Thoughts and Opinions about Programming
Fay-lee-nuh/
Source:
https://threadreaderapp.com/thread/1474325931741397000.html

Ok technically my holiday has started, but I thought it'd be fun to do a @threadapalooza on programming and all the things I have thought about over the past year!

Let's see how close I can get to a hundred opinions and thoughts about programming & education! (1/100)

Programmers in the workforce today are still largely self-taught in childhood. That still shapes our thinking on education. Implicitly or explicitly, many programmers believe that if you love programming, you will have found it as a child (2/100)

Related to the self-teaching: Many programmers never took a programming lesson and thus don't really know what one could look like. They thus implicitly think that a programming lesson should look like their experience: kids figuring out stuff by themselves (3/100)

If you are self-teaching programming, basically the compiler('s error messages) are your only teacher. Hence we have all been a bit "Stockholm syndromed" into thinking compilers are lovely teachers (4/100)

Compilers are not lovely teachers. Error messages are meant for professionals and even for them, they are often hard to understand, and only helpful because you have learned that "syntax error unexpected EOL" means you probably forgot a quotation mark somewhere (5/100)

Ok, on to the next topic: Papert and constructionism. Seymour Papert was undoubtedly an amazing teacher, but I don't think he understood how amazing he was. His way of teaching by guiding kids with exploration is only doable with lots of knowledge of programming (6/100)

Most teachers do not have the knowledge and skill to teach programming in the way that the early proponents like Papert advocated for, and that has been one of the factors that has hampered adoptions for decades (7/100)

To restate the previous point: telling teachers to teach a brand new topic and also to teach it in a radically new way is too much and does not work. We need to meet teachers where they are and integrate programming into their ways of teaching (8/100)

In many countries including mine, that means less exploration, less computer time, and more practice and direct instruction (9/100)

Saying with Papert, switching to another topic, the profound "anti-school" movement within programming education. In Mindstorms, Papert writes that things he taught himself as a kid were more valuable than "anything I was taught in elementary school" (10/100)

> BEFORE I WAS two years old I had developed an intense involvement with automobiles. The names of car parts made up a very substantial portion of my vocabulary: I was particularly proud of knowing about the parts of the transmission system, the gearbox, and most especially the differential. It was, of course, many years
>
> I believe that working with differentials did more for my mathematical development than anything I was taught in elementary school. Gears, serving as models, carried many otherwise abstract

Firstly as an aside: as a teacher I hate it with deep burning hate when people use their own school experiences as guidance for how things should be taught, because well n = 1 but also people that say these things are very often privileged and have a very limited scope (11/100)

Like people that say this often are the type of people for which their worst experience in school has been to be bored, but not be bo hungry or not to not being able the see the board for lack of glasses or to not understand a word the teacher said (12/100)

Back to Papert, let that quote sink in. All of learning in elem. school to him is worth less than his own exploration of cars. I know it is meant as an exaggeration but come on man... Anyway, the point is: anti-school and programming education are heavily related (13/100)

You still see this in our idols like Zuckerberg, Jobs and Gates being dropouts and "look they still made it" and Thiel's scholarship where kids were encouraged to not go to college. Guess it is not hard to see the glaring privilege there (14/100)

Anyway, people often ask me how it is possible that there still is no mandatory programming education, and I think one answer that we don't talk so often about the fact that at its core, "CS" (whatever it is) believes schools are bad and make kids dumb and less creative (15/100)

That just is not a really great way to get something adopted (see also tweet 8) (16/100)

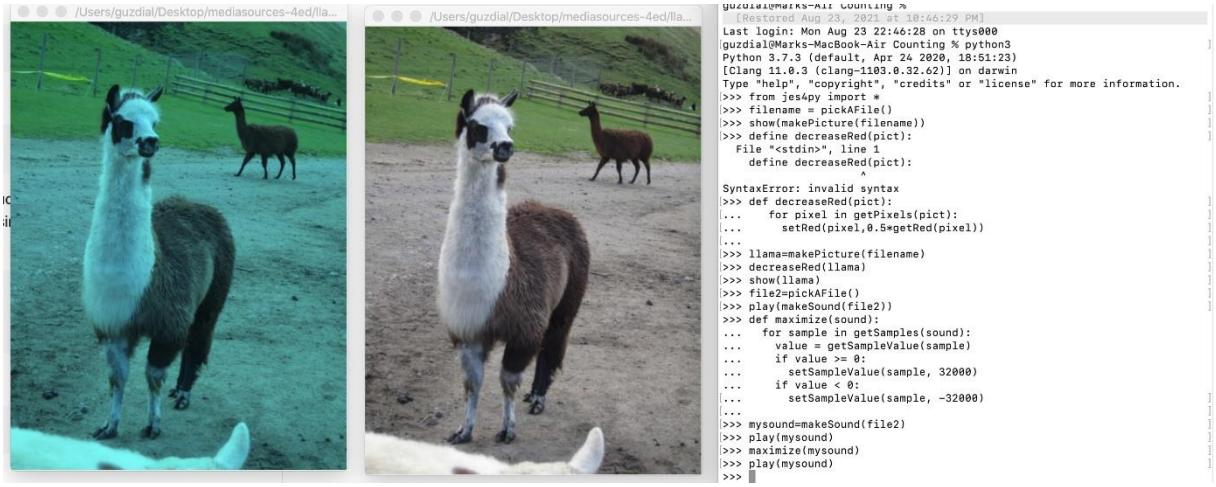
Ok, on to the next topic: the role of learning to program!

Everyone can learn to program, there is no special gene or type of brain that prevents people from learning the basics of programming. You do not have to be "born for it" (17/100)

Every person benefits from knowing *a little bit* of programming, not to become programmers, but to do some simple calculations in a spreadsheet or to be able to whip up a simple website. Like all kids learn to write but they will not all be Stephen King, that is fine (18/100)

Every field benefits from people knowing a little bit of programming. From natural language processing or image processing to being able to make an app, just knowing these things exist and are possible makes everyone more effective (19/100)

Mark @guzdial's Teaspool Languages are an amazing metaphor for this, see further: computinged.wordpress.com/2021/09/06/med… (20/100)



Media Computation today: Runestone, Snap!, Python 3, and a Teaspoon LanguageI don't get to teach Media Computation1 since I moved to the University of Michigan, so I haven't done as much development on the curriculum and infrastructure as I might like if I were teaching it…https://computinged.wordpress.com/2021/09/06/media-computation-today-runestone-snap-python-3-and-a-teaspoon-language/

There are many reasons why there are not a don't of non-white non-male programmers, and luckily nowadays we are talking about many of those reasons: lack of role models, microaggressions, and straight-up discrimination (21/100)

But there are also a few reasons we don't talk about so often, so let's do that now (need to get to a hundred, after all...)
One eye-opening framework for me has been GenderMag (see gendermag.org)
Gendermag described 3 different personas that use computers (22/100)

And it describes how women (in general of course, not all women!) tend to want to forage information and do not like to tinker with stuff until it works. So in short: "Don't bother with the manual cause developers won't read it" disproportionally affects women! (23/100)

In education, I see similar things. Girls more often want to know what the goal of an assignment is and how to "do it the right way". Just fiddling around for the fun of it is not likely to enjoy them, on the contrary, it makes them insecure and thus *less* engaged! (24/100)

This makes sense if you think about girls coming into a CS classroom with lower self-efficacy, less prior knowledge, sometimes open stereotypes ("my brother told me I would fail this course!"). So of course in that scenario, you want to know how to do it properly! (25/100)

So not only role models and topics matter in CS classes for retention: instructional strategies also impact who feels at home! Broadly speaking more exploration will benefit kids that feel like they belong and already know a bit, more likely to be white boys than others (26/100)

Ok and now for the most explosive tweet of them all (I once said this at a workshop and was laughed at out loud, but I am trying again here...):

If programming languages/tools would be made by women, they'd look radically different (27/100)

Women tend to care more about what others think others (whether you believe that is nature or nurture is not relevant here) So they think a lot more about other users, rather than the "eat your own dogfood" approach, which would lead to more useable products (28/100)

I don't want to toot my own horn (narrator: she did want to toot her own horn) but the idea of @hedycode is super simple. Anyone who has taught kids knows that learning a lot at once is hard, so I am sure several teachers came up with this idea but could not build it (29/100)

The fact no one has ever built a PL of which the syntax changes over time says a lot about the little regard for the novice user our community has in general. I can't separate that from my female identity of feeling I need to do better for kids (30/100)

Like if a kid struggled in my class, I did not think: "they are just not smart enough", rather I would take it personally and think: "I need to do better, I need to *be* better to allow them to learn" (31/100)

And of course, it is also about allowances here. When I started to teach in high school as an academic, people thought that was really weird, but I am quite sure for a man it would even be seen as weirder. For me at least if fitted people's image of the "nurturing lady" (32/100)

And yeah I know there are many male teachers and programmers out there who also care deeply about others!!! Just as a community we have generally accepted or even rewarded hard to use tools under the guise of "if you can't use this, it is your fault" (33/100)

Women tend to blame themselves if people can't understand their tools, which is generally a good thing in programming language or tool design! (34/100)

Ok since we are on the topic of programming language design... maybe this is explosive too but:

This syntax of a programming language is a user interface and should be designed as such: by experts in usability, not by compiler builders (35/100)

If you don't believe me, maybe you will believe @AndreasStefik who did an experiment showing that students using Perl and Java did not outperform students using his programming language Randomo which uses random keywords! learntechlib.org/p/167915/ (36/100)

An Empirical Investigation into Programming Language Syntax Recent studies in the literature have shown that syntax remains a significant barrier to novice computer science students in the field. While this syntax barrier is known to exist, whether and how it …https://www.learntechlib.org/p/167915/

Switching gears now a bit toward the praxis of doing educational research. I don't know why but I can't do research without being immersed in the field I am researching. When I wrote my thesis on spreadsheets, I interned at an investment bank a day a week for years. (37/100)

For education, actually, the reverse happened! I started to teach programming, cause I did not know what I wanted to do with my life, I started to research it and then

over time it became my main topic (more on that in @strangeloop_stl talk:) (38/100)
I find it increasingly hard and frustrating that I seem to be one of the few researchers that are so intertwined with the field. Partly this is envy on my part! I don't know how I could do research without teaching but if I could it'd save me a lot of time haha! (39/100)

Sometimes, I blame individual researchers for their lack of engagement, like recently someone asked if I knew schools for them to work with. I asked "to do what?" and she said, "I don't know yet, I want to do co-creation with them!" Such things annoy me to no end (40/100)

Because it shows a lack of understanding of how schools work, how you build credit there. But of course, we need to look at the system, not at individual actors in it. As researchers we are not rewarded for the deep insights we get from reading or teaching or thinking (41/100)

So if I say: "I need to teach to do research" in the eyes of some academic bean counters that does not make sense because how am I using that time in the classroom to WrITe PaPERs? Simply being there to learn, to get inspired or so counts for diddly squad in our system (42/100)

I truly deeply believe that without my years of teaching experience, I would not have been able to come up with @hedycode, cause I was scratching my own itch there (or rather scratching the kids's itches but that sounds weird). But only the Hedy *papers* matter! (43/100)

Really a senior academic close to me was getting really excited about the millions of Hedy programs that have been created, not because, I don't know, clearly we are solving a problem and helping schools but because of all the papers we could write about the data (44/100)

I feel less and less at home in that academia; for I want to *do* things, build tools, write educational materials, research *with* school what works rather than to see them at labs that we can just use. (45/100)

And yeah I can do all these things now, and I do, but it feels like I am somehow cheating, like I am slacking on my actual job to secretly do things that I am not really supposed to do, which feels bad! Anyway, as I say often... maybe I am too practical for academia :D (46/100)

Ok, new topic! Programming education in university!

I recently wrote a column (in Dutch) in which I admitted that I am slowly realizing

that not all that keen on teaching programming in university anymore. This surprises people because well don't I love teaching?? (47/100)
Firstly, I think in itself might be a bit of gender bias mixed with wishful thinking. My direct colleagues want to think I love teaching so they'll feel less guilty about giving me a big teaching load. And if people keep telling you you love a thing, it must be true...? (48/100)

And yeah, I like teaching high school, and I love teaching in the teacher's college at the @VUamsterdam I like to make slides, come up with assignments, I even find grading is fun and useful! So why don't I enjoy teaching undergrads to program? (49/100)

The penny dropped for me only when thinking about the teacher's college. Teaching there is so fun, because that it is so crystal clear what we want students to be able to do at the end of their program: teach high school! (50/100)

Using that lens to look at the computer science courses I finally understood my own frustration! There is no clear picture at all about what jobs CS students will have after college, and therefore there is no guidance in deciding what they have to learn (51/100)

Many undergrad programs assume (sometimes implicitly) that CS students will become scientists, and therefore have to master a lot of theory. In reality (like it or not) most of our graduates naturally become programmers (52/100)

This comes up every time in different ways! For example, should students learn to use an IDE? “No,” says camp education-for-science, “they will learn that later, it is not a core competency" “Yes of course” says the other side, “that is indispensable for their career!” (53/100)

The same tension also occurs over CI/CD, software testing, DDD, and so on. Making decisions costs me a lot of energy in this context; if there is no clear end goal, how to decide whether or not to explain how to fold code or apply a refactoring? (54/100)

This is also bad for students: different professors make different decisions, resulting in a messy and incoherent learning experience. Use the command line in one course, but an IDE in another. I think it is time to face the fact: most graduates won't become scientists (55/100)

Talking about our student population, let's dive into another topic. *Why* do we think our students should choose CS as their major? Lately, we see more and more students coming into CS because they (rightly!) think it is a path to a good career (56/100)

Some students that might have selected traditional well-paying careers like law or medicine, now choose CS and that does not sit well with some people. No, they say! You should *love* programming! Otherwise you do not fit in here. This is an interesting contrast here (57/100)

Most people think it's fine if a kid wants to do law for the money, or if they want to do law to "make the world a better place" both are fine reasons that most people don't think of as weird. But liking CS for the money is dirty, and to change the world feels... off (58/100)

We should embrace this new generation of students; they ask new and interesting questions, and broader our view! Also... they might not be here forever (one day the web3 bubble will burst and numbers will do down as they did in the early zeroes) and then we'll miss them! (59/100)

Also... this leads to an interesting paradox! CS programs are in top-10 lists of "quickest to get a job after graduation" every year; so as a student you do get to enjoy job security, but you should not be motivated by it. I don't know, isn't that a bit... disingenuous? (60/100)

Thanks y'all for reading along, the reception of this has been fantastic!!! Didn't think I would make it to 60 and I have run out of topics (this is more or a less all I thought of while running this morning) so I am taking a small programming break (61/100)

I got soooo many lovely questions, a coffee and some snacks, so let's do this!!
First up: Does @scratch help new learners?

The key question here is: help with what? Scratch 100% comes from the Papert philosophy: it is basically a lego box with blocks, and kids have to explore to learn the meaning of blocks (62/100)

Often teachers wonder where Scratch's lesson plan is, but that is not missing by accident but by design, according to constructionism, kids should not be limited in their creation by a trajectory a teacher has cooked up (63/100)

So if your goal is to use Scratch to teach concepts, you need a good teacher to help kids "see through" the block into generic concepts. Lacking that, I think Scratch might not help or even hinder, creating misconceptions that can spill over into other langs (64/100)

Also, Scratch entirely hides the execution model, which makes debugging excruciatingly hard. If a variable changes value unexpectedly, there is no way to add a watch on it (heck I can't even see what sprites touch the variable!) (65/100)

Hence Scratch does not teach some of the practices of programming (in textual languages) such as debugging or commenting out suspicious code. So if that is your goal, Scratch might not be helpful either. (66/100)

And because debugging is so hard because there are many hidden dependencies, some kids (and teachers!) drop out of Scratch because it is quite hard to get some simple things done (67/100)

But!!! If your goal is to show how easy programming *can be*, with a set of well-chosen and somewhat guided lesson plans, Scratch can be a great way of showing kids that programming is not scary and that code can be a great great way to express your ideas! (68/100)

Next up: bootcamps!

I think bootcamps can be a great way of teaching people employable skills helping them make money and expand the workforce. However, bootcamps have often (unknowingly) drank a big gulp of the "teach yourself" Kool-Aid (69/100)

There are wonderful counterexamples (shoutout to @jesslynnrose!! classcentral.com/cohorts/webdev…) but many bootcamps assume a lot of skills (like: start cmd or know how to install Python on Windows) so they could be better if they taught more and let people swim a bit less (70/100)

Free Web Development Bootcamp | Class Central Cohorts This six-week bootcamp with weekly live streams is led by technologist Jessica Rose and built around freeCodeCamp's Responsive Web Design Certification. You'll learn HTML, CSS, and accessible and resp…https://www.classcentral.com/cohorts/webdev-bootcamp-spring-2022

How to cater to different interests in programming?

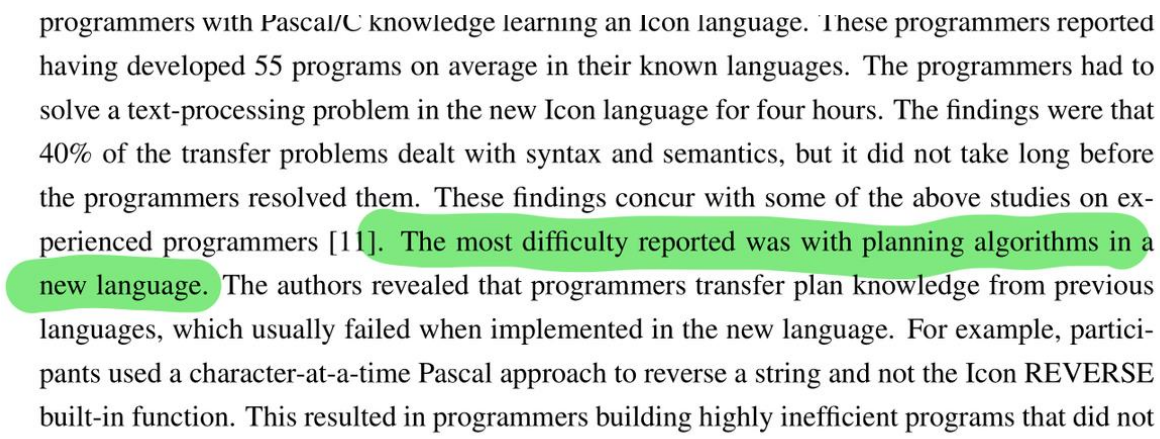
Great question!!! I struggle with that too and I have no magical solution.
With @hedycode we have introduced different "adventures" that show similar concepts (71/100)

With a loop you can draw a square, but also sing a song ("baby shark!!") or create interactive fiction (a hero walks through several different rooms). That not only caters to different audiences but can also help strengthen conceptual understanding if you do them all (71/100)

On problem-solving:

My answer is that the blocks into which you divide a problem heavily depend on the

language you use, so you need to know the language first. See f.e. Scholtz and Wiedenbeck (screenshot stolen from @EthelTshukudu) (72/100)



programmers with Pascal/C knowledge learning an Icon language. These programmers reported having developed 55 programs on average in their known languages. The programmers had to solve a text-processing problem in the new Icon language for four hours. The findings were that 40% of the transfer problems dealt with syntax and semantics, but it did not take long before the programmers resolved them. These findings concur with some of the above studies on experienced programmers [11]. The most difficulty reported was with planning algorithms in a new language. The authors revealed that programmers transfer plan knowledge from previous languages, which usually failed when implemented in the new language. For example, participants used a character-at-a-time Pascal approach to reverse a string and not the Icon REVERSE built-in function. This resulted in programmers building highly inefficient programs that did not

Secondly, solving a problem takes a lot of mental effort. Spending only a little bit of that on syntax takes energy away from solving the problem. And learning syntax, in principle, is easy. With a bit of practice, you can easily learn it so that is a good investment (73/100)

On examples with bogus names!!

Love this topic and while I do not know any research on it, I do have an opinion (surprise surprise) I think these types of examples stress the idea that people learn programming for programming's sake (74/100)

For example, take a code snippet like this:

```
def foo(x):
print(x)
```

yes, this explains what a function is... But it doesn't explain why you'd use one. This is only clear if you already know what a function is, and you just lack knowledge of how to do it in Python (75/100)

Similarly, code like this (which yes, I too have used in teaching)

```
x = 5
if x == 5:
print(“yes”)
else:
print(“no”)
```

does not make sense! Like why not just do print("yes")? It only explains what not why and that hampers learning for certain types of novices (76/100)

Metacognition!

Papert famously said: "It is hard to think about thinking, without thinking about thinking about something"

Programming is great because it does solidify thought into an executable form (77/100)

And programming education has a lot of tools at hand to teach metacognition but sadly it does not always do that in a systematic way. @gvwilson get his panties in a justified knot every once in a while about the lack of courses on debugging and I think he is right (78/100)

Teaching students to make state tables of variables, to write down and then check hypotheses, to debug in a systematic fashion, to write clear comments, to document and test well are all metacognitive tools and we do not teach them (yet??) (79/100)

On non-English programming languages!!!

One of the coolest things I learned this year has been how much programming is dominated by English, see this thread I did this summer:
(80/100)

Hedy has made some progress on keywords in Dutch and Spanish (and allowing them to be mixed with English as @guzdial suggested) but it is a lot of work and many tools are not really made to support this, like what parse generator allows you to insert different keywords? (81/100)

And we are not yet changing statements to be more natural to different mother tongues, for example, my native language says (translated of course) "if x 5 is", so I'd like to override the if to be `if x 5 =` in Dutch but yeah you see how that is a parsing nightmare :) (82/100)

Ok, one more topic because I have to start dinner in a bit:

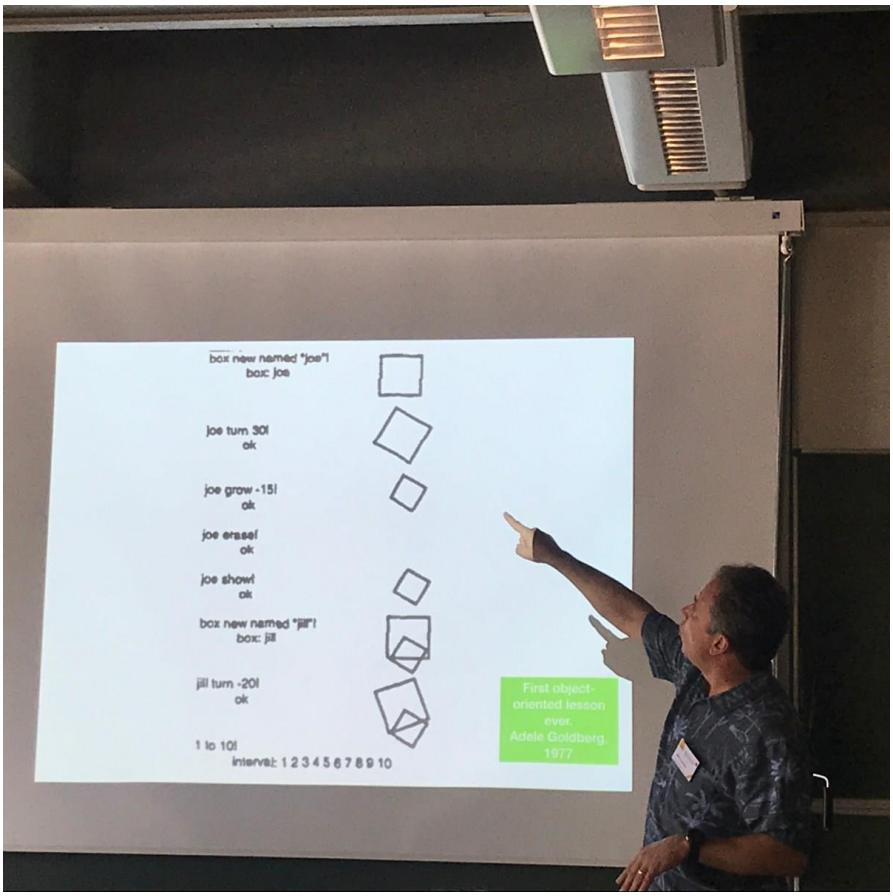
The most important thing we teach when we teach programming is the execution model of a programming language (83/100)

For example, in Hedy, loops used to show all at once (like Python does because we generate Python) That is confusing for kids because it does not stress the execution model of repetition. So now we have added a little pause (84/100)

This is why I think notebooks (while lovely for data processing!) are not appropriate tools for teaching introductory programming, they break the fundamental execution model of Python/of programming (85/100)

I consider a student "good" at Python if they can effortlessly predict the execution of a program. Of course, this involves a lot of conceptual knowledge too but ultimately you can only type code of you can predict consistently and effortlessly what the code will do (86/100)

This relates a lot to the concept of a "notional machine": the machine that you use to think with. I struggled a lot with what a notional machine is (see also felienne.com/archives/6375) but this year it clicked for me while explaining it to students! (87/100)



So… what is a notional machine?We are here for three days now, so let's discuss the actual top: notional machine. We were asked in groups to consider this and our group came up with this answer: "A notional machines …https://www.felienne.com/archives/6375

When you as a teacher say something like "and now what happens in the computer is x gets the value 5" that is a notional machine, you are telling the learner how the machine works, and if they remember it, it will become the machine they use when reasoning about code (88/100)

I also think the core difference between a metaphor ("electricity flows like water") and a notional machine ("x now has the value 5") is that programmer me really uses

the notional machine of a variable with a name having a value (rather than a binary number on a stack) (89/100)

It is not a lie I tell students to explain something, it is mostly how I really think about code; it is the actual machine of my notion (90/100)

ok room for a few more questions!

Not at all an easy read, but "Toward a Developmental Epistemology of Computer Programming" has been such an eye-opener for me! dl.acm.org/doi/10.1145/29… (91/100)

Ow.... nice new topic: physical computing, a good or a bad idea for all kids?

Physical computing is a hard question. On the one hand: tangible things are cool, and physical computing allows for really fun projects integrated in other disciplines (92/100)

Like @RolfHut's smart umbrella is a fantastic way of engaging kids that care about climate change in programming bbc.com/news/science-e… (93/100)

Smart umbrellas 'could collect rain data'A Dutch scientist has plans to turn our umbrellas into rain gauges to help fill in the gaps in weather data.https://www.bbc.com/news/science-environment-27222282

On the other hand, as a way of teaching programming concepts... I am not sure. It is brittle: a perfect program but a wire in the wrong port breaks. And there is a lot of

distance between the program and its execution, making it hard to learn about the execution model (94/100)

Also, my experience is that teaching things that look "technical" will disengage girls; they quickly worry it will be hard and not for them. Of course we should tell them that is not the case, but we still might be giving them the impression that code is "technical" (95/100)

More generically, I am a bit fan of integrating programming in other courses (see the Teaspoon languages) however in practice very often programming ends up integrated with only math or physics and that might be counterproductive for some kids (96/100)

To end on a low note (sorry!) I think some programming lessons have a negative net effect, I do not think "anything is better than nothing". Badly executed very exploratory guest lessons my (well-meaning) parents might end up confirming the idea that programming is hard (97/100)

And guest lessons my professionals are very likely to be given by white male professionals confirming harmful stereotypes (98/100)

So I guess if you are a programming parent that wants to help their kids's school implement programming, I would prefer it if you help the teacher do it rather than if you offer guest lectures (99/100)

Maybe you can help them get started with @hedycode, which is free, open source, available in 17 different languages, has really lovely error messages and implements a lot of the ideas in this thread (100/100)