blog :: Brent -> [String]

Abstraction, intuition, and the "monad tutorial fallacy"

Posted on January 12, 2009

While working on an article for the Monad.Reader, I've had the opportunity to think about how people learn and gain intuition for abstraction, and the implications for pedagogy. The heart of the matter is that people *begin with the concrete, and move to the abstract*. Humans are very good at pattern recognition, so this is a natural progression. By examining concrete objects in detail, one begins to notice similarities and patterns, until one comes to understand on a more abstract, intuitive level. This is why it's such good pedagogical practice to demonstrate examples of concepts you are trying to teach. It's particularly important to note that this process doesn't change *even when one is presented with the abstraction up front*! For example, when presented with a mathematical definition for the first time, most people (me included) don't "get it" immediately: it is only after examining some specific instances of the definition, and working through the implications of the definition in detail, that one begins to appreciate the definition and gain an understanding of what it "really says."

Unfortunately, there is a whole cottage industry of monad tutorials that get this wrong. To see what I mean, imagine the following scenario: Joe Haskeller is trying to learn about monads. After struggling to understand them for a week, looking at examples, writing code, reading things other people have written, he finally has an "aha!" moment: everything is suddenly clear, and Joe Understands Monads! What has really happened, of course, is that Joe's brain has fit all the details together into a higher-level abstraction, a metaphor which Joe can use to get an intuitive grasp of monads; let us suppose that Joe's metaphor is that Monads are Like Burritos. Here is where Joe badly misinterprets his own thought process: "Of course!" Joe thinks. "It's all so simple now. The key to understanding monads is that they are Like Burritos. If only I had thought of this before!" The problem, of course, is that if Joe HAD thought of this before, it wouldn't have helped: the week of struggling through details was a necessary and integral part of forming Joe's Burrito intuition, not a sad consequence of his failure to hit upon the idea sooner.

But now Joe goes and writes a monad tutorial called "Monads are Burritos," under the well-intentioned but mistaken assumption that if other people read his magical insight, learning about monads will be a snap for them. "Monads are easy," Joe writes. "Think of them as burritos." Joe hides all the actual details about types and such because those are scary, and people will learn bet an avoid all that difficult and confusing stuff. Of course, exactly the opposite is tr ake it *harder* for people to learn about monads, because now they have to spend ls are burritos and getting utterly confused, and then a week trying to forget about the l can actually get down to the business of learning about monads. (Of course, certainly n e this, and I don't even have any particular ones in mind, just a general impression of them, but if the shoe fits...)

What I term the "monad tutoria iling to recognize the critical role that struggling through fundamental details pla n. This, I suspect, is also one of the things that separates good teachers from pc rself frustrated and astounded that someone else does not grasp a concept as easily and intuitively as you do, even after you clearly explain your intuition to them ("look, it's really quite simple," you say...) then you are suffering from the monad tutorial fallacy.

Share this:

Related

Teaching abstraction
In "teaching"

Monads: Easy or Hard?
In "haskell"

Parsing context-sensitive languages with Applicative
In "haskell"

This entry was posted in haskell, learning, math, teaching and tagged abstraction, intuition, learning, monads, pedagogy, tutorials. Bookmark the permalink.

95 Responses to *Abstraction, intuition, and the "monad tutorial fallacy"*

OJ *says:*
January 12, 2009 at 11:59 pm

Great post mate. When writing my tutorials I try and keep this in mind. I'm certain I've suffered from it myself, though I am yet to be pulled up (probably coz I don't get a lot of readers ;)).

The heart of the point is key though. What works for you might not work for someone else. Especially when the conclusion you've drawn from your experience leads you to an analogy that wouldn't have worked for you in the first place.

I am still yet to find a Monad tutorial that does a good job.

Nice blog :)

Reply

Bill Mill *says:*
January 13, 2009 at 1:44 am

A related article I came across this week: http://betterexplained.com/articles/developing-your-intuition-for-math/

(I have nothing to add, except, yes! Agreed wholeheartedly.)

Reply

Keith Braitwaite *says:*
January 13, 2009 at 2:51 am

Absolutely. So, as a neophyte Haskell programmer I ask: where is there a catalogue of very most simple concrete examples of monad usage?

I'm currently learning the State monad. It's a rich and powerful thing, but I struggle to learn effectively from rich and powerful examples (which is what the very small number of examples I've been able to find are).

Another aspect: I'll get around to understanding how wonderfully clever monads are after I know how to be an effective programmer with them, thanks (just as I learned λ calculus after becoming proficient with Scheme).

Thus, show me how to do something very simple but useful with the State monad (even if, and this is the important bit) there is another more idiomatic way to do it, before launching into the examples that show the motivation for mondads in the first place. StgGen, I'm looking at you. Show me, oh let's say an accumulator that walks a list adding up the values. Sure, you wouldn't do it that way in practice, but what a great, simple example of using the tool!

Reply

Julian Gilbey *says:*
June 3, 2010 at 1:00 pm

Have a look at Paul Hudak's book: The Haskell School of Expression. He has a lovely example of essentially this; I stepped through the code on (lots of) paper, and it really began to help. Then having a several-hour conversation with

a computer-science-lecturer friend of mine about what a monad is just added on to this!

Good luck!

Reply

Jason *says:*
June 12, 2010 at 6:56 pm

I think the simplest example of a monad is the list monad. A slightly more complicated example is the tree monad. Here, return() yields the singleton list, or the tree of one node. I've reached my understanding from category theory, where return() is more commonly referred to as the unit morphism.

If tutorials gave examples of lists and trees before launching into the abstract treatment, they wouldn't be too far off.

The problem is, the mathematical conception is in terms of fmap (i.e. the functor), unit (aka. return), and flatten (the natural transform fmap . fmap -> Id). IO style deals with it in terms of an alternative formulation, i.e. replacing fmap and flatten with bind. The former is easier for understanding the list and tree example in my opinion, just as it is also what the category theorist is used to. It is then necessary to motivate the switch to the latter so as to explain how the IO monad works. The state monad may be a good step in the process.

Reply

Julian Gilbey *says:*
June 21, 2010 at 5:13 am

The functor (T), the unit natural transformation (eta:X->TX) and the flatten natural transformation (mu:T(TX)->TX) are indeed the more common category theory notions; however, the alternative formulation is also known, and is called a Kleisli triple; it is defined in terms of T, eta and the Kleisli map which takes a function f:X->TY to a map f*:TX->TY. So the bind operator can be thought of as follows: the binding m >>= f (using Haskell notation) is just f*(m).

Reply

Pingback: *Turulcsirip - diegoeche*

Christophe Poucet *says:*
January 13, 2009 at 7:11 am

I would say this relates to unlearning. I do no think this is particular to monads. Monads just exemplify it well since they're hard to understand in the beginning, have a steep learning curve and a definitive and very clear 'aha' point. That and the fact it does not take years to learn them. But I think this holds for all concepts we learn, just that some concepts' aha moment is less well-defined, or takes longer to attain.

Perhaps this is due to the fact that monads have a very small interface, and thus you must understand it completely to be able to use it. There's no partial understanding, and as such the 'aha' erlebnis is not smeared out.

Reply

Steve *says:*
January 13, 2009 at 10:43 am

Makes me think of Zen koans–the sudden enlightenment that's the result of an indeterminate amount of sometimes subconscious mental work, the way the insight can only be communicated by words or actions to someone who's already had it, etc.

Reply

apfelmus *says:*
January 13, 2009 at 11:11 am

Thanks for this great insight!

In other words, there is no way around discovering an abstraction oneself; of course guided by someone (teacher, text) who has already discovered it.

Reply

Mikael Vejdemo Johansson *says:*
January 13, 2009 at 12:59 pm

Look, it's really quite simple. A monad is just a monoid in the category of endofunctors on Hask. See?

Reply

Ferg *says:*
January 13, 2009 at 1:07 pm

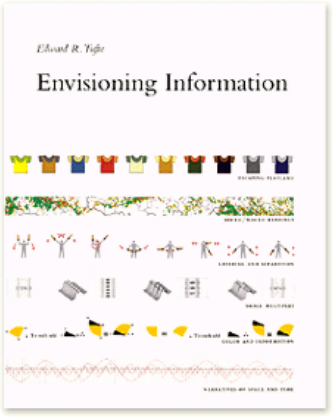
There is some confusion here in not paying attention to the difference between "How I learn" and "How everyone learns". Interesting in someone from the programming community, where differences in how people use manuals are very clear.

Reply

Jeff *says:*
January 13, 2009 at 1:31 pm

This is pretty much was good teaching is about – understanding the process of learning and crafting the lesson to optimize that process for the material.

One thing I'd love to see in every aspect of higher math (and higher math tutorials) is tutorial creators adopting one little thing from Tufte: creating not just one concrete example but a whole series of such and set them side by side to allow the "whole pattern" to present itself. You can see this in the cover of his book "Envisioning Information"



The shirts at the top illustrate "shirtness" in many canonical examples. The "feel" of flag semaphores are illustrated canonically as well. It's not that you are trying to learn every symbol represented (or may be you and you see the pattern involved also).

If someone does a tutorial on monads again (or group theory or topology or whatever), please let them use this one idea in their presentation of the tutorial.

It's actually a good test, in fact. If you can't come up with the example cases for such a canonical diagram or illustration, you probably don't understand the subject well enough to be doing a tutorial!!

Reply

Raoul Duke *says:*

January 13, 2009 at 3:31 pm

$0.02

I think I like http://ertes.de/articles/monads.html#section-4.

For me, the "laboriously threading a world as an extra argument" example makes the most sense.

Monads are a great brain exercise because they are very simple, but then quickly become complicated by all the nuances of using them in actual Haskell: which notation you choose; the error messages you get; trying to combine monads.

Also, I'd like to further your thought on the mistake people make by saying that people also learn better through different media: I really do better when there are good pictures involved, than just text and impenetrable math notation.

So, like, http://sigfpe.blogspot.com/2006/08/you-could-have-invented-monads-and.html is nice in that it has some pictures.

Reply

Jenny *says:*
January 13, 2009 at 3:33 pm

Just stumbled across this blog. I come to the idea of "monad" as a philosopher rather than a mathematician. Can anyone tell me how this mathematician's concept of "monad" is related to Leibniz's metaphysical concept of monad? That was the original definition of the term, as far as I know.

Reply

Jim Crayne *says:*
May 8, 2014 at 11:10 pm

I realize I am a few years behind replying here. But I recall the term Monad was also used by Robinson in his hyper-real calculus, that was in the 60s i think, when did the category theory usage begin? It may have been a progressive distortion where there actually was some leibnizian influence. I think Robinson's notion is slightly more reminiscent of Leibniz's idea, although certainly different, and it's name was probably more of a tribute than a refining of the concept.

http://en.wikipedia.org/wiki/Monad_%28non-standard_analysis%29

Did that play a role in the evolution of "Monad" as used by category theory, which inspired it's usage in the haskell language? I do not know the history… but…

It's not hard for me to imagine that a hyper-real monad is an example of a category theory monad with proper definitions of arrows and such.

http://en.wikipedia.org/wiki/Monad_%28category_theory%29

It's also not hard to imagine that the categoroy theory concept was devoloped completely independently and is also a tribute…

Here's a possible analogy to Leibniz, In terms of a one way monad, there is no way out, only in… If we think of analysis as a sort of getting something out of the monad, then this monad cannot be described in terms of its parts, which is what Leibniz said about his monads… Maybe that's a stretch… Would love to hear if anyone knows the true etymology.

Reply

Jim Crayne *says:*
May 8, 2014 at 11:22 pm

Another very fascinating observation, is both Haskell and Leibniz use their respective "monad" concepts to

answer this question:

How do we salvage the power of a static referentially transparent descriptive language in a universe where there is unpredictability (and free-will)?

Reply

Brent *says:*
January 13, 2009 at 4:42 pm

Jenny: it actually isn't related at all, as far as I know, except perhaps in a vague etymological sense. Mathematicians have this bad habit of taking nice-sounding words and appropriating them for their own use.

Reply

Eric *says:*
April 26, 2011 at 7:07 pm

And there in a nutshell is the problem I have with many explanations of mathematical concepts. I find that most explanations of advanced concept Q start off promising a simple explanation but then rely on the reader's deep understanding of M and C (which we won't bother to explain because what are you even doing here if you don't know the essence and uses of M and C?) In short, I have to "wikipedia" my way through the explanation one painstaking new concept at a time. I can't help but think that a less lazy teacher of concept Q would include explanations of M and C in their treatise as a matter of courtesy.

In this case,
Q = Monad
M = Lift
C = Bind
And that sort of use of wretchedly meaningless symbols (bad variable names) in math explanations is another pet peeve. Got you.

Reply

Dru Nelson *says:*
January 13, 2009 at 5:22 pm

So which monad tutorials are the best and the most concise?

Reply

Dan P *says:*
January 13, 2009 at 5:27 pm

A metaphor sets up a(n approximate) isomorphism between domains. The application of this is that a person who can reason successfully about one of the domains can then transfer this reasoning to the other. This is the primary reason for using metaphors to elucidate technical material. The first test of any proposed metaphor should be whether or not it does this. I think that in almost every case, the metaphors for monads fail.

There's an important flipside to this too. Not only should your metaphor allow you to transfer correct reasoning from one domain to another, it should also transfer incorrect reasoning to incorrect reasoning. Otherwise your metaphor is too general. So that's the second test for whether a metaphor is any good.

For example, the burrito metaphor is pretty poor as it stands. But suppose burritos had the property that whenevr you stuffed one burrito with a bunch more burritos, the tortilla around the inner burritos dissolved leaving just a single burrito. This would model the join property of monads quite nicely. That's a good candidate for passing the first test. But given a burrito you can freely scoop the contents out. So burritos fail the second test because you can do things with burritos that you can't do with all monads.

Reply

Dan P *says:*
January 13, 2009 at 5:29 pm

Jenny,

The word monad, as used by mathematicians, is a portmanteau of 'triple' (it's made from three parts) and 'monoid' (to which it is formally similar).

Reply

decourse *says:*
January 14, 2014 at 9:10 pm

I know this is five years later, but for the record, it's "triad", not "triple".

Reply

JoshG *says:*
January 13, 2009 at 5:57 pm

Monads aren't burritos they're vegemite sandwiches! Sheesh! ;-)

Do you fancy starting a pedagogical tutorial? It might even be best not to use any particular existing implementation (e.g. State as Keith asks) but to do something that develops into a vegemite sandwich each iteration and then voila! people see what a monad is by creating one themselves.

Reply

Jenny *says:*
January 13, 2009 at 5:57 pm

That's okay. Poets do the same thing, and interesting things can happen! I always liked the idea of muscling a word into a certain odd meaning.

Reply

Jenny *says:*
January 13, 2009 at 7:53 pm

Dan P.–well, if "mono" means "triple" or "three," I have to leave my portmanteau at the luggage locker.

Reply

James Craig Burley *says:*
December 22, 2010 at 10:13 am

I'm guessing "mon-" comes from "monoid", "-ad" comes from "triad".

Reply

Pingback: *Top Posts « WordPress.com*

Pseudonym *says:*
January 13, 2009 at 9:21 pm

Jenny:

It's a bit confusing here because the word "category" really was borrowed directly from philosophy, but "monad" wasn't.

Generally speaking, mathematicians like inventing words (e.g. groupoid) and naming things after people (e.g. Noetherian ring).

Computer scientists, on the other hand, like borrowing metaphors from other fields and real life. Words like "computer" and "printer", for example, used to refer to jobs that people did. A bundle of wires that "transport" data is called a "bus". You view your data through "windows". Data that has a rooted branching-type structure is called a "tree". My personal favourite is removing intermediate trees, which we call "deforestation".

We do this partly to avoid the problem in the article. The word "monad" is scary, because it's not based on an existing metaphor.

Reply

slacker *says:*
April 4, 2014 at 10:32 pm

I realize it's been 5 years… but for the benefit of new readers:

You got the etymology of at least the "bus" wrong. "Bus" was taken from the old electrical engineering concept of a "busbar", i.e. a solid rod or strip of metal meant to conduct very large currents and distribute power to multiple receivers attached to it. "Busbar" itself is a contraction of "omnibus bar", "omnibus" being Latin for "to all"

Reply

Jason Feingold *says:*
April 23, 2014 at 12:57 am

And the English word 'bus' (as in transport) is also derived from 'omnibus'.

Reply

Jenny *says:*
January 13, 2009 at 10:48 pm

I'm happy with the idea of different fields (computer scientists, mathematicians) using words in different ways, and the idea that mathematicians aren't borrowing from philosophy. But I seem to be grumpy when it comes to the word "monad," which was not a metaphor but is a simple translation of the Greek word "monos," which means "unit" or "one." Saying "monad" means"three" is, to me, like saying "monorail" means "it travels on three rails." It's not a metaphor. A semaphore, maybe. You say the word "monad" is scary. I'm not scared, are you?

Reply

Captain Oblivious *says:*
April 10, 2009 at 6:56 pm

The etymology for the mathematical monad is related to the Greek, and to Liebnitz. Monads essentially define predicates, in the sense of logic. These are the "monadic predicates" of the "monadic logic". The defining property of the monadic logic is that its predicates take only one argument — that its arity is 1. The "two argument" logic is called dyadic, and so on using Greek.

Here's the general idea, relatively abstractly. Consider a collection of things, with rules on how those things can be "connected". The properties of the collection that can be proved about the collection without constraint on what the objects "are" depends on the ways that the objects are connected. So lets consider the set (1, 2, 3), and say that x is connected to y by "<" if x is less than y. For example, we can prove that the set has 3 elements in it. It wouldn't matter if the set was (5.0, 1000, 9999). The only information you need is that there are three connections.

A monad is _a_ "totality of all things" (under consideration at a given point). It is an unescapable "environment". A domain of discourse, together with a valuation function. This ought to make the relation to Liebniz's monads pretty clear. The world as perceived, together with an "evaluator" — opinion maker, actor, and so on form a monad.

Reply

Todd Trimble *says:*

December 30, 2012 at 9:15 am

That's an invented etymology (of the *mathematical* term "monad") if I ever saw one. Much more likely, I believe, is that it's a back-formation from "monoid", a closely related concept.

Reply

Graham Wideman *says:*
February 27, 2013 at 6:08 am

You guys might be interested in my "etymology" thread at stackoverflow.
http://stackoverflow.com/questions/14090183/haskell-monad-etymology-versus-meaning

Reply

Mikael Vejdemo Johansson *says:*
January 13, 2009 at 11:26 pm

Jenny: I suspect that DanPs comment was in part a dig on the mathematics-internal history of the word Monad. The underlying concept was called, before the community kinda settled on Monad, both Triad and Triple – refering both to technical building blocks of the construction as such.

I cannot – in the few minutes I'm putting into it right now – figure out what a better (i.e. more accurate) theory for the etymology of the word would be; but if you for instance go to the Wikipedia page for Monads in category theory, and follow the external link to the lecture notes by Barr & Wells, they don't use Monad, but rather Triple in their exposition....

Reply

Jenny *says:*
January 13, 2009 at 11:48 pm

Thank you. I've looked at the Wikipedia page. That helps. I'm going to quietly disappear now. Goodbye.

Reply

Phil *says:*
January 14, 2009 at 7:01 pm

I think that this could all be easily cleared up if one of you FP guys would just show us how to write one of these monad thingies in Java...

Reply

Pingback: *The Real Adam – Monads + Ruby = crazy*

Pingback: *On keeping category theory and abstract algebra alive in Haskell « Integer Overflow*

Pseudonym *says:*
January 17, 2009 at 8:50 pm

> *I'm not scared, are you?*

No, I'm not. But apparently a lot of programmers are. Programmers like borrowing metaphors from real life, and "monad" isn't a borrowed metaphor.

Reply

Mikael Vejdemo Johansson *says:*
January 18, 2009 at 3:52 am

Phil: There are examples written, but Java doesn't lend itself particularly well to that kind of abstraction. Thus a formal monad sometimes looks really awkward in Java.

I do recall seeing, somewhere, though, code examples that show off Monad style arguments and phenomena written in Java. Buggered if I remember where, alas.

Reply

Pingback: *Fad diets are like burrito tutorials « blog :: Brent -> [String]*

Pingback: *a chicken monad « blog :: Brent -> [String]*

Michael Easter *says:*
February 27, 2009 at 11:23 pm

This is a wonderful post. You've nailed, spot-on, a complete, big idea that had only been beginning to surface in my own mind. Doubly so, as I'm preparing a talk on monads! (I feel that they are chimichangas, though ;-)

Reply

Michael Easter *says:*
March 2, 2009 at 12:33 am

I like this idea so much that I've named my talk "Monads are Burritos" (with a sense of irony), so as to remind myself of the danger of the fallacy.

Reply

Michael Easter *says:*
March 8, 2009 at 1:34 am

http://twitpic.com/1xavz

Reply

Curt Sampson *says:*
April 7, 2009 at 10:03 pm

I found a great monad tutorial: chapter 8 ("Functional Parsers") of Graham Hutton's book *Programming in Haskell*.

Interestingly enough, not only does it use no metaphors, but it doesn't even mention monads. It just walks you through the details of bind and return. I've been building my own monads quite happily since I've read it.

But it does start to beg the question, "for what purpose is a monad tutorial?" While I came out of that one happily able to write my own monadic parsers and suchlike, it did not leave me with the ability to use all the useful monads in the standard library. The "All About Monads" tutorial helped with that, but there's no question in my mind that the The Typeclassopedia is a very necessary thing.

cjs@cynic.net

Reply

Cory *says:*
April 23, 2009 at 6:26 am

I may be a bit late to the game here, but Phil, that can be rephrased:

"I think that this could all be easily cleared up if one of you OO guys would just show us how to write one of these object

thingies in Haskell…"

Of course you can, but it's a different type of abstraction for a different way of thinking about programming…

Reply

marcio *says:*
June 6, 2009 at 9:49 am

Man, i believe you stumped into something big. I mean, this "Monad Tutorial fallacy" might have single-handedly debunked Marx, quite a bunch of philosophers, a surprisingly big number of trendy scientific theories and maybe even 42! Seeing as your by-far-biggest tag is "Haskell" the blog might not be concerned about the reach of this idea beyond programming, but i am left wondering… The delicious irony is that the chosen title for the fallacy is actually an example of itself: "this fallacy is just like a bad Monad tutorial"! I would like to further explain myself, but i will have to spend some time evaluating how deeply guilty of this mistake i am myself — a lot, i fear…

Reply

Pingback: *Understanding monads « The Lumber Room*

inhahe *says:*
December 10, 2009 at 11:30 pm

ok, so are you going to write a tutorial on monads for us? ;P (or have you already?)

Reply

Ramkumar Ramachandra *says:*
December 11, 2009 at 5:18 am

Good post. Everyone is used to a certain level of abstraction, and this heightens with learning. While not going to the extent of teaching group theory to a small kindergarden kid, one should also be careful to avoid hard-coding patterns into students. Hardcoding patterns using examples inhibits further learning.

Reply

Eric *says:*
April 26, 2011 at 7:16 pm

I disagree. Abstractions should be learned bottom up by having students experience several "different in the specifics" examples which also have a commonality, and questioning/challenging them to discover what is common among the examples. Only then can you hope to give a simple expression for the essence of the abstraction and hope that it be understood.

Reply

wonk *says:*
December 15, 2009 at 1:21 pm

Mark-Jason Dominus has now proven that monads actually *are* like burritos.

Reply

Brent *says:*
December 15, 2009 at 2:38 pm

Actually—and this is an important distinction—he has only proven that burritos are monads. =)

Reply

Malcolm Gorman *says:*
December 16, 2009 at 12:13 am

I agree. I like BOTH concrete AND abstract attributes in a learning context. And that includes source code comments.

What's the best monad tutorial providing both?

Mal.

Reply

Brent *says:*
December 28, 2009 at 5:14 pm

I'm a bit biased, of course, but I recommend starting with the Typeclassopedia and then proceeding to follow up some of the citations if desired.

Reply

Andrew Winkler *says:*
December 18, 2009 at 7:56 pm

If you've got a mathematical background, then think of
a monad as any kind of completion. The real numbers as
a completion of the rationals under order, the algebraic
closure of a field, the cauchy sequences as a completion of a metric space, etc.

Any object can be embedded in its completion. That's return. Any object, completed twice, is the same completion. That's join. Maps among the objects extend
to mappings among the completions. That's fmap.

If your mathematical background is stronger, "completion" translates to "adjoint functors", and
monads are an essentially equivalent way of describing adjoint functors.

Reply

Eric *says:*
April 26, 2011 at 7:23 pm

Perhaps part of the problem here is that words like
bind
join
and
return
all have long-established meanings in the world of computer programming (join actually has two such meanings, one in concurrent programming and one in relational database querying).

Now correct me if I'm wrong but hasn't the monad stuff given completely different meanings (from the familiar-to-programmer meanings) in just about every sense to all three of these words,

Now that seems, to an experienced programmer, as just gratuitously mean, and ill-considered, when defining a relatively new PROGRAMMING concept.

Reply

Graham Wideman *says:*
February 27, 2013 at 6:28 am

... not to mention type, class and instance all referring to concepts that don't align well with their meanings in other languages.

Reply

decourse *says:*
January 14, 2014 at 9:20 pm

This is a year later, so sorry for the late contribution, but "class" is one of the few words that *does* align well with its meaning in mathematics.

The term "class" comes to us straight from Goedel-von Neumann-Bernays set theory. To overcome Russell's Paradox, they re-axiomatised set theory so that a set is any well-behaved collection of "things", but collections which aren't so well-behaved (e.g. "the set of all sets") is a "class".

Similarly, in programming languages, a "type" is a set/collection of values, and a "class" is a set/collection of types (i.e. a "set of sets").

The confusion, I think, is that when you write the word "class Foo" in many programming languages, you're actually doing three distinct things: You are declaring a new type called "Foo", and you are declaring a new collection of types also called "Foo", and you are declaring that the type "Foo" is a member of the collection "Foo". In Haskell, those three distinct things are separated out, which makes for more typing, but also more clarity.

Reply

Pingback: *Linktipps Januar 2010 :: Blackflash*

Pingback: *Tab Sweep: Monads*

Pingback: *Garbled » Blog Archive » Git Tutorials Suck, A Sucky Git Tutorial*

Jan Snajder *says:*
September 19, 2010 at 5:53 am

Very good post, thank you!

From my experience, getting to understand monads (or any abstract notion for that matter) is about developing an intuition about when and why they might be useful. This intuition has to be developed gradually on a series of examples. Only if I can see a pattern there, will I be willing to move to a more abstract level, otherwise I won't bother. Burrito analogies are useful, but only as a sort of private post hoc explanations.

Reply

Pingback: *intelligence about intelligence « Truth of the Lesser Men*

nothingmuch *says:*
January 11, 2011 at 2:16 pm

Feynman put it very well: http://www.youtube.com/watch?v=Cj4yoEUlU-Y

Reply

gvaerg *says:*
February 11, 2011 at 11:09 am

I finally understood monads today! (It was through the equivalence of bind with fmap/join that I got it.) Moreover, I thought immediately of this blog post that I have read months ago and I also understood why monads are like burritos. Thank you for this treat!

Reply

Lance Walton *says:*

February 17, 2011 at 5:48 pm

Two good introductions to monads:

* Yet Another Haskell Tutorial builds it up nicely (chapters 5 and 9 I think):
http://www.cs.utah.edu/~hal/docs/daume02yaht.pdf
* Daniel Spiewak’s blog: http://www.codecommit.com/blog/ruby/monads-are-not-metaphors

Reply

Graham Wideman *says:*
February 27, 2013 at 6:34 am

Yet another newer link: http://en.wikibooks.org/wiki/Haskell/YAHT

Reply

raving1 *says:*
September 22, 2011 at 9:58 am

“If you ever find yourself frustrated and astounded that someone else does not grasp a concept as easily and intuitively as you do, even after you clearly explain your intuition to them ….”

You describe …
… Fallacy!

Fallacy! |–> … indicates ‘Tautology’
‘Tautology’ = ‘Object’ <– Curry's paradox, Haskell
Fallacy!, 'Tautology', 'Object' = DESCRIPTION

"...implications for pedagogy. The heart of the matter is that people begin with the concrete ..."

No.
"...implications for pedagogy" ‘Gradient of descent’ for convergence]

————

“If you ever find yourself frustrated and astounded that someone else does not grasp a concept…”

There are other reasons. Your ‘monad tutorial fallacy’ is excellent!

… A fundamental reason is ‘Subject(ivity)’ [Closed DESCRIPTION]

‘Subject(ivity)’ and DESCRIPTION can be extended to ‘OPEN DESCRIPTION’ but it is difficult (to describe) and the working-of-the-process is described by your use of ‘Intuition’.

‘Subject(ivity)’ can be a timely activity involving the ‘over-arching’ span of islands of discontinuity.

A SINGLE ‘subjective experience’ is but a moment’s instantaneous (and temporally slurred) glimpsed experience and realization of reality.

DESCRIPTIONS can be formed by aggregating many ‘SINGLE momentary subjective experiences’ (DESCRIPTIONs)

————

Nice to find people who have a passion for categorization!
It’s a lonely world of understanding but not being understood.. …

Reply

Pingback: *xion.log » Adventures in Haskelland*

Pingback: *We have no idea what we are doing: exclusion in free software culture « Alex McLean*

Pingback: *Monads: Easy or Hard? | blog :: Brent -> [String]*

Alberto Gómez Corona *says:*
August 18, 2012 at 4:53 am

What you say is a disgrace of the whole educational system since at least the Enlightenment and the Age of Reason. It is derived from a wrong idea of the mind: the idea that it is a unique monolitical rational processing unit, instead of a collection of specialized modules. One of these modules is the one in charge of learning by example. The alumni have to repeat the process of discovery from examples to generality. If this is not provided by the teacher, the teacher is not a teacher but a "enlightened" rationalist exhibitionist.

Reply

slacker *says:*
April 4, 2014 at 11:37 pm

That's exactly why I call this era "Endarkenment".

Reply

Yang Jerng Hwa *says:*
October 5, 2012 at 8:10 am

Totally agree. My own thinking about this has previously led me to believe that artificial intelligence code should be architectured in this way. #2cents

Reply

Graham Wideman *says:*
February 27, 2013 at 6:24 am

I agree with the diagnosis: "failing to recognize the critical role that struggling through fundamental details plays in the building of intuition". But I'll claim that the solution isn't to force people to go through their own hours/days/weeks of wandering in the wilderness in the hopes of stumbling on the missing jigsaw pieces. It's to spell out the fundamental details properly, and leave no series of dots unconnected. The (well at least _a_) reason this is a particular problem in Haskell monads is that Haskell's treatment of monads, and the resulting syntax, is designed to _hide_ the details so as to be left with only a "beautiful" version of the mainline code that expresses the "main intent" (for some value of "intent"). All the stuff that is elided (or hidden under multiple layers of type substitution, syntactic sugar etc) in achieving that concision is the stuff that the beginner needs decompressed in order to know to see how it works. See the reader comments inline in Real World Haskell's online version for copious examples of crucial dots not joined, and that book is a relatively carefully prepared one!

Reply

Pingback: *My take on monads | The Pleasure of Trying Things Out*

Adrian May *says:*
March 18, 2013 at 10:37 pm

It's an interesting thought but I think the problem is not with abstractions in general, but whether or not those abstractions are already familiar to the reader. In the case of monads, people using the examples-first approach usually start with boxes, but that just stores up problems for later.

I think of monads as glue code. That's a pretty general abstraction that every C programmer knows about – we've all been bored by writing chains of pedantic error handlers and logging mantras. One can suggest that if you have a mechanism for factoring out the glue from the blocks, then you might try to design all the blocks to be compatible with one type of glue.

Then you can work through the parser, state, etc, monads, always emphasising that they are examples of what we said at the outset: glue. This approach also avoids the pitfall of people thinking more monads make a better program, just as OO newbies write millions of classes to earn brownie points. Often a simple applicative functor is all the glue you need.

Reply

ben *says:*
July 9, 2013 at 2:44 am

I think part of what you're saying is that monad tutorials are written for the writer and not the reader : someone has an a-hah moment and they want to share it, but what they are sharing is their a-ha moment and not knowledge, or the process of acquiring knowledge. My favorite book ever in all of my math studies is Atiyah and MacDonald's Commutative Algebra. It's basically a big list of exercises and just enough definitions and theorems to get you through them. It's remarkably short and remarkably easy to learn from, and a lot of the most interesting stuff is developed in the exercises. Instead of writing endless beautiful explanations, they just give the reader the tools to figure it out for themselves. It's a masterpiece. Perhaps someone should write the monad tutorial in this style — a series of exercises. (Perhaps someone already has.)

Reply

Brent *says:*
July 9, 2013 at 7:51 am

Hmm, that's a really good idea!

Reply

Josh Graham (@delitescere) *says:*
May 13, 2014 at 10:58 pm

Suggested a while ago… https://byorgey.wordpress.com/2009/01/12/abstraction-intuition-and-the-monad-tutorial-fallacy/#comment-1025

Reply

Pingback: *The Monad Tutorial Fallacy | Ted Goranson's ISIS-US Blog*

Pingback: *A collection of monad tutorials and a few other miscellaneous things. | BrokeTheBuild*

Pingback: *Figuring out the State Monad | SAdams in Kent*

Pingback: *Monads are hard because … | The Endeavour*

Pingback: *What are monads and why are they useful? | The Pleasure Of Finding Things Out*

Pingback: *My Math Diary*

Abstractor *says:*
July 14, 2014 at 4:44 pm

Very insightful article which went from teaching the "monad tutorial fallacy" to the more abstract "tutorial fallacy". Thank you.

Reply

Brad C. *says:*
August 20, 2014 at 4:31 pm

Hi Brent. I'm not a Haskell programmer, but I found your insight that "people begin with the concrete, and move to the

abstract" to be very fascinating. Would you mind elaborating on that–how you came to that conclusion? I'm interested in regards to other contexts, namely compassion, suffering, and group identity. Thanks!

Reply

Curt Sampson *says:*
August 20, 2014 at 10:16 pm

That learners start with the concrete and move to the abstract is actually and old and quite well-known idea in the field of learning. Start by looking in to the Dreyfus model of skill acquisition (the Wikipedia article on this is a good starting point) and you'll find plenty of places to follow up from there. The best book I've read on the model is the Patricia Benner book mentioned in the references in that article. (Even though it's about nursing, I found it extremely applicable to software development as there seem to be many similar skills and learning situations involved; it's a book that I think that all software developers should read.)

Reply

Pingback: *Types will carry you over the Monads | Funkcionálně.cz*

Pingback: *Monads are like burritos | kewl beans*