

DPI

De Programmatica *Ipsium*

DE PROGRAMMATICA IPSUM

Issue 013:
Programming
Languages

October 7th, 2019

Table of Contents

Issue 013: Programming Languages	5
De Programmatica Ipsum Volume 2	9
Choosing A Programming Language	13
Memories Of Hello World	17

Issue 013: Programming Languages



October 7th, 2019

Welcome to the thirteenth issue of *De Programmatica Ipsum*, dedicated to the subject of *Programming Languages*. In this edition, the first of Volume 2 of this magazine:

- Adrian explains some major changes¹ in the structure and flow of this magazine.
- Graham exhorts us to think about mother Earth² when choosing a programming language.
- Adrian dives into memory lane looking for his first lines of code³.

ISSUE 013: PROGRAMMING LANGUAGES

Enjoy this issue! Please subscribe to our free newsletter⁴ to stay updated about new releases.

Cover photo by Shahadat Shemul⁵ on Unsplash⁶.

REFERENCES

¹ <https://deprogrammaticaipsum.com/de-programmatica-ipsu-2/>

² <https://deprogrammaticaipsum.com/choosing-a-programming-language/>

³ <https://deprogrammaticaipsum.com/memories-of-hello-world/>

⁴ <https://deprogrammaticaipsum.com/newsletter/>

⁵ https://unsplash.com/@shemul?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

⁶ https://unsplash.com/s/photos/languages?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

De Programmatica Ipsum

Volume 2



By Adrian Kosmaczewski

Welcome to the second year of De Programmatica Ipsum. Maybe you will have noticed some subtle, small changes in the layout of this magazine. Let us explain them to you in detail.

De Programmatica Ipsum started as an idea thrown in a chat conversation back in August 2018. Graham and the author of these lines missed magazines such as

Dr. Dobb's¹ and Byte², particularly because of their closer focus on the practitioners, on the programmers. Thus was born the first issue³ of this magazine, published exactly one year ago.

Back then, the idea was to provide a tribune, an arena for the exchange of ideas by programmers for programmers in all cultures, countries, and origins. We wanted to contribute financially to those writers, for their time and effort, and we wanted to avoid advertising in these pages.

Unfortunately that model did not stand the test of time. The subscription model did not work for this magazine, and the work required to find and interact with prospective writers was far more than we expected.

Hence we have decided to change the formula of this magazine starting today. We have made the content of this magazine 100% free. We have also refunded all of our current subscribers.

Regarding contributions and financing, we still refuse to use advertising in these pages. We are looking for ethical, simple options for those who would like to help us carry this project forward. We will let you know about that soon on ~~Twitter~~⁴ Mastodon⁵ and here.

Volume 2 of De Programmatica Ipsum will be produced just by Graham and this author. We still believe that we need a different perspective about the craft of programming—a more human-centered, reflexive, and empathic, view.

We would like to thank all of our readers. We would also like to thank in particular those who took the time to subscribe, who supported this idea since day one.

But in particular, we would like to thank our writers, who have created such incredibly relevant and passionate pieces, and who trusted us with this precious content. This magazine would not be the same without your help. In alphabetical order: Agis Tsaraboulidis, Adam Jones, Adrian Tineo, Anastasiia Vixentael, Carola Nitz, Daniel Steinberg, Fernando Rodriguez, Guido de Caso, Ioana Porcarasu, Julia Cacciapuoti, Marie-Cécile Godwin Paccard, Roland Leth, and Susanna Riccardi.

Here is Volume 2 of De Programmatica Ipsum. We hope you will enjoy reading it, as much as we enjoy writing it.

Cover photo by Anastasia Zhenina⁶ on Unsplash⁷.

REFERENCES

¹ https://en.wikipedia.org/wiki/Dr._Dobb's_Journal

² [https://en.wikipedia.org/wiki/Byte_\(magazine\)](https://en.wikipedia.org/wiki/Byte_(magazine))

³ <https://deprogrammaticaipsum.com/issue/issue-001-hype/>

⁴ <https://twitter.com/deprogipsum>

⁵ <https://mas.to/@deprogrammaticaipsum>

⁶ https://unsplash.com/@disguise_truth?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

⁷ https://unsplash.com/s/photos/volume-2?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

Choosing A Programming Language



By Graham Lee

Selecting the correct programming language for a particular task often feels like a bunfight. Suggestions are thrown around based on the individual's preferences and desires. The experience is very likely to leave one feeling that programming is an art, rather than a science¹.

Let us frame a programming language selection exercise with some real-world considerations. As software engineering leaders, it's our responsibility to design our systems and processes such that they efficiently solve the problem at hand, and

of course tool selection is an integral part of such design. Here's an opportunity to practice that skill.

Which programming language should I choose to avert the climate catastrophe?

The destruction of the climate by industrial processes is a real-world problem. If programming language selection has no impact on it, then maybe we're all just wasting our time, creating shiny consumer distractions to make the last few years of humanity that little more palatable for the richest 1% of the world's human population.

Computers, of course, use energy. The Semiconductor Industry Association forecast² in September 2015 that before 2040, the world demand for computers would exceed the world's ability to supply those computers with electricity. It's not just the computers that consume the power, though. Computers sat in data centres usually need heating, ventilation and air conditioning. Where they don't, they're probably not near their users (who *do* need HVAC) so additional networking hardware is required. They may not be so close to their operators either, who need to drive out to the data centres in their (currently mostly) petrol-powered vehicles.

On the personal/mobile/IoT side, computers are synonymous with highly-polluting materials extraction³ associated with battery technology. Battery chargers themselves are inefficient. Worse, consumers often leave their smartphones charging overnight: while this isn't dangerous⁴, it does leave the phone trickle-charging, shortening the battery's lifespan and consuming additional energy. Many manufacturers do not allow for user-replaceable batteries. Therefore, when a device's battery becomes unusable, a whole new replacement device must be manufactured and installed.

Many consumer computers are connected to screens and other energy-using devices, that are inefficient at reacting to context. Some smartwatches, for example, only light their screens when they are in use, but others have inefficient, always-on displays.

There are individual applications of computers with measurable power consumption characteristics. Across the US, data centres collectively consumed 70 billion

kWh in 2014⁵. This is currently the amount of electricity estimated to power the bitcoin network⁶.

Maybe we need to carry on research into more efficient computers, and maybe select programming languages based on computational efficiency. That efficiency may not be measured in instructions needed to complete a task, but in ability to use heterogeneous computing equipment⁷ effectively. Optimising for efficiency leads us into the realm of Jevons' paradox⁸: increasing the efficiency of a resource's consumption increases the consumption of the resource. It does not reduce the consumption. Just think what we could do with all those spare CPU cycles if we deduplicated decoding of Rick Astley videos, or reduced the hash puzzle complexity of a cryptocurrency.

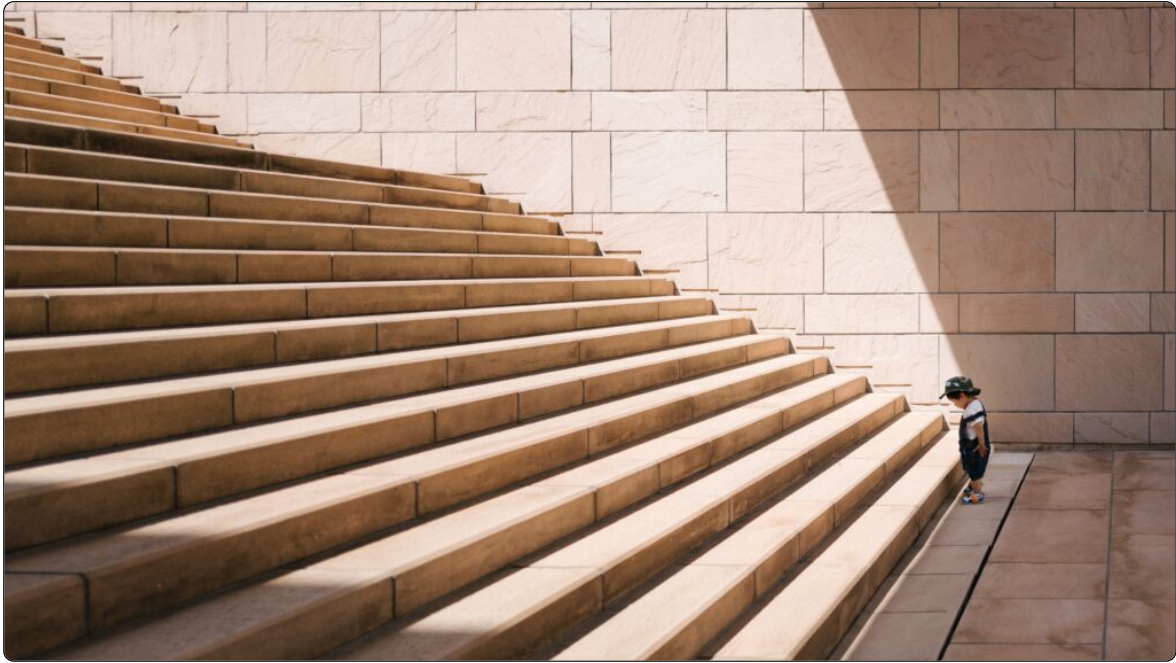
So, Javascript, Go, or Swift?

Cover photo by Martin Adams⁹ on Unsplash¹⁰.

REFERENCES

- ¹ <https://deprogrammaticaipsum.com/issue-4-programming-art-or-science/>
- ² <https://www.semiconductors.org/wp-content/uploads/2018/05/RITR-WEB-version-FINAL.pdf>
- ³ <http://corporatejustice.org/news/36-cobalt-blues-environmental-pollution-and-human-rights-violations-in-congolese-cobalt-mines>
- ⁴ <https://www.pcmag.com/news/357987/charging-your-phone-overnight-battery-myths-debunked>
- ⁵ <https://www.datacenterknowledge.com/archives/2016/06/27/heres-how-much-energy-all-us-data-centers-consume>
- ⁶ <https://digiconomist.net/bitcoin-energy-consumption>
- ⁷ <https://www.arm.com/why-arm/technologies/big-little>
- ⁸ <https://www.sciencedirect.com/science/article/pii/S0921800905001084>
- ⁹ https://unsplash.com/@martinadams?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText
- ¹⁰ https://unsplash.com/s/photos/climate-warming?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

Memories Of Hello World



By Adrian Kosmaczewski

Once upon a time, during the summer of 1992. It was, if I remember well, around the 15th of July. I wrote my first line of QBasic¹ code in a blue text editor, fueled by the eponymous PRINT command.

I pressed the F5 button to execute it.

A line of white text appeared on a dark screen.

Just that.

I still have the files somewhere. They have the .BAS extension. It was the first code I ever executed in that PC, my first real computer. A machine powered by an Intel 386 SX CPU clocked at 25 MHz, with 2 MB of RAM, a hard disk of 128 MB, and a SoundBlaster card.

But, wait. It was not the first time I programmed a computer. In school, back in the 1980s, I used a Casio fx-180p calculator² with a memory for 38 instructions. That is right, you could program any formula in it, as long as it required less than 39 instructions. I used it mostly for solving quadratic equations during exams. That's cheating, I know. In the Argentina of the 1980s, it felt like using alien technology in front of my teachers.

Technically, that calculator was a computer. Small, limited, and painfully so. But I bet it was Turing complete and all and all.

But in 1992 it was a completely different level. I had a real keyboard. I had a machine that could execute more than 38 instructions. QBasic even came bundled with sample code for a game, called Gorillas³, and the rumor went that Bill Gates himself had written it. Imagine that.

QBasic was simple. Fully procedural, with SUB and FUNCTION keywords (hint: the latter returned values, while the former did not, but essentially, they were the same thing: subroutines.) No object orientation, no closures. There were WHILE loops and IF statements.

It was very similar to JavaScript in terms of experience; no explicit compilation phase; just write the code and press F5 (which, curiously enough, still is the essence of web development, almost 30 years later.)

I programmed in QBasic day and night, and then I would play Aces of the Pacific⁴ when I got bored or frustrated with the language. Which was very often.

I went to the Librairie Ellipse⁵ in Geneva and bought a book about QBasic. I wanted to learn more. It was the first computer book I ever bought. It was not very good, actually, but little did I know.

Writing code was an exercise in patience. My second line was obviously buggy. I could not get the syntax right. I did not know that you could “debug” a program, and execute it step-by-step. I did not know that you could use breakpoints and inspect the values of variables.

I did not know anything about programming. I had no idea who Turing was. I had no idea of paradigms, patterns, IDEs, not even how many other programming languages there were.

By 1993, I added a CD-ROM player to my computer, and I ordered a couple of CD-ROMs from Walnut Creek⁶. One of them had folders and folders of source code in a variety of languages with very strange names: REXX, Forth, Basica, ADA, Lisp, Fortran. I did not know what those languages were for. I did not know why there were so many. I did not know I could create my own. There was no Stack Overflow, there was no GitHub.

There was no Internet, actually. Not yet, at least. That was in December 1994. I connected to `comp.lang.pascal` in Usenet and asked questions. I

logged anonymously to `ftp.funet.fi` to download freeware stuff via FTP. Some of that was compilers and editors.

I was still mostly using MS-DOS `edit`. I had not heard about `emacs` or `vim`.

After that came Turbo Pascal. We had to learn it in University, to simulate some laws of mechanics. We actually had exams about Turbo Pascal without a computer; we had to write the source code of a recursive function on paper, and we would have points taken away if we forgot semicolons.

This is how they taught us programming in university back then. But I do not remember them telling us how to reverse a linked list.

Turbo Pascal had “Units” which were libraries that you could reuse from project to project. I started collecting bits and pieces of code from the Walnut Creek CD-ROMs that I used in my projects, and without knowing I started applying the DRY principle in my work.

I also wrote my first class and my first objects with Turbo Pascal. I somehow figured out the Singleton and the Adapter design patterns without knowing that the Gang of Four were busy writing a book about them⁷, roughly at the same time.

Maybe some design patterns, like Singleton and Adapter, are just simply natural steps happening while writing simple code snippets. But the Abstract Factory and the Flyweight are not. Neither is MVC or React or other architectures.

Those days, we would meet with other fans of programming in the cafeteria of the “Sciences II” building of the University of Geneva. We would exchange ideas, dream about writing code for a living.

In October 1997, I got my first job as a software developer. But you know that story⁸.

If I could go back in time, I would have recommended my younger self to read the first edition of “Code Complete” in its entirety, before taking that job.

Cover photo by Jukan Tateisi⁹ on Unsplash¹⁰.

REFERENCES

¹ <https://en.wikipedia.org/wiki/QBasic>

² https://www.calculator.org/calculators/Casio_fx-180P.html

³ [https://en.wikipedia.org/wiki/Gorillas_\(video_game\)](https://en.wikipedia.org/wiki/Gorillas_(video_game))

⁴ https://en.wikipedia.org/wiki/Aces_of_the_Pacific

⁵ <http://ellipse.ch/>

⁶ <https://archive.org/details/walnutcreekcdrom>

⁷ https://en.wikipedia.org/wiki/Design_Patterns

⁸ <https://akos.ma/blog/being-a-developer-after-40/>

⁹ https://unsplash.com/@tateisimikito?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

¹⁰ https://unsplash.com/s/photos/beginning?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText