

Write- protected.

Save anyway?

Essays on Programming, Mathematics,
and Attempting to Understand the World

Mitchell Vitez

*With thanks to E. A. Chavis, a true friend and
shining wit. Your feedback made this better.*

This is (largely) a work of non-fiction. Despite that, some names, characters, places, and incidents are either the product of the author's imagination, or are used ficticiously. Any resemblance to actual persons—living, dead, or in superposition—is entirely coincidental.

The number of mistakes contained here is not zero, for which I take full responsibility. The Haskell bits are probably closest to correct, because at least they were typechecked. Read the rest at your own risk.

Copyright ©2020 Mitchell Vitez. All rights reserved. All rights outgoing. Ups and downs still undecided.

Cover design by Mitchell Vitez. Photoshop on canvas, 2020.

Compliments and cheery witticisms may be directed to mitchell@vitez.me. Likewise for well-reasoned disagreement. Scathing and unreasonable critiques may be directed to nobody@example.com

Published via leanpub.com, whose services I heartily recommend

The author may be most easily located via vitez.me

Write-protected. Save anyway?

Essays on Programming, Mathematics,
and Attempting to Understand the World

Mitchell Vitez

Preface

I have long been the kind of person who figured I'd probably write a book...someday. I sure read enough of them. How hard can it be?

Well, it turns out to be quite the endeavor just to write a bunch of words.

I began writing in earnest on January 1, 2019. For the first fifty days of that year, I wrote an essay every day, with many of them breaking the 1000 word mark. Back then, I felt that I was a little tired of being read-only, and thought it was time to write. I'm not quite so tired anymore.

This book is structured as a collection of essays, with fairly wide-ranging topics. Many of the essays are about programmingish or mathy subjects, especially functional programming and Haskell. This is because Haskell's maybe the only topic I'm anywhere close to having even a smidge of expertise about. If you're not into that kind of thing, maybe it'll spark some intrigue? But of course, feel free to breeze through. I won't be able to stop you.

The essays with topics straying farther from that core tend to be weaker, admittedly, but I had tons of fun thinking about the ideas involved. There's even a chapter dedicated to "Weird Thoughts", which probably contains the most bad arguments per capita, but also some of the most intriguing ideas. I have not gone through and updated the arguments since the time I first made them, so a few may be outdated.

I tend to write about technical (or at least vaguely technical-ish) topics in a very informal, hand-wavy style. I use the word "things" a lot, when I just need some handy

abstraction for an arbitrary referent. The writing is not anywhere close to a textbook's veracity, but instead is much more conversational. I of course encourage the reader to do their own research, especially about anything unfamiliar or that I seem especially hand-wavy about. Forming the correct conclusion is, in all cases, left as an exercise for the reader.

People sometimes say a great book exists to fill a niche. This book doesn't really fill a strong niche, so maybe it will never be a great book. I'm okay with that. As long as I give you something to think about, and you enjoy yourself, I think we'll get along just fine.

Contents

Preface	5
Contents	7
1 Computer Science	8
Infinite Recursive Permutation Generators	9
Deconstructing Hadamard Gates	12
The Map Puzzle	22
The Erlang C Formula	24
Dead Simple Password Safety	26
The CAP Theorem Misconception	29
Malevolent Ackermann	31
2 Software Craftsmanship	33
Nullity, Exceptionality, and Optionality	34
A Mental Model of the Python REPL, for Beginners . . .	40
A Few Simple Python Reductions	50
Null IN PostgreSQL	58
How Python Dictionaries Work	61
3 Programming Projects	68
LSTM XOR	69
A Curation Algorithm	74
Slacktivity	79
Concatenative Programming	90
Magic Square Generation	95
Real World Character Recognition	102
Building an AST for Lua in Haskell	109

Elm Everywhere	115
4 Functional Programming in Haskell	122
A Beginner's Guide to the Ways Haskell Helps Us	
Avoid Errors	123
Type-Enforced Exponential Trees	133
Basics of Probability Monads	140
Sum Types Are Better Than Others	144
Encoding Free Groups	146
A Few ghci Tips	149
Parity Clarity	151
Roman Numerals in Haskell	157
Building Lenses	159
Composing Coercions	179
5 More on Functional Programming	184
Currying and Macros	185
The Only Runtime Error I've Ever Seen In Elm	187
Idris Auto-Implementations	189
OCaml's Categorical Origins	191
Maybe vs. Dec	196
6 Weird Thoughts	200
Sentience Decidability Programs	201
Lerping and Slerping	203
The Badness of Death	205
Alien-Worthy Human Humor	210
My Superintelligence Can Beat Up Your Superintelligence	212
The Supreme Cleverness of Mario Kart's Mirror Mode .	215
Why Volleying Is Allowed In Tennis But Not In Ping-Pong	218
7 Music	221
Giant Steps, Tiny Steps	222
The Rise of Dark Downtempo	225
The Incredible Joys of Amateur Music-Making	227
MuseNet	229

8 Pure Math	231
Braid Isotopy	232
Invertible Bitmatrices	235
Manifesting Manifolds	239
Morphisms	242
The Chebyshev's Inequality Proof of the Weak Law of Large Numbers	245
Hidden Recursion	248
9 Applied Math, Physics, and Modeling	253
Delta Functions and Mixture Models	254
Guessing a Function	256
A Polarizing Intuition	263
Topological Data Analysis	265
A Sense of Relativity	271
Black-Scholes Under Duress	273
Dice Roll Likelihoods	277
10 Achieving Goals	279
Don't Forget Training	280
Shirking Imaginary Duties	283
Achieving Anything Easy	285
The Inner Game	287
The Ingeniousness of Genuineness	290
Fear of Repackaging	292
You Have to be Willing to Lose	294
Understanding Understanding	297
Start Predicting Everything	300
Freezing, Flattening, Rasterization, and Rendering	302
Novelty Minus Novelty	305
11 Incentivization	307
Incentives are a Hell of a Drug	308
Incentives at Scale	313
Existing Effortlessly	315
The Ins and Ins of Economics	317

12 Culture	321
Hype and Hatred	322
In Praise of Intricate Writing	325
Agape	328
Human Heroes	330
Video Game Setpieces	332
The Siren Song of Meta-Altruism	336
Cultural Affordances	339
13 Values and Meaning	341
Interpreting the Value of Money	342
Wows Per Dollar	345
Time Value of Everything	347
Forgetful Functors and Lossy Language	349
Things We Criticize	353
14 Language and Logic	356
Flip It Around	357
Hyperintrospection	372
Most People	375
Life and Other Fuzzies	377
Thought Molding	379
Semi-Conserved Quantities	381
15 Wrap-up	384
Epilogue	385

Computer Science

Infinite Recursive Permutation Generators

Consider the following ten lines of code:

```
def gen_perms(s, n=1):
    if n < 1:
        yield ''
    for prefix in gen_perms(s, n-1):
        for c in s:
            yield prefix + c
    gen_perms(s, n+1)

for perm in gen_perms('abcdefghijklmnopqrstuvwxyz'):
    print(perm)
```

In a nutshell, this prints out every possible permutation of letters of the alphabet, in “spreadsheet column name” order (a, b, c..., aa, ab, ac..., aaa, aab, aac...). Every dictionary word. Every great speech. Entire (unpunctuated) works of Shakespeare.

For now, we’ll ignore a few basic improvements. This could easily be generalized to work with more than just strings. The runtime isn’t too great either since we’re doing a bunch of recalculation. Memoization and choosing better data types would fix these, but let’s focus on the aesthetic qualities of this code, and on how it actually works.

Starting with the easy bit. Assume we have an infinite permutation generator called `gen_perms` that takes a string and gives back all the possible permutations of that string based on the order the characters were passed in. In that case, this loop will run through those infinite permutations and print out each one:

```
for perm in gen_perms('abcdefghijklmnopqrstuvwxyz'):
    print(perm)
```

Let's get into the real meat of it though. `gen_perms` takes a string `s`, and a parameter `n`. You can think of `n` as the length of the permutations currently being generated.

```
def gen_perms(s, n=1):
```

The body of the function has three parts. First of all, if `n` is less than one, we want to give back an empty string. This is our recursive base case, and should be pretty straightforward.

```
    if n < 1:
        yield ''
```

Glossing over the tricky middle bit for a second, here's the third part of the function body. Just like we need a base case for termination, if we want infinite generation we need an ever-growing case. Here, we just call `gen_perms` with the next largest permutation length so it can compute all of those for us. This will keep growing as `n` reaches ever higher towards infinity.

```
    gen_perms(s, n+1)
```

Here's the fun stuff. First, we inductively assume that `gen_perms` works for $n-1$. That should generate all the permutations of length $n-1$ for us, and we'll iterate over each of them as `prefix`. Because a prefix is of length $n-1$, and we want to generate a permutation of length n , all we need to do is add a single character to `prefix`. These single characters have an obvious source: each character in `s`, in order.

Eventually, with all the possible prefixes having all the possible characters in *s* appended to them, we manage to come up with all the possible permutations of length *n*.

```
for prefix in gen_perms(s, n-1):  
    for c in s:  
        yield prefix + c
```

This code manages to combine several fairly tricky topics—recursion, infinity, permutations, generators—into one short sample, which I think makes it a super fun example to work with. For bonus points, try writing the memoized version, or a version that works with *sets* instead of *strs*!

This is a preview. The full book is available at
leanpub.com/writeprotected