VISIONEN

www.visionen.ethz.ch Juni 2017

Purified

Magazin des Vereins der Informatik Studierenden an der ETH Zürich (VIS)

open systems

Open Systems gehört mit seinen Mission Control Security Services im Bereich IT-Sicherheit zu den europaweit anerkannten Anbietern. Wir arbeiten von Zürich und Sydney aus in einem dynamischen Umfeld in über 180 Ländern. Bei uns kannst Du Dein Wissen in einem jungen Team in die Praxis umsetzen und rasch Verantwortung übernehmen. Infos über Einstiegs- und Karrieremöglichkeiten sowie Videos findest Du auf unserer Website. **www.open.ch**

MOST ATTRACTIVE EMPLOYERS
SWITZERLAND
universum
2017

Editorial

«Purified» ist eines der Wörter, die auf Englisch wesentlich mehr Sexappeal haben als auf Deutsch. Als Übersetzung liefert «linguee.de» als Erstes das Wort «gereinigt». «Gereinigt» weckt bei mir folgende Assoziationen: Zitrusreiniger, WC, Lappen, Mop und Ähnliches. Es dürfte klar sein, niemand, der sich ein Glas Wasser aus dem Britafilter-Dingens genehmigt oder seinen Tee aufgiesst, möchte dabei an Zitrusreiniger denken.

«Purifien» kann man nicht nur Wasser. Das Wort lässt sich auch auf Code anwenden. Sollte der Code nicht rein sein, hat er unnötige Berechnungen drin, welche die Laufzeit beeinträchtigen, oder besitzt er unsaubere Hacks, die nicht den Paradigmen der gewählten oder aufgezwungenen Programmiersprache entsprechen? In diesem Fall sollte man sich als Informatikstudent der ETH nochmals hinsetzen und diese Unreinheiten ausmerzen. Am Schluss hat man wunderbaren «purified» Code. In dieser Ausgabe wird Linus Torvalds' «purified» Linked List vorgestellt, ein schönes Beispiel für wirklich guten Code.

Nicht nur Code und Wasser kann man reinigen, auch kann es helfen, den Kopf hin und wieder durchzulüften, insbesondere während der Lernphase. A propos Lernphase, in dieser Ausgabe geben wir auch Tipps, wie man Karteikarten effizient einsetzt und welche Möglichkeiten sich zu digitalen Karteikarten bieten.

Dies ist meine erste Ausgabe als Chefredaktor und ich hoffe, ihr habt viel Spass beim Lesen und wünsche viel Erfolg für die Prüfungen und Durchhaltevermögen in der Lernphase!

Euer Chefredaktor

Julian Croci

Inhalt

Purified

Offizielles

Berichte

Rätsel

Serien

Junior Software Engineer

Haben Sie einen Hochschulabschluss in Informatik auf Bachelor- oder Master-Level und Java-/Java-EE-Kenntnisse? Interessieren Sie sich für Applikationsentwicklung und Integration? Steigen Sie bei uns als Integrator oder Software Engineer ein!

Bewerben Sie sich online!

HR-Team Schweiz
+41 44 272 61 11
jobs@adnovum.ch
adnovum.ch/jobs

kununu TOP COMPANY

ürich · Bern · Lausanne · Budapest · Ho-Chi-Minh-Stadt · Singapur

Linus Torvalds' Linked List

JONATHAN UNGER - UM SCHÖNEN CODE BEMÜHT

«If I had more time, I would have written you a shorter letter.» – Jonathan Unger

Was ist guter Code? Diese Frage kennen wir mindestens schon seit dem ersten Semester und wird auf sämtlichen Informatikblogs wiederholt. Sogar Eiffel wurde mit der Motivation erschaffen, Entwicklern zu helfen, guten und korrekten Code zu schreiben. Eiffel ist dabei kein Einzelfall. Sprachen wie Rust und Haskell beantworten diese Frage auf ihre eigene Art. Vor einiger Zeit hatte sich sogar Linus Torvalds zu dieser allgegenwärtigen Frage geäussert. Er antwortete mit seinem Konzept von «Coding with Good Taste». Die Frage mag zwar schwer, vielleicht auch unmöglich zu beantworten sein, aber schauen wir uns trotzdem einmal an, was Linus dazu zu sagen hat.

Im Laufe des Studiums jedes Informatikstudenten kommt der Tag, an dem man eine Singly-Linked List implementieren soll. Studenten, die die Aufgaben sogar machen, sind das Problem wahrscheinlich so angegangen, wie jeder andere schon immer. Zuerst definiert man einen Head, der auf das erste Element zeigt, und dazu noch das Element selber, bestehend aus einem Wert und einem Pointer auf das nächste Element.

So weit, so gut.

Danach muss man Funktionalität für diese Datenstruktur implementieren, wie z. B. das Entfernen eines Elementes. Nach Torvalds lösen die meisten das Problem so, wie er es in seinem Pseudo-C-Code beschreibt:

```
remove_list_entry(entry)
{
    prev = NULL;
    walk = head;

    // Walk the list

    while (walk != entry) {
        prev = walk;
        walk = walk->next;
    }

    // Remove the entry by updating the
    // head or the previous entry

    if (!prev)
        head = entry->next;
    else
        prev->next = entry->next;
}
```

Der geübte Leser erkennt, dass hier das Element nicht wirklich gelöscht, sondern nur dereferenziert wird. Ausserdem werden Randfälle wie Nullpointer nicht mit einbezogen. Doch vom Prinzip her funktioniert dieser Pseudo-C-Code. Jeder hätte diesen Code ungefähr so geschrieben. Auch ich hätte diesen Code so geschrieben.

Linus erklärt, dass dieser Code kein guter Stil sei. Obwohl es so in «Computer Science 101» gelehrt wird und praktisch jeder den Code so schreiben würde, ist es nach seinem Geschmack schlicht nicht guter Code. Linus kritisiert somit fast alle Entwickler und wahrscheinlich den Leser inklusive. Aber wie verteidigt Torvalds seine Ansicht? Er zeigt eine zweite Art, das Problem zu lösen:

```
remove_list_entry(entry)
{
    // The "indirect" pointer points to the
    // *address* of the thing we'll update

    indirect = &head;

    // Walk the list, looking for the thing that
    // points to the entry we want to remove

    while ((*indirect) != entry)
        indirect = &(*indirect)->next;

    // .. and just remove it
    *indirect = entry->next;
}
```

Dieses Beispiel muss man länger auf sich wirken lassen. Trotzdem erkennt man nach und nach, dass dieses Beispiel objektiv besser ist. Aber was ist es, das diesen Code besser macht?

In diesem Beispiel will Linus, dass sich der Leser vor allem auf das If-Statement von vorher fokussiert. Es ist verschwunden! Die Ausnahme ist zum Normalfall geworden.

Am wichtigsten ist nach Torvalds, dass der zweite Code zeigt, dass der Entwickler ein besseres Verständnis für das grosse Bild hat. Er versteht, wie Pointer funktionieren und wie man sie korrekt anwendet. Obwohl der Entwickler des zweiten Codes seinen Horizont nur von einfachen Pointern zu zweifachen Pointern erweitern musste, konnte er den Code massiv verbessern. Aber das Beispiel gilt nicht nur für Pointer. Generell kann ein Entwickler, der ein tiefgründiges Verständnis besitzt, viel einfacher Optimierungen und Veränderungen vornehmen, so Linus.

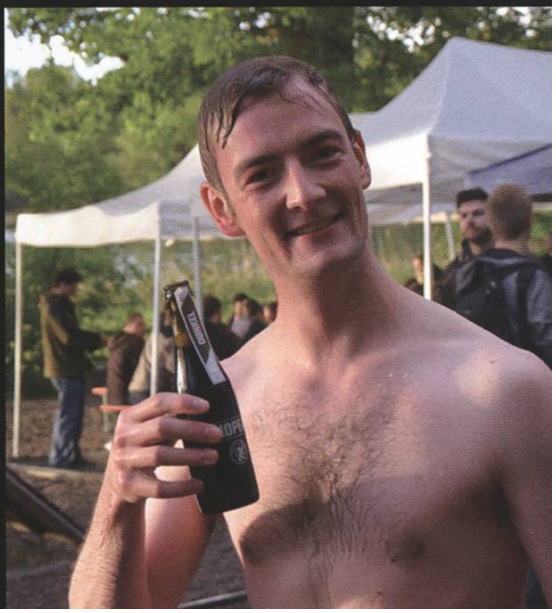
Das gute an diesem Beispiel ist, dass viele schon einmal eine Linked List implementiert haben und praktisch jeder eine gesehen oder benutzt hat. Die meisten können sich mit diesem Code identifizieren. Ausserdem wird jedem schnell klar, dass die zweite Version auf eine Art besser ist. Sei es die gewisse Cleverness des Codes, die Effizienz oder auch nur die halbierte Anzahl an Zeilen: Linus hat ein Beispiel gefunden, welches dem durchschnittlichen Coder im Nachhinein einleuchtet, aber dennoch professionell genug ist, dass es in einer Stunde in «Datenstrukturen und Algorithmen» durchgenommen werden könnte. Mir hat Linus jedenfalls gezeigt, dass mein Code wahrscheinlich ziemlich langweilig ist und wie viel Verbesserungspotential er doch hat.

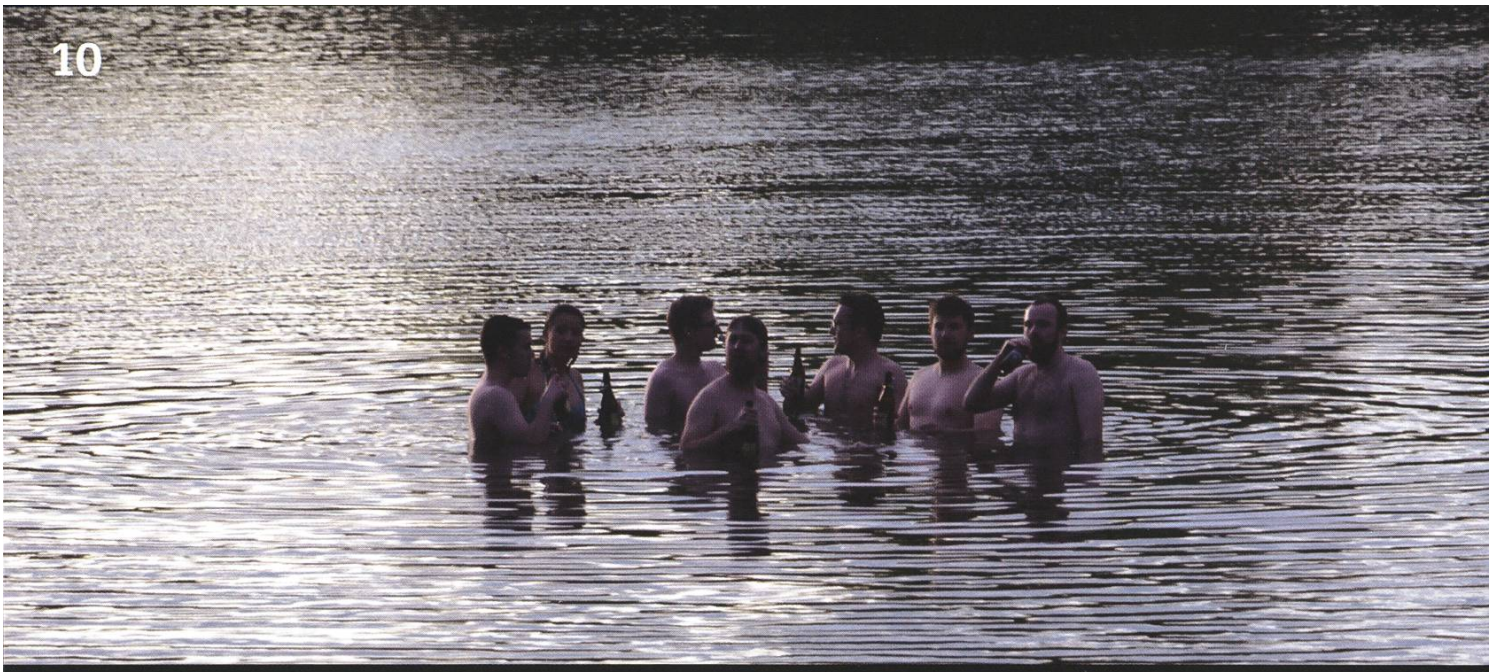
Die wichtigste Moral, die jeder Entwickler aus der ganzen Geschichte ziehen kann, ist, dass man nicht das nächste Quicksort in seinem Gebiet entwickeln muss. Es ist das Verständnis des Fachs, das einen guten von einem grossartigen Entwickler unterscheidet.

VISKAS


DIESEL
ONLY THE BRAVE
CODE


ZURICH





Bad Bash Scripts

NILS LEUZINGER - WALLOWS IN MEMORIES

Some people, when thinking of Bash scripts, see an unintelligible mess in front of their inner eye. They may go even further and claim that Bash is just a bad shell in general and that one should use other shells in its stead - PowerShell, for example. I think the latter statement is complete bogus and I would argue that Bash (or the shells that are derived from Bash) are probably some of the best shells there are. Though, I would totally agree with those people that Bash scripts *do* have a tendency to be a complete mess. So in this article, we're going to take a look at what makes certain Bash scripts unreadable, why Bash is still awesome and most importantly, how it should be used to make it awesome.

Now, I have a little confession to make: I don't actually use Bash. I use Zsh. Zsh is a shell that feels like Bash with more features, but it's actually not a superset of Bash. That is, when a Bash script is executed in Zsh, it might behave differently. I used Bash in the past, but I think Zsh is even more awesome than Bash, which is why I switched. Anyway, I feel competent enough to identify bad Bash scripts and propose improvements. I'm also of the opinion that having beautiful scripts is mostly useless, unless you plan to make them public, so let's all imagine that the scripts we're going to check out are all public and were made by some super famous Bash guy. Speaking of super famous Bash guys, I'd like to thank my friends who sent me some their scripts - more or less voluntarily.

Finally, before we start: To read this article, it's probably best if you're at least at beginner or intermediate level. I will explain some things, but this is not supposed to be an introduction to Bash. If anything is unknown to you, feel free to take a peek at the manual.

Enough said, let's go bashing!

To start off, we'll inspect a snippet of one of my earlier Bash scripts:

```
#acquire file lock
failed=1
while [[ $failed == "1" ]]; do
        #spin
        sleep 0.001
        [[ -n $(cat lock.1 2> /dev/
null) ]] && continue
        echo "$$" >> preLock.1
        echo "$$" > lock.1
        preLockContent=$(cat
preLock.1)
        if [[ $preLockContent == "$$"
]]; then
                #prelocking
successful
                failed=0
        else
                #prelocking
unsuccessful: try again
                rm preLock.1
                #accidentally locked.
should rm lock.
                [[ $(cat lock.1) ==
"$$" ]] && rm lock.1
        fi
done
```

If you take a closer look, you can already see a very grave mistake in the first row. I'm not talking about declaring `failed=1`, I'm talking about the very first row. "Acquire file lock". *What?* That's right, I'm trying to do concurrent programming in Bash. I set it up so this script is called every time I press the volume-up key, so there's bound to be multiple scripts running at the same time. Therefore, I needed to implement some locking mechanism somewhere. It's sort of readable because there are comments, but it is still way too long for what it does. C, Python or Ruby would have been better choices.

Another mistake I frequently see, is that people tend to use absolute paths too often.

```
cd /home/nils/scripts/downloadscript/
youtube-dl -x "$1"
```

Here, I'm trying to download a music video from youtube and I want to save it to the directory the script is placed in, so I use an absolute path for that. This is bad practice. If I ever wanted to move that script, I'd have to reprogram it. Much better is to use the following "trick":

```
cd "$(dirname "$0")"
```

This changes the directory to the one the script resides at. There's not even a need to use system-wide variables.

What if we wanted to save the music temporarily, play it and then delete it again, so it doesn't use up space?

```
cd "$(dirname "$0")"
youtube-dl -x "$1" -o music.opus
mpv music.opus
rm music.opus
```

If you think that this looks a bit clumsy, you're right. A cleaner solution is to use the `/tmp` directory for temporary storage:

```
youtube-dl -x "$1" -o /tmp/music.opus
mpv /tmp/music.opus
```

We don't even need to change our directory. Of course we could use `mpv` to play directly from YouTube, but maybe having the music file saved somewhere isn't bad if we decide that we want to keep it.

For the next example, I picked something that has various applications: Choosing a random file from a directory. This could be a cooking recipe, a wallpaper, a contact file of a lottery winner etc.. Let us find a rough outline for this script. Coming from an imperative standpoint (like me), you might be inclined to try counting the files in the directory, then choosing a random number N, and finally picking the Nth file. A Bash implementation could take the form of something like this:

```
rand=$(( RANDOM % $(ls -1 directory/
| wc -l) + 1))
file=$(ls -1 directory/ | head -$rand
| tail -1)
cat "directory/$file"
```

At this point I would already say that you should not be using Bash. As a matter of fact, anything random is a pain in Bash. To gain access to randomness, you can use the `RANDOM` "variable". Reading from this piece of magic returns a random value in the range of 0 to 32767. Why exactly 15 bits of randomness? I have no clue. On top of this, doing arithmetic isn't exactly a strong suit of Bash either - the syntax is simply not as clean as in other programming languages.

If we compared the previous script to an implementation done in Ruby, we would see a pretty clear difference in readability.

That said, it's not that Bash is bad at this type of stuff. We're using it wrong! Let's consider how ➔

we can improve the script. First off, if we want to pick a random wallpaper from a directory, we can swindle a bit:

```
feh --bg-center --randomize directory
```

Admittedly, there's not much Bash involved here. But this is exactly what we want! We're using what Bash excels at the most, and that is the ability to work as glue between other programs. It just so happens that we've got a program called `feh` that is readily available as a package on almost all big Linux distributions. Of course this is sort of cheating because we got lucky that there already existed a program to take care of randomizing wallpapers for us. What if we went back to our original problem of picking a random *file* from a directory?

There's still a decisively more elegant method than the `RANDOM` variable, and that is a command called `shuf`. `shuf` is incredibly cool, even though the first time I learned of this command, I seriously questioned why it was part of the GNU coreutils (the GNU coreutils make up the foundation of most Bash scripts and contain commands such as `cd`, `ls` and `wc`). Anyway, here's what the manual of `shuf` says: "Generate random permutations."

That's it. Sounds fairly useless in my opinion... Or so I used to think. So how do we use `shuf` to pick a random file?

```
file=$(ls -1 directory | shuf -n 1)
cat "directory/$file"
```

That is incredibly concise! No arithmetic, no random variables, just Bash being glue between some commands. Turns out Bash is pretty awesome after all.

Let's take a look at one final example. We'll try displaying a random headline from www.reddit.com/r/Showerthoughts.

```
showerthoughtsSite=$(wget -O
- "https://www.reddit.com/r/
Showerthoughts.json" 2> /dev/null)
echo $showerthoughtsSite | grep -oP
"\"title\": \".*?[^\\\]\"" | sed -e
's/"title": "//g' -e 's/"$//g' -e
's/\\"/"/g' > /tmp/redditheadlines
cat /tmp/redditheadlines | head -$((
RANDOM % $(cat /tmp/redditheadlines |
wc -l)) ) | tail -1
```

As you might notice, there's the same problem as before: I'm using `RANDOM`, even though `shuf` would've been a much better choice. This leads to a convoluted calculation that is barely even readable. Already, the script looks messy. At least I'm doing something right - the `/tmp` directory is happy about being put to good use. The worst part, by far, are the `grep` and `sed` commands. Nowadays, I would probably use Ruby or Python anyway, because those two languages are more readable for string processing, but since it's kind of a short script, I'll forgive myself. What I won't forgive myself for, is not knowing how escaping works. Why on earth did I think using doublequotes for my regexes was a good idea? I don't know. Here's the nicer version:

```
grep -oP '"title": ".*?[^\\]"'
```

Lastly, there's also a change that can be made on the `sed` command that has a huge impact:

```
sed -e 's_"title": "__g' -e 's_"$__g'
-e 's_\\"_"_g'
```

I would claim that this is substantially more readable. It's still not perfect - it's never going to be, regexes are ugly by design - but at least you can clearly see where the regexes start and stop. Always be on the lookout for those tiny tricks! They can make or break the readability of a script.

Unfortunately, I'm now out of examples (and time. And space.), so that's pretty much it! This has turned out to be more of a tips-and-tricks compilation, rather than a collection of good guidelines for Bash, like I initially intended to write about. I still hope you've learned a thing or two, though.

Happy bashing, everyone!

Bildnachweise

Sofern nicht anders vermerkt, wurden die Bilder und Grafiken dieser Ausgabe durch die jeweiligen Autoren oder den VIS zur Verfügung gestellt.

Seite 27: Wikipedia/Vagrant (CC BY-SA 3.0)

On the Smartness of Light Bulbs

ALINE ABLER – SHINES AND SPARKLES

I like fancy things. No point in denying it. I also have a tendency to make impulse purchases, which combines pretty badly with the first trait. But not always. Not this time.

This time, I bought a Philips Hue starter kit. That's right, smart lights. Internet-of-Things things. Fancy things.

Philips Hue is a line of remote-controlled light bulbs. They come in several variations, most notably color and ambient light bulbs, LED strips, spotlights and a few more. It's easily the most prestigious and expensive product on the market – there certainly are cheaper alternatives, but since this was an impulse purchase, I haven't spent much time comparing them.

Why Should You Care?

"Prestigious and expensive" is often synonymous to "proprietary and locked down", at least in the world of IT. The Philips Hue line is a much-appreciated exception to that rule. Although it's certainly not open source or anything, it still has an open API that is quite easy to use. But first, let me describe the Hue lighting system in more detail:

At the core of the system is the Hue bridge, which also acts as an interface between applications and the light bulbs. The bulbs themselves connect over the Zigbee Light Link protocol[1], which is an open standard for smart lighting systems. Yes, there's a standard for that.

It's actually a pretty fancy one – I downloaded the official standard, just for the heck of it. The bulbs build a multihop network, so when one bulb is too far away from the Hue bridge, you add another bulb halfway and it will connect.

Interestingly, the Zigbee Light Link (ZLL) protocol does not sit on top of Bluetooth or WiFi – it's based on the ZigBee specification, which is a wireless communication standard in itself.[2]

The Hue bridge, on the other hand, connects to your local network and orchestrates the lights via ZigBee. You can access it as a RESTful web service, which makes it extremely easy to create custom applications to control your lights.

Although it's certainly not open source or anything, it still has an open API that is quite easy to use.

In the API, lights are organized in groups, to which you can apply scenes – preset combinations of colors and brightnesses. You can also directly set each light, and turn them on or off individually or in groups. All in all, the API pretty much corresponds to what you see in the Hue app, in case you've seen that before.

On to the Tinkering

If you know me, you know that it didn't even take me a week until I found some functionality I wanted that the Hue app didn't provide. In this case, I wanted ambilight functionality for my TV. The lamps are supposed to take on the average color of my TV screen.

In fact, there already exist applications which do that in various settings. In Kodi, for example, you just need to install an addon. However, I like to tinker, which means my setup is definitely *not* standard or even common. My TV, for example, is connected to my PC via HDMI. I use mpv, a command line media player, to watch movies.

Plus, using one of the 30 already existing solutions would've been cheating.

So I started scripting. I'm actually pretty proud of the code I produced for my lights. It's neat and tidy. The first thing I wrote was a python module which controls my lights. It contains functions like `setColor(lightID, color)` or `setScene(sceneName, groupID)`. The implementation of these functions was as straightforward as can be – just send an HTTP request to the Hue bridge, and off you go. That was nice. I loved working with it.

In addition to the Python module, which I also reused for all my other Hue apps, I wrote a program that repeatedly takes a screenshot, samples the average color, and sets my lights to that color. That just sounded *way* easier than it actually was, so let me go into some more detail:

First off, repeatedly making a screenshot is not so easy. It should be fast and use little memory. I ended up creating a C library just for taking the screenshot, which was then used in the Python script. Python can directly reuse the memory the C program allocates for the screenshot – no pesky copying of buffers.

The next problem was: How do I sample the average color?

That doesn't sound hard either, but that's because "average color" is an entirely wrong term for what I wanted. In most movies, the average color is some washed-out grey or almost-black, throughout. What I wanted was the *perceived* average color – the color a human would think to be average.

That's harder than it sounds, because human perception is not exactly an exact science. I arbitrarily determined that black pixels should not

My Hue bridge, digilently doing its job.

count towards the average, which didn't help a lot. Weighing the pixel colors by their saturation gave me much better results, but depending on the kind of movie, the amount of which saturated colors had to be overweighted varied wildly. Fortunately, command line options are a thing. Coming from bash, I am amazed how easy it is to define command line options in Python.

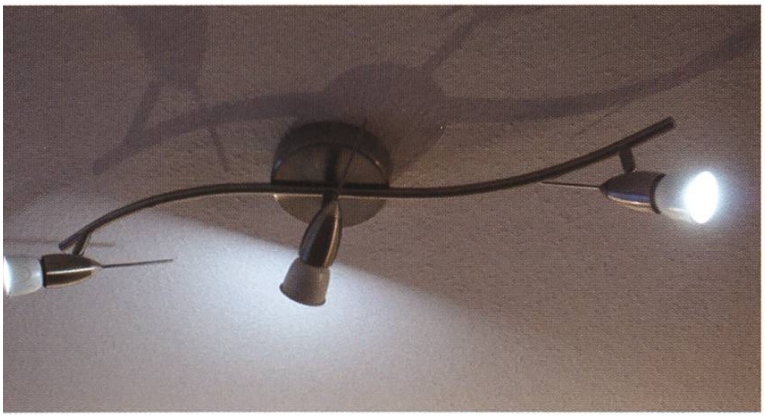
My script has some more options to play with. For example, you can set how much the brightness of the average color should be decreased. You don't want your lamps shining at full power while watching a movie.

The final result is pretty pleasing, meaning that you usually barely notice the color changes while watching. They do lag behind a little – the ➔

lamps need about half a second to switch to a new color, and my script samples the screen once per second, so the color change comes a tad late. You only see that during scene cuts, though.

Rise and Shine

But of course, I don't spend *all* of my free time watching movies, and the Hue lights have many more applications. Waking me up, for example. The official Hue app even includes a feature where you can schedule your lights to turn on or off at specific times of the day. I'm not a morning person (at all), so I figured turning on the lights before my alarm rings might help me get up.



Shining proudly. The cold white light is said to help focus, while warm white light is relaxing.

However, using the Hue app would've meant I needed to set three different alarms now instead of two, and again, using an existing solution would've been cheating.

So I started coding. The general idea was that my Android phone would synchronize the alarm times of the stock alarm app to my Raspberry Pi, which would then turn the lights on. That way, I don't need to maintain an additional alarm, since I'm using the phone alarm anyway.

Fortunately, I had a lot of experience in Android development. And luckily, the Android framework is totally easy to code in. So naturally, in order to create an app that sends a simple UDP packet over the network, I needed half a day and the entire night. The app is pretty lightweight, it only contains a SyncAdapter, a SyncService, an Authenticator, an AuthenticatorService, an AuthenticatorActivity and a Provider.

It's a fairly curious app. When you launch it, you get a blank screen that just says "Hello World". You have to go to your Android account settings, where you can add an account for my app, and specify an IP and port. The app then registers a custom sync adapter in the system, and Android takes care of running it regularly. As a result, my sync adapter regularly sends the timestamp of your next alarm to the specified IP and port as an UDP packet. Sounds simple, but it took a lot of boilerplate code to get this up.

"Syncing regularly" in this context means "syncing whenever Android feels like it", which in practice is about once every 30 minutes. This means my solution won't work for short-term alarms, i.e. when you set an alarm for in ten minutes, but it's pretty reliable for periodic once-a-day alarms, which corresponds to my use case.

Of course, when the app sends an UDP packet on its way, we need someone to receive it. The counterpart of my wake-up light application is a small server application running on my Raspberry Pi. A Python script listens for UDP packets and reads the timestamps out of them, and finally schedules a Bash script to be run at the specified time using the Linux utility `at`. The Bash script then executes a Python script which turns on the light.

Yeah, I know. It's hacky. It was 3 AM, okay? I will fix it someday.

Hue vs. Light Switch

As cool as smart lights are, they have one issue – your light switch will become useless. When I use my light switch, I cut the power to my light bulbs and subsequently can't turn them on through the Hue app – duh. So I don't have a usable light switch anymore, just the app. Annoying.

Naturally, you can buy a dedicated Hue light switch which connects to the ZigBee network – for 30 bucks. Not even I would spend 30 bucks on a light switch. However, not having a light switch is also not an option. Good thing I like to tinker.

A friend of mine had recently looted a sizeable treasure in the trash at his workplace. He found bags filled with cheap USB remotes. Dozens and dozens of cheap-ass USB remotes, battery included. Naturally, he knew that we could put these to good use and saved them from the trash can. He distributed them among his friends, so I also got a hold of two of these remotes. Thanks, Phil!

The remotes came with individually paired USB dongles and were detected by the system as keyboards. I quickly wrote up a script that listens to these "keyboards" and maps certain keypresses to Hue light actions. I got a few more remotes from my friend, plugged all the dongles into my Raspberry Pi and put the remotes all over the place – on my desk, on the sofa, next to my bed. I can control my lights from anywhere now. Take that, light switch!

By the way, in case you ever wondered – window color is wonderfully suitable to color-code USB dongles.

The Perils of IoT

This article might make it seem like I fully embrace IoT stuff, so I feel obliged to talk about the downsides of smart things here. After all, IoT botnets make the news more and more often. And while the Hue system seems to be one of the more well-constructed IoT lines – it even uses standard cryptography techniques – it is not invulnerable to attacks either[3,4].

Even so, the Internet of Things will come – saying anything else is futile. And there will be catastrophes. And thus, it is important to never blindly trust these devices. They can fail, they can be compromised.

While the Hue system seems to be one of the more well-constructed IoT lines, it is not invulnerable to attacks either.

I'm a tinkerer. I like to toy with things, and that's why I got these light bulbs. I'm not relying on them – I have other lights installed as well – and I won't lose much if they would get compromised. They don't store sensitive data like an IP camera might do. They're just lights. However, it is everybody's personal decision how much they want to entrust to smart things.

And now, go watch a video of a drone hacking the light bulbs of an office building[5].

References

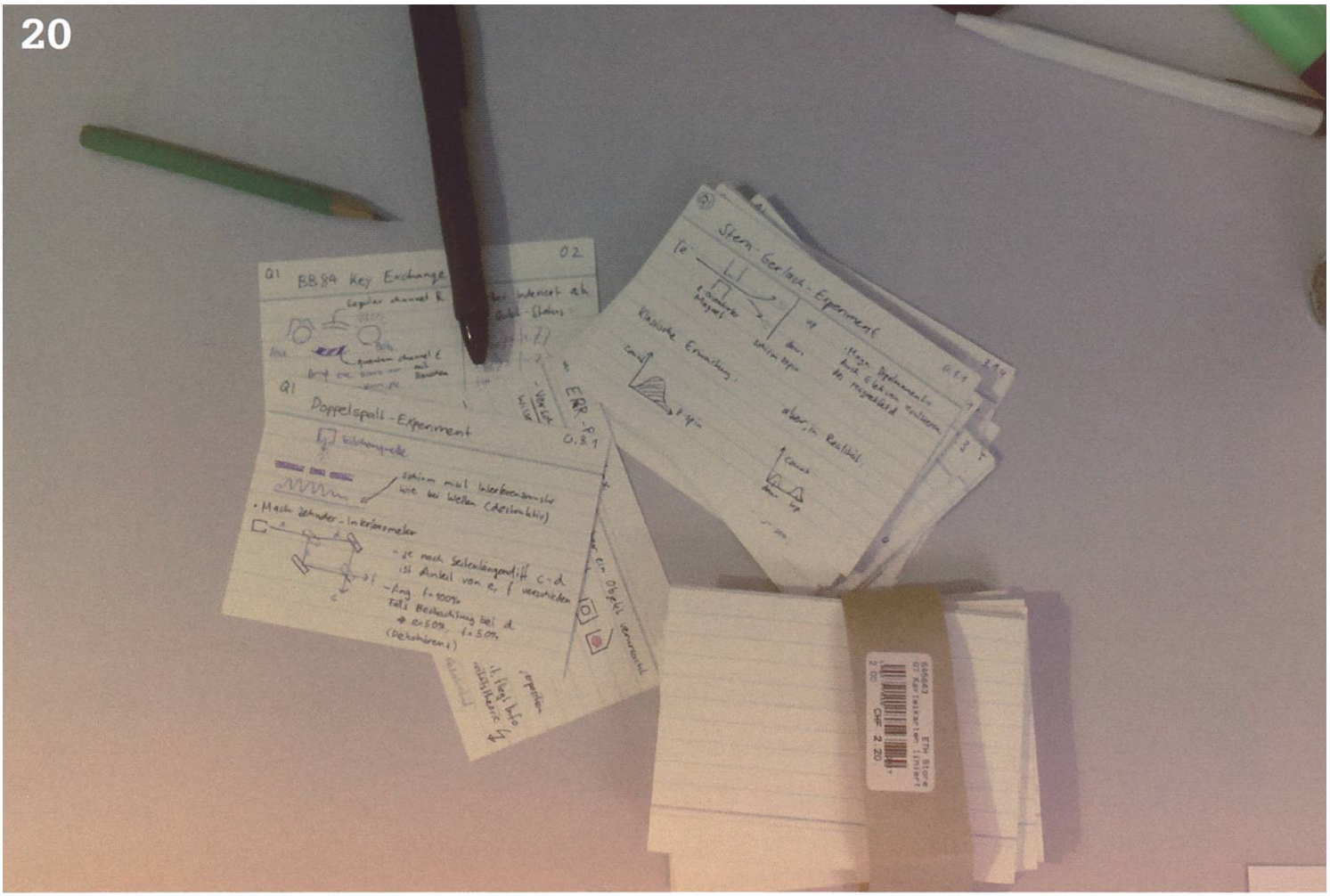
[1] http://www.zigbee.org/download/standards-zigbee-specification/

[2] https://en.wikipedia.org/wiki/ZigBee

[3] https://www.theregister.co.uk/2016/11/10/iot_worm_can_hack_philips_hue_lightbulbs_spread_across_cities/

[4] http://iotworm.eyalro.net/iotworm.pdf

[5] https://www.youtube.com/watch?v=Ed1OjAuRARU



Studying with Flash Cards like a Uni Student

ZENO KOLLER - USES ALGORITHMS FOR LEARING ALGORITHMS

This past autumn semester - the second in my Master's program - has been the first in my career as an ETH student in which I've created flash cards for studying. And it won't be the last one.[1]

Past Semester: My Starter Deck

In the past autumn semester, I created flash cards for three courses: *Network Security*, *Human Computer Interaction* and *Distributed Algorithms*. While the first two were definitely fact-heavy, *Distributed Algorithms*, with its oral exam, focuses more on the insights gained in class. Studying the material with flash cards turned out well for all of them. First came the task of creating the cards. Physical ones, that is. Not an easy task if your handwriting is hardly legible. I ended up throwing away and rewriting every fifth card or so, as I was not happy with the way I phrased a particular concept. Still, I found the task of creating the cards to be rather relaxing. It also has benefits for understanding: To extract the essence of a concept, one has to understand it. After a couple of hours of boiling down course

notes, slides and readers, I had three decks of cards: One for each topic. Holding the content of a whole lecture in the palm of one's hand feels powerful. Almost as powerful as carrying a stack of shiny Pokémon cards as a school kid…

I took the cards with me on the train, and ended up moving a lot of studying time from "dedicated" study slots to periods of time where I would be doing nothing useful anyway. Periods where I would otherwise just dick around on my smartphone. The physical cards had the advantage of not distracting me with push notifications.

My algorithm for studying the cards was just to go through the whole deck, either forward or backwards. The cards mostly had the name of a concept or an algorithm written on the front and some properties or the actual algorithm on the back. I would look at the term and try to think of the concept. Often, I would also write small questions about the concept that I would then try to answer. This emulates the exam situation, where you have to do the same thing. As the cards are much less information-dense than lecture notes or even a summary, they require a bit more focus - maybe for the better? When I did not really understand something, I would not just gloss over it and say, 'Yeah, whatevs, this won't be in the exam anyways', and skip to the next section like I might be tempted to when reading a textual summary. Instead, I would extract the card from the deck (I made sure to number them in their original order for re-sorting later) and push it to a stack of concepts I have to revisit more carefully.

What's missing from this algorithm is a more systematic evaluation of how well I know certain topics or a way of exponentially increasing the period of a certain card appearing. All in all, my rating of this studying method is: 4/5, would do again.

Going Digital

This semester, my courses include *Computational Intelligence Lab*, *Principles of Distributed Computing* and *Natural Language Understanding*. The first two are a bit less suitable for flash cards. Both require not only facts, but knowing how to apply methods really well. Natural Language Understanding has a bit more facts, but I also feel the potential of some nasty 'Apply this algorithm that no one has ever done by hand except in an exam'-type question. However, I now have another motivation for studying with them: Two weeks after the semester, I will begin my last slice of civil service, which lasts until after the exams. While the work I'll be doing probably won't be too exhausting, the chances of me studying for hours in the evening are not too great either. So what is a time-constrained student to do? My plan is to study a lot during the semester and then to create some cards again to revise. This time around, I might also go digital, despite my aforementioned skepticism. Then I could use to my advantage that today, basically everyone is a smartphone addict. This way, during breaks, I can pretend to act like a normal person while in reality cramming in some algorithms.

This way, I can pretend to act like a normal person while in reality cramming in some algorithms.

➔

Anki

The list of flashcard software is pretty long[2], but the one most people seem to use is Anki[3]. The desktop app is open source. Anki's ecosystem also has a web app which directly syncs with the desktop app. On the go, you can either use the web app to study or use one of several mobile apps.

To evaluate its usefulness for my purposes, I gave Anki a spin. With more than ten years of development time, Anki is pretty complete feature-wise. The basic cards have a question-answer format, but custom templates can be created. The cards are rendered with HTML and can include text, images, LaTeX and even sound and videos. It also supports creation of "cloze deletion" cards. Cloze[4] is a test where some _____ are deleted from the sentence and you have to guess them. I found that using Cloze is an efficient way of directly converting lecture notes to flash cards. Anki's markup for Cloze cards even allows for creating multiple cards from a single sentence - one for each deletion.

In the "study" mode, you assess for each card how well you knew the answer. Based on this feedback, the card will be presented to you either sooner or later using a spaced repetition algorithm. Anki also keeps statistics on your study habits.

Anki's age shows not only in its stability and feature-completeness. The UI feels a bit dated by today's standards. It's a cross-platform app, so it does not integrate itself well into the system - it would be nice if the LaTeX renderings would support high-resolution displays. On the other hand, the app can be controlled via keyboard, which is a big plus. This makes the repetitive task of creating cards slightly more bearable. For macOS, there is an alternative app which uses Anki's format: AnkiApp. It looks like a blown-up iOS app and cannot be navigated via keyboard, so I instantly discarded it.

There are lots of alternatives to Anki. Notable mentions are Memorang and Tinycards by Duolingo. Both have slick mobile apps and take a gamification approach to motivate you to study (as pioneered by Duolingo). These apps are more geared towards users that study using provided decks, because the interfaces for creating cards are rather poor (web and mobile only). Also, there is no support for LaTeX and the cloud-based storage might not be your cup of tea (you can create private decks, though).

In summary, I find the available software for flash cards unsatisfactory for my needs. Either I use something that is fast but does not match my expectations design-wise, or I use the latest mobile apps which are nice to study with, but lack features on the creation side. I want both!

Considering this dilemma, I might still end up creating plain old analogue cards. If you know a solution that fits my needs, it will unfortunately be too late, because I'll already have written all my cards.[5]

Links

[1] Only the second last one.

[2] https://en.wikipedia.org/wiki/List_of_flashcard_software

[3] https://apps.ankiweb.net

[4] https://en.wikipedia.org/wiki/Cloze_test

[5] But tell me anyway: kollerz@vis.ethz.ch :-)


Hast du
VISIONEN
?
Hier ist noch Platz für deinen Artikel!
Hast du etwas, das du deinen Mitstudenten mitteilen willst? Warst du im Ausland und willst von deinen grossartigen Erfahrungen berichten? Hast du ein Praktikum gemacht und möchtest erzählen, was du erlebt hast? Dann nutze die VISIONEN als Plattform und schick uns deinen Artikel! Alle Infos unter:
http://www.vis.ethz.ch/de/visionen/articles
Oder möchtest du aktiv mithelfen? Dann werde Mitglied der VISIONEN-Kommission. Einfach E-Mail an:
visionen@vis.ethz.ch
V I S
http://www.vis.ethz.ch

Command Line Case Studies: vagrant

PASCAL WACKER - DOESN'T LIKE COFFEE

Last time, Zeno wrote about a small program called awk that lets you process text files. In this episode of Command Line Case Studies, I'm going to talk about vagrant. Vagrant isn't only available for Unix machines like awk, but can also be used on Windows. So what is it? It's open-source software that lets you build and maintain portable virtual development environments.

Before I started my bachelor here at ETH, I did an apprenticeship and then worked for about 5 years as a web developer. Back in the days when I wrote my first websites, IE 6 was a requirement. Even worse, IE on Mac was a requirement and just believe me, it was horrible. When you wrote code for a website, you'd usually test the HTML stuff just on your computer, upload it using FTP and hope for the best. Later a tool called XAMPP[1] came along. You had a local Apache, a DB and PHP running on it. There were a few issues here and there, but it was such an improvement in my eyes. You didn't have to write PHP files, upload them to a server just to test it, you just had your website running in http://localhost!

Often this setup ended in "It works on my machine, there has to be something wrong on the server!" discussions.

So ... everything's good then, right? End of story! But it wasn't. Often this setup ended in "It works on my machine, there has to be something wrong on the server!" discussions. Another issue with XAMPP is that it's just PHP. What if you want to use Python, Java, Ruby or NodeJS? Sure, you could make it work somehow, but there just has to be a better solution!

Virtualisation

There is a thing called a hypervisor[2]. It enables you to run a virtual OS, let's say a virtual server, on your computer which, if you configure it properly, has the same configuration as the web server sitting somewhere in the cloud. One of the best-known Hypervisors is VirtualBox[3]. Through the ETH IT Shop[4] you can also get the VMWare products. With those hypervisors, you can spin up your VM, use it, stop it and destroy it. So far, so good, so how do you get the VM to do what you want it to do, let's say install Apache, PHP and MySQL on it? You could simply boot a Debian system on it and then install everything by hand. The next step would be to have a bash script that takes care of it for you, or you could use an automation tool like Puppet, Salt, Chef

or Ansible. But still, you need to setup a VM by yourself. Is there even more room for improvement?

Enter Vagrant

Vagrant does this for you. It doesn't matter what hypervisor you'll use, what method for provisioning or OS you use. There will be just a few simple vagrant commands you have to keep in mind. You can create a Vagrant file, run it on Linux using Virtualbox, copy it to a Windows machine using VMWare Workstation and run it there. Both host machines will have a VM running with the exact same configuration. Neat, isn't it? If you ask Wikipedia what a vagrant is, it tells you: "[...] A vagrant could be described as being "a person without a settled home" [...]". In our case, it's a VM, but otherwise, that description fits pretty well.

How does it Work?

Vagrant builds on top of all the existing software and basically just provides a command line interface to manage it. The commands you'll need are `vagrant init` to create a new Vagrantfile, `vagrant up` to start a VM, `vagrant halt` to stop a VM and `vagrant destroy` to delete a VM. There are other commands like `vagrant share` to share the VM with the internet or `vagrant provision`, but you can read about them in the docs of vagrant.

Let's for example just spin up a VM with a Ubuntu 12.04 LTS 64-bit inside. First, we have to create a folder we can work in. Let's just assume we already did that and are in that folder. What we have to do now is simply:

```
$ vagrant init hashicorp/precise64
$ vagrant up
```

This will create a Vagrantfile in that folder and create the VM for you. After the VM was created and is booted, you can SSH into it by using

```
$ vagrant ssh
```

There it is, your new VM. To exit the VM, simply type `exit` and you'll be in you computer's regular command line again. From there, to shut down the VM, simply type:

```
$ vagrant halt
```

And to start it again, use:

```
$ vagrant up
```

as you did the first time when creating it. To delete a VM, you simply use:

```
$ vagrant destroy
```

which you will have to confirm.

Installing Software on Your VM

Note: I'm assuming in this section that you destroyed your VM and you're running it for the first time, since only then provisioning is done automatically!

That was fun, but what good is a VM without anything on it? Let's install Apache, shall we? First, we create a file called `installApache.sh`. This file should look like this:

```
#!/usr/bin/env bash

apt-get update
apt-get install -y apache2
if ! [ -L /var/www ]; then
  rm -rf /var/www
  ln -fs /vagrant /var/www
fi
```

What we do here is get updates, install Apache, delete the `/var/www` folder and create a symlink from `/vagrant` to `/var/www`. Why? Simple: Vagrant automatically maps your folder on the host machine to `/vagrant` in ➔

the guest. But to do this, we also have to tell vagrant about that file! We need to edit our Vagrantfile, and it should look something like this after we're done:

```
Vagrant.configure("2") do |config|
  config.vm.box = "hashicorp/
precise64"
  config.vm.provision :shell, path: "
installApache.sh"
  config.vm.network :forwarded_port,
guest: 80, host: 8080
end
```

In the Vagrantfile, we tell vagrant to use the installApache.sh file for provisioning and we also tell it to map port 8080 on the host to port 80 in the VM, so we can call http://localhost:8080 to access a website in our VM. Let's try it out by creating a file called `index.html` with any valid HTML inside, for example

```
<!doctype html>
<html>
    <head>
        <title>It's alive!</title>
    </head>
    <body>
        <h1>It works!</h1>
    </body>
</html>
```

Start the VM using `vagrant up` and check your browser! Beautiful, isn't it? If you change your provision script, you can simply call `vagrant provision` (on a running machine), to have it run. There is practically no limit to what you can do when provisioning. You could, for example, clone a git repo and initialize a DB, or install Python. More about provisioning and what languages can be used other than plain bash scripts can be found in the docs[5]. If you feel like it, you can specify additional folders to be synced. I, for example, usually create a folder `source` on my host which is synced to `/var/www` on the guest. You can find more information about synced folders, and about how to use NFS or SMB instead of the basic syncing, in the docs[6].

That's pretty much it. The official website with all the docs is located at https://vagrantup.com.

Happy Coding!

Bonus

If you're lazy and don't want to deal with provisioning yourself, have a look at https://puphpet.com/. Under the hood, it uses Puppet to provision your vagrant machine, but it lets you configure what you need with a simple GUI.

"**Vagrant** is a tool for building and managing virtual machine environments in a single workflow. With an easy-to-use workflow and focus on automation, Vagrant lowers development environment setup time, increases production parity, and makes the "works on my machine" excuse a relic of the past."

Links

[1] https://apachefriends.org/

[2] https://en.wikipedia.org/wiki/Hypervisor

[3] https://www.virtualbox.org/

[4] https://idesnx.ethz.ch/

[5] https://www.vagrantup.com/docs/provisioning/

[6] https://www.vagrantup.com/docs/synced-folders/basic_usage.html

„Unsere Softwarelösungen setzen neue Standards in der Sensorik."

Eduard Rudi,
Software Engineer

„Become part of the Sensirion success story". Wollen Sie Ihrer Karriere den entscheidenden Kick geben und sich neuen Herausforderung stellen? Dann heissen wir Sie herzlich willkommen bei Sensirion.

Sensirion steht für Hightech, Innovation und Spitzenleistungen. Wir sind der international führende Hersteller von hochwertigen Sensor- und Softwarelösungen zur Messung und Steuerung von Feuchte, Gas- und Flüssigkeitsdurchflüssen. Unsere Sensoren werden weltweit millionenfach in der Automobilindustrie, der Medizintechnik und der Konsumgüterindustrie eingesetzt und tragen zur stetigen Verbesserung von Gesundheit, Komfort und Energieeffizienz bei. Mit unserer Sensorik liefern wir damit einen aktiven Beitrag an eine smarte und moderne Welt.

Schreiben Sie Ihre eigenen Kapitel der Sensirion Erfolgsgeschichte und übernehmen Sie Verantwortung in internationalen Projekten. Stimmen Sie sich auf www.sensirion.com/jobs auf eine vielversprechende Zukunft ein.

www.sensirion.com/jobs

SENSIRION
THE SENSOR COMPANY

Logikrätsel. Heute: Rechenspiele

Das Gedankenspiel

Matthias denkt sich eine Zahl, verdreifacht sie, addiert 792 zum Resultat, teilt das neue Resultat durch 144 und erhält eine ganze Zahl (ohne Rest). Danach zieht Matthias vom Resultat 2 ab und multipliziert das Resultat mit 100´000. Das Resultat ist eine Million. Mit welcher Zahl hat er gestartet?

Teiler oder Vielfaches

Didier und Muriel spielen mit Karten die von 1 bis 10 nummeriert sind. Didier beginnt und nimmt die Karte mit der 2 aus dem Kartenstapel. Abwechslungsweise wählen nun beide Spieler je eine Karte, deren Nummer immer jeweils ein Teiler oder ein Vielfaches der Zahl der letzten Karte des vorherigen Spielers sein muss. Kann ein Spieler nicht mehr ziehen, hat der andere gewonnen. Nach der 2, welche Karte muss Muriel nehmen, damit sie sicher das Spiel gewinnen wird? Bemerkung: 1 ist ein Teiler aller Zahlen.

Wer verliert verdoppelt

Drei Spieler haben drei Runden des Spiels «Wer verliert verdoppelt» gespielt. In jeder Runde gibt es einen Verlierer und dieser muss das Guthaben der anderen beiden Spieler verdoppeln (das Spiel endet, wenn dies nicht mehr möglich ist). Nach diesen drei Runden besitzt jeder Spieler 24 Franken. Welches Guthaben hatten die Spieler vor Spielbeginn (in aufsteigender Reihenfolge) wenn kein Spieler zu Beginn mehr als 40 Franken hatte? Zusatzfrage: Wie viele Lösungen gibt es?

Dreimal mehr «Kopf»

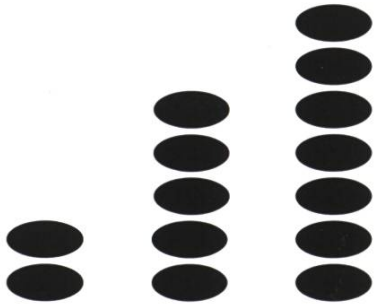
Mathilda wirft mehrmals einen Einfränkler in die Luft und notiert sich die Folge von «Kopf» (K) und «Zahl» (Z), die sie dabei erhält. In dieser Folge von «K» und «Z» sind die Folgen von jeweils vier aufeinanderfolgenden Würfen immer unterschiedlich und Total hat es genau drei Mal so viele «K» wie «Z». Wie viele Buchstaben enthält die Folge maximal?

Division mit 11

Mathilda teilt eine dreistellige Zahl durch 11. Überraschung! Der Quotient ist gleich der Summe der Quadrate der drei Ziffern der ursprünglichen Zahl. Wie lautet die Anfangszahl? Zusatzfrage: Wie viele Lösungen gibt es?

Nehmen oder teilen

Christian und Philippe spielen folgendes Spiel. Sie haben drei Stapel zur Verfügung, die aus 2, 5 und 7 Spielsteinen bestehen. Mögliche Spielzüge sind:



- Entweder ein oder mehrere Spielsteine von einem Stapel entfernen (der Stapel darf danach auch leer sein)

- Oder einen Stapel mit mindestens zwei Spielsteinen in zwei Stapel aufteilen (die Stapel müssen nicht gleich gross sein) ohne einen Spielstein zu entfernen.

Die Spieler spielen abwechslungsweise einen Zug, gewonnen hat der Spieler, welcher den letzten Spielstein nimmt. Philippe beginnt. Welches muss sein erster Zug sein, damit er das Spiel immer gewinnen kann, egal wie Christian spielt? Zusatzfrage: Wie viele Lösungen gibt es? Die Startaufstellung wird als (2; 5; 7) beschrieben. Die Antwort soll die Stellung nach dem ersten Zug beschreiben, in der Form (a; b) oder (a; b; c) oder (a; b; c; d) wobei a, b, c und d die Anzahl Spielsteine (in aufsteigender Reihenfolge) in den jeweiligen Stapeln angeben.

Lösung der letzten Ausgabe

Französischtest
TRENTE
Philippes Palindromzeit
70 Minuten
Der alte Taschenrechner
16 Mal (353 + 553 + 553 + 553 =)
Von eins bis (fast) unendlich
56
Selbstreferenz
2359
Die unendliche Geschichte
8/27

Alle Rätsel stammen aus dem Archiv der internationalen Mathematik- und Logikmeisterschaft, der Meisterschaft mit weltweit über 120'000 Teilnehmern. Mehr Information unter http://www.smasv.ch.

Thorben denkt: Am Arbeitsplatz

THORBEN BOCHENEK – MAG KOMPLIZIERTE UND KOMPLEXE KUNDEN

Diesmal schreibe ich von meinem Arbeitsplatz aus. Es ist relativ praktisch, wenn einem die Firma gehört: Man kann sich einfach hinsetzen und machen was man will. Zum Beispiel einen VISIONEN-Artikel schreiben. Zumindest solange die Firma auch noch irgendwie Geld hat. Wenn das ausgeht, wird es dann doch eher stressig. Oder wenn ein Kunde wieder mal was will.

Kunden sind schon lustig: Am wenigsten Stress hätte man ohne sie, aber dann fehlen Spass und Geld. Was dann wieder den Stress erhöht. Aber etwas Stress ist natürlich wiederum spassig. Aber ist zu viel Spass auch stressig?

Kunden sind schon lustig: Am wenigsten Stress hätte man ohne sie, aber dann fehlen Spass und Geld.

Auf jeden Fall ist mein Schreibtisch ziemlich sauber. Wir haben so einen statischen Staubwedel gekauft. Es ist seltsam befriedigend, damit seinen Monitor abzustauben. Kann ich nur empfehlen. Toll ist auch ein schwarzer Tisch. Darauf sieht man Staub viel besser. Das ist ideal, wenn man gerne putzt: Eine wahre Wonne! Zuhause putze ich eigentlich fast gar nicht. Ich habe mir bereits vor ein paar Jahren einen Staubsaugerroboter gekauft. Er heisst "Putzi". Am Anfang fand ich ihn einfach interessant, jetzt sehr praktisch. Ich habe gar keinen normalen Staubsauger. Das verwirrt Leute häufig. Das macht aber nichts. Leute dürfen auch mal verwirrt sein.

Verwirrung ist auch so ein cooles Wort. Ich hab mal einen tollen Vortrag gehört, dass Komplex das Gegenteil von Simplex sei und nichts anderes bedeutet, als dass viele Fäden ineinander verwoben sind. Also quasi verwirrt. Seitdem erkläre ich immer allen, das komplex und kompliziert sehr unterschiedlich sind. Komplex sind im Prinzip viele einfache Teile zusammengesetzt. Kompliziert ist dagegen ein einzelner nicht einfacher Teil. Cool, oder? Ein Auto ist zum Beispiel eher komplex und weniger kompliziert. Eine Fourier-Transformation eher kompliziert und ein Atomkraftwerk irgendwie beides. Ist übrigens eine tolle Geschichte für Parties.

Never Heard of It #17

BALZ GUENAT – HAS FROZEN UP IN SPACE

I recently stumbled upon "Music from Fortitude" when it popped up as a 'recommended new release'. I gave it a shot and ended up really liking it. It reminded me very much of "Dark Jovian", a short album by Amon Tobin who was featured here before. Both albums feel like soundtracks, but only one of them actually is one. It's time for another double feature! If you're looking for a 4/4 beat, see ya next time, because you won't find it here!

Amon Tobin - Dark Jovian &
Ben Frost - Music from Fortitude

Curiously, both of these albums explicity specify a setting—an environment where they belong. The setting of "Music from Fortitude" is clear: it's the soundtrack to Fortitude, a TV series set in the Antarctic, so this is where it belongs. For "Dark Jovian", it's less obvious, that is, unless you know what Jovian means: Jovian is the adjectival form of Jupiter and is another term for 'gas giant'. So to get into the mood for "Dark Jovian", we have to think ourselves into orbit. This leaves us with a frozen wasteland and an inhospitable planet in dark and empty space. Two harsh environments of solitude: a great match in my opinion.

When listening to "Dark Jovian", I always have this image inside my head: The bridge of a lone spaceship, with its machinery churning away deep inside, and outside the windows, slowly appearing from behind an unlit moon, a massive planet with its surface features frozen in perfect chaos. I think this fits the music perfectly and vice versa.

These two albums share many aspects in terms of feel, but they are a bit different in the instrumentation. In both, we get lots of electric rumbling, humming and buzzing. Ben Frost combines this with more organic sounds like strings and choir, whereas Amon Tobin keeps mostly to synthetic sounds or heavily distorted samples.

Alright, I wrote too much already. So, if you, like me, like to close your eyes and let yourself float on a bed of noise, give these two albums a try.

Year: 2015 (Dark Jovian),
2017 (Music from Fortitude)

Length: 38:45,
70:14

Spotify: tinyurl.com/NeverHeardSpotify

Google Play Music: tinyurl.com/
NeverHeardGoogle

Die Welt gemäss Beni Koller

Ausgleichsrechnung

MICHAEL GROSSNIKLAUS - NOCH IMMER TEMPORÄRER REDAKTOR

Beni Koller steht im Müller an der Markstätte in Konstanz und versteht die Welt nicht mehr. Es ist Samstagmorgen kurz nach 9 Uhr und Beni war extra früh aufgestanden, um vor den Massen an Schweizer Einkaufstouristen vom vorteilhaften Eurokurs und den günstigen Preisen in Deutschland zu profitieren. Sein Erstaunen rührt aber nicht daher, dass es ihm nicht gelungen war, seine Mitbürger auf die hinteren Ränge zu verweisen, oder dass er sich vor lauter Menschen im Geschäft kaum noch bewegen kann. Sprachlos gemacht hat ihn die Parfümflasche, die er in seiner rechten Hand hält. Es handelt sich dabei um eine 200ml-Flasche des Parfüms «Cool Water» von Davidoff. Genau die gleiche Flasche hatte Beni vor weniger als zwei Wochen in Zürich in der Import-Parfümerie zum stolzen Preis von 130 Franken gekauft. Was ihm an der Flasche im Müller in Konstanz sauer aufstösst ist, dass sie nur 49,95 Euro kostet.

Natürlich wusste Beni zum Zeitpunkt, als er das Parfüm in Zürich gekauft hatte, dass er in absehbarer Zeit nach Konstanz zum Einkaufen fahren würde. Und natürlich war Beni damals schon bewusst, dass es insbesondere bei Drogerieprodukten teilweise erhebliche Preisunterschiede zwischen der Schweiz und Deutschland gibt. Trotzdem entschied er sich, das Parfüm in Zürich zu kaufen, da einerseits sein Vorrat bereits erschöpft war und er andererseits niemals erwartet hätte, dass der Preisunterschied mehr als das Doppelte ausmachen würde. Als ihm so richtig klar wird, dass er mit dieser Kaufentscheidung rund 80 Franken in den Sand gesetzt hat, ist Benis Einkaufslaune für eine Weile völlig verflogen. Als er zudem bemerkt, dass Müller auch einen Onlineshop hat, über den er selbst mobil den Preis des Produkts in Deutschland ganz einfach hätte herausfinden können, ärgert sich Beni noch mehr über sich selber. Besonders nervt ihn, dass es überhaupt nicht das erste Mal passiert, dass er sein Geld verschenkt, weil er sich nicht genügend informiert hat.

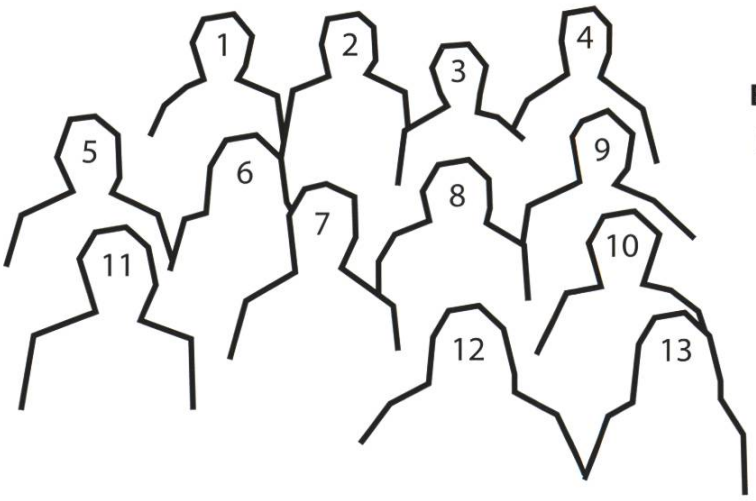
Doch anstatt sich besser zu informieren, hatte sich Beni in der Vergangenheit lieber über Leute lustig gemacht, die jeweils alle Angebote verglichen und dann das günstigste kauften. Gerne schimpfte er solche Leute «Rappenspalter» und hat ihnen überheblich vorgerechnet, dass die Lebenszeit, die sie mit dem endlosen Preisvergleichen vergeuden würden, viel wertvoller sei, als die erhoffte Ersparnis, die sie dadurch gewinnen könnten. Ein weiteres Argument, das er in solchen, meist polemisch geführten Diskussionen gerne einbrachte, basierte auf einer Art Finanzkarma, das Beni zu beobachten glaubte. So behauptete er stets unbeirrbar, dass in seiner Erfahrung allfällige Verluste, die durch mangelnde Information entstanden, sich im Durchschnitt immer durch Situationen ausglichen, bei denen man zufällig profitiert. Einen anekdotischen Beweis, den er für

die Existenz solcher Situationen gerne anführte, war die 100-Franken-Note, die er als Kind einst vor dem Gartentürchen seines Elternhauses fand. Diese musste er zwar, wie es sich gehört, erst auf dem Polizeiposten abgeben, durfte sie aber nach Ablauf einer Frist, in der sich der Eigentümer hätte melden können, behalten. War die Diskussion bzw. sein Monolog dann am Punkt angekommen, an dem er keine Widerrede mehr erhielt, zuckte er typischerweise noch lässig mit den Schultern und lachte siegesgewiss: «You win some, you lose some, alles kein Problem».

Doch an diesem Samstagmorgen hat Beni ein Problem. So sehr er sich auch anstrengt, es wollen ihm ausser der 100-Franken-Note keine Beispiele einfallen, bei denen er signifikant finanziell profitiert hat, ohne gross etwas dafür zu tun. Ja, natürlich, fand er mal einen Skipass auf der Piste, für den er an der Talstation eine Depotgebühr von 5 Franken zurückerhielt. Und klar ist es auch schon vorgekommen, dass am Ticketautomat der SBB der Kunde vor ihm ein paar Münzen im Ausgabefach hat liegen lassen. Doch je länger Beni darüber nachdenkt und beginnt nachzurechnen, desto mehr muss er einsehen, dass diese kleinen Beträge die Fehlausgaben, die er mit seinem «Don't worry, be happy»-Ansatz eingefahren hat, in der Summe nicht ausgleichen können. Zähneknirschend beschliesst Beni also, die Konsequenzen aus dieser Erkenntnis zu ziehen und sich in Zukunft besser zu informieren. Dass diese Verhaltensänderung mehr Aufwand für ihn bedeuten wird, ist allerdings nicht der Punkt, der ihn am meisten daran stört. Viel mehr nervt ihn, dass er sich dadurch eingestehen muss, dass Bekannte von ihm, die er jahrelang verspottete, besser Bescheid wussten als er selbst. Langsam kehrt auch Benis Einkaufslust wieder zurück und er will sich gleich ans Werk machen, indem er ein paar Fehlausgaben der Vergangenheit mit Schnäppchen in Konstanz kompensiert.

Am frühen Nachmittag sitzt Beni mit zwei vollen Taschen und einem gut gepackten Rucksack im Zug und wartet, bis dieser Konstanz in Richtung Zürich verlässt. Ein Zöllner, der den Zug kontrolliert, fragt ihn, was er alles eingekauft habe. Gehorsam überreicht ihm Beni alle Belege und beteuert, dass er unter der Freigrenze von 300 Euro geblieben sei. Der Zöllner schaut ihn etwas verdutzt an, informiert ihn, dass die Freigrenze bei 300 Franken liegt, und beginnt, eine Rechnung für die zusätzlich fällige Mehrwertsteuer auszustellen.

Visionäre



Blockieren gerne Treppen: Noah Delius[1], Dominic Sonderegger[2], Silvia Siegrist[3], Pascal Wacker[4], Aline Abler[5], Sarah Kamp[6], Andreas Brombach[7], Balz Guenat[8], Mickey Vänskä[9], Zeno Koller[10], Robin Bisping[11], Julian Croci[12] und Lena Csomor[13].

Impressum

VISIONEN

Magazin des Vereins der Informatik Studierenden an der ETH Zürich (VIS)

Ausgabe Juni 2017

Periodizität 6x jährlich
Auflage 2200

Chefredaktion
Julian Croci
chefredaktor@vis.ethz.ch

Cover
Layout-Team

Layout
Aline Abler
Pascal Wacker
Robin Bisping
layout@vis.ethz.ch

Inserate
Balz Guenat
inserate@vis.ethz.ch

Anschrift Redaktion & Verlag
Verein Informatik Studierender (VIS)
CAB E31
Universitätsstr. 6
ETH Zentrum
CH–8092 Zürich

Inserate (4-farbig)

½ Seite	CHF 1000.–
1/1 Seite	CHF 1800.–
2/2 Doppelseite	CHF 4000.–
1/1 Seite, Umschlagsseite (U2)	CHF 3000.–
½ Seite, Rückumschlag (U4)	CHF 3000.–

Andere Formate auf Anfrage.

Lektorat
Felice Serena
Dominic Sonderegger
Noah Delius
Nils Leuzinger
Silvia Siegrist
Lena Csomor
Mickey Vänskä
lektorat@vis.ethz.ch

Redaktion
Stefan Dietiker
Lukas Widmer
Andreas Brombach
Sarah Kamp
Pascal Wiesmann
Manuel Braunschweiler
Zeno Koller
Michael Grossniklaus
Julian Croci
redaktion@vis.ethz.ch

und freie Mitarbeiterinnen und Mitarbeiter

Druck
Sprüngli Druck AG
5612 Villmergen
http://www.spruenglidruck.ch/

Die Visionen werden klimaneutral gedruckt.

FSC www.fsc.org **Mix** Papier aus verantwortungsvollen Quellen **FSC® C007061**

Swiss Climate Klimaneutral gedruckt

Der VIS ist Teil des Verbandes der Studierenden an der ETH (VSETH).

AZB
PP/Journal
CH – 8092 Zürich

Falls unzustellbar, bitte zurück an:
Verein der Informatik Studierenden
CAB E31
Universitätsstr. 6
ETH Zentrum
CH-8092 Zürich