KEVIN FOCKE

THE CORE monster MANUAL

Vol 6

"I want to order exactly 6 breads," said the Exactor.

"Quack, did you say 9 breads? How generous of you to support a hard-working breadwinner..."

THE CODE MONSTER MANUAL VOLUME 0 (Sample)

Escape From Number Hell

Computer fundamentals, the book I wish I had.
The creative glue for intermediate-level programming concepts.

Improve your programming versatility by learning the nuances of numbers & their representations.
Not a math wizard? No problem!

Surrealistic drawings and cover background by Darío Mekler.

Monster drawings and cover cartoon by Thais Freire Sanchez.

TABLE OF CONTENTS

CHAPTER 0: Intuition

NO NEED TO BE A MATH WIZARD

How can you prevent the 125 million dollar mistake that caused the NASA Climate Orbitor to crash?

You don’t need to be a rocket scientist or mathematician to understand it. In fact, the mistake was very simple: They misunderstood what a number was.

Anyone can make this mistake and we will learn how to prevent it.

Now you might think: “I still have a math trauma from high school. Do we *really* need to learn about numbers?”

Fret not, dear astronaut. You don’t need to be a math wizard or multi-dimensional alien to be a great programmer. This book does not hide insights under cryptic mathematical equations. Nor is there even a single line of code in the whole book. Instead, we start from first principles as we build up our knowledge of timeless computing concepts that are relevant in whichever programming language(s) you end up using.

“Why do I need to learn about numbers, anyway? I just want to make programs. Can’t we just use functions for that?”

Biologists learn about the core principles of atoms, even though they are more interested in the structures emerging from atoms: Proteins, cells, etc.

Similary, a function, and more generally software, emerges from numbers. Yet, unlike the inflexible rules of chemistry, the rules of programming are not limited by functions; we gain the flexibility to change how the functions themselves work by deeply understanding numbers!

“But coding bootcamps promise me I can learn how to code in 30 days or less. And they don’t bother me with numbers, either.”

The complexity of computers is hidden away by programming languages and operating systems. This gives a false sense of confidence until, one day, you encounter problems that can only be solved by deeply understanding how the computer itself works—including numbers.

Sure, you can make programs until you encounter those problems, but that’s like paddling your surfboard into a whirpool… Fun, if you’re a fan of suddenly sinking to the bottom of the terrible abyss. Invitations to “The

Sinking Crew” are available in an “easy” programming book or bootcamp near you.

Furthermore, the book is spiced up with eclectic insights because concepts are often relevant beyond the boundaries of an industry or field.

For example:

`Here Be Monsters`, the old naval maps used to say about uncharted territories. It’s helpful to know where the dangers are, but software is not a stormy sea nor a dark alley in a city. Software has no location; it’s not a physical building you can walk in. Software is more like air; it’s everywhere.

Computers are monsters because they think different than us. The Code Monster Manual helps you understand where the monsters lurk, and gives practical guidance on preventing monster mayhem. The book provides a helicopter-view of the most bang-for-your-buck coding concepts.

While I try to keep the content approachable for ambitious beginners, there is quite a lot of ground to cover and the pace is fast. It’s natural to have a lot of questions in the beginning and it takes time to fully answer them.

Look, you may even feel discouraged when learning about computers because the subject is so broad. But remember then: Nothing worthwhile is ever easy. A mountain is climbed step by step, and so it is with computers.

Along the way, you’ll learn about bright ideas and bits of trickery.

Bright Idea

Repeat what you've learned in your own words to better retain information.

Bit Of Trickery

Incremental change is hard to notice, and yet it's the basis of most progress. Perfection is for the gods and the delusional. Us mere mortals, we iterate!

As software developers do, we start counting the volumes from 0. Why do developers count from 0? It's related to a positional starting point called an `index`. The position you are currently at is 0. If you move forward one step from the starting point, you are at position 1. If you move backwards one step from the starting point, you are at position -1. A teaser to mull on, we will learn how to represent negative numbers with positive numbers which sounds paradoxical—and yet that *is* how we do it.

In the 0'th chapter, we will first build our intuition with some dogs, before graduating to misunderstood monsters.

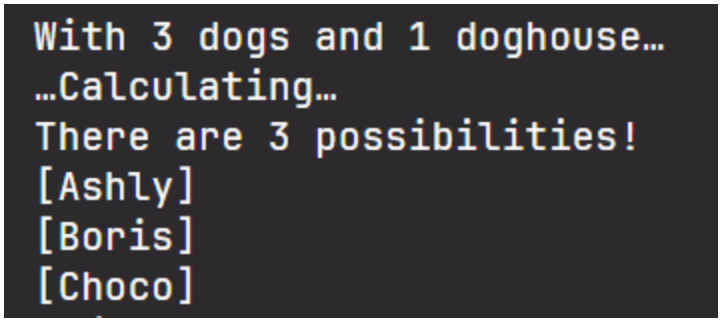
LET SLEEPING DOGS LIE

Somewhere in the Savanna, there are 3 dogs; Ashly, Boris, and a puppy called Choco. The caretaker Johhny starts an animal shelter for the dogs. He builds a lovely little doghouse and paints it with blue clouds. He then builds another doghouse, painted in a gnarly green. Boris grumps a little.

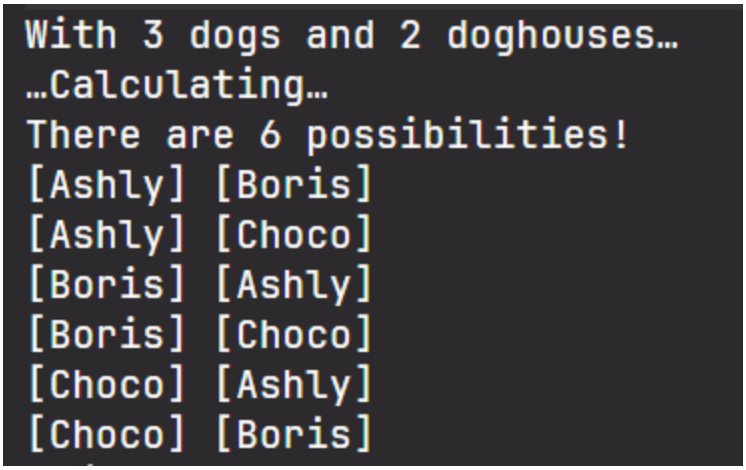
Johhny starts to wonder, as programmers do: How could I assign these dogs to their doghouses?

“So… let’s think about it… A dog can either be in a doghouse, or roam in the Savanna. If a doghouse is available, it *must be assigned to a dog*. Unfortunately, I can’t put multiple dogs in the same doghouse because they’re very territorial animals.”

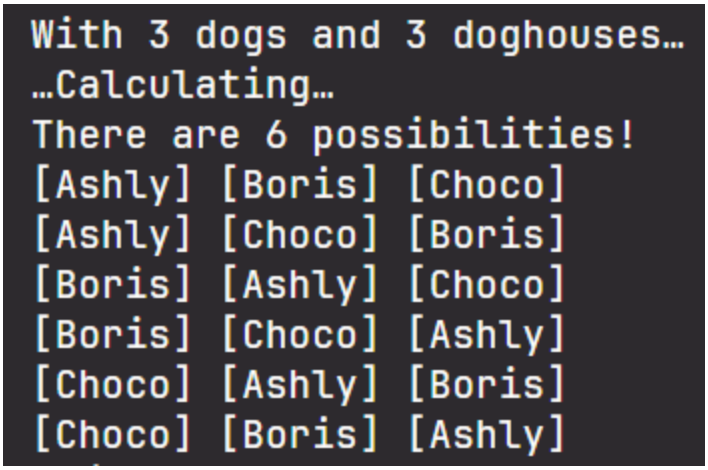
Johhny writes up a program to calculate the possibilities and it spits out the answer:



Well that was obvious. But what happens if we add more doghouses?



Johhny builds a third doghouse, painted with red roses. Boris *woofs*.



You may notice there are still 6 possibilities—how can that be? What is missing?

Well… we have pretended dogs don't exist unless they're inside a doghouse. Later on, we will see that the concept of `absence` is so tricky for programmers that it cost "a billion dollars of pain and damage…"

Johhny builds another doghouse and paints it purple. Boris starts barking loudly. He is mad and trying to communicate, although he finds it a tad difficult because he's a dog. If dogs could talk, Boris would say:

"Before I was happy with any doghouse; it gets cold here at night. However, now there are more than enough doghouses and I want the one painted with roses! And if I can't have that one, I'd rather have the one with the blue

clouds than the gnarly green."

Johhny runs the program.

```
With 3 dogs and 4 doghouses…
…Calculating…
There are 24 possibilities!
[Ashly] [Boris] [Choco] [Empty]
[Ashly] [Boris] [Empty] [Choco]
[Ashly] [Choco] [Boris] [Empty]
[Ashly] [Choco] [Empty] [Boris]
[Ashly] [Empty] [Boris] [Choco]
[Ashly] [Empty] [Choco] [Boris]
[Boris] [Ashly] [Choco] [Empty]
[Boris] [Ashly] [Empty] [Choco]
[Boris] [Choco] [Ashly] [Empty]
[Boris] [Choco] [Empty] [Ashly]
[Boris] [Empty] [Ashly] [Choco]
[Boris] [Empty] [Choco] [Ashly]
[Choco] [Ashly] [Boris] [Empty]
[Choco] [Ashly] [Empty] [Boris]
[Choco] [Boris] [Ashly] [Empty]
[Choco] [Boris] [Empty] [Ashly]
[Choco] [Empty] [Ashly] [Boris]
[Choco] [Empty] [Boris] [Ashly]
[Empty] [Ashly] [Boris] [Choco]
[Empty] [Ashly] [Choco] [Boris]
[Empty] [Boris] [Ashly] [Choco]
[Empty] [Boris] [Choco] [Ashly]
[Empty] [Choco] [Ashly] [Boris]
[Empty] [Choco] [Boris] [Ashly]
```

Johhny assigns the dogs to their house, gives them food adapted to their diets, and goes to sleep.

In the middle of the night, all hell breaks

loose! Boris scares away Ashly and takes over her house. Boris celebrates his victory by eating Ashly’s meal and promptly has an allergic reaction.

Bright Idea

Differentiate between a mild allergic reaction `a recoverable error` and a horribly allergic reaction `an unrecoverable error`. The unrecoverable error is also called a `panic`. This distinction matters a great deal to programmers and the dogs they love.

Ashly complains to her friend Choco, interrupting Choco's beauty sleep—and now Choco is mad too! Ashley completely panics and runs away into the Savanna.

Johhny wakes up—such chaos! He falsely believed the situation was stable, but the dogs moved around. So much trouble with just 3 dogs and 4 doghouses. Now imagine adding more dogs and doghouses. Yeah, that'd be a mess…

In a nutshell: Everything in a computer is a bunch of numbers interpreted to be something.

It gets complicated because the same number can represent different things, different

numbers can represent the same things, and the numbers can move around causing all sorts of chaos.

If this makes no sense yet, don’t worry; we have a whole book left to explain!

A bonus round, in case you’re curious…

Question: how many possibilities are there with 3 dogs and 10 doghouses?

Answer: 720 possibilities!

Q: 3 dogs and 20 doghouses?

A: 6 840 possibilities!

Q: 3 dogs and 30 doghouses?

A: 24 360 possibilities!

Q: 4 dogs and 30 doghouses?

A: 657 720 possibilities!

Q: 5 dogs and 30 doghouses?

A: 17 100 720 possibilities!

Q: 6 dogs and 30 doghouses?

A: Way too many possibilities, it crashes the

computer!

There might be a way to improve Johhny's program so it doesn't crash, but that's not the point. The point to remember is that the amount of possibilities, or `state space`, keeps increasing at an ever-faster pace. This will become relevant in a later volume. For now, let's focus on understanding numbers.

HIGH-LEVEL PROGRAMMING VS. LOW-LEVEL PROGRAMMING

“Developers, Developers, Developers,” Steve Ballmer, then-CEO of Microsoft, famously chanted at the turn of the millenium. Yet all software developers must ultimately bow down to: Numbers.

A number can be a count of things (5 rabbits), it can represent an ordering (the 3rd duck), it can represent a location (lives in rabbithole number 42), or a time (Bunion, The Rabbit, woke up at 3 AM in the morning). Even a function itself is composed of a bunch of numbers that carry out actions by activating hardware circuits with instruction numbers.

Numbers, numbers, numbers.

To avoid thinking about numbers, programmers abstract away the underlying complexity of numbers by creating simpler interfaces. Similar

to how a subway map shows the routes while hiding the jaggedness of the real routes. To keep an overview & navigate better, we hide information that is not relevant.

There is a distinction between `high-level programming` and `low-level programming`.

High-level programming uses more abstractions whereas low-level programming uses fewer abstractions. In other words, low-level programming works closer to the foundation. Both kinds of programming are useful in different scenarios.

High-level programming is useful when the pre-defined abstractions align with the functionality you want; why reinvent the wheel? In contrast, low-level programming is the most difficult and useful when you want maximum flexibility.

Low-level programming requires you to understand how the system *actually* works rather than understanding simplified abstractions built on top of it. High-level programming is like learning the buttons of an all-in-one cooking robot—handy if that's the functionality you need.

To maximize your flexibility, the book series is focused on low-level programming.

SUMMONING THE NUMBEROMICON

In volume 0, *The Numberomicon*, we are laying lifelong software foundations by thoroughly understanding what a number is. Similar to constructing a building, laying the foundation of drab gray concrete is not the most interesting part. And yet, it is essential if you want to build something impressive.

You have summoned the Numberomicon.

Chapter 1, The Exactor: Why do computer need exactness and why do they use binary numbers?

Chapter 2, The Contraducktion: The versatility of numbers is confusing; the same number can represent different things, and a different number can represent the same thing. How can we avoid this confusion and stop thinking about numbers?

Chapter 3: Where and how are numbers stored?

Who can access them?

Chapter 4: Computers are ultimately a tool. The underlying complexity is abstracted away to improve usability; the messiness is hidden and you can always peel back another layer. What are the broad considerations of using abstractions at different scales?

CHAPTER 1: THE EXACTOR

Spirit Of The Monster: Computers do exactly what you tell them to. If you expect the computer to just act reasonable, you will be surprised by the results…

FABLE: AN AMBIGUOUS ENCOUNTER

“Okay. What *exactly* do you mean with that?” the Exactor asked for the umpteenth time.

“Well look, I just want you to add these numbers,” said the Programmer.

“Okay. What do you mean with `just`?” the Exactor pondered.

“Ignore that word,” said the programmer, curtly.

“Okay. What do you mean with `add`?”

“Add means you combine one or more numbers.”

“Okay. What do you mean with `numbers`?”

“A number is… well it’s obvious isn’t it?” Responded the Programmer, nervously tapping his foot as if he’s a drummer in a long-haired metal band.

“Unfortunately, it’s not obvious at all, sir. Could you explain what a `numbers` is?”

“A number is how we count an amount of something,” said the Programmer.

“Okay. And what do you mean with `numbers`?”

Asked the Exactor.

“I just explained…”

“With all due respect, I don’t think that you did sir nor did you answer my question.” The Exactor said with a slight air of vicarious embarrassment.

“What?!” The Programmer shouted in disbelief.

“Please keep your tone down,” said the Exactor, calmly. “You explained what a `number` is and I am very appreciative that you did. But it still remains unclear what a `numbers` is. A `numbers` is not a `number`, is it?”

“You’re making a joke aren’t you?”

“No, sir. You see, I want to avoid *any* possibility of ambiguity. Please explain: What is a `numbers` *exactly*?”

Enjoyed this sample?

Get the full book at:

kevinfocke.com