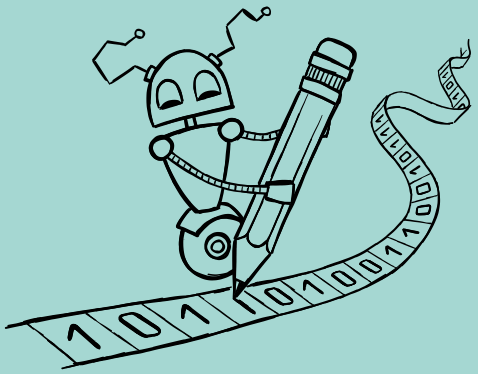
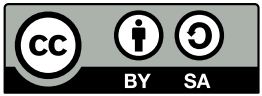


2^4 Things I Learned in Computer Science

University of Basel, Dept of Mathematics and Computer Science



2^4 Things I Learned in Computer Science

Department of Mathematics and Computer Science

University of Basel

Author’s Note

Studying Computer Science is like taking a deep dive into an endless ocean of ideas, concepts, and tools. It is a vast ecosystem where everything is interconnected, where exploring one thing opens doors to a plethora of other, equally interesting, things. It is easy to get lost in this wide landscape, even more so if it is not one’s main field of expertise.

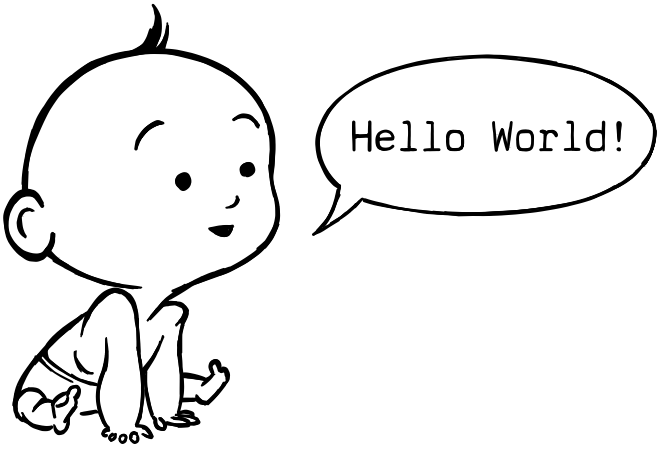
This is why Computer Science students of the University of Basel set out to identify the essential concepts and ideas they discovered during their studies. The intention was to go beyond the scope of traditional textbooks and also include the pecularities that are not formally part of a usual Computer Science curriculum. Finally, the goal was to present the fruit of this work to people outside of our field.

The creation of this booklet was organized in the form of a Master's level seminar, which had its first iteration during the spring semester 2019. We have listed the participants on the inside back cover.

The result is **2^4 Things I Learned in Computer Science**, a collection of short illustrated articles. It is available in the form of this small booklet, and online on the companion website [1]. Due to the success and the positive feedback, we will continue this work and are now planning for a next edition of the seminar.

We hope that you enjoy these info bytes
and that you pass them on to your friends!

[1] `https://tilics.dmi.unibas.ch/`


Hello World!

0

The first words of a computer scientist

The first piece of code you usually find in a programming book or tutorial is a “Hello World!” program. The purpose of this program is to display the text “Hello World!”. Because this is rather simple for nearly all programming languages, it is a good example to see the basic flavor of a programming language. Due to this tradition, these are usually the first lines of code a programmer writes in their live.

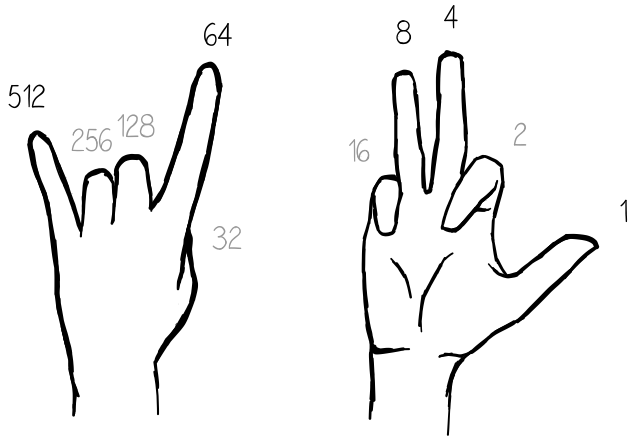
1

Being lazy is a good thing

A lazy person is always searching for ways to make things faster and more efficient.

Instead of doing the same thing over and over again, a good computer scientist tries to automate the task. This not only improves the reliability, but also saves time in the long run. Furthermore, a good computer scientist is also lazy when it comes to writing code. Rather than solving a specific problem, a good programmer develops a generic solution that can be reused for similar problems.

Good programmers are diligent in order to be lazy. Or with other words: They are strategically lazy. And that’s a good thing.

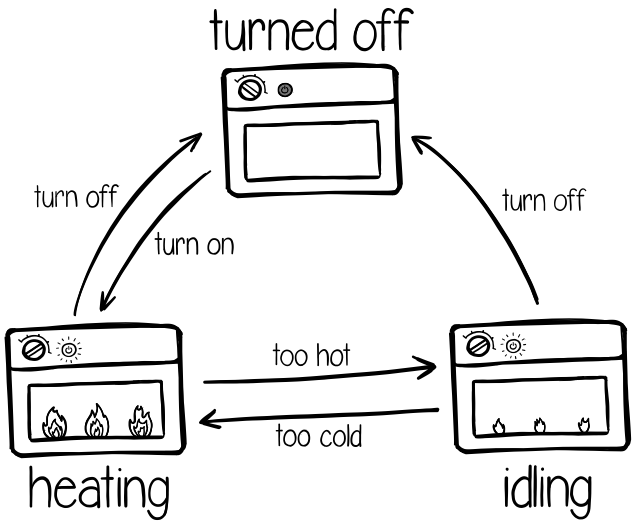


$$512 + 64 + 8 + 4 + 1 = 589$$

2

Counting to more than ten with two hands

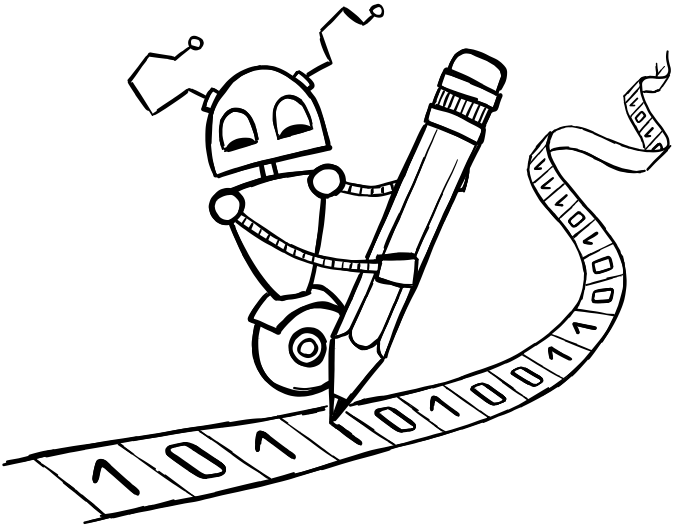
Normally you count with your fingers by stretching them out one after another. You don’t care about which ones are stretched out, but only about the amount of them. Each of them is as significant as the other. By changing the significance in the way that the right thumb has significance 1 and the left neighbor neighbor is always twice as significant we can get 1024 different numbers. 1+2+4+8+16+32+64+128+256+512=1023 and 0. This is called binary counting and exactly the way a computer counts but with zeros and ones instead of bent and stretched fingers.


turned off
turn off
turn on
turn off
too hot
too cold
heating
idling

Think about the different states of your system

State machines are a concise way of describing systems with clearly distinguishable phases or states. The concept of state machines were developed in the context of computer systems but is now also used by engineers from other disciplines.

A state machine is a system which defines its current status through a set of states. The current state determines what the system is doing. In case of the example, this means whether the heating element is activated or not. The transition between the states happens based on events. These events can be triggered by external factors like a button pressed by the user or a sensor reading but can also be internal like the result of a computation.

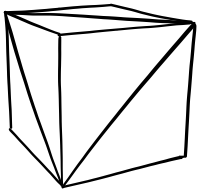

1 0 1 1 0 1 0 0 1 1 0 0 1 0 1 1 0 1

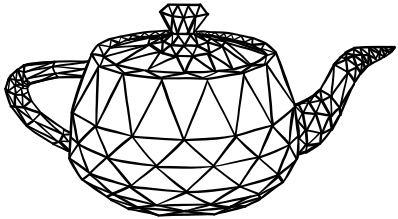
The Essence of Computation: The Turing Machine

The Turing machine is an imaginary model of a machine. It describes what can be computed by following an algorithm. It prints information in a coded form on a tape. This tape is divided into squares. Each of these squares contains either a 0 or a 1.

As a state machine, the Turing machine reads the content of the currently selected square and executes the algorithm associated with its current state. Every possible state of the Turing machine has an algorithm associated. Depending on the algorithm, the machine modifies the content of a square and moves the tape to the left or the right.

The Turing machine is a quite simple concept but it turns out, it contains the essence of computation: Every problem that can be computed, can theoretically also be computed by a Turing machine. This makes the Turing machine an important concept in theoretical computer science.

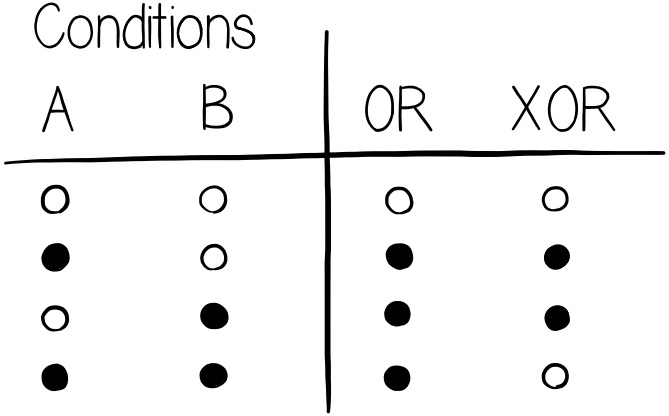




World of triangles

The 3D pictures seen in computer animated films or computer games consist of many small triangles. The shape of a figure in such an image comprises many points. To visualize a surface, a set of those points is connected by lines. The mathematical figure which emerges through this process is called polygon. The smallest polygon possible is the triangle, where only three points are connected. Furthermore, a triangle guarantees that all the points of the polygon are on the same plane.

The more triangles are used, the more detailed a surface is. Whereas a simple object with flat surfaces like a cube can be modeled by using only a few triangles, a more complex structure like a teapot requires many more triangles for representing its round shape.

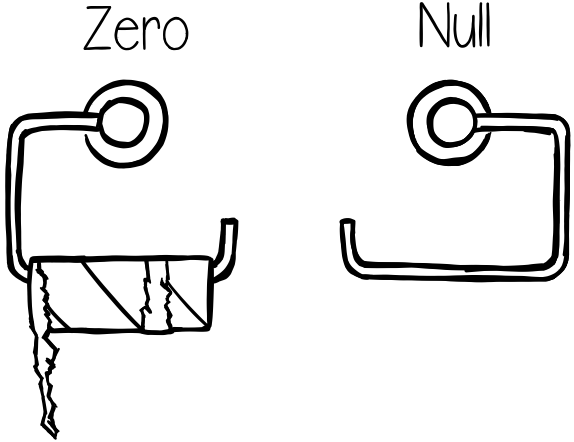

Conditions
A
B
OR
XOR

There are two different kinds of OR

The question whether an “or” can also be interpreted as an “and” is a common source of confusion. While as humans we can often resolve this confusion by looking at the context, computers require a strict definition.

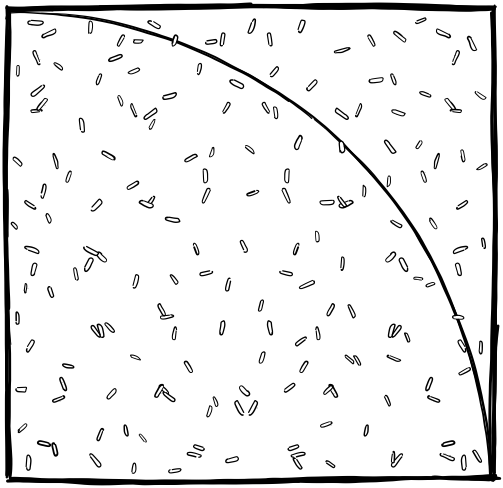
In mathematical logic, the foundation of Computer Science, it is therefore distinguished between “OR” and “exclusive OR”. The latter is often abbreviated as XOR. In English, we can express an XOR by using an “either . . . or . . . ” construction.

The OR operator is always defined as an “and/or”. It is fulfilled if at least one of its conditions is fulfilled. It is therefore also called “inclusive or”. An exclusive or on the other hand is fulfilled if (and only if) exactly one of the conditions is fulfilled. If both of its conditions are met, the exclusive or is no longer fulfilled.


Zero
Null

Zero is not null

The null value has a special meaning in computer science. It is used to declare that a value is not yet defined or known. The specialty of null is that although it shows us that a value is missing, it itself is a value. So, it is possible to compare two nonexistent values. Suppose we have data about a patient; it does make a difference whether the patient has no disease (0) or is not yet diagnosed (null).



How to calculate Pi with rice

Draw a square and a quatercircle in it on a paper and throw a hand full of rice on it. Done!

Well almost done. Now you count how many grains of rice landed inside the square and how many landed inside the quatercircle. The ratio of these two numbers is an approximation of the ratio of the two areas Pi*r$^{2/r}$2=Pi/4. Multiply the ratio by 4 and you get Pi, well an approximation. If you are not happy with the accuracy just use more rice.

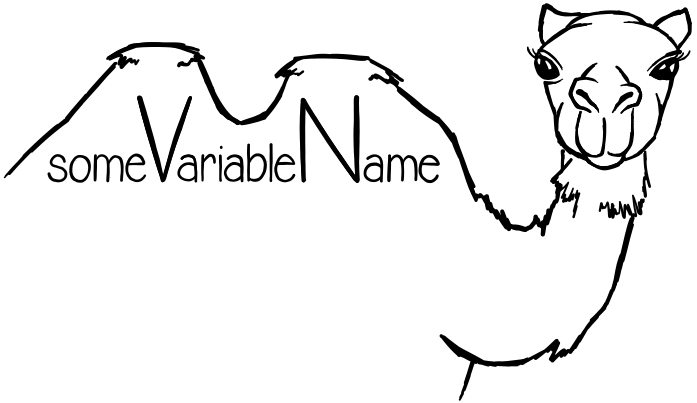
This is one example for the Monte Carlo method. Key is to avoid computing a complicated formula by using random samples in a clever way.


41
DAYS WITHOUT
ACCIDENT
42

Why Uptime Matters

The time since a computer was started is referred to as its **uptime**. When the computer is rebooted (or crashes), its uptime is reset to zero. The time the computer is not running is called **downtime**.

When running a **service**, such as a website, we want it to be available at all times. This means that the **server** the website is running on has to be available all the time. As a high uptime means that a server has been running a long time without interruption, it can indicate that the provided services are also available with few interruptions.

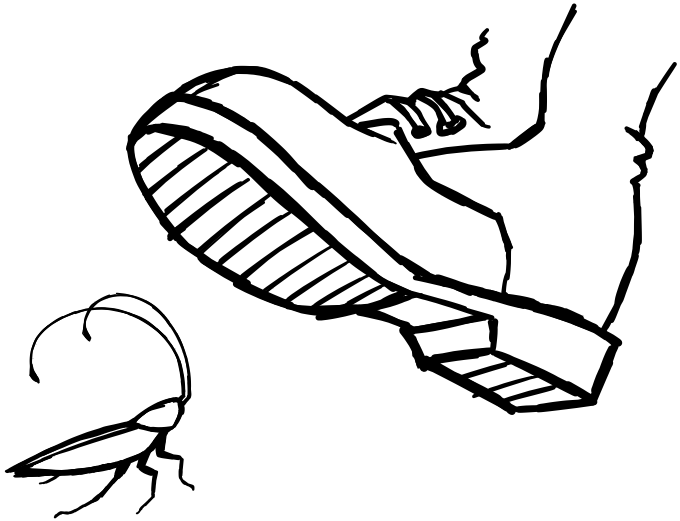

someVariableName

CamelCase, or why coding conventions matter

One problem that often arises when writing code is the naming of variables. Often multi-word names are used for variables but in most programming languages it is not possible to use spaces in a variable name.

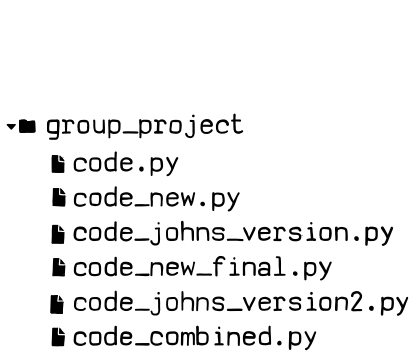
One programming style that tries to structure this is called CamelCase. It is the practice of writing each word in the middle of a phrase with a Capital letter. This greatly improves readability.

```
thisGreatlyImprovesReadabilityComparedTo
    whenwewouldusenocapitalizationatall
```

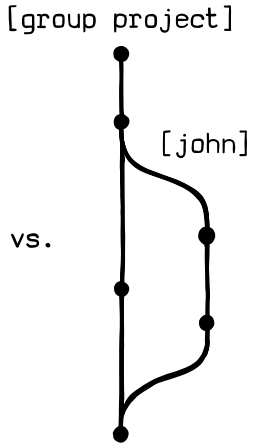


We all hate Bugs

Bugs creep into your house without notice, they eat your food and clog your sink. They hide in corners and behind large furniture and are most of the times hard to find. If you find a bug that was bothering you for a long time the feeling of success is rewarding. A bug in software is very simmilar. It is a behavior of the software which is not intended. It is often very hard to find bugs in software and can sometimes be attributed to one single character that was wrongly placed. The act of finding bugs is called debugging. Like pest control, the code is searched and cleaned from every bug that influences the behavior negatively. Unfortunately, while fixing software bugs, the programmer accidentally adds other bugs. The Cycle is endless.


group_project
code.py
code_new.py
code_johns_version.py
code_new_final.py
code_johns_version2.py
code_combined.py

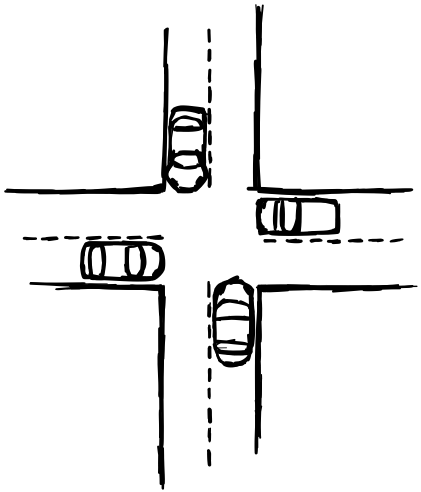
vs.


[group project]
[john]

Always use a Version Control System

It’s no secret that backing up your data is a good idea, and the same goes for your code. However, keeping a lot of slightly renamed versions of your files for every little modification is probably not the way to go.

Version Control Systems (VCS) allow not only to back up your files, but more importantly help you to keep track of the different versions and what changed in each of them. Even better, multiple people can simultaneously work on one file and then easily combine the different changes.

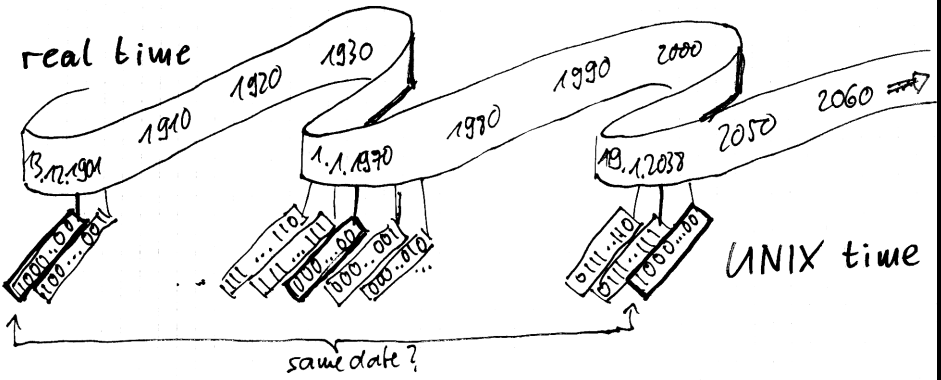


Deadlock

Imagine an uncontrolled intersection without any traffic signs. Here, drivers have to respect the priority to the right system. In the situation shown in the left picture, none of the cars can continue. In practice, this is solved by one driver giving up his priority using a hand-sign.

A similar situation can occur when several computer processes running in parallel want to access shared resources. In a *circular wait* scenario, processes block each other from continuing execution as they wait for a resource held by one of their peers.

Since computer processes cannot hand wave at each other, clear rules are needed to avoid this situation in the first place.


real time
1910
1920
1930
1970
1990
2000
2050
2060
13.12.1901
1.1.1970
19.1.2038
UNIX time
same date?

No Mercy: The Year 2038 Problem

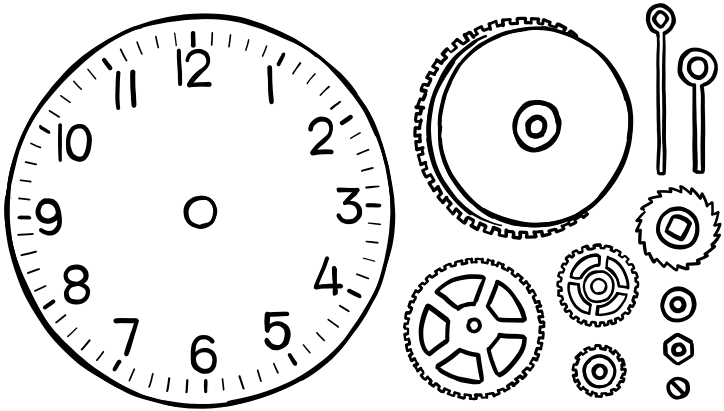
The date of Jan 1, 1970 is to UNIX developpers what year 0 is for Christians, just that in UNIX time you count in seconds, not in days as we do with calendars.

Now here is a problem:

- in UNIX time, two billion seconds *after* Jan 1, 1970 is Jan 19, 2038
- in UNIX time, two billion seconds *before* Jan 1, 1970 is Dec 13, 1901

Many UNIX systems can hold time values only up to four billions, and then values start to “wrap around”. This means that on Jan 19, 2038, these UNIX systems will think it is the year 1901!

The clock is now ticking for finding all these old UNIX systems and to update their software, giving them more bits for storing *UNIX* time. For sure, some systems will be missed and some systems will be too old to be fixed.
Real time will have no mercy . . .


12
1
2
3
4
5
6
7
8
9
10
11

Reverse Engineering

Reverse Engineering describes the process of analyzing an existing apparatus to figure out how it works and how it was made. The same can be done for a computer program, but instead of looking at screws and cogwheels, we are looking at the machine instructions. These are effectively the 0’s and 1’s that tell the computer what to do. This is obviously much harder to understand than the code that was used to generate the program. A skilled reverse engineer can still use this information to gain knowledge about the software. This can be used to detect security flaws, which can then either be reported to the programmers for them to eliminate or it can be exploited maliciously. Additionally, *Reverse Engineering* is used to change old programs to make them compatible with new hardware or to decipher old file formats.

List of topics

0 Hello World
1 Automation
2 Binary system
3 Finite state machine
4 Turing machine
5 Polygon mesh
6 XOR
7 Zero vs Null
8 Monte Carlo method
9 System vs service uptime
10 Camel case
11 Programming bugs
12 Version control systems
13 Deadlock
14 Year 2038 problem
15 Reverse engineering

Seminar participants (spring semester 2019)

Fabricio Arend Torres
Nils Bühlmann
Rebecca Dold
Simon Dold
Omnia Kahla
Patrick Kahr
Marcel Lüthi
Sebastian Philipps
Linard Schwendener
Tim Steindl
Christian Tschudin
Marco Vogt
Moira Zuber

https://tilics.dmi.unibas.ch/