Coding Is for Everyone—as Long as You Speak English
Code depends on English—for reasons that are entirely unnecessary at a technical level

This year marks the 30th anniversary of the World Wide Web, so there's been a lot of pixels spilled on "the initial promises of the web"—one of which was the idea that you could select "view source" on any page and easily teach yourself what went into making it display like that. Here's the very first webpage, reproduced by the tinker-friendly programming website Glitch in honor of the anniversary, to point out that you can switch to the source view and see that certain parts are marked up with <title> and <body> and <p> (which you might be able to guess stands for "paragraph"). Looks pretty straightforward—but you're reading this on an English website, from the perspective of an English speaker.
Now, imagine that this was the first webpage you'd ever seen, that you were excited to peer under the hood and figure out how this worked. But instead of the labels being familiar words, you were faced with this version I created, which is entirely identical to the original except that the source code is based on Russian rather than English. I don't speak Russian, and assuming you don't either, does <заголовок> and <заглавие> and <тело> and <п> still feel like something you want to tinker with?

In theory, you can make a programming language out of any symbols. The computer doesn't care. The computer is already running an invisible program (a compiler) to translate your IF or <body> into the 1s and 0s that it functions in, and it would function just as effectively if we used a potato emoji 🥔 to stand for IF and the obscure 15th century Cyrillic symbol multiocular O ❁ to stand for <body>. The fact that programming languages often resemble English words like *body* or *if* is a convenient accommodation for our puny human meatbrains, which are much better at remembering commands that look like words we already know.
But only some of us already know the words of these commands: those of us who speak English. The "initial promise of the web" was only ever a promise to its English-speaking users, whether native English-speaking or with access to the kind of elite education that produces fluent second-language English speakers in non-English-dominant areas.
It’s true that software programs and social media platforms are now often available in some 30 to 100 languages—but what about the tools that make us creators, not just consumers, of computational tools? I'm not even asking whether we should make programming languages in small, underserved languages (although that would be cool). Even huge languages that have extensive literary traditions and are used as regional trade languages, like Mandarin, Spanish, Hindi, and Arabic, still aren't widespread as languages of code.
I've found four programming languages that are widely available in multilingual versions. Not 400. Four (4).
Two of these four languages are specially designed to teach children how to code: Scratch and Blockly. Scratch has even done a study showing that children who learn to code in a programming language based on their native language learn faster than those who are stuck learning in another language. What happens when these children grow up? Adults, who are not exactly famous for how much they enjoy learning languages, have two other well-localized programming languages to choose from: Excel formulas and Wiki markup.
Yes, you can command your spreadsheets with formulas based on whatever language your spreadsheet program's interface is in. Both Excel and Google Sheets will let you write, for example, =IF(condition,value_if_true,value_if_false), but also the Spanish equivalent, =SI(prueba_lógica,valor_si_es_verdadero,valor_si_es_falso), and the same in dozens of other languages. It's probably not the first thing you think of when you think of coding, but a spreadsheet can technically be made into a Turing machine, and it does show that there's a business case for localized versions.

Similarly, you can edit Wikipedia and other wikis using implementations of Wiki markup based on many different languages. The basic features of Wiki markup are language-agnostic (such as putting square brackets [[around a link]]), but more advanced features do use words, and those words are in the local language. For example, if you make an infobox about a person, it has parameters like "name = " and "birth_place = " on the English Wikipedia, which are "име = " and "роден-място = " on the Bulgarian Wikipedia.

In addition to these four widely available, multilingual programming languages, there are several dozen, maybe a hundred or so, programming languages that are available in a language or two other than English, such as Qalb (Arabic), Chinese Python, farsinet (Persian), Hindawi Programming System (Bengali, Gujarati, and Hindi), and even a version of Perl but in Latin. Several non-English programming languages even date back to the era of government-sponsored room-sized megacomputers, such as several Soviet-era programming languages that were based on Russian and the multilingual languages ALGOL 68 (1960s) and 4th Dimension (1980s). But many newer languages, like Python, Ruby, and Lua, come from non-English speaking countries (the Netherlands, Japan, and Brazil) and still use English-based keywords. The initial promise of the web is, for many people, more of a threat—speak English or get left out of the network.

These languages exist because it's not difficult to translate a programming language. There are plenty of converters between programming languages—you can throw in a passage in JavaScript and get out the version in Python, or throw in a passage in Markdown and get out a version in HTML. They're not particularly hard to create. Programming languages have limited, well-defined vocabularies, with none of the ambiguity or cultural nuance that bedevils automatic machine translation of natural languages. Figure out the equivalents of a hundred or so commands and you can automatically map the one onto the other for any passage of code.

Indeed, it's so feasible to translate programming languages that people periodically do so for artistic or humorous purposes, a delightful type of nerdery known as esoteric programming languages. LOLCODE, for example, is modeled after lolcats, so you begin a program with HAI and close it with KTHXBAI, and Whitespace is completely invisible to the human eye, made up of the invisible characters space, tab, and linebreak. There's even Pikachu, a programming language consisting solely of the words *pi*, *pika*, and *pikachu* so that Pikachu can—very hypothetically—break away from those darn *Pokémon* trainers and get a high-paying job as a programmer instead.

When you put translating code in terms of *Pokémon*, it sounds absurd. When you put translating code in terms of the billions of people in the world who don't speak English, access to high-paying jobs and the ability to tinker with your own device is no longer a hypothetical benefit. The fact that code depends on English blocks people from this benefit, for reasons that are entirely unnecessary at a technical level.

But a programming language isn't just its technical implementation—it's also a human community. The four widespread multilingual programming languages have had better luck so far with fostering that community than the solitary non-English-based programming languages, but it's still a critical bottleneck. You need to find useful resources when you Google your error messages. Heck, you need to figure out how to get the language up and running on your computer at all. That's why it was so important that the first web browser let you edit—not just view—websites, why Glitch has made such a point of letting you edit working code from inside a browser window and making it easy to ask for help. But where's the Glitch for the non-English-speaking world? How do we make the web as tinker-friendly for the people who are joining it now (or who have been using it as a consumer for the past decade) as it was for its earliest arrivals?

Here's why I still have hope. In medieval Europe, if you wanted to access the technology of writing, you had to acquire a new language at the same time. Writing meant Latin. Writing in the vernacular—in the mother tongues, in languages that people already spoke—was an obscure, marginalized sideline. Why would you even want to learn to write in English or French? There's

nothing to read there, whereas Latin got you access to the intellectual tradition of an entire lingua franca.

We have a tendency to look back at this historical era and wonder why people bothered with all that Latin when they could have just written in the language they already spoke. At the time, learning Latin in order to learn how to write was as logical as learning English in order to code is today, even though we now know that children learn to read much faster if they're taught in their mother tongue first. The arguments for English-based code that I see on websites like Stack Overflow are much the same: Why not just learn English? It gains you access to an entire technological tradition.

We know that Latin's dominance in writing ended. The technology of writing spread to other languages. The technology of coding is no more intrinsically bound to English than the technology of writing was bound to Latin. I propose we start by adjusting the way we talk about programming languages when they contain words from human languages. The first website wasn't written in HTML—it was written in *English* HTML. The snippet of code that appears along the bottom of Glitch's reproduction? It's not in JavaScript, it's in *English* JavaScript. When we name the English default, it becomes more obvious that we can question it—we can start imagining a world that also contains Russian HTML or Swahili JavaScript, where you don't have an unearned advantage in learning to code if your native language happens to be English.

This world doesn't exist yet. Perhaps in the next 30 years, we'll make it.