HOW TO SPEAK MACHINE

COMPUTATIONAL THINKING FOR THE REST OF US

JOHN MAEDA

HOW TO SPEAK MACHINE

Computational Thinking for the Rest of Us

John Maeda

PORTFOLIO // PENGUIN

Portfolio/Penguin
An imprint of Penguin Random House LLC
penguinrandomhouse.com

ISBN 9780399564420 (hardcover)
ISBN 9780399564437 (ebook)
ISBN 9780593086322 (international edition)

Library of Congress Control Number: 2019949382

Printed in the United States of America
10 9 8 7 6 5 4 3 2 1

BOOK DESIGN BY NICOLE LAROCHE

Contents

Introduction

It was a typical cold, snowy New England winter day on December 17, 2004, when I started a WordPress "web log" on the topic of simplicity. There was a whimsical motivation to turning my research at MIT in that direction—namely, how the letters MIT occurred in perfect sequence in the words SIMPLICITY and COMPLEXITY. But there was a less Dr. Seussian motivation at play as well, which I wrote about in my first blog entry:

> I have always been interested in how the computer (which is an object of great complexity) and design (which is traditionally about simplicity) tend to mix poorly together like the proverbial "oil and water."

Subsequently, that blog turned into a book titled *The Laws of Simplicity*, which was rapidly translated into fourteen languages. Why was it unusually impactful? I think because it arrived at a time when computing technology was just starting to impact everyday lives back in the pre-iPhone era. That book's overwhelming momentum and the concurrent rise of Apple's successful fusing of design and technology oddly drove me to head in the opposite direction of computing's inherent complexities and instead toward designing for simplicity.

I wanted to somehow get closer to the essence of design and move away from computers the way I had done once prior in my early career—back in the nineties, when I was a practicing graphic designer in Japan with a mismatched MIT pedigree. I'd somehow managed to escape the "T" (Technology) of MIT as an engineering student, and then made a U-turn into the thick of it as an MIT Media Lab professor leading the intersection of design and advanced computing technologies. Perhaps it was dealing with the weight of earning tenure that made me feel stuck somewhere in the future of design. I wanted to reconnect with the classics. I think it was a mix of not knowing what to do with the MBA I'd earned as a part-time hobby and an overwhelming mood in 2008 of Barack Obama's "Yes, we can," combined with my desire to rekindle the past, that resulted in my becoming the sixteenth president of the esteemed "temple" of the art and design world: Rhode Island School of Design, or RISD.

A series of later milestones, which built upon the work I had long led at the MIT Media Lab, gave me a reputation as a fierce defender of design. These included, for instance, appearing before Congress to encourage putting an "A," for Art, in STEM education, turning it into STEAM, and later launching the "Design in Tech Reports" while working in Silicon Valley at venture capital firm Kleiner Perkins. So in 2019, when a popular business magazine announced in a headline that I'd said, "In reality, design is not that important," it did not come as a surprise to me that I would be dragged through the internet mud by all lovers of design.

There were many short- and long-form responses to the interview that cast my responses as every shade of clueless and ignorant. I knew that the controversy had hit a peak when my idol,

Hartmut Esslinger, the design force behind Apple's original design language, started coming after me on social media. If I had earned some sort of "badge" in the design world, the internet was now ruling that it was my public duty to turn that badge in, and for a period of time I would be unwelcome at any Temple of Design out there. How did I feel? Terrible.

My words had been taken out of context from a twenty-minute phone interview—and, frankly, when the article came out, I immediately admired the editorial team's choice of headline as brilliant clickbait. Apparently, my interview was the highest-performing article on their website for quite some time, as evidenced by the myriad spears and cleavers that were continuously being lobbed in my direction. What stung the most was knowing that few people had really read the entire interview, so the headline was all that stuck in anyone's mind. To them, I had completely demeaned the work designers do every day. So I needed to be punished.

Here's the reality: I honestly don't believe that design is the most important matter today. Instead, I believe we should focus first on understanding computation. Because when we combine design with computation, a kind of magic results; when we combine business with computation, great financial opportunities can emerge. What is computation? That's the question I would get asked anytime I stepped off the MIT campus when I was in my twenties and thirties, and then whenever I left any technology company I worked with in my forties and fifties.

Computation is an invisible, alien universe that is infinitely large and infinitesimally detailed. It's a kind of raw material that doesn't obey the laws of physics, and it's what powers the internet at a level that far transcends the power of electricity. It's a

ubiquitous medium that experienced software developers and the tech industry control to a degree that threatens the sovereignty of existing nation-states. Computation is not something you can fully grasp after training in a "learn to code" boot camp, where the mechanics of programming can be easily learned. It's more like a foreign country with its own culture, its own set of problems, and its own language—but where knowing the language is not enough, let alone if you have only a minimal understanding of it.

There's been a conscious push in all countries to promote a greater understanding of how computers and the internet work. However, by the time a technology-centered educational program is launched, it is already outdated. That's because the pace of progress in computing hasn't moved at the speed of humans—it's been moving at the exponential speed of the internet's evolution. Back in 1999, when a BBC interviewer made a dismissive comment about the internet, the late musician David Bowie presciently offered an alternate interpretation: "It's an alien life form . . . and it's just landed here." Since the landing of this alien life form, the world has not been the same—and design as it has conventionally been defined by the Temple of Design no longer feels to me like the foundational language of the products and services worlds. Instead, it's ruled by new laws that are governed by the rising Temple of Tech in a way that intrinsically excludes folks who are less technically literate.

A new form of design has emerged: computational design. This kind of design has less to do with the paper, cotton, ink, or steel that we use in everything we physically craft in the real world, and instead has more to do with the bytes, pixels, voice, and AI that we use in everything we virtually craft in the digital world powered

by new computing technologies. It's the text bubble that pops up on your screen with a message from your loved one, or the perfect photo you shot in the cold rain with your hands trembling and yet which came out perfectly, or the friendly "Here you go, John" that you hear when you ask your smart stereo to play your favorite Bowie tunes. These new kinds of interactions with our increasingly intelligent devices and surroundings require a fundamental understanding of how computing works to maximize what we can make.

So I came to wonder if I could find a way for more nontechie people to start building a basic understanding of computation. And then, with that basic conceptual grounding, to show how computation is transforming the design of products and services. For much of the twentieth century, computation by itself was useful only to the military to calculate missile trajectories. But in the twenty-first century, it is *design* that has made computation relevant to business and, more so, to our everyday lives. Design matters a lot when it is leveraged with a deep understanding of computation and the unique set of possibilities it brings. But achieving an intuitive understanding of an invisible alien universe doesn't come so easily.

This book is the result of a six-year journey I have traveled away from "pure" design and into the heart of what is impacting design the most: computation. I will take you on a tour through the minds and cultures of computing machines from how they once existed in a simpler form to how they've evolved into the much more complex forms we know today. Keep in mind that this book is not designed to turn you into a computer science genius—I've vastly simplified, and in some cases oversimplified, technical

concepts in a way that will surely raise some experts' eyebrows and might cause them to cringe. But I hope that, armed with even my rough approximations, you will learn how computation has expanded both in technical capability and in sociocultural impact—something you may regard simultaneously as hugely impressive and terribly frightening.

Computation brings its share of problems, but most of them have to do with us—how we use it—rather than by the underlying technology itself. We've entered an era in which the computing machines we use today are powered not just by electricity and mathematics, but by our every action and with insights gained in real time as we use them. In the future we'll have only ourselves to blame for how computation evolves, but we're more likely to succumb to a victim mentality if we remain ignorant as to what is really going on. So it will become only natural to want to pin the blame on a handful of tech company leaders, if not all of them—a fairly likely scenario, since fear of the invisible or unknown is far more powerful than any fear that has physical form, like a pack of wild wolves or a threatening tornado. The intangible, invisible alien force that is the internet represents the perfect object of fear that already lives in your neighborhood and teaches your children while secretly seeking to do you all harm, which explains why the eeriness of tech is so widely portrayed in TV and movies today.

I have always believed that being curious is better than being afraid—for when we are curious we get inventive, whereas when we are afraid we get destructive. Something about my experience between the Temple of Design and the Temple of Tech has kept me curious all these years. When I think more deeply about it, it's

the many career failures I've fortunately experienced in between my few successes that drive me to still remain a little hungry and foolish. But to be honest, I'm just like anyone else—tired, a little lazy, and all too eager to wait for a hero to rise who will protect me and fight for all of us. There's a common lack of understanding of what computation fundamentally can and cannot do. Rather than give away your power of understanding to someone else, I invite you to be curious about the computational universe.

Perhaps I wrote this book for you. Perhaps *you* are the hero the world has been waiting for. Perhaps you are one of the many who will find a way to wield the power of computation with inventiveness and wonder. Those kinds of heroes are now desperately needed in order to advance computation beyond what it is today in its superpowerful, albeit running with the conflicted conscience of a teenager, form. Being new to the computational universe, you just might discover something that we first-generation techies have not yet been able to imagine. When you find it and make it into a success, it will set an example for the rest of us. I wish that heroic moment upon you someday, but first let me start you on the path to speaking the language of the machine.

Chapter 1

MACHINES RUN LOOPS

1 // Computers excel at repeating themselves through loops.

2 // Hard machines are visible; soft machines are invisible.

3 // Human computers are the original computing machines.

4 // Recursion is the most elegant means to repeat oneself.

5 // Loops are indestructible unless a programmer has made an error.

1. Computers excel at repeating themselves through loops.

Physical education was never my forte growing up. Not only was I the second-fattest kid in class, but I also couldn't for the life of me throw or catch a ball of any size. My one special power was staying awake for a long time, thanks to the influence of my father and the boot camp mentality with which he ran the family business. But that little bit of physical pride meant nothing when running loops around the school and constantly getting lapped by all my classmates. I always felt it was so boring, not to mention tiring, to run those loops around the school grounds. Although I wasn't particularly good at it, I knew I wasn't the only exhausted one—even the fastest runners looked winded by the time I got to the finish line. Athletic or not, we're all animals that, no matter how physically fit, will tire eventually.

There's one thing that a computer can do better than any human, animal, or machine in the real world: repetition. It doesn't get bored and complain if you tell it to count from one to a thousand, or even to a billion. You just tell it to start at zero, add one, and repeat until your target number is reached. And the computer just goes off and does it. For example, I type in these three lines of code into my computer:

```
top = 1000000000
i = 0
while i < top: i = i + 1
```

which compels the computer to count to one billion, within one minute, and it's then waiting for me to give it a new set of instructions to execute under my complete control. It's eager to please me.

Let's think about that for a second. The hamster on the treadmill spins it round and round fast, but eventually tires. The Formula One racing car races the track at high speeds but eventually runs out of gas and needs to stop. If the hamster were to not stop, we would be rather concerned. Our first thought would be, *Freaky!* If the F1 race continued with no need for the cars to make a pit stop, we'd think to ourselves, *Magic!*

But a computer running a program, if left powered up, can sit in a loop and run forever, never losing energy or enthusiasm. It's a metamechanical machine that never experiences surface friction and is never subject to the forces of gravity like a real mechanical machine—so it runs in complete perfection. This property is the first thing that sets computational machines apart from the living, tiring, creaky, squeaky world.

I first came in touch with the power of computational loops in 1979, when I was in seventh grade. I had just encountered my first computer. That in itself was unusual for someone with my background growing up on the poor side of town; thanks to desegregation efforts motivated by the civil rights movement, I was bused to a school an hour away from my home that was much better than the run-down schools in my neighborhood.

Commodore was a big name in computing at the time. When I say big, of course I mean big for maybe a few thousand people in the United States and Europe. Personal computers weren't really personal back then, because the average family could not afford one.

The Commodore PET (Personal Electronic Transactor) was manufactured in the United States with a small screen displaying text in only fluorescent green, a tiny tactile keyboard, and a tape cassette drive for storage. It had 8 kilobytes of total memory, and its processing speed was 1 megahertz. In comparison, the average mobile phone has 8 gigabytes and runs at 2 gigahertz, which is a millionfold increase in memory and a thousandfold increase in speed.

There was no internet yet, so you couldn't search for anything. There was no Microsoft, so you couldn't do your schoolwork on a word processor or spreadsheet. There was no touch screen or mouse, so you couldn't directly interact with what was on the monitor. There were no color or grayscale pixels to display images with, so you couldn't visually express information. There was only one font, and text was only uppercase. You would "navigate" the computer screen by pressing the cursor keys: up, down, left, right. And any functionality that you wanted it to have, you would need to type in a program to create it yourself or type it in line by line from a book or magazine.

As you can imagine, the computer sat in the classroom generally unused—it was not only useless, it was soulless as an experience. No expressive or informative images. No stereo sound or the latest tunes. No utility or empowerment with an amazing set of apps. It just blinked at you, constantly, with its cursor rectangle—awaiting you to type in instructions for it to follow. And when you did raise the courage to type something into it, you would likely be rewarded with, in all caps, SYNTAX ERROR—which essentially translated to YOU'RE WRONG. I DON'T UNDERSTAND.

It's no surprise that the computer attracted only a few kinds of students—perhaps those who grew up a bit lower on the empathy scale (like me), or those who could tolerate the crushing blow of

being told they were wrong with each keystroke. My friend Colin, whose parents worked with computers at Boeing, showed me my first program. He rapidly typed the following program into the PET, free from any syntax errors:

```
10 PRINT "COLIN"
20 GOTO 10
```

Then he told me to go ahead and type RUN . . . What happened next astounded me. The computer began printing "COLIN" continuously. I asked when it was going to stop. Colin said, "Never." This worried me. He then proceeded to break the program with CONTROL-C. And then the blinking text prompt came back.

Colin then retyped the first line of code, but this time with one space and a semicolon.

```
10 PRINT "COLIN ";
```

And he typed RUN to display the following:

```
COLIN COLIN COLIN COLIN COLIN COLIN COLIN COLIN COLIN COLIN COLIN
COLIN COLIN COLIN COLIN COLIN COLIN COLIN COLIN COLIN COLIN COLIN
COLIN COLIN COLIN COLIN COLIN COLIN COLIN COLIN COLIN COLIN COLIN
COLIN COLIN COLIN COLIN COLIN . . .
```

And again, it kept printing and scrolling by. I tried it myself, and typed:

```
10 PRINT "MAEDA ". . .
```

to experience the incredibly affirming display of my name being said forever and ever:

```
MAEDA MAEDA MAEDA MAEDA MAEDA MAEDA MAEDA MAEDA MAEDA
MAEDA MAEDA MAEDA MAEDA MAEDA MAEDA MAEDA . . .
```

Thereafter, I would perform this magic "say my name" trick for anyone who was curious about the computer. I did it for my classmate Jessica, who I kind of had a thing for. The limits of my computer expertise became evident when she asked, "What else can you do?" *Uh-oh*.

But my curiosity was piqued. I started to read *Byte* magazine (one of maybe two computing magazines available). Since there was close to no software available, it was really important to learn to write programs. *Byte* regularly included entire computer programs printed out across many pages and ready to be manually typed in to a computer yourself—the only problem was that I didn't have a computer to regularly use.

Luckily, my mother, Elinor, was always forward-looking and hopeful that her children could do bigger and better things in life. She set aside enough money from our small, family-operated tofu shop in Seattle to buy me an Apple II computer and an Epson line printer. To express my gratitude to her, I wanted my first computer program to help her in some way at the tofu shop. Thus I set out to write a monthly billing program that I hoped could save her some time. It would manually take in our regular customers' orders each week and print out an invoice at the end of the month.

I was a fast typist by the tenth grade, and so with zeal I wrote this program that I felt could help my mother. It took me maybe

three months of programming every day after school. My conundrum was figuring out how to deal with leap years—I figured that if I made an input routine for 365 days in a year, I would run into a problem every four years. In the end I saw that as a bridge to cross when I came to it, so instead I just kept typing and typing until I had completed all 365 input routines (there were no text editors with copy and paste functions). It was a manual, laborious project. I recall the deep satisfaction I felt when my mother first used it to print invoices for the month.

Shortly after this moment of success, my tenth-grade math teacher, Mr. Moyer, encouraged me to come to his after-school computer club. I had gained a reputation for being skilled at computer programming—perhaps what I should really say is that I was a computer nerd. Having successfully written my first thousand-line computer program, I thought it would be beneath me to go to Mr. Moyer's club meeting as I would surely be too much of an expert compared with the others attending. But on the day I showed up, I vividly remember Mr. Moyer talking about LOOPS using a command called FOR . . . NEXT. Upon learning about it, I broke into a sweat—the kind of sweat you feel when you've done something terribly stupid.

When I got home that evening, I looked at my long computer program, which was 365 distinct input statements of the form:

```
10 DIM T(365), A(365) : HOME
100 REM GET THE NUMBER OF TOFU AND SUSHI AGE FOR EACH
    DAY OF THE YEAR
110 REM COMMENTS LIKE THIS ARE HOW PROGRAMMERS TALK TO
    THEMSELVES
```

```
120 PRINT "IT'S DAY 1"
130 PRINT "HOW MANY TOFU"
140 INPUT T(1)
150 PRINT "TOFU ORDER IS", T(1)
160 PRINT "HOW MANY DOZEN SUSHI AGE"
170 INPUT A(1)
180 PRINT "SUSHI AGE ORDER IS", A(1)
290 PRINT "CONTINUE? HIT 0 TO EXIT OR 1 TO CONTINUE"
200 INPUT ANSWER
210 IF (ANSWER = 0) GOTO 9999
220 PRINT "IT'S DAY 2"
230 PRINT "HOW MANY TOFU"
240 INPUT T(2)
250 PRINT "TOFU ORDER IS", T(2)
260 PRINT "HOW MANY DOZEN SUSHI AGE"
270 INPUT A(2)
280 PRINT "SUSHI AGE ORDER IS", A(2)
290 PRINT "CONTINUE? HIT 0 TO EXIT OR 1 TO CONTINUE"
300 INPUT ANSWER
310 IF (ANSWER = 0) GOTO 9999
320 PRINT "IT'S DAY 3"
330 PRINT "HOW MANY TOFU"
340 INPUT T(3)
350 PRINT "TOFU ORDER IS", T(3)
360 PRINT "HOW MANY DOZEN SUSHI AGE"
370 INPUT A(3)
380 PRINT "SUSHI AGE ORDER IS", A(3)
390 PRINT "CONTINUE? HIT 0 TO EXIT OR 1 TO CONTINUE"
400 INPUT ANSWER
```

```
410 IF (ANSWER = 0) GOTO 9999
420 PRINT "IT'S DAY 4"
430 PRINT "HOW MANY TOFU"
440 INPUT T(4)
450 PRINT "TOFU ORDER IS", T(4)
460 PRINT "HOW MANY DOZEN SUSHI AGE"
470 INPUT A(4)
480 PRINT "SUSHI AGE ORDER IS", A(4)
490 PRINT "CONTINUE? HIT 0 TO EXIT OR 1 TO CONTINUE"
500 INPUT ANSWER
510 IF (ANSWER = 0) GOTO 9999
520 PRINT "IT'S DAY 5"
530 PRINT "HOW MANY TOFU"
540 INPUT T(5)
550 PRINT "TOFU ORDER IS", T(5)
560 PRINT "HOW MANY DOZEN SUSHI AGE"
570 INPUT A(5)
580 PRINT "SUSHI AGE ORDER IS", A(5)
590 PRINT "CONTINUE? HIT 0 TO EXIT OR 1 TO CONTINUE"
600 INPUT ANSWER
610 IF (ANSWER = 0) GOTO 9999
620 REM CONTINUE SIMILARLY FOR 360 MORE TIMES BY TYPING
    AS FAST AS YOU CAN
9999 PRINT "ALL DATA ENTERED"
```

and so on and so on for 360 more times . . .

I was referring to days of the year as numbers from 1 to 365 and had strung them together as a long program. We sold five or six products in addition to tofu and sushi age (*abura-age*), and I

included a lot more dialogue text. So it was plenty to type for each day of the year. And there were many GOTO statements to route the logic in ways that could give greater clarity to my mother as she input the data that I'd left out in the above depiction.

I then rewrote the program with my newly learned technique from Mr. Moyer:

```
10 DIM T(365), A(365) : HOME
100 REM THANK YOU, MISTER MOYER
110 FOR I = 1 TO 365
120 PRINT "IT'S DAY", I
130 PRINT "HOW MANY TOFU"
140 INPUT T(I)
150 PRINT "TOFU ORDER IS", T(I)
160 PRINT "HOW MANY DOZEN SUSHI AGE"
170 INPUT A(I)
180 PRINT "SUSHI AGE ORDER IS", A(I)
190 PRINT "CONTINUE? HIT 0 TO EXIT OR 1 TO CONTINUE"
200 INPUT ANSWER
210 IF (ANSWER = 0) GOTO 9999
220 NEXT
230 REM NO NEED TO TYPE ANY MORE ENTRIES. RELAX!
9999 PRINT "ALL DATA ENTERED"
```

Done!

I stared incredulously at the 365 separate programming chunks, which comprised 40 or so lines of programming per entry point—totaling roughly 14,600 lines of code. In under half an hour I was

Index