root, tiny superscript characters are expected to be tucked inside the crook of the radical $\sqrt{}$ character, as in $\sqrt[3]{}$. Writing logarithms the traditional way is similarly complicated.

All of this places annoying limits on what can be written with a keyboard, even when you have a computer that can draw

such characters on the screen (older computers couldn't). Sophisticated software exists (L^AT_EX for example) to help you write traditional math notation, but that software is about *writing the characters*, not *doing the math*.

DragginMath is about *doing the math*. For you to do the math easily, I had to make a few changes to traditional math notation. This was not undertaken lightly. I understand the magnitude of what I am asking, and the consequences. The choice came down to *Write it the traditional way* vs. *Do a **lot** more math a **lot** more simply*. You can try to do both in the same app if you want. Good luck. After thinking about this problem for weeks, I chose to just do the math.

Are these changes huge? No. You will barely notice some of them. Two require greater attention. Translating back and forth by a human is easy once it is explained.

DragginMath has a visible exponentiation operator. It is called *raise*. Its symbol is \uparrow . If a hardware keyboard is attached to your iPad, you can also use a *caret* \wedge (shift-6), which becomes \uparrow at the appropriate time.

$a\uparrow b$ is pronounced "a raise b".

This kind of notation is not new. You can find it in some programming languages, all spreadsheets, and Google's calculator feature.

Traditional math notation uses superscripted exponents, such as x^y . Superscripted exponents of any complexity are difficult to write, especially on a computer. DragginMath doesn't write any superscripted exponents, and you don't either. If the left operand is complicated, put it in parentheses. If the right operand is complicated, put it in parentheses. In DragginMath, your focus is on the operator tree anyway.

\uparrow also has a unary form, written $\uparrow x$. Its implicit base is e . This is equivalent to the exponential function e^x .

$a\sqrt{b}$ is pronounced “a root b”.

This is the greatest divergence from traditional notation, in which $a\sqrt{b}$ means *a times the square root of b*. In DragginMath, $a\sqrt{b}$ means *the a root of b*. **There is no way around this that does not fundamentally cripple DragginMath.** If you mean to write *a times the square root of b*, write $a*2\sqrt{b}$ or $a*\sqrt{b}$.

Traditional math notation uses superscripted radicals, such as $\sqrt[3]{}$. Superscripted radicals of any complexity are difficult to write, especially on a computer. DragginMath doesn't write any superscripted radicals, and you don't either. If the left operand is complicated, put it in parentheses. If the right operand is complicated, put it in parentheses. In DragginMath, your focus is on the operator tree anyway.

$\sqrt{}$ also has a unary form, written \sqrt{x} . Its implicit root is 2. This is equivalent to the traditional square root \sqrt{x} .

$a\downarrow b$ is pronounced “a log b”.

This is the other significant divergence from traditional notation. $a\downarrow b$ means $\log_a b$. The symbol makes sense because $\log\downarrow$ is the inverse of $\text{raise}\uparrow$.

Traditional math notation uses subscripted bases, such as $a_{}$. Subscripted bases of any complexity are difficult to write, especially on a computer. DragginMath doesn't write any subscripted bases, and you don't either. If the left operand is complicated, put it in parentheses. If the right operand is complicated, put it in parentheses. In DragginMath, your focus is on the operator tree anyway.

\downarrow also has a unary form, written $\downarrow x$. Its implicit base is e . This is equivalent to the natural log function $\ln x$.

Earlier versions of DragginMath wrote logs differently. The current way described here is more useful to humans.

All of these operators, whether binary or unary, have the same operator precedence: higher than multiplication and lower than the sign operators.

Unlike the four arithmetic operators $+ - * \div$, which are left associative, it is traditional for *raise* to be right associative. By default, this is the case in DragginMath, too, but it can be reconfigured to left associative because spreadsheets work that way. This configuration affects all of these operators $\uparrow \sqrt{} \downarrow$, both binary and unary. If your DragginMath is configured one way, but you need the other way for a particular expression, writing parentheses can always give you what you need.

Reconfiguring associativity affects how expressions are written, but it has no effect on operator trees. Because the text at the top of the screen is regenerated out of the operator trees, you can flip this configuration back and forth, and the text form of your expressions will always be correct for the current configuration.

Obvious Simplifications

Some things are just easy. Why should we make them harder? Obviously, we shouldn't.

Enter **a+0**. Drag **0** up onto **+**. Obvious, yes?

Enter **a*1**. Drag **1** up onto *****. Obvious, yes?

Enter **0÷3**. Drag **0** up onto **÷**. Obvious, yes?

Enter **5↑1**. Drag **1** up onto **↑**. Obvious, yes?

DragginMath knows a lot of special cases like this, and it handles them directly. But sometimes *you* must be careful. For example, enter $0 \div x$. Drag 0 up onto \div . This may be obvious, but it is not always correct: what if x is zero? As you will learn later, DragginMath doesn't really object to dividing by zero. But to do the right thing in that rude case, DragginMath must *know* it is dividing by zero. It doesn't know that here, and you don't either. *You* must take note of this as you solve a problem. If anything special must happen because of this, *you* must do it. This app is your *assistant*, not your *replacement*.

Converting Binary and Unary Equivalents

DragginMath has lots of binary and unary operators. Most of these go together in pairs. For example, unary minus (negate) is the same as subtracting from zero. Unary divide (reciprocate) is the same as dividing into one. And raise, root, and log all have both binary and unary forms.

If you have a unary form and you need the corresponding binary form, *flick down* on it (this may require a little practice). The binary form appears with the correct default operand. For example, flicking down on the $-$ in $-x$ changes it into $0-x$.

Or you might need to go the other direction. If you have a binary form with its default unary operand, drag the unary operand up onto the operator. The corresponding unary form appears. For example, $0-x$ becomes $-x$ when you drag 0 onto $-$.

These moves can be useful when factoring or cancelling: DragginMath tries to recognize when things that are not very different are actually the same, but sometimes it needs a little help from you.

More About Replication

Earlier, we saw that multiplication can be converted into addition by replication. For example, enter $3x$, then drag 3 onto $*$ to see $2x+x$. Repeat the process as far as needed.

This also works for raise. For example, enter x^3 , then drag 3 onto \uparrow to see $x*x^2$. Repeat the process as far as needed.

This also works for addition. For example, enter $3+x$, then drag 3 onto $+$ to see $2+1+x$. Once again, repeat the process as far as needed. Yes, this is useful, and sometimes necessary.

Expanding Numbers

Sometimes a number must be broken into smaller parts to solve a problem. Double-tap a number to expand it into prime factors in the operator tree. Future versions will have this and other options for breaking a number into parts.

Rules of Exponents

The operators $\uparrow \sqrt{} \downarrow$ can be commuted, associated, distributed, and factored in various ways. Although these words are not traditionally used in this context, I use them here because it is hard to see why they are not, at least in a casual sense. Traditional language places all of these things under the general heading *Rules of Exponents*.

$$a \uparrow b = 1/\ b \sqrt{a}$$

$$a \uparrow 1/\ b = b \sqrt{a}$$

$$a \uparrow (b+c) = a \uparrow b * a \uparrow c$$

$$a \uparrow (b-c) = a \uparrow b \div a \uparrow c$$

$$a \uparrow (b*c) = (a \uparrow b) \uparrow c$$

$$a \uparrow (b \div c) = c \sqrt{a \uparrow b}$$

$$(a*b) \uparrow c = a \uparrow c * b \uparrow c$$

$$(a \div b) \uparrow c = a \uparrow c \div b \uparrow c$$

$$(a \uparrow b) \uparrow c = a \uparrow (b*c)$$

$$(a \sqrt{b}) \uparrow c = a \sqrt{(b \uparrow c)}$$

$$a \uparrow b * a \uparrow c = a \uparrow (b+c)$$

$$a \uparrow b \div a \uparrow c = a \uparrow (b-c)$$

$$a \uparrow c * b \uparrow c = (a*b) \uparrow c$$

$$a \uparrow c \div b \uparrow c = (a \div b) \uparrow c$$

by dragging **a** or **b** sideways.

by dragging **a** or **1/** sideways.

by dragging **+** up onto **↑**.

by dragging **-** up onto **↑**.

by dragging ***** up onto **↑**.

by dragging **÷** up onto **↑**.

by dragging ***** up onto **↑**.

by dragging **÷** up onto **↑**.

by dragging **↑** up onto **↑**.

by dragging **↑** up onto **↑**.

by dragging **a** up onto **a**.

by dragging **a** up onto **a**.

by dragging **c** up onto **c**.

by dragging **c** up onto **c**.

$$a \sqrt{b} = b \uparrow 1/\ a$$

$$1/\ a \sqrt{b} = b \uparrow a$$

$$a \sqrt{(b*c)} = a \sqrt{b} * a \sqrt{c}$$

$$a \sqrt{(b \div c)} = a \sqrt{b} \div a \sqrt{c}$$

$$a \sqrt{b} \uparrow c = (a \sqrt{b}) \uparrow c$$

$$a \sqrt{b} \sqrt{c} = (a*b) \sqrt{c}$$

$$(a*b) \sqrt{c} = a \sqrt{b} \sqrt{c}$$

$$(a \div b) \sqrt{c} = a \sqrt{c} \uparrow b$$

$$a \sqrt{b} * a \sqrt{c} = a \sqrt{(b*c)}$$

$$a \sqrt{b} \div a \sqrt{c} = a \sqrt{(b \div c)}$$

$$a \sqrt{c} * b \sqrt{c} = 1/\ (1/\ a + 1/\ b) \sqrt{c}$$

$$a \sqrt{c} \div b \sqrt{c} = 1/\ (1/\ a - 1/\ b) \sqrt{c}$$

by dragging **a** or **b** sideways.

by dragging **1/** or **b** sideways.

by dragging ***** up onto **√**.

by dragging **÷** up onto **√**.

by dragging **↑** up onto **√**.

by dragging **√** up onto **√**.

by dragging ***** up onto **√**.

by dragging **÷** up onto **√**.

by dragging **a** up onto **a**.

by dragging **a** up onto **a**.

by dragging **c** up onto **c**.

by dragging **c** up onto **c**.

$a \downarrow b = 1/ (b \downarrow a)$	by dragging a or b sideways.
$1/ (a \downarrow b) = b \downarrow a$	by dragging a or b sideways.
$a \downarrow (b * c) = a \downarrow b + a \downarrow c$	by dragging * up onto ↓.
$a \downarrow (b \div c) = a \downarrow b - a \downarrow c$	by dragging ÷ up onto ↓.
$a \downarrow b \uparrow c = a \downarrow b * c$	by dragging ↑ up onto ↓.
$a \downarrow b \sqrt{c} = a \downarrow c \div b$	by dragging √ up onto ↓.
$a \downarrow b * c = a \downarrow b \uparrow c$	by dragging ↓ up onto *.
$a \downarrow b \div c = a \downarrow c \sqrt{b}$	by dragging ↓ up onto ÷.
$a \downarrow b + a \downarrow c = a \downarrow (b * c)$	by dragging a up onto a .
$a \downarrow b - a \downarrow c = a \downarrow (b \div c)$	by dragging a up onto a .

There are so many of these, I might easily have missed some, either when making this list, or when implementing DragginMath. If you know a transformation that DragginMath doesn't do, please tell me about it.

If you think DragginMath might be able to do things with raise, root, and log that are more complicated than these basic examples, you would be right about that. For example, do you remember the different ways of invoking general and specific distribution of multiplication over addition? Those work here, too. Enter $a \uparrow (b + c * d)$, then drag + up onto ↑. The result is $a \uparrow b * (a \uparrow c) \uparrow d$. But if you drag **b** or * up onto ↑, the result is $a \uparrow b * a \uparrow (cd)$. The first example seeks out *everything* that can work in that subtree, but the second example *stops* where you tell it.

As always, if you want to know if DragginMath does something, think up an example and *try it*.

Introducing Absolute Value

If you ever make a list of the things in traditional math notation that never should have happened, be sure to put the $|x|$ representation of absolute value at the top of your list. Yes, we need the idea and some way to write it. But it is interesting that *no programming language has ever tried to implement this notation*. They always go about writing it a different way. There are good reasons for that. For example, consider the traditional expression

$$|a-2|b+3|c|4-d||$$

There are two ways to interpret this, and there is no generally accepted rule for deciding which to use. You might blithely assert that people don't or shouldn't write expressions like that, but a tool like DragginMath doesn't get to make such assertions. It must reliably do something sensible, and I must be able to tell you why.

DragginMath puts absolute value in the same syntax category as negate and reciprocate: a prefixed unary operator. Its symbol is **||**. As with all unary operators, you can apply it to either a simple operand or to complicated things inside parentheses.

While DragginMath *reads* absolute value *only in this way*, you can choose how it *writes* absolute value: either the same way it reads it (for example **||x**), or the way traditional math writes it (for example $|x|$). The traditional way is the default.

Perhaps it seems that, if it can write the traditional way, it should be able to read that way, too. Unfortunately, computer code that reads is completely different from computer code that writes: they have nothing to do with each other, and each has its own set of limitations to deal with.

Just as redundant signs are always cleaned out of an expression before DragginMath does anything else with it, redundant absolute values are cleaned out, too. And just as DragginMath can move sign operators around to equivalent places, it can move absolute values, too.


Doing Arithmetic

Yes, DragginMath does arithmetic. But it is not a calculator in the ordinary sense. The screen keyboard does not even have a decimal point. This is not an oversight. By design, DragginMath does integer arithmetic *only*. This is appropriate for a tool whose main focus is algebra. When doing arithmetic, DragginMath may do things that surprise you, but they won't surprise your algebra teacher... much.

Up to this point, we have seen Red Mode, Purple Mode (commute and cancel), and Blue Mode (most other algebra). Now we will use Green Mode, whose main purpose is doing arithmetic.

Enter $2*3+8\div 2$. Drag **3** *down* until it turns green. Now drag it *up* onto $*$ and drop. The result is $6+8\div 2$. Drag **8** *down* until it turns green. Now drag it *up* onto \div and drop. The result is $6+4$. Drag **6** *down* until it turns green. Now drag it *up* onto $+$ and drop. The result is **10**.

This process allows you to see as much arithmetic as you need to be sure you understand the result. But most of us understand arithmetic pretty well, and doing one operation at a time is tedious. So let's try something that goes a little faster.

Use  to go back to the beginning of this exercise. Drag *any* of the numbers in $2*3+8\div 2$ down until it turns green. Now drag it up onto $+$. The result is **10**. You can evaluate a subtree of any size this way. This kind of action is typical of DragginMath operations.

But there are even more efficient ways if you care to learn.

Undo back to the beginning again. Drag **3** down until it turns green. Now drag it back up onto **8**. The result is **10**.

What just happened here? DragginMath evaluated the *smallest common subtree* between **3** and **8**. In this case, that is *everything under the $+$* . As far as DragginMath is concerned, this is the same as what you did before, because $+$ subsumes everything under it, including the **8**. You just didn't know that was the reason. If that doesn't make sense to you, don't worry about it. You have choices in how to make arithmetic happen, and you can choose whichever one you like best.

Enter $2*3*4$. Drag the **4** down onto the **3**. The result is **24**. Once again, DragginMath evaluated the smallest common subtree between **3** and **4**. But in this case, you didn't have to *drag down, then back up*. You could just drag directly from one operand to the other. The direction happened to be downward, so you entered Green Mode without any extra motion. Things often just happen to work out that way. But due to the shape of some operator trees, these *Green Mode For Free* moves are not always possible. With a little practice, you will soon become adept at

invoking arithmetic with the smallest motion, whatever that happens to be. And even if you don't, the results are the same.

The Nature of DragginMath Arithmetic

DragginMath only does integer arithmetic. There are no decimals. There are no fractions. If there were fractions, they would be unit fractions only (look for this in a future release).


The result of addition, subtraction, and multiplication of integers is known in advance: it will always be another integer. But the result of division might not be. If you evaluate $12 \div 4$, the result is 3 , an integer. But if you evaluate $4 \div 12$, the result is $1 \div 3$. This is as far as DragginMath will go with this. For classical algebra problems, *this is the answer*. Numbers like 0.333333333 are of no interest here. Understanding why $1 \div 3$ is the answer here is one of the more challenging things students have to learn.

When you ask DragginMath to divide, the result always comes to you in lowest terms. You don't have to ask for this. DragginMath just does it. Even complicated fractional arithmetic is easy. For example, enter $3 \div 5 + 4 \div 7 - 2 \div 3 - 3 \div 105$. Drag 105 down onto 5 . The result is $10 \div 21$.

Division is not the only troublemaker when it comes to arithmetic. Root and log cause their own problems. For example, enter and evaluate $\sqrt{49}$. The result is 7 . Enter and evaluate $\sqrt{50}$. The result is $5 * \sqrt{2}$ (note the $*$, which is optional in traditional notation, but necessary in DragginMath). Enter and evaluate $\sqrt{51}$. The result is $\sqrt{51}$. All of these results are correct in the context of algebra.

These nice integer results happen efficiently because DragginMath keeps a Factorization Cache. Whenever it needs to

divide, reduce roots, or evaluate logarithms, it looks in the Cache for the necessary factors, then uses them the same way you would if you were doing this work on paper. The Cache computes and then remembers all factors your work actually needs. When a factorization is needed again, it is already there. The Cache expands as needed, constrained only by the amount of memory in your device. In extreme circumstances, you may be aware of a small delay caused by Cache expansion. Usually, you will not notice. It is possible to kill this app by working with numbers so large that the resulting Cache won't fit in your machine. On our test machines, this requires numbers greater than 100,000,000. Your results may vary. Of all the things DragginMath *could* do to contain this risk (it is *not* a bug), simply letting it die suddenly when the Cache becomes too large might be the least obnoxious. Sorry, but this is a case of raw reality overriding all other considerations.

There are other problem areas in arithmetic. For example, what is $1 \div 0$? What is $0 \div 0$? These are questions that classical algebra doesn't like to answer, and with good reason. But computer programmers need answers to these questions, also with good reason. DragginMath has answers. They are derived from the IEEE-754 Math Standard, which has been tucked away inside most computers since the 1990s. The problems classical algebra has with these questions haven't gone away, but even beginning mathematicians need to be aware of these issues and what the modern world is doing about them. To learn more, see the article "About Writing Math on a Computer" found on DragginMath's main  page, or on the brising.com website.

What about $\sqrt{-1}$? Look for complex arithmetic in a future version of DragginMath.

More About Signs

Traditional notation uses the dash – in three different ways: subtraction, negation, and negativity. Subtraction and negation are *operators*, but negativity is a *property* of actual numbers.

Sometimes you need to separate a number from its sign. Flick down on a negative number to do this.

Sometimes you need to combine a number with a sign operator. Drag the number up onto the operator to do this.

Evaluation Does More Than Arithmetic

It would be easy to say that Green Mode exists only to do arithmetic, but it does other things, too. If you are the curious type, this is an opportunity to explore the relationship between arithmetic *evaluation* and algebraic *expansion*.

Enter $(x+y)(x-y)$. You could use the Distributive Property (remember: that's in Blue Mode) to expand this expression by dragging either + or – up onto *. Instead of doing that, you can drag an x down into Green Mode, then drop it on the other x . The result may surprise you, but it is correct. No arithmetic is performed, but DragginMath's evaluation strategy can deliver results like this in many situations. Try evaluating $(a+b)\uparrow 3$ by dragging 3 down onto b . As Steve Jobs used to say in his demos:

Blammo!

Screen Management

The result of the last example, evaluating $(a+b)^3$, uses more screen space than many devices have to offer.

One way to deal with this is to switch to the small font, which is done via the \equiv button. The change is immediate upon returning from the \equiv screen. If you use an iPhone instead of an iPad, the small font is the only one available. This is intentional.

You can also move an operator tree as a whole by dragging its root. Be aware that some DragginMath actions cause your trees to redraw at the upper edge of the screen, so your careful placements may be in vain. Perhaps a future version of DragginMath won't do this.

You can also move all trees on the screen by dragging your finger on whitespace, assuming there is any.

You can also cause all trees on the screen to line up in the upper left corner by double-tapping on whitespace.

If your work involves several expressions on the screen at the same time, you might not need all of them all of the time. If an expression is no longer useful, while others on the screen are still necessary, flick left on the root of the tree you no longer need. A dialog then asks if you really want to discard it.


Solving Equations

After all this time, you are probably wondering how to solve an equation using DragginMath.

Enter $3x+4=10$. Drag 4 up onto $=$. The result is $3x=10-4$. Evaluate $10-4$. The result is $3x=6$. Drag 3 up onto $=$. The result is $x=6\div 3$. Evaluate $6\div 3$. The result is $x=2$.

The various Properties of Equality at the foundation of “solving for x” are all invoked by dropping things on $=$. What

DragginMath then does internally is equivalent to the classic mantra of algebra: “*Do the same thing to both sides.*” You don’t see that dual action happening, but you do see the result. That’s all you need to know to solve in DragginMath. Really.

Use  to rewind the previous example to the beginning. Let’s do it a little faster this time. Starting from $3x+4=10$, drag **3** up onto $=$, then drag **3** down onto **4**. The result is still $x=2$.

In complicated examples, some people have a hard time remembering what to drag onto $=$. This analogy might help.

Imagine there is a loose pile of sticks in your yard. It is quite a pile, maybe several feet high. And for some reason, you drop your phone onto the pile of sticks. It falls farther and farther into the pile, until finally you see it through the gaps in all those sticks, lying on the ground. Of course, you want to get it back. There are two ways you might try to do this:

- 1) Reach down into the pile, hoping your arm is long enough to grab your phone down there on the ground.
- 2) Pull the sticks away, one or several at a time, until your phone is left by itself with no more sticks around or over it.

When solving equations in DragginMath, we use the second method: pull everything else *away* from what you want to find. Whatever you pull away, drag it up in Blue Mode and drop it on $=$. The relevant operators will be inverted onto the other side. You can do this one at a time, or several at a time. DragginMath is good at this.

There is one possible complication. In a minimal example, let’s say you have $x=2$. Drag **2** up onto $=$. DragginMath shows a dialog asking if you want to use subtraction or division to make this change, or maybe if you want to cancel the change. If you choose subtraction, the result is $x-2=0$. If you choose division,

the result is $x \div 2 = 1$. Sometimes, this is useful. Usually, it means you tried to pick up the whole pile of sticks and walk away with it when you didn't really intend to do that. If this is what happened, choose to cancel the change and think about what you really meant to do. In this minimal example, there isn't anything else you *can* do.

Full vs. Partial Solution

Enter $x+2=5$, then drag **2** up onto $=$. The result is $x=5-2$. But what about “*Do the same thing to both sides*”? It appears that DragginMath did two *different* things, one to each side.

That's because, by default, DragginMath likes to clean up after itself. Would you like to see the whole process? You can. Tap the hamburger button \equiv , then choose **Solve Part** and tap **OK**. Let's solve this equation again.

Enter $x+2=5$, then drag **2** up onto $=$ to get $x+2-2=5-2$. Now you can see that DragginMath subtracted 2 from *both* sides. On the left side, drag $+$ up onto $-$. This associates the **2**s together as $x+(2-2)=5-2$. Drag either of the **2**s sideways past the other to cancel into $x+0=5-2$. Drag **0** up onto $+$ to simplify as $x=5-2$. You will quickly tire of doing these repetitive cleanup steps, but you must understand that they need to happen.

When you first install DragginMath, it assumes you want to **Solve Full**, where equations are automatically cleaned up after each move. But if you are truly new to algebra, you may want to use **Solve Part** for a while so you can see and be involved with everything that has to happen for algebra to work correctly. Be sure to clean up *immediately* after each move involving $=$. If you don't, your equations will quickly become too complicated to

understand. After you become comfortable with this aspect of algebra, switch to **Solve Full**, but always remember these steps that DragginMath is doing for you.

More Absolute Value & Plus-and-Minus

Solving equations involving absolute value can be tricky. So it is helpful to have a working relationship between absolute value $\|$ and plus-and-minus \pm . In DragginMath, if you drag $\|$ onto $=$, it becomes \pm on the other side. If you drag \pm onto $=$, it becomes $\|$ on the other side.

How does DragginMath evaluate $\|$? First, it eliminates any $-$ or \pm or $\|$ operators within reach, as they no longer have any effect under $\|$. Whatever remains under the $\|$ is evaluated. If the result is a negative number, it is converted to positive. Anything else is left as it is.

How does DragginMath evaluate \pm ? It doesn't. Evaluating an expression containing \pm may result in a simpler expression, but it will still contain \pm . To move beyond this, *double-tap* \pm in the operator tree. The result is *two* operator trees, separate and almost equal: one for the positive case, and one for the negative case. You may then proceed to evaluate the separate cases.

Still More About Signs

Mentioned earlier: **whenever you ask DragginMath to do *anything* in an operator tree, it *first* tries to simplify the sign operators in that part of the tree.** If you already know algebra, you will recognize that you do this, too. There are rules for this, and fully simplifying a long chain of sign operators all at once can be complicated business. **Experienced humans often make subtle adjustments to exactly *how* they simplify signs based on what they intend to do *next*.**

But DragginMath isn't human, and it doesn't know what you intend to do next. It only knows what you are telling it to do *now*, so it simplifies sign chains in only one way that is known to be safe in all cases. Because of this, changes to expressions with complicated sign chains always give a result that is correct, but it may not be the exact result that you want.

Sometimes, DragginMath moves part way through a multi-step change, then stops because there is no clear path forward through what remains. It is even possible that the node you dragged was simplified away and no longer exists. It is up to you to notice this has happened. Then you must tell DragginMath how to proceed from where it stopped.

If you move only one operator at a time, you will not encounter this issue. For some problems, that is the right thing to do, even for sophisticated users who like to make large changes in single moves.

Combinatorics

DragginMath implements *factorial* in its traditional symbolic form: a postfix **!** operator, for example **5!**. Its precedence is higher than the exponential operators and lower than the sign operators.

Using current Apple hardware, values up to **20!** can be computed in Green Mode. Larger values remain in symbolic form. The factorial of **0** is **1**. The factorial of any negative integer is **?** (*NaN* or *Not-a-Number*, once again from the IEEE-754 Math Standard).

What about Blue Mode? Drag the **5** in **5!** up onto **!**. The result is **4!*5**. Now drag the **4** up onto **!**. The result is **3!*4*5**. You can do this as far as needed.

Factorial has no widely recognized inverse, so attempting to solve for **x** in **x!=6** currently does nothing.

Permute and Combine operators are in development.

More Relations

You have seen how to solve in DragginMath: drag things up onto **=**. There are also *inequality* relations: **< ≤ ≠ ≥ >**. Use them just as you use **=**.

Relations are not really operators, but DragginMath works with them as if they were. The formal language of mathematics is different for relations and operators, but for all intents and purposes, relations are composable, commutable, and associable. This means DragginMath can build them into operator trees, just as with true operators like **+** or *****. And you can drag them around, just as with true operators.

What won't a relation do? It won't evaluate. This might change in the future, but for now DragginMath leaves it to you

to recognize that $3=3$ and $5<7$ are true, and that $2>9$ just can't be right. If you try to evaluate relations, DragginMath will do *everything but* the relations. For example, evaluating $2*2=1+3$ gives the result $4=4$.

If you commute any relation in a chain, for example by dragging 3 sideways in $3<5<7$, the whole chain reverses the sense of the relation, leaving the result $7>(5>3)$. You can then reassociate to get $7>5>3$.

DragginMath solves simple linear inequalities correctly, just as it does with equalities. **More complicated inequalities are another matter.** For any relationship that has a piecewise solution, stronger tools are needed; most non-linear relationships and anything involving absolute value are in this category. DragginMath can help you carry out the techniques required for their solution, but *you* must know what they are, when they are needed, and how to use them.

Substituting Values in Expressions

Sometimes you must substitute one value for another in an expression. DragginMath can do this.

When substituting, you must have at least two expressions on the screen, and at least one of them *must* be an equation. Enter $5x^2+6x-7$; $x=3$. From the equation, drag x into *either* Blue Mode or Green Mode, then onto $-$ in the other expression. The result is $5*3^2+6*3-7$; $x=3$. All occurrences of x in the expression were replaced with 3 . Since we are just playing with this, you could then use the other side of the equation to replace all occurrences of 3 with x .

The old value you are replacing does not have to be simple. The new value you are substituting does not have to be simple. For example, enter $5x^2+6x-7$; $x^2=a+b$. Drag \uparrow from the second expression onto $-$ in the first. The result is $5(a+b)+6x-7$.

Remember that substitution is a Blue or Green Mode operation. If you are in another mode, substitution doesn't work. Users often forget this and have a frustrating experience.

Substitution is both powerful and dangerous. It is safer in DragginMath than in some other systems that provide this kind of behavior, but you must still be careful here. It is so easy to do so much without envisioning all the consequences or noticing all the results. For example, enter $3x\div 2$; $x=3$. Drag x in the equation up onto \div . The result is $3*3\div 2$; $x=3$. No problem there. Now drag 3 in the equation up onto \div to reverse the process. Do you see what happened? This is *exactly what you asked for* but *probably not what you wanted*. What can be done about this? You don't have to drop a substitution on the root of a tree: you can drop it anywhere. So you can drag the 3 in the equation up and directly onto the x you actually want to change in the other expression. Then you must deliberately look for any other instances you want to change, instead of letting DragginMath find them all for you from the root.

This is not to deter you from using substitution, but please be attentive and cautious (and remember: **Blue Mode or Green Mode, but not Red or Purple Mode**). The equation you save may be your own.


Named Expressions

If you do a lot of work with a particular expression, for example the Pythagorean Equation or Quadratic Equation, you might get tired of typing it all the time. It would be nice to enter an expression once, give it a name, and have DragginMath remember it that way. So let's do that.

You might have noticed a \equiv symbol on the screen keyboard. You also see this symbol in the upper right corner of the screen: the hamburger button. But on the screen keyboard, this symbol has a more mathematical meaning: *define*. Both meanings have a lot of tradition behind them, each in their own context.

In DragginMath, \equiv is actually a binary operator. The left operand is the expression you want to remember. The right operand is the expression's name. When you tap \equiv , the screen keyboard shifts to uppercase. Enter a single uppercase letter to name your expression, for example P or Q.

The text you enter might look like this: $a^2+b^2\equiv P$

DragginMath remembers this until you change it. When you want to *use* P, tap  to shift the keyboard, then tap P. The defined expression a^2+b^2 appears in your input when you do this, and the keyboard shifts back to lowercase.

Would you like to review your definitions? Instead of tapping \equiv , swipe your fingertip sideways off it. This raises a screen showing all definitions. Swipe up and down to see the whole list. If you want to delete a definition, swipe left on it to expose the delete button. If you want to use a definition, double tap it in the list. If you only want to look, tap **OK** at the bottom when you are done.

Notice some things about how defined expressions appear as operator trees. You don't see the \equiv operator in the tree. Only

the uppercase name is there, set off to the side. That uppercase name tag is the root of its visible operator tree. If you want to move a named expression to a different place on the screen, you must drag from its name tag.

\equiv is not equivalent to $=$. It exists for this purpose only: to define expressions. This operator is not commutable, and the expression's name comes *after* the operator. The precedence of \equiv is very low and it cannot be embedded in larger expressions. However, named expressions can be used inside the definitions of other named expressions, including redefinitions of themselves, for example $\mathbf{P}=\mathbf{c}\uparrow\mathbf{2}\equiv\mathbf{P}$. As seen throughout all of DragginMath, you may only enter expressions that are *correct* and *complete*. That means you can't define incomplete expressions that expect completion by the context in which they appear. There are situations where you might want that, but DragginMath won't do it.

Bringing It Forward (Implied Result)

Most problems are solved in DragginMath by entering an expression, then moving toward a solution just by dragging things around. Sometimes, that is not enough. You may discover that your current expression must be embedded in some *larger* expression in order to move forward. It would be bothersome and error-prone to retype the previous expression into the new one, so DragginMath doesn't make you do that.

If you ever want to include the previous expression in the new one, enter empty parentheses () wherever the previous expression needs to sit in the new one. The previous expression then appears inside these parentheses. This is a special case in

DragginMath's keyboard language that means only this. If the previous top text was actually several expressions separated by semicolons, a dialog appears asking which one you want to use.

This use of empty parentheses originated in the HP-71 calculator. It turns out to be useful here, too.

Distributing Over Relations

The classic mantra of elementary algebra is "*Do the same thing to both sides of the equation.*" For most problems, you do this in DragginMath whenever you drag something up onto = or any other relation.

Sometimes, the thing you need to do to both sides isn't in the relation. It may not even exist anywhere on your current screen, and if it is not there, you can't work with it.

Let's say (just for an example) you have worked a problem down to $3x+1<7$. Now, you realize you need to multiply both sides by 2 (Why? Remember: this is just an example!).

Tap the top text to start a new expression, then enter $()*2$. The result is $(3x+1<7)*2$. Now drag $<$ up onto $*$ and see $(3x+1)*2<7*2$. Both sides of the relation have been multiplied by 2, just as you needed to do. This is like the Distributive Property, but it distributes *any operator over any relation*. You can even do this with complicated multi-operators like $+ab\div c$.

Some caution is required. If you negate a relation $x<y$, for example, the two operands of the relation automatically swap with each other: $-(x<y)$ becomes $-y<-x$. This is correct. But what if you multiply an inequality by $-a$? DragginMath swaps the operands, but this is only true if a is positive, and DragginMath

has no way of knowing if that is really the case. Maybe you don't know either, so pay attention.

This is not the only way this problem arises. You must be aware of these issues, which DragginMath cannot fix for you. This is a tool, and any tool has limitations. This is one of them. You must understand the algebra that DragginMath is helping you do and make sure it has done the right thing.

Adding Equations (Superpose)

Some solution techniques require you to add one equation to another. If you have $\mathbf{a=b}$; $\mathbf{c=d}$, drag the = of the first equation onto the = of the second equation (this works in **either Blue Mode or Green Mode, but not Red or Purple Mode**). The result is $\mathbf{a=b}$; $\mathbf{c+a=d+b}$.

The dragged node must be =, but the target node can be any relation.