What Is a Computational Essay?

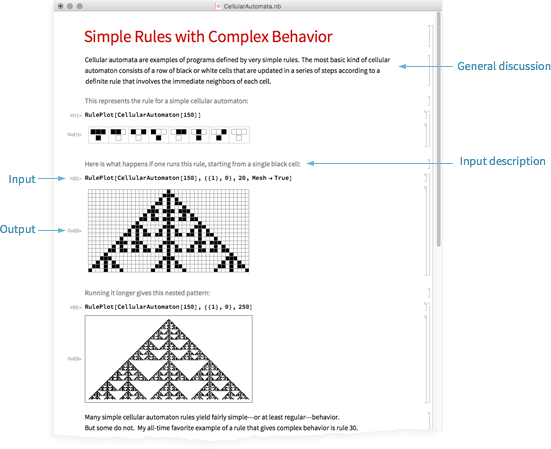
November 14, 2017

*A Powerful Way to Express Ideas*

People are used to producing prose—and sometimes pictures—to express themselves. But in the modern age of computation, something new has become possible that I’d like to call the *computational essay*.

I’ve been working on building the technology to support computational essays for several decades, but it’s only very recently that I’ve realized just how central computational essays can be to both the way people learn, and the way they communicate facts and ideas. Professionals of the future will routinely deliver results and reports as computational essays. Educators will routinely explain concepts using computational essays. Students will routinely produce computational essays as homework for their classes.

Here’s a very simple example of a computational essay:



There are basically three kinds of things here. First, ordinary text (here in English). Second, computer input. And third, computer output. And the crucial point is that these all work together to express what’s being communicated.

The ordinary text gives context and motivation. The computer input gives a precise specification of what’s being talked about. And then the computer output delivers facts and results, often in graphical form. It’s a powerful form of exposition that combines computational thinking on the part of the human author with computational knowledge and computational processing from the computer.

But what really makes this work is the [Wolfram Language](http://www.wolfram.com/language)—and the succinct representation of high-level ideas that it provides, defining a unique bridge between human computational thinking and actual computation and knowledge delivered by a computer.

In a typical computational essay, each piece of Wolfram Language input will usually be quite short (often not more than a line or two). But the point is that such input can communicate a high-level computational thought, in a form that can readily be understood both by the computer and by a human reading the essay.

It’s essential to all this that the Wolfram Language has so much built-in knowledge—both about the world and about how to compute things in it. Because that’s what allows it to immediately talk not just about abstract computations, but also about real things that exist and happen in the world—and ultimately to provide a true computational communication language that bridges the capabilities of humans and computers.

*An Example*

Let’s use a computational essay to explain computational essays.

Let’s say we want to talk about the structure of a human language, like English. English is basically made up of words. Let’s get a list of the common ones.

Generate a list of common words in English:

|  |
| --- |
| WordList[]  ✕  WordList[] |

How long is a typical word? Well, we can take the list of common words, and make a histogram that shows their distribution of lengths.

Make a histogram of word lengths:

|  |
| --- |
| Histogram[StringLength[WordList[]]]  ✕  Histogram[StringLength[WordList[]]] |

Do the same for French:

|  |
| --- |
| Histogram[StringLength[WordList[Language -> "French"]]]  ✕  Histogram[StringLength[WordList[Language -> "French"]]] |

Notice that the word lengths tend to be longer in French. We could investigate whether this is why documents tend to be longer in French than in English, or how this relates to quantities like entropy for text. (Of course, because this is a computational essay, the reader can rerun the computations in it themselves, say by trying Russian instead of French.)

But as something different, let’s compare languages by comparing their translations for, say, the word “computer”.

Find the translations for “computer” in the 10 most common languages:

|  |
| --- |
| Take[WordTranslation["computer", All], 10]  ✕  Take[WordTranslation["computer", All], 10] |

Find the first translation in each case:

|  |
| --- |
| First /@ Take[WordTranslation["computer", All], 10]  ✕  First /@ Take[WordTranslation["computer", All], 10] |

Arrange common languages in “feature space” based on their translations for “computer”:

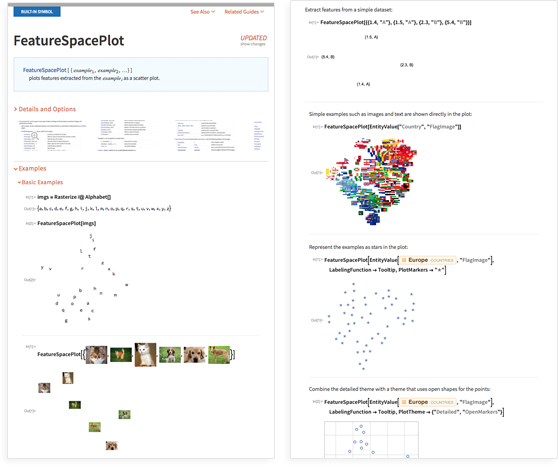
|  |
| --- |
| FeatureSpacePlot[First /@ Take[WordTranslation["computer", All], 40]]  ✕  FeatureSpacePlot[First /@ Take[WordTranslation["computer", All], 40]] |

From this plot, we can start to investigate all sorts of structural and historical relationships between languages. But from the point of view of a computational essay, what’s important here is that we’re sharing the exposition between ordinary text, computer input, and output.

The text is saying what the basic point is. Then the input is giving a precise definition of what we want. And the output is showing what’s true about it. But take a look at the input. Even just by looking at the names of the Wolfram Language functions in it, one can get a pretty good idea what it’s talking about. And while the function names are based on English, one can use “[code captions](http://reference.wolfram.com/language/workflow/ConfigureNonEnglishCodeCaptions.html.en)” to understand it in another language, say Japanese:

|  |
| --- |
| FeatureSpacePlot[First/@Take[WordTranslation["computer",All],40]]  ✕  FeatureSpacePlot[First /@ Take[WordTranslation["computer", All], 40]] |

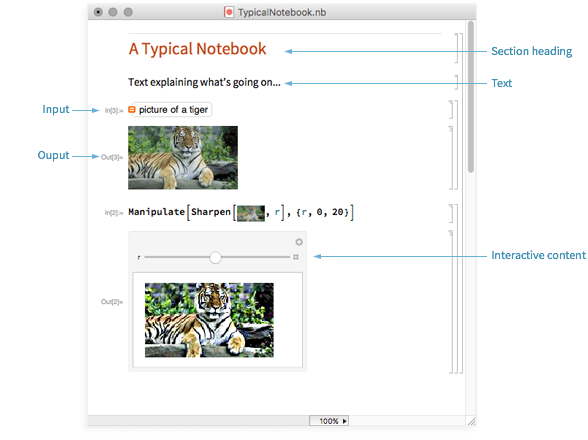
But let’s say one doesn’t know about [**FeatureSpacePlot**](http://reference.wolfram.com/language/ref/FeatureSpacePlot.html). What is it? If it was just a word or phrase in English, we might be able to look in a dictionary, but there wouldn’t be a precise answer. But a function in the Wolfram Language is always precisely defined. And to know what it does we can start by just looking at its documentation. But much more than that, we can just run it ourselves to explicitly see what it does.

[](http://reference.wolfram.com/language/ref/FeatureSpacePlot.html)

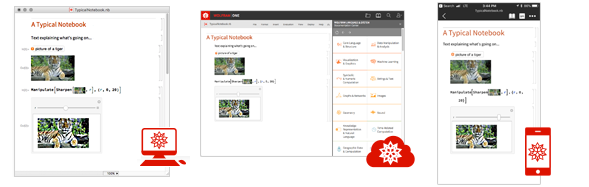
And that’s a crucial part of what’s great about computational essays. If you read an ordinary essay, and you don’t understand something, then in the end you really just have to ask the author to find out what they meant. In a computational essay, though, there’s Wolfram Language input that precisely and unambiguously specifies everything—and if you want to know what it means, you can just run it and explore any detail of it on your computer, automatically and without recourse to anything like a discussion with the author.

*Practicalities*

How does one actually create a computational essay? With the technology stack we have, it’s very easy—mainly thanks to the concept of notebooks that we introduced with the [first version of Mathematica](http://mathematica25.com/) all the way back in 1988. A notebook is a structured document that mixes cells of text together with cells of Wolfram Language input and output, including graphics, images, sounds, and interactive content:



In modern times one great (and very hard to achieve!) thing is that full Wolfram Notebooks run seamlessly across desktop, cloud and mobile. You can author a notebook in the native [Wolfram Desktop](https://www.wolfram.com/desktop/) application (Mac, Windows, Linux)—or on the web through any web browser, or on mobile through the [Wolfram Cloud app](http://www.wolfram.com/cloud-app/). Then you can share or publish it through the [Wolfram Cloud](https://www.wolfram.com/cloud/), and get access to it on the web or on mobile, or download it to desktop or, now, iOS devices.



Sometimes you want the reader of a notebook just to look at it, perhaps opening and closing groups of cells. Sometimes you also want them to be able to operate the interactive elements. And sometimes you want them to be able to edit and run the code, or maybe modify the whole notebook. And the crucial point is that all these things are easy to do with the cloud-desktop-mobile system we’ve built.

*A New Form of Student Work*

Computational essays are great for students to read, but they’re also great for students to write. Most of the current modalities for student work are remarkably old. Write an essay. Give a math derivation. These have been around for millennia. Not that there’s anything wrong with them. But now there’s something new: write a computational essay. And it’s wonderfully educational.

A computational essay is in effect an intellectual story told through a collaboration between a human author and a computer. The computer acts like a kind of intellectual exoskeleton, letting you immediately marshall vast computational power and knowledge. But it’s also an enforcer of understanding. Because to guide the computer through the story you’re trying to tell, you have to understand it yourself.

When students write ordinary essays, they’re typically writing about content that in some sense “already exists” (“discuss this passage”; “explain this piece of history”; …). But in doing computation (at least with the Wolfram Language) it’s so easy to discover new things that computational essays will end up with an essentially inexhaustible supply of new content, that’s never been seen before. Students will be exploring and discovering as well as understanding and explaining.

When you write a computational essay, the code in your computational essay has to produce results that fit with the story you’re telling. It’s not like you’re doing a mathematical derivation, and then some teacher tells you you’ve got the wrong answer. You can immediately see what your code does, and whether it fits with the story you’re telling. If it doesn’t, well then maybe your code is wrong—or maybe your story is wrong.

What should the actual procedure be for students producing computational essays? At this year’s [Wolfram Summer School](https://education.wolfram.com/summer/school/) we did the experiment of asking all our students to write a computational essay about anything they knew about. We ended up with [72 interesting essays](https://wolfr.am/TopicExplorations-SummerSchool2017)—exploring a very wide range of topics.

In a more typical educational setting, the “prompt” for a computational essay could be something like “What is the typical length of a word in English” or “Explore word lengths in English”.

There’s also another workflow I’ve tried. As the “classroom” component of a class, do livecoding (or a live experiment). Create or discover something, with each student following along by doing their own computations. At the end of the class, each student will have a notebook they made. Then have their “homework” be to turn that notebook into a computational essay that explains what was done.

And in my experience, this ends up being a very good exercise—that really tests and cements the understanding students have. But there’s also something else: when students have created a computational essay, they have something they can keep—and directly use—forever.

And this is one of the great general features of computational essays. When students write them, they’re in effect creating a custom library of computational tools for themselves—that they’ll be in a position to immediately use at any time in the future. It’s far too common for students to write notes in a class, then never refer to them again. Yes, they might run across some situation where the notes would be helpful. But it’s often hard to motivate going back and reading the notes—not least because that’s only the beginning; there’s still the matter of implementing whatever’s in the notes.

But the point is that with a computational essay, once you’ve found what you want, the code to implement it is right there—immediately ready to be applied to whatever has come up.

*Any Subject You Want*

What can computational essays be about? Almost anything! I’ve often said that for any field of study X (from archaeology to zoology), there either is now, or soon will be, a “computational X”. And any “computational X” can immediately be explored and explained using computational essays.

But even when there isn’t a clear “computational X” yet,  computational essays can still be a powerful way to organize and present material. In some sense, the very fact that a sequence of computations are typically needed to “tell the story” in an essay helps define a clear backbone for the whole essay. In effect, the structured nature of the computational presentation helps suggest structure for the narrative—making it easier for students (and others) to write essays that are easy to read and understand.

But what about actual subject matter? Well, imagine you’re studying history—say the history of the English Civil War. Well, conveniently, the Wolfram Language has a lot of knowledge about history (as about so many other things) built in. So you can present the English Civil War through a kind of dialog with it. For example, you can ask it for the geography of battles:

|  |
| --- |
| http://blog.stephenwolfram.com/data/uploads/2017/11/InOutImg12.png  ✕  GeoListPlot[\!\(\\*  NamespaceBox["LinguisticAssistant",  DynamicModuleBox[{Typeset`query$$ = "English Civil War",  Typeset`boxes$$ = TemplateBox[{"\"English Civil War\"",  RowBox[{"Entity", "[",  RowBox[{"\"MilitaryConflict\"", ",", "\"EnglishCivilWar\""}], "]"}],  "\"Entity[\\\"MilitaryConflict\\\", \  \\\"EnglishCivilWar\\\"]\"", "\"military conflict\""}, "Entity"],  Typeset`allassumptions$$ = {{  "type" -> "Clash", "word" -> "English Civil War",  "template" -> "Assuming \"${word}\" is ${desc1}. Use as \  ${desc2} instead", "count" -> "3",  "Values" -> {{  "name" -> "MilitaryConflict",  "desc" -> "a military conflict",  "input" -> "\*C.English+Civil+War-\_\*MilitaryConflict-"}, {  "name" -> "Word", "desc" -> "a word",  "input" -> "\*C.English+Civil+War-\_\*Word-"}, {  "name" -> "HistoricalEvent", "desc" -> "a historical event",  "input" -> "\*C.English+Civil+War-\_\*HistoricalEvent-"}}}, {  "type" -> "SubCategory", "word" -> "English Civil War",  "template" -> "Assuming ${desc1}. Use ${desc2} instead",  "count" -> "4",  "Values" -> {{  "name" -> "EnglishCivilWar",  "desc" -> "English Civil War (1642 - 1651)",  "input" -> "\*DPClash.MilitaryConflictE.English+Civil+War-\_\*\  EnglishCivilWar-"}, {  "name" -> "FirstEnglishCivilWar",  "desc" -> "English Civil War (1642 - 1646)",  "input" -> "\*DPClash.MilitaryConflictE.English+Civil+War-\_\*\  FirstEnglishCivilWar-"}, {  "name" -> "SecondEnglishCivilWar",  "desc" -> "Second English Civil War",  "input" -> "\*DPClash.MilitaryConflictE.English+Civil+War-\_\*\  SecondEnglishCivilWar-"}, {  "name" -> "ThirdEnglishCivilWar",  "desc" -> "Third English Civil War",  "input" -> "\*DPClash.MilitaryConflictE.English+Civil+War-\_\*\  ThirdEnglishCivilWar-"}}}}, Typeset`assumptions$$ = {},  Typeset`open$$ = {1, 2}, Typeset`querystate$$ = {  "Online" -> True, "Allowed" -> True,  "mparse.jsp" -> 1.305362`6.5672759594240935,  "Messages" -> {}}},  DynamicBox[ToBoxes[  AlphaIntegration`LinguisticAssistantBoxes["", 4, Automatic,  Dynamic[Typeset`query$$],  Dynamic[Typeset`boxes$$],  Dynamic[Typeset`allassumptions$$],  Dynamic[Typeset`assumptions$$],  Dynamic[Typeset`open$$],  Dynamic[Typeset`querystate$$]], StandardForm],  ImageSizeCache->{265., {7., 17.}},  TrackedSymbols:>{  Typeset`query$$, Typeset`boxes$$, Typeset`allassumptions$$,  Typeset`assumptions$$, Typeset`open$$, Typeset`querystate$$}],  DynamicModuleValues:>{},  UndoTrackedVariables:>{Typeset`open$$}],  BaseStyle->{"Deploy"},  DeleteWithContents->True,  Editable->False,  SelectWithContents->True]\)["Battles"]] |

You could ask for a timeline of the beginning of the war (you don’t need to say “first 15 battles”, because if one cares, one can just read that from the Wolfram Language code):

|  |
| --- |
| http://blog.stephenwolfram.com/data/uploads/2017/11/InOutImg13.png  ✕  TimelinePlot[Take[\!\(\\*  NamespaceBox["LinguisticAssistant",  DynamicModuleBox[{Typeset`query$$ = "English Civil War",  Typeset`boxes$$ = TemplateBox[{"\"English Civil War\"",  RowBox[{"Entity", "[",  RowBox[{"\"MilitaryConflict\"", ",", "\"EnglishCivilWar\""}], "]"}],  "\"Entity[\\\"MilitaryConflict\\\", \\\"EnglishCivilWar\\\"]\  \"", "\"military conflict\""}, "Entity"],  Typeset`allassumptions$$ = {{  "type" -> "Clash", "word" -> "English Civil War",  "template" -> "Assuming \"${word}\" is ${desc1}. Use as \  ${desc2} instead", "count" -> "3",  "Values" -> {{  "name" -> "MilitaryConflict",  "desc" -> "a military conflict",  "input" -> "\*C.English+Civil+War-\_\*MilitaryConflict-"}, {  "name" -> "Word", "desc" -> "a word",  "input" -> "\*C.English+Civil+War-\_\*Word-"}, {  "name" -> "HistoricalEvent",  "desc" -> "a historical event",  "input" -> "\*C.English+Civil+War-\_\*HistoricalEvent-"}}}, {  "type" -> "SubCategory", "word" -> "English Civil War",  "template" -> "Assuming ${desc1}. Use ${desc2} instead",  "count" -> "4",  "Values" -> {{  "name" -> "EnglishCivilWar",  "desc" -> "English Civil War (1642 - 1651)",  "input" -> "\*DPClash.MilitaryConflictE.English+Civil+War-\_\  \*EnglishCivilWar-"}, {  "name" -> "FirstEnglishCivilWar",  "desc" -> "English Civil War (1642 - 1646)",  "input" -> "\*DPClash.MilitaryConflictE.English+Civil+War-\_\  \*FirstEnglishCivilWar-"}, {  "name" -> "SecondEnglishCivilWar",  "desc" -> "Second English Civil War",  "input" -> "\*DPClash.MilitaryConflictE.English+Civil+War-\_\  \*SecondEnglishCivilWar-"}, {  "name" -> "ThirdEnglishCivilWar",  "desc" -> "Third English Civil War",  "input" -> "\*DPClash.MilitaryConflictE.English+Civil+War-\_\  \*ThirdEnglishCivilWar-"}}}}, Typeset`assumptions$$ = {},  Typeset`open$$ = {1, 2}, Typeset`querystate$$ = {  "Online" -> True, "Allowed" -> True,  "mparse.jsp" -> 1.305362`6.5672759594240935,  "Messages" -> {}}},  DynamicBox[ToBoxes[  AlphaIntegration`LinguisticAssistantBoxes["", 4, Automatic,  Dynamic[Typeset`query$$],  Dynamic[Typeset`boxes$$],  Dynamic[Typeset`allassumptions$$],  Dynamic[Typeset`assumptions$$],  Dynamic[Typeset`open$$],  Dynamic[Typeset`querystate$$]], StandardForm],  ImageSizeCache->{275., {7., 17.}},  TrackedSymbols:>{  Typeset`query$$, Typeset`boxes$$, Typeset`allassumptions$$,  Typeset`assumptions$$, Typeset`open$$,  Typeset`querystate$$}],  DynamicModuleValues:>{},  UndoTrackedVariables:>{Typeset`open$$}],  BaseStyle->{"Deploy"},  DeleteWithContents->True,  Editable->False,  SelectWithContents->True]\)["Battles"], 15]] |

You could start looking at how armies moved, or who won and who lost at different points. At first, you can write a computational essay in which the computations are basically just generating custom infographics to illustrate your narrative. But then you can go further—and start really doing “computational history”. You can start to compute various statistical measures of the progress of the war. You can find ways to quantitatively compare it to other wars, and so on.

Can you make a “computational essay” about art? Absolutely. Maybe about art history. Pick 10 random paintings by van Gogh:

|  |
| --- |
| http://blog.stephenwolfram.com/data/uploads/2017/11/InOutImg14.png   van Gogh paintings output  ✕  EntityValue[RandomSample[\!\(\\*  NamespaceBox["LinguisticAssistant",  DynamicModuleBox[{Typeset`query$$ = "van gogh", Typeset`boxes$$ =  TemplateBox[{"\"Vincent van Gogh\"",  RowBox[{"Entity", "[",  RowBox[{"\"Person\"", ",", "\"VincentVanGogh::9vq62\""}], "]"}],  "\"Entity[\\\"Person\\\", \\\"VincentVanGogh::9vq62\\\"]\"",  "\"person\""}, "Entity"],  Typeset`allassumptions$$ = {{  "type" -> "Clash", "word" -> "van gogh",  "template" -> "Assuming \"${word}\" is ${desc1}. Use as \  ${desc2} instead", "count" -> "4",  "Values" -> {{  "name" -> "Person", "desc" -> "a person",  "input" -> "\*C.van+gogh-\_\*Person-"}, {  "name" -> "Movie", "desc" -> "a movie",  "input" -> "\*C.van+gogh-\_\*Movie-"}, {  "name" -> "SolarSystemFeature",  "desc" -> "a solar system feature",  "input" -> "\*C.van+gogh-\_\*SolarSystemFeature-"}, {  "name" -> "Word", "desc" -> "a word",  "input" -> "\*C.van+gogh-\_\*Word-"}}}},  Typeset`assumptions$$ = {}, Typeset`open$$ = {1, 2},  Typeset`querystate$$ = {  "Online" -> True, "Allowed" -> True,  "mparse.jsp" -> 0.472412`6.125865914333281,  "Messages" -> {}}},  DynamicBox[ToBoxes[  AlphaIntegration`LinguisticAssistantBoxes["", 4, Automatic,  Dynamic[Typeset`query$$],  Dynamic[Typeset`boxes$$],  Dynamic[Typeset`allassumptions$$],  Dynamic[Typeset`assumptions$$],  Dynamic[Typeset`open$$],  Dynamic[Typeset`querystate$$]], StandardForm],  ImageSizeCache->{227., {7., 17.}},  TrackedSymbols:>{  Typeset`query$$, Typeset`boxes$$, Typeset`allassumptions$$,  Typeset`assumptions$$, Typeset`open$$,  Typeset`querystate$$}],  DynamicModuleValues:>{},  UndoTrackedVariables:>{Typeset`open$$}],  BaseStyle->{"Deploy"},  DeleteWithContents->True,  Editable->False,  SelectWithContents->True]\)["NotableArtworks"], 10], "Image"] |

Then look at what colors they use (a surprisingly narrow selection):

|  |
| --- |
| ChromaticityPlot[%]  ✕  ChromaticityPlot[%] |

Or maybe one could write a computational essay about actually creating art, or music.

What about science? You could rediscover Kepler’s laws by looking at properties of planets:

|  |
| --- |
| http://blog.stephenwolfram.com/data/uploads/2017/11/InOutImg17.png  ✕  \!\(\\*  NamespaceBox["LinguisticAssistant",  DynamicModuleBox[{Typeset`query$$ = "planets", Typeset`boxes$$ =  TemplateBox[{"\"planets\"",  RowBox[{"EntityClass", "[",  RowBox[{"\"Planet\"", ",", "All"}], "]"}],  "\"EntityClass[\\\"Planet\\\", All]\"", "\"planets\""},  "EntityClass"],  Typeset`allassumptions$$ = {{  "type" -> "Clash", "word" -> "planets",  "template" -> "Assuming \"${word}\" is ${desc1}. Use as \  ${desc2} instead", "count" -> "4",  "Values" -> {{  "name" -> "PlanetClass", "desc" -> " referring to planets",  "input" -> "\*C.planets-\_\*PlanetClass-"}, {  "name" -> "ExoplanetClass",  "desc" -> " referring to exoplanets",  "input" -> "\*C.planets-\_\*ExoplanetClass-"}, {  "name" -> "MinorPlanetClass",  "desc" -> " referring to minor planets",  "input" -> "\*C.planets-\_\*MinorPlanetClass-"}, {  "name" -> "Word", "desc" -> "a word",  "input" -> "\*C.planets-\_\*Word-"}}}},  Typeset`assumptions$$ = {}, Typeset`open$$ = {1, 2},  Typeset`querystate$$ = {  "Online" -> True, "Allowed" -> True,  "mparse.jsp" -> 0.400862`6.054539882441674, "Messages" -> {}}},  DynamicBox[ToBoxes[  AlphaIntegration`LinguisticAssistantBoxes["", 4, Automatic,  Dynamic[Typeset`query$$],  Dynamic[Typeset`boxes$$],  Dynamic[Typeset`allassumptions$$],  Dynamic[Typeset`assumptions$$],  Dynamic[Typeset`open$$],  Dynamic[Typeset`querystate$$]], StandardForm],  ImageSizeCache->{171., {7., 17.}},  TrackedSymbols:>{  Typeset`query$$, Typeset`boxes$$, Typeset`allassumptions$$,  Typeset`assumptions$$, Typeset`open$$, Typeset`querystate$$}],  DynamicModuleValues:>{},  UndoTrackedVariables:>{Typeset`open$$}],  BaseStyle->{"Deploy"},  DeleteWithContents->True,  Editable->False,  SelectWithContents->True]\)[{"DistanceFromSun", "OrbitPeriod"}] |

|  |
| --- |
| ListLogLogPlot[%]  ✕  ListLogLogPlot[%] |

Maybe you could go on and check it for exoplanets. Or you could start solving the equations of motion for planets.

You could look at biology. Here’s the first beginning of the reference sequence for the human mitochondrion:

|  |
| --- |
| GenomeData[{"Mitochondrion", {1, 150}}]  ✕  GenomeData[{"Mitochondrion", {1, 150}}] |

You can start off breaking it into possible codons:

|  |
| --- |
| StringPartition[%, 3]  ✕  StringPartition[%, 3] |

There’s an immense amount of data about all kinds of things built into the Wolfram Language. But there’s also the [Wolfram Data Repository](https://datarepository.wolframcloud.com/), which contains all sorts of specific datasets. Like here’s a map of state fairgrounds in the US:

|  |
| --- |
| GeoListPlot[  ResourceData["U.S. State Fairgrounds"][All, "GeoPosition"]]  ✕  GeoListPlot[  ResourceData["U.S. State Fairgrounds"][All, "GeoPosition"]] |

And here’s a word cloud of the constitutions of countries that have been enacted since 2010:

|  |
| --- |
| http://blog.stephenwolfram.com/data/uploads/2017/11/InOutImg22.png  ✕  WordCloud[  StringJoin[  Normal[ResourceData["World Constitutions"][  Select[#YearEnacted > \!\(\\*  NamespaceBox["LinguisticAssistant",  DynamicModuleBox[{Typeset`query$$ = "year 2010", Typeset`boxes$$ =  RowBox[{"DateObject", "[",  RowBox[{"{", "2010", "}"}], "]"}],  Typeset`allassumptions$$ = {{  "type" -> "MultiClash", "word" -> "",  "template" -> "Assuming ${word1} is referring to \  ${desc1}. Use \"${word2}\" as ${desc2}.", "count" -> "2",  "Values" -> {{  "name" -> "PseudoTokenYear", "word" -> "year 2010",  "desc" -> "a year",  "input" -> "\*MC.year+2010-\_\*PseudoTokenYear-"}, {  "name" -> "Unit", "word" -> "year", "desc" -> "a unit",  "input" -> "\*MC.year+2010-\_\*Unit-"}}}},  Typeset`assumptions$$ = {}, Typeset`open$$ = {1},  Typeset`querystate$$ = {  "Online" -> True, "Allowed" -> True,  "mparse.jsp" -> 0.542662`6.186074404594303,  "Messages" -> {}}},  DynamicBox[ToBoxes[  AlphaIntegration`LinguisticAssistantBoxes["", 4, Automatic,  Dynamic[Typeset`query$$],  Dynamic[Typeset`boxes$$],  Dynamic[Typeset`allassumptions$$],  Dynamic[Typeset`assumptions$$],  Dynamic[Typeset`open$$],  Dynamic[Typeset`querystate$$]], StandardForm],  ImageSizeCache->{86., {7., 18.}},  TrackedSymbols:>{  Typeset`query$$, Typeset`boxes$$,  Typeset`allassumptions$$, Typeset`assumptions$$,  Typeset`open$$, Typeset`querystate$$}],  DynamicModuleValues:>{},  UndoTrackedVariables:>{Typeset`open$$}],  BaseStyle->{"Deploy"},  DeleteWithContents->True,  Editable->False,  SelectWithContents->True]\) &], "Text"]]]] |

Quite often one’s interested in dealing not with public data, but with some kind of local data. One convenient source of this is the [Wolfram Data Drop](https://datadrop.wolframcloud.com/). In an educational setting, particular databins (or cloud objects in general) can be set so that they can be read (and/or added to) by some particular group. Here’s a databin that I accumulate for myself, showing my heart rate through the day. Here it is for today:

|  |
| --- |
| DataListPlot[TimeSeries[Databin[  ✕  DateListPlot[TimeSeries[YourDatabinHere]] |

Of course, it’s easy to make a histogram too:

|  |
| --- |
| Histogram[TimeSeries[Databin[  ✕  Histogram[TimeSeries[YourDatabinHere]] |

What about math? A key issue in math is to understand why things are true. The traditional approach to this is to give proofs. But computational essays provide an alternative. The nature of the steps in them is different—but the objective is the same: to show what’s true and why.

As a very simple example, let’s look at primes. Here are the first 50:

|  |
| --- |
| Table[Prime[n], {n, 50}]  ✕  Table[Prime[n], {n, 50}] |

Let’s find the remainder mod 6 for all these primes:

|  |
| --- |
| Mod[Table[Prime[n], {n, 50}], 6]  ✕  Mod[Table[Prime[n], {n, 50}], 6] |

But why do only 1 and 5 occur (well, after the trivial cases of the primes 2 and 3)? We can see this by computation. Any number can be written as 6*n*+*k* for some *n* and *k*:

|  |
| --- |
| Table[6 n + k, {k, 0, 5}]  ✕  Table[6 n + k, {k, 0, 5}] |

But if we factor numbers written in this form, we’ll see that 6*n*+1 and 6*n*+5 are the only ones that don’t have to be multiples:

|  |
| --- |
| Factor[%]  ✕  Factor[%] |

What about computer science? One could for example write a computational essay about implementing Euclid’s algorithm, studying its running time, and so on.

Define a function to give all steps in Euclid’s algorithm:

|  |
| --- |
| gcdlist[a_, b_] :=   NestWhileList[{Last[#], Apply[Mod, #]} &, {a, b}, Last[#] != 0 &, 1]  ✕  gcdlist[a\_, b\_] :=  NestWhileList[{Last[#], Apply[Mod, #]} &, {a, b}, Last[#] != 0 &, 1] |

Find the distribution of running lengths for the algorithm for numbers up to 200:

|  |
| --- |
| Histogram[Flatten[Table[Length[gcdlist[i, j]], {i, 200}, {j, 200}]]]  ✕  Histogram[Flatten[Table[Length[gcdlist[i, j]], {i, 200}, {j, 200}]]] |

Or in modern times, one could explore machine learning, starting, say, by making a feature space plot of part of the MNIST handwritten digits dataset:

|  |
| --- |
| FeatureSpacePlot[RandomSample[Keys[ResourceData["MNIST"]], 50]]  ✕  FeatureSpacePlot[RandomSample[Keys[ResourceData["MNIST"]], 50]] |

If you wanted to get deeper into software engineering, you could write a computational essay about the HTTP protocol. This gets an HTTP response from a site:

|  |
| --- |
| URLRead["https://www.wolframalpha.com"]  ✕  URLRead["https://www.wolfram.com"] |

And this shows the tree structure of the elements on the webpage at that URL:

|  |
| --- |
| TreeForm[Import["http://www.wolframalpha.com", {"HTML", "XMLObject"}],   VertexLabeling -> False, AspectRatio -> 1/2]  ✕  TreeForm[Import["http://www.wolframalpha.com", {"HTML", "XMLObject"}],  VertexLabeling -> False, AspectRatio -> 1/2] |

Or—in a completely different direction—you could talk about anatomy:

|  |
| --- |
| AnatomyPlot3D[left foot]  ✕  AnatomyPlot3D[Entity["AnatomicalStructure", "LeftFoot"]] |

*What Makes a Good Computational Essay?*

As far as I’m concerned, for a computational essay to be good, it has to be as easy to understand as possible. The format helps quite a lot, of course. Because a computational essay is full of outputs (often graphical) that are easy to skim, and that immediately give some impression of what the essay is trying to say. It also helps that computational essays are structured documents, that deliver information in well-encapsulated pieces.

But ultimately it’s up to the author of a computational essay to make it clear. But another thing that helps is that the nature of a computational essay is that it must have a “computational narrative”—a sequence of pieces of code that the computer can execute to do what’s being discussed in the essay. And while one might be able to write an ordinary essay that doesn’t make much sense but still sounds good, one can’t ultimately do something like that in a computational essay. Because in the end the code is the code, and actually has to run and do things.

So what can go wrong? Well, like English prose, Wolfram Language code can be unnecessarily complicated, and hard to understand. In a good computational essay, both the ordinary text, and the code, should be as simple and clean as possible. I try to enforce this for myself by saying that each piece of input should be at most one or perhaps two lines long—and that the caption for the input should always be just one line long. If I’m trying to do something where the core of it (perhaps excluding things like display options) takes more than a line of code, then I break it up, explaining each line separately.

Another important principle as far as I’m concerned is: be explicit. Don’t have some variable that, say, implicitly stores a list of words. Actually show at least part of the list, so people can explicitly see what it’s like. And when the output is complicated, find some tabulation or visualization that makes the features you’re interested in obvious. Don’t let the “key result” be hidden in something that’s tucked away in the corner; make sure the way you set things up makes it front and center.

Use the structured nature of notebooks. Break up computational essays with section headings, again helping to make them easy to skim. I follow the style of having a “caption line” before each input. Don’t worry if this somewhat repeats what a paragraph of text has said; consider the caption something that someone who’s just “looking at the pictures” might read to understand what a picture is of, before they actually dive into the full textual narrative.

The technology of Wolfram Notebooks makes it straightforward to put in interactive elements, like [**Manipulate**](http://reference.wolfram.com/language/ref/Manipulate.html), into computational essays. And sometimes this is very helpful, and perhaps even essential. But interactive elements shouldn’t be overused. Because whenever there’s an element that requires interaction, this reduces the ability to skim the essay.

Sometimes there’s a fair amount of data—or code—that’s needed to set up a particular computational essay. The cloud is very useful for handling this. Just deploy the data (or code) to the Wolfram Cloud, and set appropriate permissions so it can automatically be read whenever the code in your essay is executed.

Notebooks also allow “reverse closing” of cells—allowing an output cell to be immediately visible, even though the input cell that generated it is initially closed. This kind of hiding of code should generally be avoided in the body of a computational essay, but it’s sometimes useful at the beginning or end of an essay, either to give an indication of what’s coming, or to include something more advanced where you don’t want to go through in detail how it’s made.

OK, so if a computational essay is done, say, as homework, how can it be assessed? A first, straightforward question is: does the code run? And this can be determined pretty much automatically. Then after that, the assessment process is very much like it would be for an ordinary essay. Of course, it’s nice and easy to add cells into a notebook to give comments on what’s there. And those cells can contain runnable code—that for example can take results in the essay and process or check them.

Are there principles of good computational essays? Here are a few candidates:

0. Understand what you’re talking about (!)

1. Find the most straightforward and direct way to represent your subject matter

2. Keep the core of each piece of Wolfram Language input to a line or two

3. Use explicit visualization or other information presentation as much as possible

4. Try to make each input+caption independently understandable

5. Break different topics or directions into different subsections

*Learning the Language*

At the core of computational essays is the idea of expressing computational thoughts using the Wolfram Language. But to do that, one has to know the language. Now, unlike human languages, the Wolfram Language is explicitly designed (and, yes, that’s what I’ve been doing for the past 30+ years) to follow definite principles and to be as easy to learn as possible. But there’s still learning to be done.

One feature of the Wolfram Language is that—like with human languages—it’s typically easier to read than to write. And that means that a good way for people to learn what they need to be able to write computational essays is for them first to read a bunch of essays. Perhaps then they can start to modify those essays. Or they can start creating “notes essays”, based on code generated in livecoding or other classroom sessions.

As people get more fluent in writing the Wolfram Language, something interesting happens: they start actually expressing themselves in the language, and using Wolfram Language input to carry significant parts of the narrative in a computational essay.

When I was writing [*An Elementary Introduction to the Wolfram Language*](https://www.wolfram.com/language/elementary-introduction/2nd-ed/) (which itself is written in large part as a sequence of computational essays) I had an interesting experience. Early in the book, it was decently easy to explain computational exercises in English (“Make a table of the first 10 squares”). But a little later in the book, it became a frustrating process.

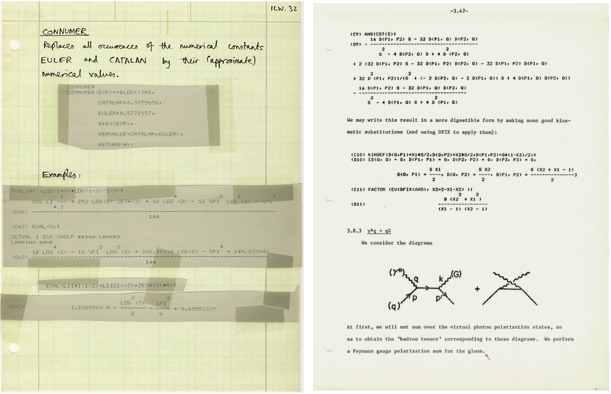
It was easy to express what I wanted in the Wolfram Language. But to express it in English was long and awkward (and had a tendency of sounding like legalese). And that’s the whole point of using the Wolfram Language, and the reason I’ve spent 30+ years building it: because it provides a better, crisper way to express computational thoughts.

It’s sometimes said of human languages that the language you use determines how you think. It’s not clear how true this is of human languages. But it’s absolutely true of computer languages. And one of the most powerful things about the Wolfram Language is that it helps one formulate clear computational thinking.

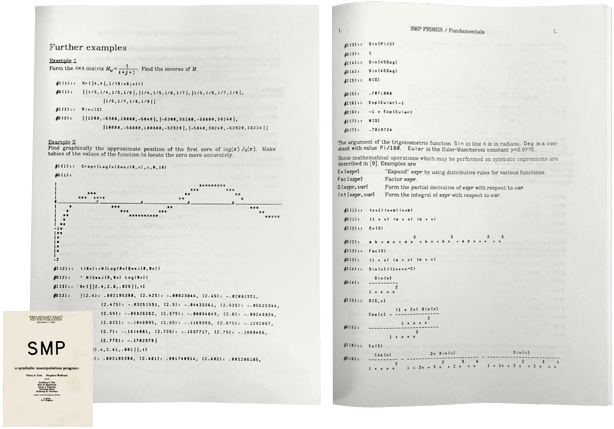
Traditional computer languages are about writing code that describes the details of what a computer should do. The point of the Wolfram Language is to provide something much higher level—that can immediately talk about things in the world, and that can allow people as directly as possible to use it as a medium of computational thinking. And in a sense that’s what makes a good computational essay possible.

*The Long Path to Computational Essays*

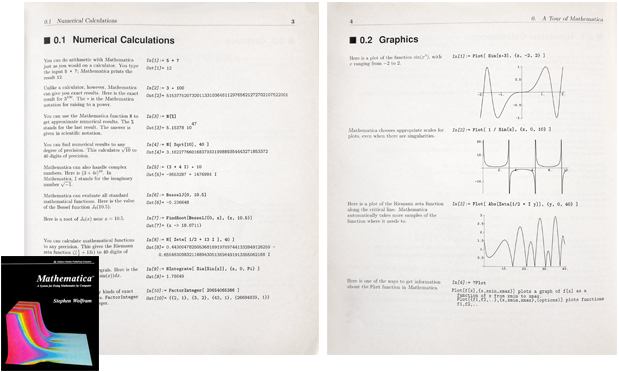
Now that we have full-fledged computational essays, I realize I’ve been on a path towards them for [nearly 40 years](http://blog.stephenwolfram.com/2016/04/my-life-in-technology-as-told-at-the-computer-history-museum/). At first I was taking interactive computer output and Scotch-taping descriptions into it:

[](http://blog.stephenwolfram.com/data/uploads/2017/11/InOutImg38-zoom.png)

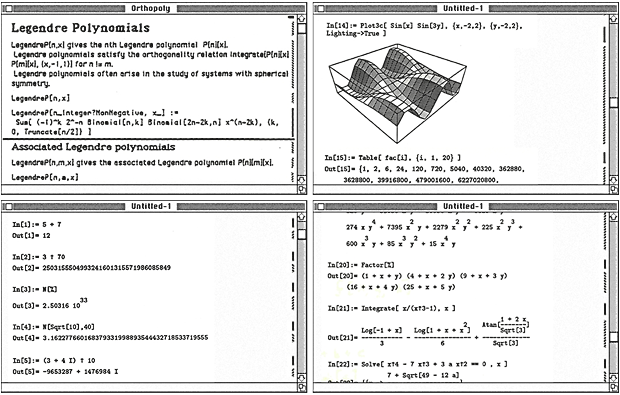
By 1981, [when I built SMP](http://blog.stephenwolfram.com/2013/06/there-was-a-time-before-mathematica/), I was routinely writing documents that interspersed code and explanations:

[](http://blog.stephenwolfram.com/data/uploads/2017/11/InOutImg39-zoom.png)

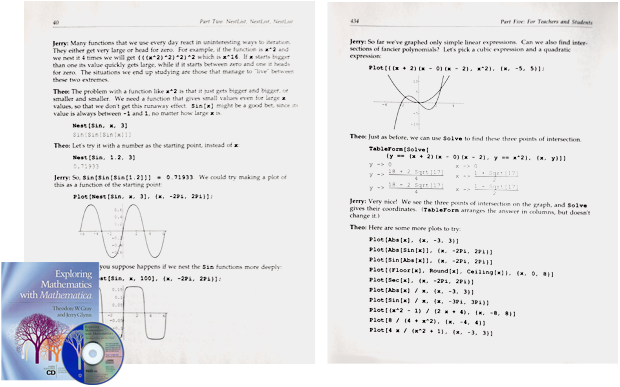
But it was only in 1986, when I started documenting what became Mathematica and the Wolfram Language, that I started seriously developing a style close to what I now favor for computational essays:

[](http://blog.stephenwolfram.com/data/uploads/2017/11/InOutImg40-zoom.png)

And with the release of [Mathematica 1.0 in 1988](http://mathematica25.com/) came another critical element: the invention of Wolfram Notebooks. Notebooks arrived in a form at least superficially very similar to the way they are today (and already in many ways more sophisticated than the imitations that started appearing 25+ years later!): collections of cells arranged into groups, and capable of containing text, executable code, graphics, etc.



At first notebooks were only possible on Mac and NeXT computers. A few years later they were extended to Microsoft Windows and X Windows (and later, Linux). But immediately people started using notebooks both to provide reports about they’d done, and to create rich expository and educational material. Within a couple of years, there started to be courses based on notebooks, and books printed from notebooks, with interactive versions available on CD-ROM at the back:

[](http://blog.stephenwolfram.com/data/uploads/2017/11/InOutImg42-zoom.png)

So in a sense the raw material for computational essays already existed by the beginning of the 1990s. But to really make computational essays come into their own required the development of the cloud—as well as the whole broad range of computational knowledge that’s now part of the Wolfram Language.

By 1990 it was perfectly possible to create a notebook with a narrative, and people did it, particularly about topics like mathematics. But if there was real-world data involved, things got messy. One had to make sure that whatever was needed was appropriately available from a distribution CD-ROM or whatever. We created a [Player for notebooks](https://www.wolfram.com/cdf-player/) very early, that was sometimes distributed with notebooks.

But in the last few years, particularly with the development of the [Wolfram Cloud](https://www.wolfram.com/cloud/), things have gotten much more streamlined. Because now you can seamlessly store things in the cloud and use them anywhere. And you can work directly with notebooks in the cloud, just using a web browser. In addition, thanks to lots of user-assistance innovations (including natural language input), it’s become even easier to write in the Wolfram Language—and there’s ever more that can be achieved by doing so.

And the important thing that I think has now definitively happened is that it’s become lightweight enough to produce a good computational essay that it makes sense to do it as something routine—either professionally in writing reports, or as a student doing homework.

*Ancient Educational History*

The idea of students producing computational essays is something new for modern times, made possible by a whole stack of current technology. But there’s a curious resonance with something from the distant past. You see, if you’d learned a subject like math in the US a couple of hundred years ago, a big thing you’d have done is to create a so-called ciphering book—in which over the course of several years you carefully wrote out the solutions to a range of problems, mixing explanations with calculations. And the idea then was that you kept your ciphering book for the rest of your life, referring to it whenever you needed to solve problems like the ones it included.

Well, now, with computational essays you can do very much the same thing. The problems you can address are vastly more sophisticated and wide-ranging than you could reach with hand calculation. But like with ciphering books, you can write computational essays so they’ll be useful to you in the future—though now you won’t have to imitate calculations by hand; instead you’ll just edit your computational essay notebook and immediately rerun the Wolfram Language inputs in it.

I actually only learned about ciphering books quite recently. For about 20 years I’d had essentially as an artwork a curious handwritten notebook (created in 1818, it says, by a certain George Lehman, apparently of [Orwigsburg, Pennsylvania](http://www.wolframalpha.com/input/?i=orwigsburg)), with pages like this:

[](http://blog.stephenwolfram.com/data/uploads/2017/11/InOutImg43-zoom.png)

I now know this is a ciphering book—that on this page describes how to find the “height of a perpendicular object… by having the length of the shadow given”. And of course I can’t resist a modern computational essay analog, which, needless to say, can be a bit more elaborate.

Find the current position of the Sun as azimuth, altitude:

|  |
| --- |
| SunPosition[]  ✕  SunPosition[] |

Find the length of a shadow for an object of unit height:

|  |
| --- |
| 1/Tan[SunPosition[][[2]]]  ✕  1/Tan[SunPosition[][[2]]] |

Given a 10-ft shadow, find the height of the object that made it:

|  |
| --- |
| Tan[SunPosition[][[2]]]10ft  ✕  Tan[SunPosition[][[2]]]10ft |

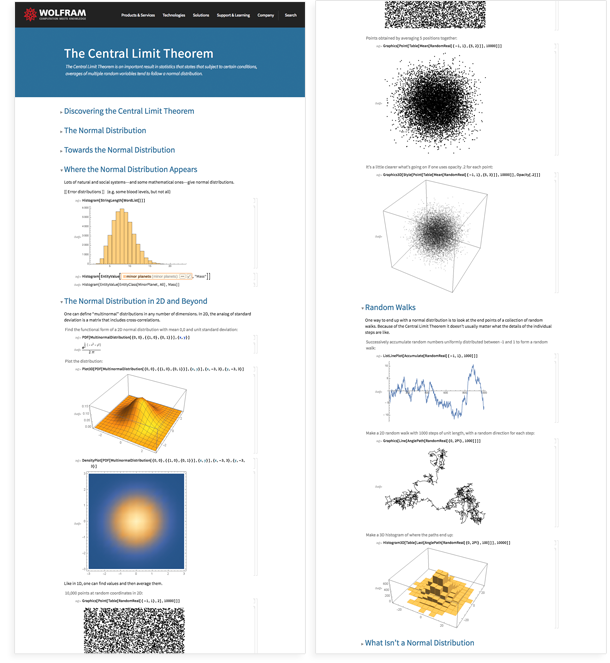
*The Path Ahead*

I like writing textual essays (such as blog posts!). But I like writing computational essays more. Because at least for many of the things I want to communicate, I find them a purer and more efficient way to do it. I could spend lots of words trying to express an idea—or I can just give a little piece of Wolfram Language input that expresses the idea very directly and shows how it works by generating (often very visual) output with it.

When I wrote my big book [*A New Kind of Science*](https://www.wolframscience.com/nks/) (from 1991 to 2002), neither our technology nor the world was quite ready for computational essays in the form in which they’re now possible. My research for the book filled thousands of Wolfram Notebooks. But when it actually came to putting together the book, I just showed the results from those notebooks—including a little of the code from them in notes at the back of the book.

But now the story of the book can be told in computational essays—that I’ve been starting to produce. (Just for fun, I’ve been [livestreaming](https://go.twitch.tv/stephen_wolfram) some of the work I’m doing to create these.)  And what’s very satisfying is just how clearly and crisply the ideas in the book can be communicated in computational essays.

There is so much potential in computational essays. And indeed we’re now starting the project of collecting “topic explorations” that use computational essays to explore a vast range of topics in unprecedentedly clear and direct ways. It’ll be something like our [Wolfram Demonstrations Project](http://demonstrations.wolfram.com/) (that now has 11,000+ Wolfram Language–powered Demonstrations). Here’s a typical example I wrote:

[](https://wolfr.am/TheCentralLimitTheorem)

Computational essays open up all sorts of new types of communication. Research papers that directly present computational experiments and explorations. Reports that describe things that have been found, but allow other cases to be immediately explored. And, of course, computational essays define a way for students (and others) to very directly and usefully showcase what they’ve learned.

There’s something satisfying about both writing—and reading—computational essays. It’s as if in communicating ideas we’re finally able to go beyond pure human effort—and actually leverage the power of computation. And for me, having built the Wolfram Language to be a computational communication language, it’s wonderful to see how it can be used to communicate so effectively in computational essays.

It’s so nice when I get something sent to me as a well-formed computational essay. Because I immediately know that I’m going to get a straight story that I can actually understand. There aren’t going to be all sorts of missing sources and hidden assumptions; there’s just going to be Wolfram Language input that stands alone, and that I can take out and study or run for myself.

The modern world of the web has brought us a few new formats for communication—like blogs, and social media, and things like Wikipedia. But all of these still follow the basic concept of text + pictures that’s existed since the beginning of the age of literacy. With computational essays we finally have something new—and it’s going to be exciting to see all the things it makes possible.