Stuff

Michael Langbein

November 16, 2021

Contents

Preface

I think that the world is in the process of becoming something amazing. Things that I saw in animated series as a kid are becoming reality. I don't just want to be a passive spectator of this process, but rather have my own active part in it, be it ever so small.

Chapter 1

Math

1.1 Logic

1.1.1 Algebra

$\wedge$ and $\vee$ are distributive:

$$(A \wedge B) \vee C \equiv (A \vee C) \wedge (B \vee C)$$

$$(A \vee B) \wedge C \equiv (A \wedge C) \vee (B \wedge C)$$

This is easiest seen by drawing a Venn-diagram.

1.1.2 Quantifiers

$$\exists! x : Q(x) \equiv (\exists x : Q(x)) \wedge (\forall x, y : Q(x) \wedge Q(y) \rightarrow y = x)$$

1.1.3 Inference

Simple if statements:

$$A \rightarrow B \equiv \neg A \vee B$$

$$\neg(A \rightarrow B) \equiv A \wedge \neg B$$

Applying this to function-statements yields:

$$\neg(\forall x : A(x) \rightarrow B(x)) \equiv \exists x : A(x) \wedge B(x)$$

If-and-only-if statements:

$$A \leftrightarrow B \equiv (A \rightarrow B) \wedge (B \rightarrow A) \equiv (A \wedge B) \vee (\neg A \wedge \neg B)$$

$$\neg(A \leftrightarrow B) \equiv (A \wedge \neg B) \vee (B \wedge \neg A) \equiv (\neg A \rightarrow B) \wedge (A \rightarrow \neg B) \equiv A \leftrightarrow \neg B$$

Applying this to function-statements yields:

$$\neg(\forall x : A(x) \leftrightarrow B(x)) \equiv \exists x : (\neg A(x) \rightarrow B(x)) \wedge (A(x) \rightarrow \neg B(x))$$

1.1.4 Consistency and well-definedness

Whenever you receive a set of axioms (like *imagine there was a complex number i*), you should only accept working with them when they don't lead to any logical inconsistencies. However, it is hard to prove that a set of axioms doesn't lead to inconsistencies. Usually, you start with a really basic set (ZFC axioms) and prove that you can construct structures (like for example Dedekind cuts) that have as properties the new axioms you want to establish. Then your axioms are consistent as long as ZFC is consistent.

A concept is well defined when it is a function, that is, when for every input-data there is always at most one output-data.

1.1.5 Experimental and protocol design

This section deals with logic-puzzles.

Multistep processes At every step of your experiment, utilize the full spectrum of your measuring device. This way, each measurement contains the maximal information content. For example, in the "twelve coins" problem, at every one of your three steps the scale should be able to tip left, right or not at all.

1.2 Set theory

1.2.1 Relations

A relation is injective if any y belongs only to one x - if at all.

Definition 1. *Injective (1-1):* $\forall x_1, x_2 \in X : x_1 R y_0 \wedge x_2 R y_0 \rightarrow x_1 = x_2$

A relation is surjective if any y belongs to at least one x.

Definition 2. *Surjective (onto):* $\forall y \in Y : \exists x \in X : xRy$

Note that none of the above definitions means that there is a y for any x. Under both definitions, there can be X's that have none or more than one y.

While these definitions are useful, it is sometimes easier to use the following definitions based on the number of in- or outgoing arrows:

Table 1.1: Types of binary relations

	out	in
≤ 1	function	injective (1-1)
≥ 1	total	surjective (onto)

- function: # out ≤ 1
- injective: # in ≤ 1
- total: # out ≥ 1
- surjective: # in ≥ 1
- bijective: # out = # in = 1

Theorem 1. *Let R be a relation on $A \times B$. Then:* $R : injective \wedge R : function \rightarrow |A| \leq |B|$

Proof. Suppose that R:injective and R:function. Proof that $|A| \leq |B|$

> R:injective, thus: # edges $\leq |B|$
> R:function, thus: $|A| \leq$ # edges
> Thus $|A| \leq$ # edges $\leq |B|$

Thus $|A| \leq |B|$

□

Theorem 2. *Let R be a relation on $A \times B$. Then:* $R : surjective \wedge R : total \rightarrow |A| \geq |B|$

Theorem 3. *Let R be a relation on $A \times B$. Then:* $R : bijective \rightarrow |A| = |B|$

The combination of function and surjectivity is sometimes written as A surj B; the combination of totlity and injectivity as A inj B.

This begs a question: $\neg AsurjB \leftrightarrow AinjB$? No, this doesn't hold. A counterexample would be ... But we can proof that $AsurjB \leftrightarrow BinjA$:

1.2.2 Orders

Definition 3. *Partial order: A relation R on a set S is called a partial order if it is reflexive, antisymmetric and transitive.*

- *reflexive:* $\forall x \in S : xRx$
- *antisymetric:* $\forall x, y \in S : xRy \wedge yRx \rightarrow x = y$

- *transitive:* $\forall x, y, z \in S : xRy \wedge yRz \rightarrow xRz$

Definition 4. *Total order: a relation R on a set S is called a total order if it is a partial order and also comparable*

- *comparable, a.k.a. total:* $\forall a, b \in S : aRb \vee bRa$

Definition 5. *Topological order*

Definition 6. *Closure:*

*A set S and a binary operator * are said to exhibit closure if applying the binary operator to two elements S returns a value which is itself a member of S.*

The closure of a set A is the smallest closed set containing A. Closed sets are closed under arbitrary intersection, so it is also the intersection of all closed sets containing A. Typically, it is just A with all of its accumulation points.

1.2.3 Partitions

1.2.4 Recursion

1.3 Combinatorics

1.3.1 Sums and asymptotics

We begin with a few trivial results. Closed form on the right. The methods we develop for sums will also work for products, since any product can be converted into a sum by taking its logarithm.

$$\sum_{n=0}^{N} n = \frac{N(N+1)}{2}$$

$$\sum_{n=0}^{N} x^n = \frac{1-x^{N+1}}{1-x}$$

We might get to these results with the perturbation method:

- $1 + x + x^2 ... + x^N = S$
- $-x - x^2 ... - x^{N+1} = -xS$
- $1 - x^{N+1} = S - xS$

It is very important to note that this method only works for calculating *finite* partial sums. Nothing guarantees us that such a method would work as well for infinite sums. In fact, we can disprove this.

Theorem 4. *A example where using the perturbation method on infinite series leads to a contradiction would be ...*

However, we are always allowed to calculate a partial sum by some variant of the perturbation method and then taking the limit of $n \to \infty$.

1.3.2 Products of sums and convolutions

Let $S_A = \sum_n a_n$ and $S_B = \sum_m b_n$. To calculate $S_A S_B$ we sum all elements in the following table:

Table 1.2: Multiplication of sums

	a_0	a_1	a_2	a_3
b_0	a_0b_0	a_1b_0	a_2b_0	...
b_1	a_0b_1	a_1b_1	...	
b_2	a_0b_2	...		
b_3	...			

Note that the diagonal from ll to ur in this table is a convolution.

1.3.3 Cardinality

1.3.4 Counting

Permutations are the number of rearrangements where order matters. Safe-combinations should really be called safe permutations. Permutations of unique items are relatively straightforward, but things get a little involved when we deal with permutations of grouped items.

- with repetition (*4-slot lock*): n^4
- without repetition (*first 3 places in race*): $\frac{n!}{(n-3)!}$

The number of unique permutations of a string consisting of n_a a's, n_b b's, n_c c's, etc. is

$$\frac{(\sum n_i)!}{\prod(n_i!)}$$

Here, $(\sum n_i)!$ is the number of permutations if every letter in the string was unique, that is, if we could distinguish between two a's. We then reduce this number by the number of duplicates by dividing through $n_a!$, the number of rearrangements of a's.

Note how the binomial coefficient is a special case of our string-permutation: it is the number of unique permutations in a string of length n with only two letters in it. In fact, the number of string-permutations is called the *multi*nomial coefficient. The formula is easily proven by induction on the number of groups i. In the base-case with two groups we trivially get the binomial coefficient. Then do the induction step.

Combinations are sets where the order does *not* matter: $(2,4,3) = (2,3,4) = (3,2,4) = ...$

- without repetition (*lottery of 6*): $\frac{n!}{(n-6)!6!} = \binom{n}{6}$, aka. the *binomial coefficient*, aka. n choose k. Note the term 6! in the denominator. n choose k is exactly a permutation without repetition (that is, where order matters), reduced by the different ways to order things (6!).
- with repetition (*draw 5 coins out of your pocket*): $\binom{(n-1+5)}{5}$, where n is the number of different coin-denominations. Its pretty cool how this formula is obtained: write all coin-denominations in a list. Under that list, write a program, consisting of arrows for moving right and x'es for picking once. This is the same as picking 5 slots for x'es out of $(n-1+5)$.

1.3.5 Generating functions

ner scenario
ng functions
ndy: finding
a recursive
formula.

In the previous section we have found a bunch of formulas for getting the possible selections in different simple situations. But what if we need to combine a few simple selection-scenarios into one complicated scenario?

Imagine you need to buy n items. There are two kinds of items: apples and bananas. Apples always come in packs of six, and bananas have two different sub-kinds. How many ways are there then to get n items?

This is where we use generating functions.

Algebraically, what's happening is that taking an ordinary generating function is a bijection between the vector space of sequences and the ring of formal power series. While in a vector space we have vector addition and scalar multiplication, in a ring we have element addition and element multiplication (and even division and differentiation!). So working in the powerseries-ring allows for ohter operations than working in the series-vectorspace. This is very simmilar to what we do when doing the Fourier-transform: we dive into a different domain where some operations are easier to do. However, Fourier is merely a change of basis, going from one vector space to another. Here, we move from a vector space to a ring - a whole different universe.

Let $A = (a_0, a_1, ...)$ be a sequence where a_n represents the number of ways to select from group $\mathcal{A}$. Then the generating function G_A would be defined as:

$$G_A(x) = a_0 + a_1 x + a_2 x^2 + ... = \sum_n a_n x^n$$

For any sequence we can define a generating function. Now, to get the desired result, we first transform several sequences into their generating functions, find their closed forms, multiply them, and transform the result of the multiplication back into a sequence.

In detail:

- From $(a_0, a_1, ...)$ create $G_A(x)$ and find its closed form.
- From $(b_0, b_1, ...)$ create $G_B(x)$ and find its closed form.
- Get $G_C = G_A G_B$
- Get $(c_0, c_1, ...)$ from G_C

This works because of the following theorem:

Theorem 5 (Convolutions with generating functions)**.** *Let $A = (a_0, a_1, ...)$ be a sequence where a_n represents the number of ways to select from group $\mathcal{A}$. Let $B = (b_0, b_1, ...)$ be a sequence where b_n represents the number of ways to select from group $\mathcal{B}$. Let $\mathcal{A} \cap \mathcal{B} = \emptyset$. Then c_n is the number of ways to select n items from $\mathcal{A} \cup \mathcal{B}$. It can be obtained by getting the n-th coefficient of $G_C = G_A G_B$*

1.4 General algebra

In the following sections, we will take care to differentiate between the definition of a concept (like of an inner product) and its implementation. The definition of an inner product is based on properties that a thing has to fulfill, whereas the implementation begins with a definition and is then followed by a proof that the properties hold under that definition.

We will mention the following implementations: the vectorspace of coordinate free oriented lengths, the vectorspace $\mathbb{R}^n$, which is the same as oriented length but put inside of a coordinate system(euclidian or angular), the vectorspace of matrices, and $C_{[a,b]}$, the vectorspace of continuous functions.

1.4.1 Vector spaces

Definition 7 (Vector space)**.** *A vector space V is a set closed over two operations: scalar multiplication and vector addition. These two operations must fullfill the following properties:*

- *V is closed under scalar multiplication and vector-addition: $a\vec{v} \in V, \vec{v} + \vec{w} \in V$*
- *vector-addition is commutative: $\vec{v} + \vec{w} = \vec{w} + \vec{v}$*
- *vector-addition is associative: $(\vec{u} + \vec{v}) + \vec{w} = \vec{u} + (\vec{v} + \vec{w})$*
- *$\vec{0}$ is the additive identity: $\vec{v} + \vec{0} = \vec{v}$*
- *$a(b\vec{v}) = (ab)\vec{v}$*
- *Vector-addition and scalar-multiplication are distributive (part 1): $a(\vec{v} + \vec{w}) = a\vec{v} + a\vec{w}$*
- *Vector-addition and scalar-multiplication are distributive (part 2): $(a + b)\vec{v} = a\vec{v} + b\vec{v}$*

The most common vectorspaces are certainly $\mathbb{R}^n$ and functionspaces. The implementations of the above defined scalar product and vector addition are trivial.

Subspaces of vectorspaces A set U is a subspace of a vectorspace V, iff $U \subset V$ and U is a vectorspace.

linear independence The elements in A, a subset of a vectorspace, are linearly independent iff $\forall \alpha_1, ..., \alpha_n : [(\sum \alpha_i \vec{a}_i = 0) \leftrightarrow (\alpha_1 = \alpha_2 = ... = 0)]$. Note that this reduces automatically to $\forall \alpha_1, ..., \alpha_n : [\sum_i^n \alpha_i b_i = 0 \rightarrow (\alpha_1 = ... = \alpha_n = 0)]$, because the $\leftarrow$ case is always true.

Consequently, linear dependence is defined as $B : ld \equiv \exists \alpha_1, ..., \alpha_2 : [(\alpha_1 \neq 0 \vee ... \vee \alpha_n \neq 0) \wedge (\sum_i^n \alpha_i b_i = 0)]$.

Proof. Prove that iff $B : ld$, then one of the elements of B is a linear combination of the others. □

Consider the span of two integers (like in the die-hard water-jug problem). They are always linearly dependent. For example a base $B = [2, 3]$ is linearly dependent because $\frac{-3}{2}2 + 1 \cdot 3 = 0$.

Bases A set B is a base to a vectorspace V iff $B \subseteq V \wedge \forall v \in V : \exists! \alpha_1, ..., \alpha_n : v = \sum_i \alpha_i b_i$. It is easy to prove that this means that $B : baseV \leftrightarrow (B : li \wedge B : spanV)$.

Definition 8 (The dimension of a vectorspace S)**.** *is defined as the size of its base B_S: $dim_S = |B_S|$.*

That means to get a base for $\mathbb{R}^2$, we never need more than two vectors. We'll prove this for the example of $S = \mathbb{R}^2$:

Proof. For any three vectors chosen from $\mathbb{R}^2$, at least one must be a linear combination of the others.
$\vec{a}, \vec{b}, \vec{c} \in \mathbb{R}^2$ Proof that $\vec{a} : ld(\vec{b}, \vec{c}) \vee \vec{b} : ld(\vec{a}, \vec{c}) \vee \vec{c} : ld(\vec{a}, \vec{b})$

> Without loss of generality, assume $\vec{a} : li(\vec{b}, \vec{c})$ and $\vec{b} : li(\vec{a}, \vec{c})$ Proof that $\vec{c} : ld(\vec{a}, \vec{b})$
>
> > $\vec{a}$ and $\vec{b}$ form a base for $\mathbb{R}^2$. That means any $\vec{x} \in \mathbb{R}^2$ is a linear combination of these two ... including $\vec{c}$.
>
> Thus $\vec{c} : ld(\vec{a}, \vec{b})$

Thus $\vec{a} : ld(\vec{b}, \vec{c}) \vee \vec{b} : ld(\vec{a}, \vec{c}) \vee \vec{c} : ld(\vec{a}, \vec{b})$

□

Theorem 6 (Every vectorspace has a basis)**.**

Theorem 7 (All bases of a vectorspace S have the same size)**.**

1.4.2 Inner product spaces

Vector spaces don't define anything about lengths, angles or projections. This lack is alleviated by adding a inner product.

Definition 9 (Inner product space)**.** *The inner product is defined as any operation that has the following properties:*

- $(a\vec{u}) \cdot \vec{v} = a(\vec{u} \cdot \vec{v})$
- $(\vec{u} + \vec{v}) \cdot \vec{w} = \vec{u} \cdot \vec{w} + \vec{v} \cdot \vec{w}$
- $\vec{u} \cdot \vec{v} = \vec{v} \cdot \vec{u}$
- $\vec{v} \neq \vec{0} \rightarrow \vec{v} \cdot \vec{v} > 0$

In inner product spaces, we can define norms, orthogonality, and angles.
As for norms:

$$|\vec{v}|^2 = \vec{n} \cdot \vec{n}$$

And orthogonality:

$$\vec{v} \perp \vec{u} \leftrightarrow \vec{v} \cdot \vec{u} = 0$$

And finally angles:

$$cos\theta = \frac{\vec{v} \cdot \vec{u}}{|\vec{v}||\vec{u}|}$$

As one nice little application, consider this statement.

Proof. Suppose $|\vec{u}| = |\vec{v}|$. Proof that $\vec{u} + \vec{v} \perp \vec{u} - \vec{v}$

> This is to prove that $(\vec{u} + \vec{v}) \cdot (\vec{u} - \vec{v}) = 0$
> The above can be rewritten to $\vec{u} \cdot \vec{u} - \vec{u} \cdot \vec{v} + \vec{v} \cdot \vec{u} - \vec{v} \cdot \vec{v}$
> The two middle terms cancel out, and the two outer terms equal $|\vec{u}|^2$ and $|\vec{v}|^2$, respectively.
> Usign the given, these terms are equal.

Thus $\vec{u} + \vec{v} \perp \vec{u} - \vec{v}$

□

Other properties of the norm are also easily proved:

- $|a\vec{v}| = |a||\vec{v}|$
- $|\vec{v} + \vec{w}| = |\vec{v}| + |\vec{w}|$
- $|\vec{v}| \geq 0$
- $|\vec{v}| = 0 \leftrightarrow \vec{v} = \vec{0}$

Two more important statements that can be proven for the general inner product spaces are the pythagorean theorem and the Cauchy-Schwartz inequality.

Proof. Pythagorean theorem.
Suppose $\vec{u} \perp \vec{v}$. Proof that $|\vec{u}|^2 + |\vec{v}|^2 = |\vec{u} + \vec{v}|^2$

Thus $|\vec{u}|^2 + |\vec{v}|^2 = |\vec{u} + \vec{v}|^2$

□

Proof. Cauchy-Schwartz inequality.
Proof that $|\vec{u}||\vec{v}| \geq |\vec{u} \cdot \vec{v}|$

Thus $|\vec{u}||\vec{v}| \geq |\vec{u} \cdot \vec{v}|$

□

An implementation of this inner product in dimension-free oriented length-space would be:

$$\vec{v} \cdot \vec{w} = |\vec{v}||\vec{w}| cos\theta$$

Based on this definition, the projection of $\vec{v}$ onto $\vec{u}$ is defined as:

$$P_{\vec{u}}(\vec{v}) = |\vec{v}| cos\theta \frac{\vec{u}}{|\vec{u}|} = \frac{\vec{u} \cdot \vec{v}}{|\vec{u}|^2} \vec{u}$$

The direct equivalent of this inner product from oriented length space to $\mathbb{R}^n$ would be:

$$\vec{v} \cdot \vec{w} = \sum_n v_n w_n$$

The following is a proof that the two implementations of inner product are equivalent.

Proof. Suppose $\vec{u} = u_1\vec{e_1} + u_2\vec{e_2} + u_3\vec{e_3}$ and $\vec{v} = v_1\vec{e_1} + v_2\vec{e_2} + v_3\vec{e_3}$. Proof that $\vec{u} \cdot \vec{v} = \sum_n v_n u_n$

$\vec{u} \cdot \vec{v} = (u_1\vec{e_1} + u_2\vec{e_2} + u_3\vec{e_3}) \cdot (v_1\vec{e_1} + v_2\vec{e_2} + v_3\vec{e_3})$

This is written out as $u_1v_1(\vec{e_1} \cdot \vec{e_1}) + u_1v_2(\vec{e_1} \cdot \vec{e_2}) + ...$

Of this, almost all terms cancel out, leaving $u_1v_1 + u_2v_2 + u_3v_3$

Thus $\vec{u} \cdot \vec{v} = \sum_n v_n u_n$

□

Note that if we were to chose a *non*orthonormal basis, the inner product would not be reduced so nicely.

Table 1.3: Important implemantations of vector spaces, inner product spaces and algebras

	Vector space		Inner product space		Algebra
	scalar prod	addition	inner prod	norm	outer prod
$\mathbb{R}^n$			$\sum_n u_i v_i$	$\sqrt{\sum_i v_i^2}$	
$\mathbb{R}^n$ in polar					
$C_{[a,b]}$			$\int_a^b u(t)v(t)\,dt$		
X on Ω			E[XY]	E[X]	
matrices					$(\sum_x \sum_y a_x b_y)_{i,j}$ (aka.linear algebra)
G^3			$\vec{v} \cdot \vec{u}$		$\vec{v} \wedge \vec{u}$ (aka.geometric algebra)
$\mathbb{R}$					(aka. ordinary algebra)
Booleans					(aka. boolean algebra)

A few comments to the different spaces shown here. X on Ω is a inner product space very similar to $C_{[a,b]}$, but note that Ω itself is not neccessarily even a vectorsace.

Here is one more example of an inner product. Consider the vector-space $L^2(\mathbb{R}^2 \to \mathbb{R}^3)$, that is, the space of square-integrable functions that take two input arguments $\vec{x}$ and return a 3-vector $\vec{y}$. How would you define an inner product to turn this into an inner-product-space? Well, the choice is up to you, but most often one choses:

$$< f, g >:= \int_{\mathbb{R}^2} < f(\vec{x}), g(\vec{x}) >_i \, d\vec{x}$$

, where $< \vec{a}, \vec{b} >_i$ is defined as $\sum_{i=0}^{3} a_i b_i$. You can prove for yourself that this is indeed an inner product!

1.4.3 Excursion: More on orthogonality and preview of function decompositions

Orthogonality turns out to be an important concept for statistics and signal analysis, so we'll look at it in a little more detail here. Why is orthogonality so important though? Linear independence allows us to take a complex signal and decompose it into simpler, independent parts. Orthogonality ensures that these parts are easy to calculate.

Although conceptually similar, orthogonality is a stricter concept than linear independence. It requires an inner product space instead of just a vector space. Also, two vectors may be linearly independent, but not orthogonal (allthough we can use Gram-Schmidt orthogonalisation to make any li vectors orthogonal).

Proof. If a set of vectors is orthogonal, then it is linearly independent: $B : orth \rightarrow B : li$.
Suppose that $\forall b_i, b_j \in B : b_i \cdot b_j = 0$ Proof that $\forall \alpha_1, ..., \alpha_n : \sum \alpha_i b_i = 0 \rightarrow \alpha_1 = ... = \alpha_n = 0$

> Let $\alpha_1^0, ..., \alpha_n^0$ be chosen. Suppose $\sum \alpha_i^0 b_i = 0$ Proof that $\alpha_1 = ... = \alpha_n = 0$
>
> > Proof that $\alpha_j^0 = 0$ for any $j \in [1, n]$
> >
> > > $\sum \alpha_i^0 b_i = 0$
> > > Multiplied by b_j:
> > > $\sum \alpha_i^0 b_i \cdot b_j = 0 \cdot b_j$
> > > With $B : orth$:
> > > $\alpha_j^0 = 0$
> >
> > Thus $\alpha_j^0 = 0$ for any $j \in [1, n]$
>
> Thus $\alpha_1 = ... = \alpha_n = 0$

Thus $\forall \alpha_1, ..., \alpha_n : \sum \alpha_i b_i = 0 \rightarrow \alpha_1 = ... = \alpha_n = 0$

□

This is profound. For example, the cos-sin-Fourier-basis is hard to prove to be linearly independent. But we can use the *stricter* property of orthogonality to prove that it must also be linearly independent.

It is good to know that although orthogonality helps us to prove linear independence, it doesn't help us to prove that a set is a base, because for that we also need the set to span the whole space.

Proof. In the infinite dimensional case, a orthogonal set $B \subseteq V$ does not have to be a base of V. A good example would be a set of linear functions in $V = C_{[a,b]}$ - they can never span quadratic functions. □

Fourier decomposition In every vectorspace a vector can be expressed as a sum of the basevectors like this: $v = \alpha_1 b_1 + \alpha_2 b_2 + ...$ If the base is orthonormal, we additionally get the benefit that the coefficients α are very easy to calculate: $\alpha_i = v \cdot b_i$. This way of calculating the coefficients is called the Fourier decomposition.

Proof. Let B be an orthonormal base of V. Then for any $\vec{v} \in V$ the nth coefficient α_n can be easily calculated as $< \vec{v}, \vec{b_n} >$

For any vectorspace it holds that $\forall \vec{v} \in V : \exists \alpha_1, ..., \alpha_N : \sum \alpha_k \vec{b_k} = \vec{v}$.
Suppose B to be orthonormal. Proof that $\alpha_n =< \vec{v}, \vec{b_n} >$

> $< \vec{v}, \vec{b_n} >=< \sum \alpha_k \vec{b_k}, \vec{b_n} >$
>
> $= \alpha_0 < \vec{b_0}, \vec{b_n} > +\alpha_1 < \vec{b_1}, \vec{b_n} > +... + \alpha_n < \vec{b_n}, \vec{b_n} > +... + \alpha_N < \vec{b_N}, \vec{b_n} >$
>
> $= \alpha_0 0 + \alpha_1 0 + ...\alpha_n 1 + ...\alpha_N 0$

Thus $\alpha_n =< \vec{v}, \vec{b_n} >$

□

This is much easier than the case where the base is *not* orthonormal. If that is the case, we have to calulate the coefficients α_n by using the projections:

$$\vec{v} = \sum \alpha_k \vec{b_k}, \text{ with } \alpha_k = \gamma_k P_{\vec{b_k}}(\vec{v}) = \gamma_k \frac{\vec{b_n} \cdot \vec{v}}{|\vec{b_n}|^2} \vec{b_n}, \text{ with } \gamma_k \text{ to be determined.}$$

An alternative, but equally expensive method would be to use linear algebra:

$$\vec{v} = [\vec{b_1}, \vec{b_2}, ..., \vec{b_N}]\vec{\alpha}$$

Calling the matrix $[\vec{b_1}, \vec{b_2}, ..., \vec{b_N}]$ the basematrix B, we get:

$$\vec{v} = B\vec{\alpha}$$

$$B^{-1}\vec{v} = \vec{\alpha}$$

This looks simple enough, but unfortunately, inverting a matrix is a $\Theta(N^3)$ operation, and matrix multiplication is still a $\Theta(N^{>2.3})$ operation. Contrary to that, the evaluation of a plynomal is a $\Theta(N)$ operation when using Horners method.

Gram-Schmidt orthogonalisation For every set of li vectors we can find a set of orthogonal vectors like this:

- $b_1 = v_1$
- $b_2 = v_2 - prj(v_2, b_1) = v_2 - \frac{v_2 \cdot b_1}{b_1 \cdot b_1} b_1$
- $b_3 = v_3 - prj(v_3, b_2) - prj(v_3, b_1)$
- ...

Orthogonal complement

1.5 Linear algebra

In the previous section on general algebra we dealt with vector spaces (subspaces, linear independence, bases) and inner product spaces (norms, orthogonality). Here we'll mostly deal with the inner-product space of vectors and occasionally with the vector-space of matrices (but really, almost exclusively with the former ...).

1.5.1 Spaces

We'll work a lot with a few vector-spaces and transfromations from and to those spaces.

Definition 10. *Let W be a space. Then $V \subseteq W$ is a subspace, if:*

- $\forall v_1, v_2 \in V : v_1 + v_2 \in W$
- $\forall v \in V : \forall r \in R : rv \in V$

Definition 11 (Nullspace)**.** *$\mathcal{N}_A$ is the nullspace of A. It is defined as $\mathcal{N}_A = \{x | Ax = 0\}$*

Definition 12 (Columnspace)**.** *$\mathcal{C}_A$ is the collumnspace of A. It is defined as $\mathcal{C}_A = \{y | Ax = y\}$*

Definition 13 (Rowspace)**.** *$\mathcal{R}_A$ is the rowspace of A. It is defined as $\mathcal{R}_A = \{y | A^T x = y\}$*

Definition 14 (Rank)**.** *r_A is the rank of A. It is defined as the dimension of $\mathcal{C}_A$, $dim_{\mathcal{C}_A}$*

Definition 15 (Nullity)**.** *n_A is the nullity of A. It is defined as the dimension of $\mathcal{N}_A$, $dim_{\mathcal{N}_A}$*

rops and cops as matrix multiplication ...

$$\text{rops}(A) = \text{rops}(I)A = RA$$

$$\text{cops}(A) = A\,\text{cops}(I) = AC$$

rops don't change $\mathcal{N}_A$ Row- and column-operations (rops and cops) seem somewhat trivial at first and not worth any proof writing efforts. However, many theorems of linear algebra are much easier proved if we first reduce the matrices to their RRLE (reduced row linear echelon) form.

Theorem 8. *Row-operations on A don't change $\mathcal{N}_A$.*

Proof. Let $A' = \text{rops}(A) = RA$ Proof that $\mathcal{N}_A = \mathcal{N}_{A'}$

> Let $x_0 : Ax_0 = 0$ Proof that $\exists y_0 : A'y_0 = 0$
>
> > Try $y_0 = x_0$ Proof that $A'y_0 = 0$
> >
> > > $RAy_0 = 0$
> >
> > Thus $A'y_0 = 0$
>
> Thus $\exists y_0 : A'y_0 = 0$
>
> Let $y_0 : A'y_0 = 0$ Proof that $\exists x_0 : Ax_0 = 0$
>
> > This is equivalent to stating that $\exists x_0 : R^{-1}A'x_0 = 0$
> > Try $x_0 = y_0$ Proof that $R^{-1}A'x_0 = 0$
> >
> > > $R^{-1}A'x_0 = 0$
> >
> > Thus $R^{-1}A'x_0 = 0$
>
> Thus $\exists x_0 : Ax_0 = 0$

Thus $\mathcal{N}_A = \mathcal{N}_{A'}$

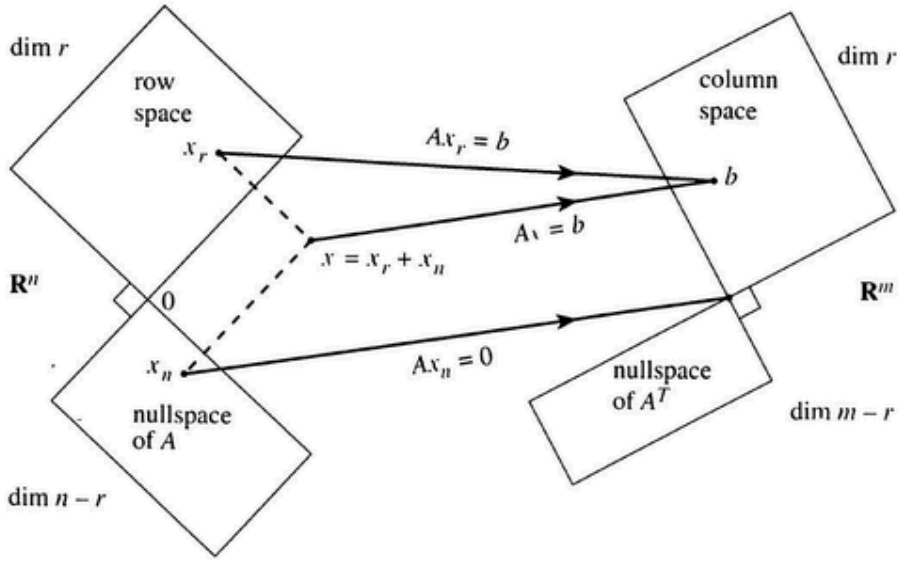
□

Therefore, when searching for the special solutions to a problem $Ab = 0$, we can use Gauss-Elemination and RREF without any problems.

cops don't change C(A) ...

rops don't change C(A) if A is invertible ...

We can now print an overview of the different spaces that are associated with a matrix A of dimension $m \cdot n$ and rank r.



The rowspace of A can be visualized using the line-intersection view of matrix-equations: it contains all the points that lie in the intersection of all the lines that make up the matrix. The columspace of A can be visulaized using the vector-image of A: it contains all the points that are spanned by A. Notice how we included the previous theorem: any combination x of a particular sollution x_r and any vector in the nullspace x_n is also a solution.

We should look in more detail at this graphic. Note, for example, that $\mathcal{N}_A$ and $\mathcal{C}_{A^T}$ seem to be orthogonal. Indeed:

Theorem 9 $(\forall v \in \mathcal{N}_A, w \in \mathcal{C}_{A^T} : v \perp w)$**.**

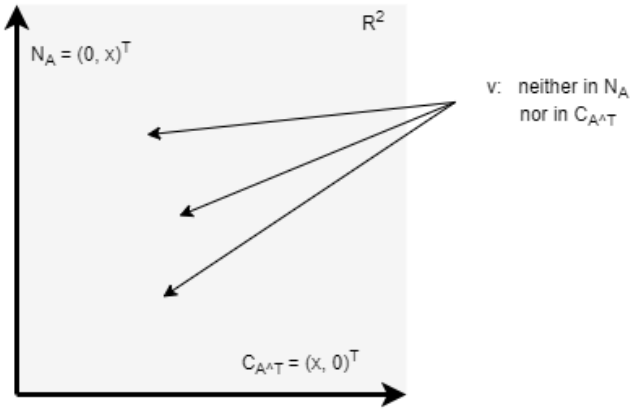
Proof.

$$\begin{aligned} w^T v = 0 \qquad & \text{with } \exists u : \mathbf{A}^T u = w \\ u^T \mathbf{A} v = 0 \qquad & \text{with } \mathbf{A}v = 0 \\ u^T 0 = 0 \qquad & \text{which is trivially true.} \end{aligned} \tag{1.1}$$

Thus, $\mathcal{N}_A$ and $\mathcal{C}_{A^T}$ only intersect in $\vec{0}$. □

However, note that $\mathbb{R}^n \geq \mathcal{N}_A \cup \mathcal{C}_{A^T}$. As an example, consider $\mathbf{A} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$. We get $\mathcal{N}_A = \begin{bmatrix} 0 \\ x \end{bmatrix}$ and $\mathcal{C}_{A^T} = \begin{bmatrix} x \\ 0 \end{bmatrix}$. Now consider $\vec{v} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, which is neither in $\mathcal{N}_A$ nor in $\mathcal{C}_{A^T}$. As illustration, look at fig. 1.5.1.

Figure 1.1: When A is not full rank, there are infinitely many vectors v that are neither in $\mathcal{R}_A$ nor in $\mathcal{N}_A$. When it comes to spatial dimensions, $1 + 1 \neq 2$!



1.5.2 Change of basis

Let V be a vector space. Let 0 be the canonical basis for that vector space. Let $A = \{\vec{a}_1, ..., \vec{a}_N\}$ and $B = \{\vec{b}_1, ..., \vec{b}_N\}$ be two other basis for that vectorspace. Let $\mathbf{A}$ be the matrix $[\vec{a}_1 ... \vec{a}_N]$ and $\mathbf{B} = [\vec{b}_1 ... \vec{b}_N]$

Every vector $\vec{v}$ can be expressed as a linear sum of the basisvectors in A, that is $\vec{v} = \sum_n \alpha_n \vec{a}_n$. That same thing in matrix notation: $\vec{v} = \mathbf{A}(\vec{v})_A$, where $(\vec{v})_A$ is the coordinates α of $\vec{v}$ with respect to the basis A. Correspondingly, for B we have $\vec{v} = \sum_n \beta_n \vec{b}_n = \mathbf{B}(\vec{v})_B$.

Note that within A and B, we express the basevectors with respect to the canonical basis 0, that is, we should really write $\mathbf{A} = [(\vec{a}_1)_0 ... (\vec{a}_N)_0]$. Note also that $[(\vec{a}_1)_A ... (\vec{a}_N)_A] = \mathbf{I}$, the identity matrix.

We can use this to obtain a simple formula for the change of basis.

$$\vec{v} = \mathbf{A}(\vec{v})_A$$

$$\vec{v} = \mathbf{B}(\vec{v})_B$$

$$(\vec{v})_B = \mathbf{B}^{-1}\mathbf{A}(\vec{v})_A$$

But inverses are notoriously hard to calculate. Fortunately, there is another approach. Call $\mathbf{T}_{BA} = [(\vec{a}_1)_B ... (\vec{a}_N)_B]$ the *transition matrix*. We can prove that $\mathbf{B}^{-1}\mathbf{A} = \mathbf{T}_{BA}$:

$$\begin{aligned}
\mathbf{B}^{-1}\mathbf{A}(\vec{v})_A &= \mathbf{T}_{BA}(\vec{v})_A \\
&= \sum_n (v_n)_A (\vec{a}_n)_B \\
&= \sum_n (v_n)_A \mathbf{B}^{-1}\mathbf{A}(\vec{a}_n)_A \\
&= \mathbf{B}^{-1}\mathbf{A} \sum_n (v_n)_A (\vec{a}_n)_A \\
&= \mathbf{B}^{-1}\mathbf{A}\ \mathbf{I}(\vec{v})_A \\
(\vec{v})_A &= (\vec{v})_A
\end{aligned} \tag{1.2}$$

Using $\mathbf{T}_{BA} = \mathbf{B}^{-1}\mathbf{A}$, a lot of statements are trivial to prove:

- $\mathbf{T}_{BA} = \mathbf{T}_{AB}^{-1}$
- $\mathbf{T}_{CA} = \mathbf{T}_{CB}\mathbf{T}_{BA}$

1.5.3 Linear transformations

Definition 16. *Let U and V be two vector spaces and $f : U \to V$. Then f is a* linear transform *if*

- *f preserves scalar multiplication: $f(\alpha\vec{u}) = \alpha f(\vec{u})$*

- *f preserves vector addition:* $f(\vec{u}_1 + \vec{u}_2) = f(\vec{u}_1) + f(\vec{u}_2)$

There are a bunch of properties to linear transformations that can be useful to us.

There is a unique linear transform from the basis of U to any set of vectors in V that we want. In other words, any linear transform f is completely deterimed by the matrix $[f(\vec{b}_1)...f(\vec{b}_N)] = [\vec{v}_1...\vec{v}_N]$. That means if we dont know the transform, but we do know the results of the transform on a basis, then we can reconstruct the transform with certainty.

Let $B = \{\vec{b}_1, ..., \vec{b}_N\}$ be a basis for U. Let $\{\vec{v}_1, ..., \vec{v}_N\}$ be any vectors in V that we may chose. Proof that there is a unique function $f : U \to V$ such that $f(\vec{b}_i) = \vec{v}_i$

Try $f(\vec{x}) = \mathbf{V}(\vec{x})_B$ Proof that $f(\vec{b}_i) = \vec{v}_i$ and f is a linear transform.

Part 1: Proof that $f(\vec{b}_i) = \vec{v}_i$

$f(\vec{b}_i) = \mathbf{V}(\vec{b}_i)_B = \mathbf{V}\vec{e}_i = \vec{v}_i$

Thus $f(\vec{b}_i) = \vec{v}_i$

Part 2: Proof that f is unique

We could not have obtained any other form of f than $f(\vec{x}) = \mathbf{V}(\vec{x})_B$. This is because for *any* linear transform from $U \to V$ we have:
$f(\vec{x}) = f(\mathbf{B}(\vec{x})_B) = f(\sum_n (x_n)_B \vec{b}_n) = \sum_n (x_n)_B f(\vec{b}_n)$
Using the result from part 1, this cannot be any other function than:
$\sum_n (x_n)_B f(\vec{b}_n) = \sum_n (x_n)_B \vec{v}_n = \mathbf{V}(\vec{x})_B$

Thus f is unique

Part 3: Proof that f is a linear tansform

Thus f is a linear tansform

Thus $f(\vec{b}_i) = \vec{v}_i$ and f is a linear transform.

Thus there is a unique function $f : U \to V$ such that $f(\vec{b}_i) = \vec{v}_i$

□

A whole bunch of other properties are now easily proved. Let f and g be linear transforms from U to V. The following are also linear transforms:

- αf
- $f + g$
- f^{-1} (if it exists)
- fg (here $g : V \to W$)

Let $f : U \to V$ be a linear transform. Then the following are equivalent:

- If $f(\vec{u}) = \vec{0}$, then $\vec{u} = \vec{0}$
- f is one-to-one
- f maps linearly independent vectors to linearly independent vectors.

Prove that a transform can be split up into mulitple transforms on the basis vectors. As an examle, consider the case of a rotation. A diagonal rotation can be reproduced by a rotation first around one, then around another axis.

A linear transformation can also be a change of basis when it is on a vectorspace and invertible.

1.5.4 Linear independence

Definition 17. *B is linearly independent if $\forall b \in B : b \neq \sum r_n b_n$, with $b_n \in B/b$.*

Theorem 10. *A better definition could be stated as such: B is linearly independent, if the only solution to $Bx = 0$ is the zero-vector.*

Proof. Suppose $\forall \vec{b} \in \mathbf{B} : \vec{b} \neq \sum_{B/b} \alpha_i \vec{b_i}$ Proof that $\mathbf{B}\vec{x} = \vec{0} \rightarrow \vec{x} = \vec{0}$

> Suppose $\mathbf{B}\vec{x} = \vec{0}$ Proof that $\vec{x} = \vec{0}$
>
> > By contradiction. Assume $\vec{x} \neq \vec{0}$ Proof that there is a contradiction.
> >
> > > We have $\mathbf{B}\vec{x} = \vec{0}$
> > > Consider the vector $\vec{b_3}$.
> > > Thus: $\sum_{B/b_3} x_i \vec{b_i} = -x_3 \vec{b_3}$
> > > Or: $\sum_{B/b_3} \frac{-x_i}{x_3} \vec{b_i} = -x_3 \vec{b_3}$
> > > This contradicts the statement that $\forall \vec{b} \in \mathbf{B} : \vec{b} \neq \sum_{B/b} \alpha_i \vec{b_i}$
> >
> > Thus there is a contradiction.
>
> Thus $\vec{x} = \vec{0}$

Thus $\mathbf{B}\vec{x} = \vec{0} \rightarrow \vec{x} = \vec{0}$
Suppose $\mathbf{B}\vec{x} = \vec{0} \rightarrow \vec{x} = \vec{0}$ Proof that $\forall \vec{b} \in \mathbf{B} : \vec{b} \neq \sum_{B/b} \alpha_i \vec{b_i}$

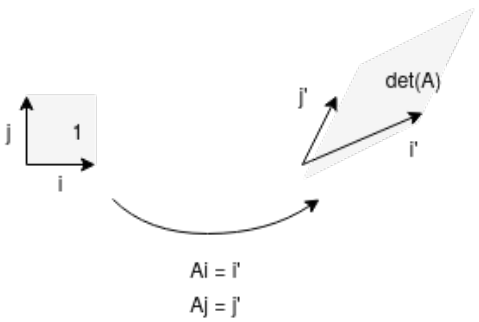
> Let $\vec{b_0} \in \mathbf{B}$ Proof that $\vec{b_0} \neq \sum_{B/b_0} \alpha_i \vec{b_i}$
>
> > By contradiction. Assume $\vec{b_0} = \sum_{B/b_0} \alpha_i \vec{b_i}$ Proof that there is a contradiction.
> >
> > > Since $\vec{b_0} = \sum_{B/b_0} \alpha_i \vec{b_i}$
> > > we get $\vec{0} = \sum_{B} \alpha_i \vec{b_i}$, where $\alpha_0 = -1$
> > > In matrix notation: $\vec{0} = \mathbf{B}\vec{\alpha}$, with a non-zero $\vec{\alpha}$.
> > > This contradicts our assumption that $\mathbf{B}\vec{x} = \vec{0} \rightarrow \vec{x} = \vec{0}$
> >
> > Thus there is a contradiction.
>
> Thus $\vec{b_0} \neq \sum_{B/b_0} \alpha_i \vec{b_i}$

Thus $\forall \vec{b} \in \mathbf{B} : \vec{b} \neq \sum_{B/b} \alpha_i \vec{b_i}$

□

Definition 18. *Let V be a subspace. $B \subseteq V$ is a basis for V if B is linearly independent and $\forall v \in V : v = \sum r_n b_n$, with $b_n \in B$*

1.5.5 Determinant



Definition Consider a 2×2 matrix $\mathbf{A}$

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

$\det \mathbf{A}$ is defined as $a_{11}a_{22} - a_{12}a_{21}$. The definitions for higher order matrix-determinants are similar, but so extremely tedious that we just ignore them here.

Theorem 11. *Let* $\mathbf{A}\vec{x} = 0$ *and* $\vec{x} \neq \vec{0}$. *Then it must hold that* $\det \mathbf{A} = 0$. *In fact, this holds in two directions:*

$$\exists \vec{x} \neq \vec{0} : \mathbf{A}\vec{x} = 0 \leftrightarrow \det \mathbf{A} = 0$$

Proof. We can prove this making use of theorem 10. From the given it follows that $\mathbf{A}$ is linearly dependent. Thus $\exists \vec{a} \in \mathbf{A} : \vec{a} = \sum_{A/a} \alpha_i \vec{a}_i$. In the case of a 2×2 matrix this expression becomes very simple: $\vec{a}_1 = \alpha \vec{a}_2$. Breaking $\vec{a}_1$ into its components

- $a_{11} = \alpha a_{12}$
- $a_{21} = \alpha a_{22}$

Putting this into the definition of a determinant we get:

$$\begin{aligned} \det \mathbf{A} &= a_{11}a_{22} - a_{12}a_{21} \\ &= \alpha a_{12}a_{22} - \alpha a_{12}a_{22} \\ &= 0 \end{aligned} \tag{1.3}$$

For higher order matrices, the proof follows from induction or something like that. □

Interpretation The size of the determinant can be seen as the scaling-factor of the transformation described by the matrix A. It is also a measure of how much linearly independent the rows/cols of A are. The size of the determinant equals the size of the (hyper-)parallelogram spanned by the columns. If two vectors are almost linearly dependent, they will be almost parallel, leading to a very small area of the parallelogram. So if you have a small determinant, your columns are almost dependent. If you have a large one, your columns are very orthogonal. If your determinant is zero, that means that your matrix A is a transformation Ax that squishes (at least) one of the dimensions of x into nothing.

Properties

- $\det \mathbf{AB} = \det \mathbf{A} \det \mathbf{B}$
- $\det \mathbf{A} + \mathbf{B} \neq \det \mathbf{A} + \det \mathbf{B}$
- $\det \mathbf{A} = \prod \lambda_i$ (see eivenvalues, later)

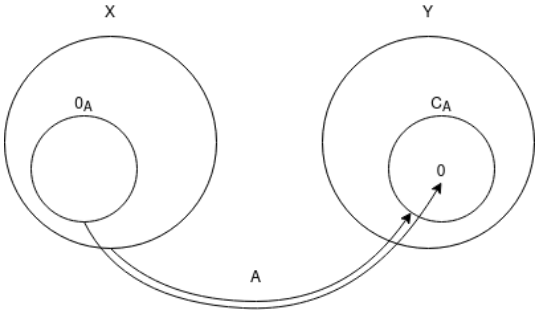
1.5.6 Rank nullity Theorem

Theorem 12. $\mathcal{N}_A = \{0\} \rightarrow \det A \neq 0$

Proof. This is almost self-proving. $\mathcal{N}_A = \{0\}$ is equivalent to writing $\mathbf{A}\vec{x} = \vec{0} \rightarrow \vec{x} = \vec{0}$. By theorem 10 this means that $\mathbf{A}$ is linearly independent. And by theorem 11 this means that $\det A \neq 0$. □

Theorem 13. *Let* $A : X \rightarrow Y$. *Then:*

$$\mathcal{N}_A = \{0\} \rightarrow \mathcal{C}_A = Y$$



Theorem 14. *Rank nullity: this is a fundamental theorem of linear algebra. Let A be of dimension $r \times c$ and full rank. Then:*

$$r_A + n_A = c$$

Theorem 15.

$$r_A = dim_{\mathcal{C}_A} = dim_{\mathcal{R}_A}$$

Theorem 16.

$$A : basis_{\mathcal{C}_A} \rightarrow r_A = n$$

1.5.7 Special matrices

Symmetric matrices are simple but very useful.

Definition 19 (A matrix $\mathbf{A}$ is symmetric)**.** *iff* $\mathbf{A} = \mathbf{A}^T$

Positive (semi-)definite matrix are a special case of symmetric matrices.

Definition 20 ($\mathbf{A}$: PSD $\leftrightarrow$ $\mathbf{A}$: sym $\wedge \forall z : z^T\mathbf{A}z \geq 0$)**.** *From the first condition,* $\mathbf{A}$ *: sym, it follows that the eigenvectors are orthogonal (see eq. 20), from the second one,* $\forall z : z^T\mathbf{A}z \geq 0$*, it follows that the eigenvalues are all greater or equal than* 0 *(see eq. . . .).*

Gram matrices appear often in important theorems. They turn out to be PSD.

Definition 21 (A Gram matrix)**.** *is a matrix* $\mathbf{A} = \mathbf{B}^T\mathbf{B}$

Theorem 17 ($\text{rank}_A = \text{rank}_{A^TA}$)**.**

Theorem 18 (A^TA : PSD)**.**

Orthogonal matrices make tons of calculations simpler.

Definition 22 (A matrix $\mathbf{A}$ is orthogonal)**.** *iff* $\forall x_1, x_2 \in \mathbf{A}, x_1 \neq x_2 : x_1 \perp x_2$

Theorem 19 (If A is orthogonal, than $A^{-1} = A^T$.)**.** *This is trivially proven. By the definition of orthogonality we have* $\mathbf{A}^T\mathbf{A} = \mathbf{I}$

1.5.8 Eigenvalues and eigenvectors

$$\begin{aligned} Ae &= \lambda e \\ (A - \lambda I)e &= 0 \\ \det A - \lambda I &= 0 \text{ using theorem 11} \end{aligned} \tag{1.4}$$

Any *square* matrix can be eigenvector (aka. spectrally) decomposed: $\mathbf{A} = \mathbf{E}\mathbf{\Lambda}\mathbf{E}^{-1}$.

Once at a sprint-discussion I wondered why principal components (see about those later) were all orthogonal. Here's the proof why. (Note that the proof only holds for symmetric matrices - which, in PCA, C_A is indeed symmetric.)

Theorem 20. $\mathbf{A}$: *sym* $\rightarrow \mathbf{E}_A : \perp$

For any matrix A, symmetric or not, and for any vectors x and y, eigenvectors or not, it holds: $\forall A, \forall x, y : < Ax, y > = < x, A^Ty >$ *This is true because*

$$\begin{aligned} < x, y > &= x^Ty \\ < Ax, y > &= (Ax)^Ty \\ &= x^TA^Ty \\ < x, A^Ty > &= x^TA^Ty \end{aligned} \tag{1.5}$$

Then we can use that finding as follows for $\mathbf{A}$ *: sym and* $x, y \in \mathbf{E}_A$*:*

$$\begin{aligned} < Ax, y > &=< x, A^T y > \\ \textit{Since } A : \textit{sym} : < Ax, y > &=< x, Ay > \\ \textit{Since } x, y \textit{ are eigenvalues of } A : < \lambda_x x, y > &=< x, \lambda_y y > \\ \lambda_x < x, y > &= \lambda_y < x, y > \end{aligned} \tag{1.6}$$

Thus, either $\lambda_x = \lambda_y$ *or* $x \perp y$

Theorem 21 (As a corollary, the eigenvalue decomposition can be simplified)**.** *:* $\mathbf{A}$ *:* *sym* $\rightarrow \mathbf{A} = \mathbf{X\Lambda X}^{-1} = \mathbf{X\Lambda X}^T$

Interpretation of eigenvalues

- λ is complex: $\mathbf{A}$ contains a rotation
- λ is negative: $\mathbf{A}$ contains some mirroring
- λ is 0: $\det \mathbf{A} = 0$, i.e. $\mathbf{A}$ squishes a dimension

1.5.9 Singular value decomposition

Let $\mathbf{A}$ be of size $m \times n$. We will prove that also such non-square matrices can be decomposed, too, namely into:

$$\mathbf{A} = \mathbf{U\Sigma V}^T \tag{1.7}$$

where:

- $\mathbf{U}$ and $\mathbf{V}$ are orthonormals of sizes $m \times m$ and $n \times n$, respectively
- $\mathbf{\Sigma}$ is a diagonal matrix of size $m \times n$

In other words:

$$\mathbf{AV} = \mathbf{U\Sigma} \tag{1.8}$$

We'll find an orthonormal basis $\mathbf{V} \subset R_A$ which is transformed into an orthonormal basis $\mathbf{U} \subset C_A$.

We'll work with a practical example alongside the proof. Let $\mathbf{A} = \begin{bmatrix} 2 & 0 \\ 0 & 1 \\ 0 & 2 \end{bmatrix}$.

Consider $\mathbf{A}^T\mathbf{A}$. Contrary to $\mathbf{A}$, this is a psd matrix. It has eigenvectors $[\vec{e}_1, \vec{e}_2, ...] = \mathbf{E}_{A^T A} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ and eigenvalues 4 and 5.

Now consider the vector $\vec{y}_i = \mathbf{A}\vec{e}_i$. Multiplying both sides with λ_i we observe that:

$$\begin{aligned} \lambda_i \vec{y}_i &= \lambda_i \mathbf{A}\vec{e}_i \\ &= \mathbf{A}\lambda_i \vec{e}_i \\ &= \mathbf{A}\mathbf{A}^T\mathbf{A}\vec{e}_i \\ &= \mathbf{A}\mathbf{A}^T\vec{y}_i \end{aligned} \tag{1.9}$$

In other words: $\vec{y}_i$ is an eigenvector of $\mathbf{AA}^T$ with eigenvalue λ_i.

Indeed: $\mathbf{AA}^T = \begin{bmatrix} 4 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 2 & 4 \end{bmatrix}$ and $\mathbf{AA}^T \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix} = 5 \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}$.

A note on the size of $\vec{y}_i$: $|\vec{y}_i| = |\mathbf{A}\vec{e}_i| = \sqrt{|\mathbf{A}\vec{e}_i|^2} = \sqrt{\vec{e}_i^T \mathbf{A}^T \mathbf{A}\vec{e}_i} = \sqrt{\lambda_i}|\vec{e}_i|$. If we want $\vec{y}_i$ to have normalized size we redefine it as $\vec{y}_i = \frac{1}{\sqrt{\lambda_i}}\mathbf{A}\vec{e}_i$

Also, if $\lambda_i = 0$ then $\vec{y}_i = \vec{0}$.

Now, let $k = \text{rank}_{\mathbf{A}}$ and define the following new matrices:

$$\begin{aligned} \hat{\mathbf{U}} &= [\vec{y}_i / \text{ where } \lambda_i = 0], \text{ size } m \times k &&= \begin{bmatrix} 0 & 1 \\ \frac{1}{\sqrt{5}} & 0 \\ \frac{2}{\sqrt{5}} & 0 \end{bmatrix} \\ \hat{\mathbf{V}} &= [\vec{e}_i / \text{ where } \lambda_i = 0], \text{ size } b \times k &&= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{aligned} \tag{1.10}$$

Consider $\hat{\mathbf{U}}^T \mathbf{A} \hat{\mathbf{V}} \vec{b}_i$, where $\vec{b}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \vec{b}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$:

$$\begin{aligned} \hat{\mathbf{U}}^T \mathbf{A} \hat{\mathbf{V}} \vec{b}_i &= \hat{\mathbf{U}}^T \mathbf{A} \vec{e}_i \\ &= \frac{1}{\lambda_i} \hat{\mathbf{U}}^T \mathbf{A} \lambda_i \vec{e}_i \\ &= \frac{1}{\lambda_i} \hat{\mathbf{U}}^T \mathbf{A} \mathbf{A}^T \mathbf{A} \vec{e}_i \\ &= \frac{1}{\lambda_i} \hat{\mathbf{U}}^T \mathbf{A} \mathbf{A}^T \vec{y}_i \sqrt{\lambda_i} \\ &= \frac{1}{\sqrt{\lambda_i}} \hat{\mathbf{U}}^T \mathbf{A} \mathbf{A}^T \vec{y}_i \\ &= \frac{1}{\sqrt{\lambda_i}} \hat{\mathbf{U}}^T \lambda_i \vec{y}_i \\ &= \sqrt{\lambda_i} \hat{\mathbf{U}}^T \vec{y}_i, \text{ now, making use of } \hat{\mathbf{U}} \vec{b}_i = \vec{y}_i \\ &= \sqrt{\lambda_i} \vec{b}_i \end{aligned} \tag{1.11}$$

Generally:

$$\begin{aligned} \hat{\mathbf{U}}^T \mathbf{A} \hat{\mathbf{V}} \mathbf{I} &= \sqrt{\lambda_i} I \\ \hat{\mathbf{U}}^T \mathbf{A} \hat{\mathbf{V}} &= \mathbf{\Sigma} \end{aligned} \tag{1.12}$$

where in our case $\mathbf{\Sigma} = \begin{bmatrix} 0 & \sqrt{5} \\ \sqrt{4} & 0 \end{bmatrix}$

We'll now expand

- $k \times m$ matrix $\hat{\mathbf{U}}$ to $m \times m$ matrix $\mathbf{U} = [\vec{y}_i]$, even where $\lambda_i = 0$
- $k \times n$ matrix $\hat{\mathbf{V}}$ to $n \times n$ matrix $\mathbf{V} = [\vec{e}_i]$, even where $\lambda_i = 0$
- $k \times k$ matrix $\mathbf{\Sigma}$ to $m \times n$ matrix $\mathbf{\Sigma} = \begin{cases} \text{if } i = j \wedge i < k : \sqrt{\lambda_i} \\ \text{else } 0 \end{cases}$

With that we obtain

$$\mathbf{A} = \mathbf{U} \mathbf{\Sigma} \mathbf{V}^T \tag{1.13}$$

In our example case we have

$$\mathbf{\Sigma} = \begin{bmatrix} \sqrt{5} & 0 \\ 0 & 0 \\ 0 & \sqrt{4} \end{bmatrix}$$

Example for a shear-matrix : SVD means that any matrix-transformation can be rewritten as a rotation, scale and another rotation. Even a shear-matrix!

```
M = [
    [1, .5],
    [0, 1]
]

U, S, VT = np.linalg.svd(M)

U @ (np.eye(2) * S) @ VT
```

SVD is often used for image compression, too. Just leave out some lower σ_i's, then you can also leave out equally many columns / rows of U and V^T, respectively.

1.5.9.1 Principal component analysis

Consider a $m \times n$ matrix $\mathbf{A}$. Let $\mathbf{X} = \mathbf{A} - \mu$, the matrix with its column-mean subtracted. Then we can express the covariance of $\mathbf{X}$ as $\mathbf{C}_X = \mathbf{X}^T\mathbf{X}$.

We want to find a transformation $\mathbf{P}$ such that $\mathbf{Y} = \mathbf{XP}$ is uncorrelated. We'll prove that such a matrix does exist.

$$\exists \mathbf{P} : \mathbf{Y} = \mathbf{XP} \wedge \mathbf{C}_Y : \text{diag} \tag{1.14}$$

Try $\mathbf{P} = \mathbf{E}_{C_X}$. Then:

$$\begin{aligned} \mathbf{C}_Y &= \mathbf{Y}^T\mathbf{Y} \\ &= (\mathbf{XP})^T\mathbf{XP} \\ &= \mathbf{P}^T\mathbf{X}^T\mathbf{XP} \\ &= \mathbf{P}^T C_{\mathbf{X}}\mathbf{P} \\ &= \mathbf{E}_{C_X}^T C_X \mathbf{E}_{C_X} \end{aligned} \tag{1.15}$$

Thus $C_Y = \Lambda_{C_X}$, which is diagonal, as required.

Note: Sometimes we care not about $cov(n_1, n_2)$ but about $cov(m_1, m_2)$. Then we use: $\mathbf{X} = \mathbf{A} - \mu$, the matrix with its *row*-mean subtracted. With that $C_X = \mathbf{XX}^T$ and $\exists \mathbf{P} : Y = \mathbf{PX}$. Try $\mathbf{E}_{C_X}$

Example: Eigenfaces are a very fun example of PCA.

```
import numpy as np
import matplotlib.pyplot as plt
import imageio
import os


(X, Y, S) = (119, 95, 50)
P = X * Y

allFaces = np.zeros((P, S))
for s, path in enumerate(paths[:S]):
    image = imageio.imread(path)
    allFaces[: , s] = image.reshape((P))

meanFace = np.mean(allFaces, axis=1, keepdims=True)
allFacesNorm = allFaces - meanFace
cov = allFacesNorm @ allFacesNorm.transpose()
evals, evecs = np.linalg.eigh(cov, k=cutoff)
plt.imshow(np.abs(evecs[:, 0].reshape((X, Y))))

cutoff = 100
evecsC = evecs[:, -cutoff:]


face = allFaces[:, 45]
faceEncoded = face @ evecsC
faceReconstr = meanFace[0, :] + (evecsC @ faceEncoded)
```

Figure 1.2: First eigenfaces

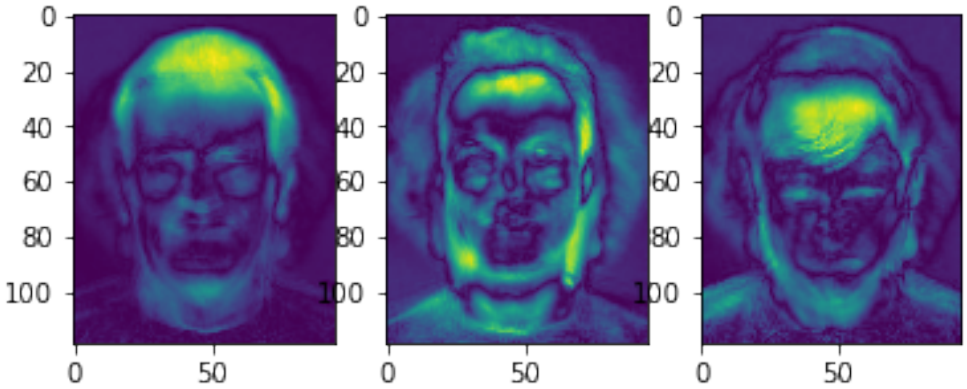
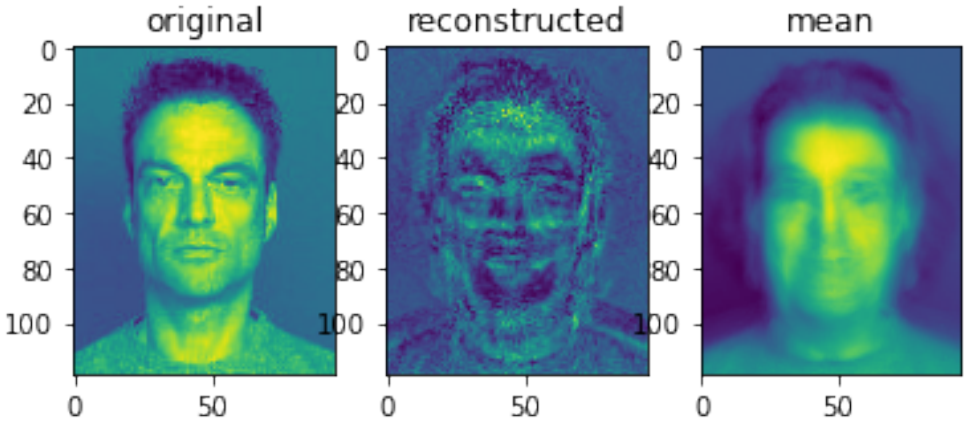


Figure 1.3: Reconstructing a faces from its percentages of the eigenfaces



1.5.10 Inverse

If A has dimensions $n \times n$, then the inverse A^{-1} such that

$$AA^{-1} = A^{-1}A = I$$

exists iff $det_A \neq 0$.

Nonsqare matrices do not have an inverse, but they might have a right- or left-inverse. Consider $C = AA^T$. This is a square matrix, so it might have an inverse:

$$AA^T(AA^T)^{-1} = I$$

Calling $A^T(AA^T)^{-1} = A^{RI}$ the right inverse:

$$AA^{RI} = I$$

For C^{-1} to exist, we require C to be full-rank, which means that A must be full row rank. This requires $r \leq c$, in other words, A being a broad matrix.

Proof. If A^{RI} exists, then A^{LI} does not □

Proof. A^{RI} is not unique. Let A be of dimension $r \times c$ and full rank, with $r < c$. By rank nullity we have

$$n_A = c - r > 0$$

That is, the nullspace is nonempty. Thus $\exists x : Ax = 0$ Now try $B' = B + x$ We get

$$AB' = AB + Ax = I + 0$$

□

1.5.11 Applications

1.5.11.1 Systems of linear equations

If there are more variables than equations, the system is underdetermined. If there are more eqations than variables, the system is overdetermined. A potentially solveable system is one where there are equally many variables as equations. But even then we must distinguish two cases.

Solving well determined systems There are two cases: a system is either consistent or inconsistent. The follwing statements are all equivalent, meaning that any one of them is related to any other one in an if-and-only-if way.

- the system is consistent
- the matrix is invertible
- the determinant is nonzero
- there is exactly one sollution

We proof a few of those equivalences just for the hell of it.

Proof. There is exactly one solution if and only if the determinant is nonzero.
Proof that $|A| = 0 \leftrightarrow \exists! \vec{x} : A\vec{x} = \vec{b}$
|
Thus $|A| = 0 \leftrightarrow \exists! \vec{x} : A\vec{x} = \vec{b}$

□

On the other hand, in the inconsistent case, the follwing statements are equivalent:

- the system is inconsistent
- the matrix is singular (aka. noninvertible)
- the determinant equals zero
- one (or more) row (or column) is lineraly dependent of the others

Solving overdetermined systems: least squares

Solving underdetermined systems: geometric bodies I like to think of underdetermined systems as (linear) geometric bodies, written in their parameterized form. A line in $\mathbb{R}^3$ is described by a 3×1 matrix (or rather, it's column space), a plane in $\mathbb{R}^3$ by a 3×2 matrix. However, it is important to note one distinction: geometric objects don't need to go through the origin, a matrix system however does. A line that does not go through the origin needs a base vector, like so:

$$\vec{x} = \left[\vec{d}\right] [\alpha] + \vec{b}$$

A plane that does not go through the origin also needs a base vector $\vec{b}$, like so:

$$\vec{x} = \begin{bmatrix} \vec{p}_1 & \vec{p}_2 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} + \vec{b}$$

Here is a problem that bothered me for a while: a line needs one parameter, a plane two. An ellipsoid, too, needs two parameters. Are there any linear geometric objects in $\mathbb{R}^3$ that require more than three parameters? The answer is: no. Here is the proof.

Proof. For any 3×4 matrix, there is a 3×3 matrix that has the same column space, that is, that describes the same geometrical body.

Proof that $\forall A(3 \times 4) \exists A'(3 \times 3) : \mathcal{C}_A = \mathcal{C}_{A'}$
| Let $A_0(3 \times 4)$. Proof that $\exists A' : \mathcal{C}_{A_0} = \mathcal{C}_{A'}$
| | We know from 1.4.1 that $\exists \vec{a}_0 \in A_0 : \vec{a}_0 : ld.$ So Try $A' = A_0/\vec{a_0}$.
| | Indeed, now A_0 and A' both form a base of the same space. So they must have the same column space.
| Thus $\exists A' : \mathcal{C}_{A_0} = \mathcal{C}_{A'}$
|
Thus $\forall A(3 \times 4) \exists A'(3 \times 3) : \mathcal{C}_A = \mathcal{C}_{A'}$

□

However, there are *non*linear objects in $\mathbb{R}^3$ that require more than three parameters! Many curves in 3d require many parameters. *But* those curves don't form a vector-space, while lines and planes do (as long as they go through the origin).

Table 1.4: Influence of rank on solutions

	m <n	m = n	m >n
r <m	n - m free variables m – r conditions on $b \in \mathcal{C}_A$		
r = m (full row rank)	1 pivot per row n – r = n – m free variables 0 conditions on $b \in \mathcal{C}_A$		X
r <n			m – r conditions on $b \in \mathcal{C}_A$ n – r free variables
r = n (full column rank)	X		1 pivot per column m – r = m – n conditions on $b \in \mathcal{C}_A$ 0 free variables, thus $\mathcal{N}_A = \{0\}$

Summary

Ax = b reduces to A'x' = 0

Theorem 22. *A problem of the form $Ax = b$ can be re-expressed as $A'x' = 0$, where $A' = [A, -b]$*

Proof. Proof that $Ax = b$ can be re-expressed as $A'x' = 0$

$Ax = b$
$Ax - b = 0$
$A_1x_1 + A_2x_2 + ... - b = 0$
Let $A' = [A, b]$ and $x_{n+1} = -1$. Then:
$A'x' = 0$
Now we can use the nullspace of A' to find the solutionspace of A.
$\mathcal{N}_{[Ab]} = \{x' | [Ab]x' = 0\}$
$= \{x' | Ax'_{1:m} = -bx'_{m+1}\}$
A subset of that nullspace equals the solutionset for $Ax = b$:
$\mathcal{N}_{[Ab]}$ where $[x'_{m+1} = -1] = \{x' | Ax'_{1:m} = b\}$

Thus $Ax = b$ can be re-expressed as $A'x' = 0$

□

Solving Ax = b

Theorem 23. *If we can only find any one particular solution x_p such that $Ax_p = b$, then we get the whole solutionspace as $\mathcal{N}_A + x_p$.*

Proof. Let $x_p : Ax_p = b$. Proof that $\mathcal{S}_{Ax=b} = \mathcal{N}_A + x_p$

We'll make use of the fact that $\mathcal{N}_A + x_p = \{x + x_p | Ax = 0\} = \{x | Ax = Ax_p\}$
Let $x_0 \in \mathcal{N}_A + x_p$. Proof that $x_0 \in \mathcal{S}_{Ax=b}$, i.o.w. $Ax_0 = b$

$x_0 \in \mathcal{N}_A + x_p$
$x_0 \in \{x + x_p | Ax = 0\}$
$x_0 \in \{x | A(x - x_p) = 0\}$
Thus $Ax_0 = Ax_p$
Since $Ax_p = b$, it must be that $Ax_0 = b$.

Thus $x_0 \in \mathcal{S}_{Ax=b}$, i.o.w. $Ax_0 = b$
Let $x_0 \in \mathcal{S}_{Ax=b}$. Proof that $x_0 \in \mathcal{N}_A + x_p$, i.o.w. $Ax_0 = Ax_p$

> > Because $x_0 \in \mathcal{S}_{Ax=b}$, we have $Ax_0 = b$.
> > Also, it was given that $Ax_p = b$.
>
> Thus $x_0 \in \mathcal{N}_A + x_p$, i.o.w. $Ax_0 = Ax_p$

Thus $\mathcal{S}_{Ax=b} = \mathcal{N}_A + x_p$

□

1.5.11.2 Matrix factorisation

Eigenvalue decomposition

$$A = V\Lambda V^{-1}$$

Singular value decomposition is eigenvalue decomposition, generalized to non-square matrices.

$$A = U\Sigma V^T$$

Nonnegative matrix factorisation : Consider a dataset A, mapping people (rows) to properties (columns). You are looking for some hidden, small set of features, that groups of people have in common. In neural networks we can reconstruct images from a minimal amount of hidden features by funnelling the image through a very small hidden layer out to a large output layer. We can do the very same thing here!

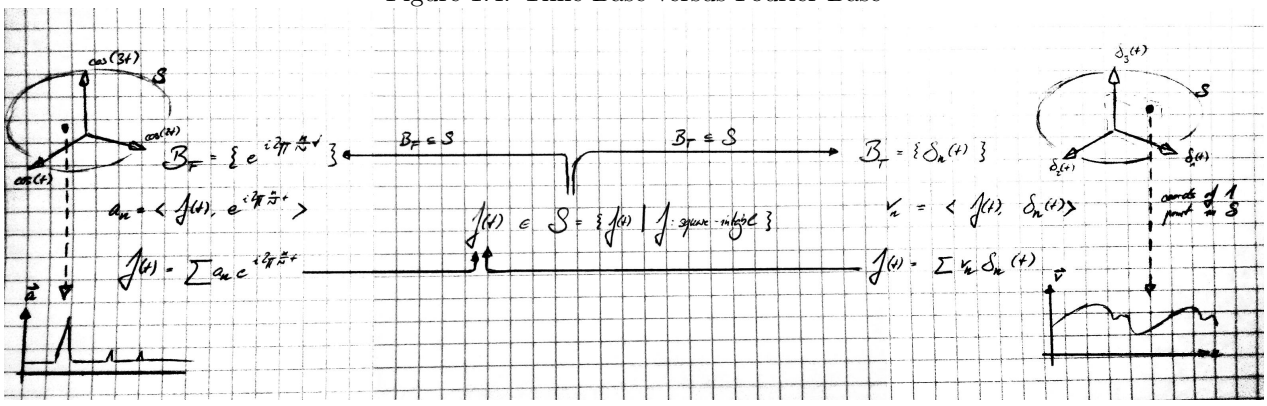
$$A = UV$$

Where A has dimension $r \times c$, U associates people with their hidden features/groups $r \times f$ and V associates features/groups with the properties $f \times c$.

1.6 Fourier and Laplace

Some qualitative intuition before we begin. The Fourier transform scans a signal for sinusoidal terms. The Laplace transform does the same, but *also* scans the signal for exponential terms. That's why the latter is so popular in differential equations: many differential equations have solutions that contain both sinusoidal and exponential terms.

Figure 1.4: Time Base versus Fourier Base



1.6.1 Discrete Fourier Analysis

We'll begin with the discrete case. For one, this case is the basis of the FFT. Also, the math is a lot easier in the discrete case. Let N be the sample-size and δ be the sample-frequency (often equal to 44100 Hz or 48000 Hz). Let V be the sample-space of signals of size N. This is a vector-space, equipped with an inner product of the form $<\vec{a},\vec{b}>=\frac{1}{N}\sum_{k=0}^{N-1} a_k(b_k)^*$, where $()^*$ is the complex complement.

1.6.1.1 The Fourier Base

We'll use the base $B=\{\vec{w}_N^n | n\in[0,N-1]\}$. Here, $\vec{w}_N^n$ is the vector we get by evaluating $e^{i2\pi\frac{n}{N}k}$ at the points $k=0, k=1, ..., k=N-1$. Thus if we call $w_N=e^{i2\pi\frac{1}{N}}$, then the vector $\vec{w}_N^n$ consists of the elements w_N^{nk}. Throughout this chapter, k will be the index of time/the signal-vector and n will be the index of frequency/the base-vectors.

In the case of $N=4$ we obtain the matrix:

$$\begin{bmatrix} e^{0.25i\pi} & e^{0.5i\pi} & e^{0.75i\pi} & e^{1.0i\pi} \\ e^{0.5i\pi} & e^{1.0i\pi} & e^{1.5i\pi} & e^{2.0i\pi} \\ e^{0.75i\pi} & e^{1.5i\pi} & e^{2.25i\pi} & e^{3.0i\pi} \\ e^{1.0i\pi} & e^{2.0i\pi} & e^{3.0i\pi} & e^{4.0i\pi} \end{bmatrix}$$

Note that Fourier-bases are symmetric matrices.

The Fourier-base is a very particular choice: each element of each base-vector turns out to be one of the N complex Nth roots of one. We could have chosen any base-vectors, but these complex roots will turn out to have a few useful properties that we will exploit to speed up the Fourier transformation. These properties are:

1. $w_N^x=(w_N)^x$
2. $w_N^{2x}=w_{N/2}^x$
3. $w_N^{2x+N}=w_N^{2x}$
4. $w_N^{x+N/2}=-w_N^x$

It is also easy to prove that this base is orthogonal.

Proof. The vectors $\vec{w}_N^n$ are orthogonal.
Suppose $n\neq m$. Proof that $<\vec{w}_N^n,\vec{w}_N^m>=0$

$< \vec{w}_N^n, \vec{w}_N^m >= \frac{1}{N} \sum_{k=0}^{N-1} e^{i2\pi \frac{n-m}{N} k}$

Using the result $\sum_{n=0}^{N-1} e^{xn} = \frac{1-e^{xN}}{1-e^x}$, we find:

$\frac{1}{N} \sum_{k=0}^{N-1} e^{i2\pi \frac{n-m}{N} k} = \frac{1}{N} \frac{1-e^{i2\pi(n-m)}}{1-e^{i2\pi \frac{n-m}{N}}}$

It is easy to see that $e^{i2\pi(n-m)} = 1$. So the above term must equal 0.

Thus $< \vec{w}_N^n, \vec{w}_N^m >= 0$
Suppose $n = m$. Proof that $< \vec{w}_N^n, \vec{w}_N^m >= 1$

We use the same line of reasoning as above to obtain:

$< \vec{w}_N^n, \vec{w}_N^m >= \frac{1}{N} \frac{1-e^{i2\pi(0)}}{1-e^{i2\pi \frac{0}{N}}}$

In the limit, this equals 1.

Thus $< \vec{w}_N^n, \vec{w}_N^m >= 1$

□

Proof of spanning

1.6.1.2 Obtaining the amplitudes

After having proven that B is an orthonormal base for V, we can get to the core of the Fourier analysis: given a signal $\vec{v}$, how do we obtain the amplitudes in $\vec{v} = \sum_{n=0}^{N-1} \alpha_n \vec{w}_n$? Well, from paragraph 1.4.3 we know that

$$\alpha_n =< \vec{v}, \vec{w}_N^n >= \frac{1}{N} \sum_{k=0}^{N-1} v_k e^{i2\pi \frac{n}{N} k}$$

Using Horner's method, this is a $\Theta(n)$ operation. Executing it on all n coefficients, the whole process becomes a $\Theta(n^2)$ operation. A naive implementation might look like this:

```
import numpy as np

def amplitudes(signal):
    N = len(signal)
    amps = []
    for n in range(N):
        sm = 0
        for k in range(N):
            wnk = np.exp(-1j * 2 * np.pi * n * k / N)
            sm += signal[k] * wnk
        amps.insert(n, sm/N)
    return amps


sig = [2, 1, 1, 4]
print amplitudes(sig)
```

Notice that this is a matrix operation.

1.6.2 As a matrix operation

Proof. The Fourier transform of $\vec{a} + \vec{b}$ equals the transform of $\vec{a}$ plus the transform of $\vec{b}$
Note that the Fourier transform can be rewritten in matrix form.

$$< \vec{a}, \vec{f_m} > (m) = \begin{bmatrix} f_{1,1} & \dots & f_{F,T} \\ \dots & & \dots \\ f_{F,1} & \dots & f_{1,T} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \dots \\ a_T \end{bmatrix} = \begin{bmatrix} < \vec{a}, \vec{f_1} > \\ < \vec{a}, \vec{f_2} > \\ \dots \\ < \vec{a}, \vec{f_F} > \end{bmatrix}$$

This means that the rules for matrix multiplication apply to the Fourier transformation, amongst which that matrix multiplication is distributive:

$$A(\vec{a} + \vec{b}) = A\vec{a} + A\vec{b}$$

Remember that a matrix-multiplication is always a change of basis: the vector $\vec{v}_I$ (as displayed in the Euclidean system **I**) can also be expressed relative to the Fourier-base **F** as $\vec{v}_F$.

$$\mathbf{F}^{-1}\vec{v}_I = \vec{v}_F$$

where (according to 1.5.2):

$$\mathbf{F}^{-1} = \begin{bmatrix} \vec{x}_F & \vec{y}_F & \vec{z}_F \end{bmatrix}$$

□

1.6.3 Fast Fourier Transform

In the previous section we have evaluated the polynomial $\sum_{k=0}^{N-1} v_k e^{i2\pi\frac{n}{N}k}$ like this:

```
for k in range(N):
    wnk = np.exp(1j * 2 * np.pi * n * k / N)
    sm += signal[k] * wnk
```

Really, this is just the same as evaluating the polynomial $A(x) = \sum_{k=0}^{N-1} v_k x^k$, where $x = e^{i2\pi\frac{n}{N}1} = w_N^n$ (using property 1). In other words, we calculated $A(w_N^n)$.

However, it turns out that we can simplify this process. Before we detail how exactly this section is to be simplified, we need to realize that for any polynomial $A(x)$, it holds that $A(x) = A_{even}(x^2) + xA_{odd}(x^2)$

Knowing that, we can split the evaluation in half:

$$A(w_N^n) = A_{even}(w_N^{2n}) + w_N^n A_{odd}(w_N^{2n})$$

And using the properties 2-4 of w_N^n, we can calculate:

$$\begin{aligned} A(w_N^n) &= A_{even}(w_N^{2n}) + w_N^n A_{odd}(w_N^{2n}) \\ &= A_{even}(w_{N/2}^n) + w_N^n A_{odd}(w_{N/2}^n) \end{aligned} \tag{1.16}$$

$$\begin{aligned} A(w_N^{n+N/2}) &= A_{even}(w_N^{2n+N}) + w_N^{n+N/2} A_{odd}(w_N^{2n+N}) \\ &= A_{even}(w_N^{2n}) - w_N^n A_{odd}(w_N^{2n}) \\ &= A_{even}(w_{N/2}^n) - w_N^n A_{odd}(w_{N/2}^n) \end{aligned} \tag{1.17}$$

This way, we obtain:

```
def fft(signal):
    N = len(signal)

    if N == 1:
        return signal

    sigE = []
    sigU = []
    for k in range(N):
        if k%2 == 0:
            sigE.append(signal[k])
        else:
            sigU.append(signal[k])

    ampsE = fft(sigE)
    ampsU = fft(sigU)

    amps = []
    for n in range(N/2):
        wn = np.exp(-1j * 2 * np.pi * n / N)
        an   = ampsE[n] + wn * ampsU[n]
        an2N = ampsE[n] - wn * ampsU[n]
        amps.insert(n,       an   )
        amps.insert(n + N/2, an2N )

    return amps
```

1.6.3.1 Backtransformation

1.6.4 Fourier for musical frequencies

Musical notes are special. Here, we don't want to get the full spectrum that FFT delivers, but only a few selected frequencies.

1.6.4.1 Goertzel

Goertzel continues to work with the Fourier base. It differs from FFT in that only a few of the elements that FFT returns are actually evaluated.

1.6.4.2 PCI

PCI (pre-calculated inverse algorithm) discards of the Fourier base and instead uses the musical frequencies in the base. This results in a non-orthogonal base, but that won't hinder us.

$$\begin{aligned} \vec{s} &= \sum_F a_f e^{-2\pi i f t} \text{ , with F = \{440, 465, ...\}} \\ &= \begin{bmatrix} ... & & ... \\ & e^{-2\pi i f t} & \\ ... & & ... \end{bmatrix} \vec{a} \\ \vec{a} &= \begin{bmatrix} ... & & ... \\ & e^{-2\pi i f t} & \\ ... & & ... \end{bmatrix}^{-1} \vec{s} \end{aligned} \tag{1.18}$$

1.6.5 Continuous Fourier Analysis

In Fourier analysis, we deal with the inner-product space $C^2_{[a,b]}$: the space of square-integrable functions that are continuous from a to b. An orthogonal base for this vector-space would be the Fourier base $\{sin(\frac{j2\pi}{b-a}t), cos(\frac{j2\pi}{b-a}t) | j \in \mathbb{N}\}$. Often, making use of Euler's formula, we instead write this base as $\{e^{i\frac{n2\pi}{b-a}t} | j \in \mathbb{N}\}$.

1.6.5.1 Comparing Fourier to other important series

Fourier versus Taylor Another famous expansion of functions is the Taylor-expansion. It is important to note that the Taylor expansion is a completely different beast from the Fourier expansion.

- Taylor works on locally differentiable functions, whereas Fourier works on globally integrable functions. You cannot recover the full function from its Taylor-expansion.
- The Taylor-base is non-orthogonal[1]

Table 1.5: Fourier versus Taylor

	coefficients	base-vectors
Fourier	$f(t) \cdot e^{i\frac{n2\pi}{b-a}t}$	$e^{i\frac{n2\pi}{b-a}t}$
Taylor	$f^{(n)}(t)\|t_0$	$\frac{(x-x_0)^n}{n!}$

Fourier versus PCA PCA comes a lot closer to Fourier in that in PCA we represent our data as a linear combination of orthogonal base-vectors There are two differences though. In PCA the data is usually a matrix instead of a vector. And in PCA we don't know in advance what the set of base-vectors is going to be, but much rather chose the most fitting one for our data-matrix. It turns out that we can use the set of eigenvectors of the data-matrix[2].

Proof. There is a set of eigenvectors E of a matrix M that is an orthogonal basis for $\mathcal{S}_M$ Proof that

[1] proof required

[2] Strictly speaking, we don't work with the raw data-matrix, but rather its correlation matrix. This is because for the eigenvector thing to work , we need the matrix to be symmetric and centered around the origin, which the raw data matrix usually isn't.

|

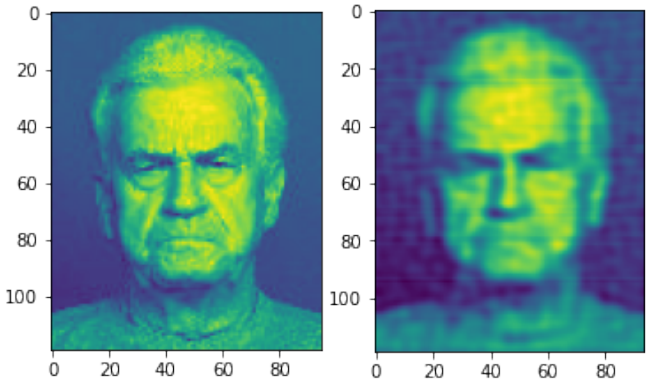
Thus

□

Fourier analysis is often a lot better at compressing. The below image has been created with only 321 frequencies out of a 119*95 image.

```
def compressImage(image, threshold):
    freqs = fft.rfft2(image)
    mags = np.abs(freqs)
    thresholdValue = threshold * np.max(mags)
    indices = mags >= thresholdValue
    freqsFiltered = freqs * indices
    imageRestored = fft.irfft2(freqsFiltered)

    print("entries above threshold: " + str(np.count_nonzero(indices)))
    return imageRestored

image = imageio.imread(files[0])
restored = compressImage(image, 0.003)

fix, axes = plt.subplots(1, 2)
axes[0].imshow(image)
axes[1].imshow(restored)
```

Figure 1.5: Image restored from top 321 frequencies after Fourier transform



1.6.5.2 Proving that Fourier functions form a basis

We can use Fourier to describe any function-space - if we can prove that the Fourier functions do indeed form a basis for that space. This takes two steps: proving orthogonality and proving span.

Orthogonality This is very straightforward:

Span This requires more work. We're first going to have to prove The Weierstrass-Theorem. This theorem states that the set $B = \{x^j | j \in \mathbb{N}\}$ forms a basis for $C_{[a,b]}$. https://psychedai.wordpress.com/2016/11/15/the-intuition-behind-bernsteins-proof-of-the-weierstrass-approximation-theorem/

1.6.5.3 Fourier transform on vector valued functions

As long as a vector-valued function can be decomposed over a set of basis-vectors (never mind what basis vectors exactly, this works with any basis), we can still use the normal, scalar Fourier transform on them.

$$
\begin{aligned}
g &: A^N \to A^M \\
g(\vec{x}) &= \sum_m^M g_m(\vec{x})\vec{e}_m \text{ ... decomposing the function over the output basis} \\
\mathscr{F}_g(\vec{u}) &= \sum_m^M \vec{e}_m \mathscr{F}_{g_m}(\vec{u}) \text{ ... Fourier transform over the input basis}
\end{aligned}
\tag{1.19}
$$

		1D	2D
continuous	Transform	$\mathscr{F}_f(u) = \int_{-\infty}^{\infty} f(t)e^{-2\pi itu}\,dt$	$\mathscr{F}_f(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)e^{-2\pi i(ux+vy)}\,dx\,dy$
	Convolution	$(f*g)(t) = \int_{-\infty}^{\infty} f(u)g(t-u)\,du$	$(f*g)(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(u,v)g(x-u,y-v)\,du\,dv$
discrete	Transform	$\mathscr{F}_f(u) = \sum_{-\infty}^{\infty} f(t)e^{-2\pi itu}$	$\mathscr{F}_f(u,v) = \sum_{-\infty}^{\infty}\sum_{-\infty}^{\infty} f(x,y)e^{-2\pi i(ux+vy)}$
	Convolution	$(f*g)(t) = \sum -\infty^{\infty} f(u)g(t-u)$	$(f*g)(x,y) = \sum_{-\infty}^{\infty}\sum_{-\infty}^{\infty} f(u,v)g(x-u,y-v)$

Consider the example of a rgb-image.

$$\begin{aligned} c(x,y) &= \vec{e}_1 r(x,y) + \vec{e}_2 g(x,y) + \vec{e}_3 b(x,y) \\ \mathscr{F}_c\begin{bmatrix} f_x \\ f_y \end{bmatrix} &= \int_X \int_Y c(x,y)e^{-2\pi i(f_x x + f_y y)}\,dy\,dx \\ &= \int_X \int_Y \vec{e}_1 r(x,y)e^{-2\pi i(f_x x + f_y y)}\,dy\,dx + \ldots \\ &= \vec{e}_1 \mathscr{F}_r\begin{bmatrix} f_x \\ f_y \end{bmatrix} + \vec{e}_2 \mathscr{F}_g\begin{bmatrix} f_x \\ f_y \end{bmatrix} + \vec{e}_3 \mathscr{F}_b\begin{bmatrix} f_x \\ f_y \end{bmatrix} \end{aligned} \tag{1.20}$$

Note that the resulting fourier spectrum consist of complex 3d-vectors.

Sometimes, however, you either have an input function that contains *multi-vector* elements, or you want to apply a filter to the fourier spectrum based on multi-vectors. In that case, you can find an introduction to geometric-algebra Fourier transforms here: https://pdfs.semanticscholar.org/41ce/67428ee60748a4142dee0eea28ed997855e6.pdf

1.6.6 Fourier Transforms and Convolutions

One major reason that Fourier transforms are so important in image processing is the convolution theorem which states that

If $f(x)$ and $g(x)$ are two functions with Fourier transforms $F(u)$ and $G(u)$, then the Fourier transform of the convolution $f(x) * g(x)$ is simply the product of the Fourier transforms of the two functions, $F(u)G(u)$.

Thus in principle we can undo a convolution. e.g. to compensate for a less than ideal image capture system:

- Take the Fourier transform of the imperfect image,
- Take the Fourier transform of the function describing the effect of the system,
- Divide the former by the latter to obtain the Fourier transform of the ideal image.
- Inverse Fourier transform to recover the ideal image.

This process is sometimes referred to as de-convolution.

Motion blur, for example, can be simulated by convoluting an original image x with a simple line-shaped kernel g, yielding the blurred image y. Thus we have:

$$\begin{aligned} y &= x * g \\ Y &= XG \\ X &= Y/G \end{aligned} \tag{1.21}$$

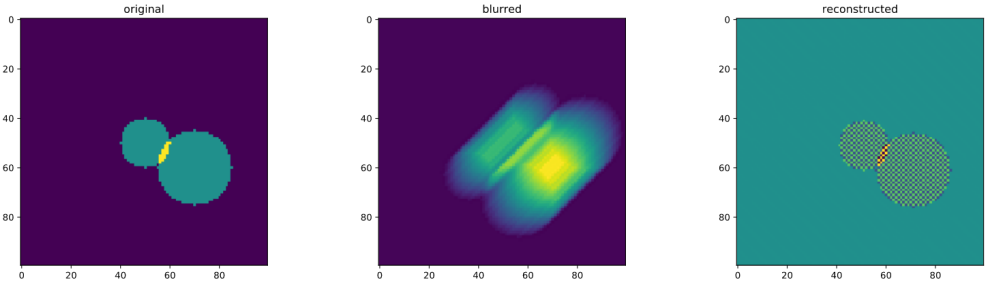
Note that the division in the last line is point-wise (i.e. not a matrix division, which would require being able to compute an inverse).

We can implement this in python.

```
def motionBlurKernel(degrees, n):

    # rotated line
    c = int((n-1)/2)
    kernel = np.zeros((n, n))
    kernel[c, :] = np.ones(n)
    kernel = scn.rotate(kernel, degrees)

    # crop
    N, _ = kernel.shape
    delta = int((N - n) / 2)
```

```
    if delta > 0:
        kernel = kernel[delta:-delta, delta:-delta]

    # normalize
    kernel = kernel / np.sum(kernel)
    return kernel


def centerPad(kernel, n):
    l, _ = kernel.shape
    m = np.zeros((n, n))
    delta = int((n - l) / 2)
    m[delta:delta+l, delta:delta+l] = kernel
    return m


def filterMotionBlur(image, angle, width):
    L, _ = image.shape
    kernel = motionBlurKernel(angle, width)
    kernel_p = centerPad(kernel, L)
    F_image = scf.fftshift( np.fft.fft2(image) )
    F_kernel = scf.fftshift( np.fft.fft2(kernel_p) )
    F_reconstr = F_image / F_kernel
    reconstr = scf.ifftshift( np.fft.ifft2(F_reconstr) )
    return reconstr



# image
point = pointMtrx(100, 100, 50, 50, 10)
point += pointMtrx(100, 100, 60, 70, 15)
noiseK = motionBlurKernel(45, 40)
image = scs.convolve2d(point, noiseK, 'same')

# reconstruction
pointReconstr = filterMotionBlur(image, 45, 40)

# plotting
fig, axes = plt.subplots(1, 3, figsize=(20, 5))
axes[0].imshow(point)
axes[0].set_title('original')
axes[1].imshow(image)
axes[1].set_title('blurred')
axes[2].imshow(np.real(pointReconstr))
axes[2].set_title('reconstructed')
```

Figure 1.6: Reconstructing an original image



1.6.6.1 Important transformations

Both Fourier and Laplace make a few operations much simpler. They turn differentiations and exponentials into algebraic terms, and convolutions into products. As such they are very popular in differential equations, because you can transform a differential equation to Fourier/Laplace domain, solve for the desired function algebraically instead of having to solve the actual differential equation, and then transform back. Laplace is more popular in the differential equation world because it works for non-stable systems, too, whereas Fourier assumes that the system never explodes at any point.

Figure 1.7: Important Fourier transformations

Function	Its Fourier transform
$f(t)$	$F(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(t) e^{-i\omega t}\,\mathrm{d}t$
e^{-at^2}	$\frac{e^{-\omega^2/(4a)}}{\sqrt{2a}}$
$\theta(t)e^{-at}$	$\frac{1}{\sqrt{2\pi}(a+i\omega)}$ if $a > 0$
$\delta(t)$	$\frac{1}{\sqrt{2\pi}}$
$f(t) + g(t)$	$F(\omega) + G(\omega)$
$cf(t)$	$cF(\omega)$
$f(t-a)$	$e^{-ia\omega}F(\omega)$
$f(t)e^{iat}$	$F(\omega - a)$
$f^{(n)}(t)$	$(i\omega)^n F(\omega)$
$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\mathrm{d}\tau$	$\sqrt{2\pi}F(\omega)G(\omega)$

Figure 1.8: Important Laplace transformations

Function	Its Laplace transform
$f(t)$	$F(s) = \int_0^{\infty} f(t)e^{-st}\,\mathrm{d}t$
c	c/s
ct^n	$cn!/s^{n+1}$
$\sin(bt)$	$b/(s^2+b^2)$
$\cos(bt)$	$s/(s^2+b^2)$
e^{at}	$1/(s-a)$
$t^n e^{at}$	$n!/(s-a)^{n+1}$
$\sinh(at)$	$a/(s^2-a^2)$
$\cosh(at)$	$s/(s^2-a^2)$
$e^{at}\sin(bt)$	$b/[(s-a)^2+b^2]$
$e^{at}\cos(bt)$	$(s-a)/[(s-a)^2+b^2]$
$t^{1/2}$	$\frac{1}{2}(\pi/s^3)^{1/2}$
$t^{-1/2}$	$(\pi/s)^{1/2}$
$\delta(t-t_0)$	$e^{-t_0 s}$
$\theta(t-t_0)$	$e^{-t_0 s}/s$
$f(t) + g(t)$	$F(s) + G(s)$
$cf(t)$	$cF(s)$
$f'(t)$	$sF(s) - f(0)$
$f^{(n)}(t)$	$s^n F(s) - s^{n-1}f(0) - s^{n-2}f'(0) - \ldots - f^{(n-1)}(0)$
$tf(t)$	$-F'(s)$
$e^{at}f(t)$	$F(s-a)$
$f(t-a)\theta(t-a)$	$e^{-as}F(s)$ if $a > 0$
$(f * g)(t) = \int_0^t f(\tau)g(t-\tau)\mathrm{d}\tau$	$F(s)G(s)$

1.7 Probability

1.7.1 Basics

1.7.1.1 Probability space

Probability works on some basic entites:

- Ω is a nonempty set called the samplespace.
- $\omega \in \Omega$ is called an outcome
- $E \subseteq \Omega$ is called an event

Definition 23 (Probability). *Probability is a measure on* Ω *. It is a total function* $\Pr : \Omega \to \mathbb{R}$ *such that:*

- $\forall \omega \in \Omega : \Pr[\omega] \geq 0$
- $\sum_{\omega \in \Omega} \Pr[\omega] = 1$

A probability measure together with a samplespace is called a probability space.
We define the probability of an event as:

$$\Pr[E] = \sum_{\omega \in E} \Pr[\omega]$$

Definition 24 (Random variable). *A random variable is a function mapping a* ω *from* Ω *to the reals.*

$$X(\omega) : \Omega \to \mathbb{R}$$

Note that a random variable strictly takes a single ω as argument, not a set of outcomes.
We then calculate the probability that a random variable X has a certain value x as such:

$$\Pr[X = x] = \sum_{X^{-1}(x)} \Pr[\omega]$$

Definition 25 (Expectation). *The expectation of a random variable is defined as*

$$E[X] = \sum_{\Omega} X(\omega) P(\omega)$$

Definition 26 (Conditional Probability).

$$\Pr[A|B] = \frac{\Pr[A \cap B]}{\Pr[B]}$$

As a nice little exercise, we prove the formula for the conditional probability of the *complement* of B.

Proof.

$$\Pr[A|\overline{B}] =$$

□

As an illustrative example, consider the following probabilities. People can be *small* (S) or *tall* (T). They can be *good* (G) or *bad* (B) at basketball. Here are the tables of probabilities:

	S	T
	0.6	0.4

			S	T
B	0.14	B	0.54	0.08
G	0.86	G	0.6	0.32

$$\begin{array}{llll} P(B|S) & 0.9 & P(B|T) & 0.8 \\ P(G|S) & 0.1 & P(G|T) & 0.2 \end{array}$$

Notice the following facts:

- notice how $P(G|T) \neq 1 - P(G|S)$
- notice how $P(G|T) = 1 - P(B|T)$
- $P(A, B) = P(A|B)P(B) = P(B|A)P(A)$
- $\Sigma_A \Sigma_B P(A, B) = 1.0$

As yet another exercise, here is the formula of the probability of a union of arbitrary events:

Proof.

$$\Pr(\cup A_i) = \sum_i \Pr(A_i) - \sum_i \sum_{j>i} \Pr(A_i \cap A_j) + \sum_i \sum_{j>i} \sum_{k>j} \Pr(A_i \cap A_j \cap A_k) - ...$$

This is proven by induction.
Base case: Proof that $\Pr(A_1 \cup A_2) = \Pr(A_1) + \Pr(A_2) - \Pr(A_1 \cap A_2)$

> This is tirivally true when looking at a Venn Diagramm.

Thus $\Pr(A_1 \cup A_2) = \Pr(A_1) + \Pr(A_2) - \Pr(A_1 \cap A_2)$
Induction step. Suppose that... Proof that

Thus

□

1.7.1.2 A few lemmas on conditional probability

In a "causal" chain of events A, B, C we can integrate out the middle-event B.

$$\begin{aligned} p(A, B, C) &= \frac{p(A, B, C)}{p(A, B)} \frac{p(A, B)}{p(A)} p(A) \\ &= p(C|AB)p(B|A)p(A) \end{aligned} \tag{1.22}$$

$$p(A, C) = \Sigma_B p(A, B, C) \tag{1.23}$$

$$\begin{aligned} p(C|A) &= \frac{p(A, C)}{p(A)} \\ &= \Sigma_B p(C|A, B)p(B|A) \end{aligned} \tag{1.24}$$

We can take the expression for conditional probability and condition *every term* on a third event.

$$\begin{aligned} p(B|A, C) &= \frac{p(A, B, C)}{p(A, C)} \\ p(A|B, C) &= \frac{p(A, B, C)}{p(B, C)} \\ p(A|B, C) &= \frac{p(B|A, C)p(A|C)p(C)}{p(B|C)p(C)} \\ &= \frac{p(B|A, C)p(A|C)}{p(B|C)} \end{aligned} \tag{1.25}$$

1.7.2 Decomposing variance - the road to sensitivity analysis

Expressing variance as expectation ...

$$\begin{aligned} V_X &= E_{(X-E_X)^2} \\ &= E_{X^2 - 2XE_X + E_X^2} \\ &= E_{X^2} - E_X^2 \end{aligned} \tag{1.26}$$

Conditional expectation and variance ...

Definition 27. *Conditional expectation:*

$$E_{Y|x} = \Sigma y P(y|x)$$

Definition 28. *Conditional variance:*

$$V_{Y|x} = E_{(Y-E_{Y|x})^2|x}$$

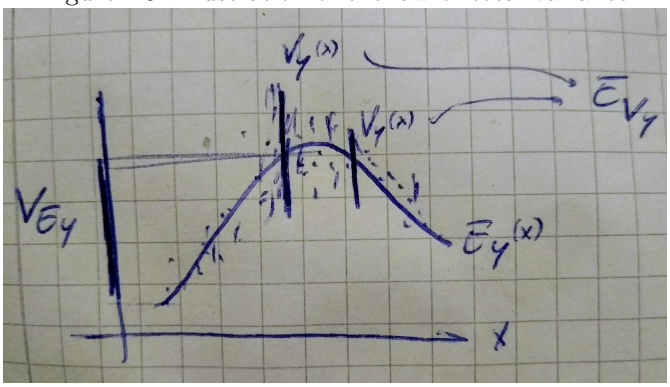
Law of total expectation ...

$$\begin{aligned} E_Y &= \Sigma_Y y P(y) \\ &= \Sigma_Y y \Sigma_X P(y|x) P(x) \\ &= \Sigma_X (\Sigma_Y y P(y|x)) P(x) \\ &= \Sigma_X E_{Y|x} P(x) \\ &= E_{E_{Y|x}} \end{aligned} \tag{1.27}$$

Law of total variance ...

$$\begin{aligned} V_Y &= E_{Y^2} - E_Y^2 \\ &= E_{E_{Y^2|X}} - E^2_{E_{Y|X}} \\ &= E_{V_{Y|X}+E^2_{Y|X}} - E^2_{E_{Y|X}} \\ &= E_{V_{Y|X}} + E_{E^2_{Y|X}} - E^2_{E_{Y|X}} \\ &= E_{V_{Y|X}} + V_{E_{Y|X}} \end{aligned} \tag{1.28}$$

Figure 1.9: Illustration of the law of total variance



1.7.3 Probability density functions

Up to now we have been dealing with probability mass functions on discrete variables. That works just as well with discrete variables, but we need to accommodate some details. For example, we can only define probability density in therms of cumulative probability functions.

Let $P(x) := Pr(X > x)$ be a cumulative probability function. Then the probability density at x is $\frac{dP}{dx}(x)$.

As an exercise, consider $x\tilde{E}xp(x)$. We want to calculate $E(x|x > x_0)$. We'll start with $P(x|x > x_0)$. We have:

$$\begin{aligned} P(X|X > x_0) &= \left(\text{ using the fact that } P(B|A) = \frac{P(A \wedge B)}{P(A)} \right) \\ &= \frac{P(X = x \wedge X > x_0)}{P(X > x_0)} \\ &= \frac{s_{x_0} P(X = x)}{P(x > x_0)} (\text{ with } s_{x_0} \text{ the step-function at } x_0) \\ &= \frac{s_{x_0} P(X = x)}{\int_{x_0}^{\infty} p(x) dx} \end{aligned} \tag{1.29}$$

This leads us to the expectation:

$$\begin{aligned} E(X|X > x_0) &= \int_{-\infty}^{\infty} xp(X|X > x_0)dx \\ &= \frac{s_{x_0} \int_{-\infty}^{\infty} xp(x)dx}{\int_{x_0}^{\infty} p(x)dx} \\ &= \frac{\int_{x_0}^{\infty} xp(x)dx}{\int_{x_0}^{\infty} p(x)dx} \\ &= \frac{s_{x_0} E(X)}{P(X > x_0)} \end{aligned} \tag{1.30}$$

1.7.4 Probability distributions

A probability distribution is a function from the domain of a random variable to its probability - in other words, a probability distribution yields the probability that a random variable will take on a certain value.

There is an abundance of ready made probability distributions to chose from, covering virtually all important situations. But care must be taken when deciding which distribution to apply to a certain problem.

The Bernoulli family based on modelling a series of coin-tosses.

- Bernoulli: heads or tails?
- Binominal: k heads in n trials
- Poisson: k heads in ∞ trials.

A remarkable feature of the Poisson-distribution is that it has only a parameter for the mean, but always the same variance.

The geometric family based on repeating an experiment until it succeeds.

1.7.4.1 Probabilistic fallacies

- T-Test interpretation: If $\Pr[A|B] = x$, then this does *not* mean that $\Pr[A|\overline{B}] = 1 - x$.
- Prosecutors fallacy aka. inverse fallacy: $P(A|B) \neq P(B|A)$

$\Pr[A|\overline{B}] \neq 1 - \Pr[A|B]$.

Proof. By contradiction.

$$\begin{aligned} \Pr[A|B] &= 1 - \Pr[A|\overline{B}] \\ &= \frac{\Pr[B] - \Pr[A \cap \overline{B}]}{\Pr[B]} \\ \Pr[A|B]\Pr[B] &= \Pr[B] - \Pr[A \cap \overline{B}] \\ \Pr[A \cap B] &= \Pr[B] - \Pr[A \cap \overline{B}] \\ \Pr[A \cap B] + \Pr[A \cap \overline{B}] &= \Pr[B] \\ \Pr[A] &= \Pr[B] \end{aligned} \tag{1.31}$$

Thus $\Pr[A|\overline{B}] \neq 1 - \Pr[A|B]$. But not that it *does* hold true that $\Pr[\overline{A}|B] = 1 - \Pr[A|B]$ □

1.8 Statistics

1.8.1 Correlation

Assume an inner product space.

1.8.2 Linear regression

Assume that reality can me modelled by a model like this one:

$$\vec{y} = \mathbf{X}\vec{w} + \vec{e}$$

Not knowing $\vec{e}$, our best guess at the outcome would be

$$\hat{\vec{y}} = \mathbf{X}\vec{w}$$

We can find the gradient of w by:

$$\frac{dE}{d\vec{w}} = -(\vec{y} - \hat{\vec{y}})\mathbf{X}$$

1.8.2.1 You are absolutely allowed to invert a linear regression relation

Linear regressions are focussed on $P(y|x)$ ... but you can absolutely revert that relation. For example, you are allowed to predict size from basketball scores, even though in reality it's likely the other way round.

Assume

$$y = \alpha x + \epsilon$$

and

$$\epsilon\ N(0, \sigma_\epsilon^2), x\ N(\mu_x, \sigma_x^2)$$

This yields $P(y|x) = N(\alpha x, \sigma_\epsilon^2)$. But in the same vain we may also write

$$x = \frac{y}{\alpha} - \frac{\epsilon}{\alpha}$$

With:

$$P(x|y) = N(\frac{\mu_y}{\alpha}, \frac{\sigma_\epsilon^2}{\alpha^2})$$

We can also arrive at the same joint distribution $P(x, y)$ coming from x as well as from y.

$$\begin{aligned} P(x,y) = P(y|x)P(x) &= N(\alpha x, \sigma_\epsilon^2)N(\mu_x, \sigma_x^2) \\ = P(x|y)P(y) &= N(\frac{y}{\alpha}, \frac{\sigma_\epsilon^2}{\alpha^2}) \int_X P(x,y) \end{aligned} \tag{1.32}$$

All the regression-model tells us, is that there is a relation $y = \alpha x + \epsilon$ between x and y. It does *not* imply a direction or causality.

1.8.3 Spatial modelling

1.8.3.1 Generalized least squares

like linear regression, but errors are not iid, but allowed to be correlated.

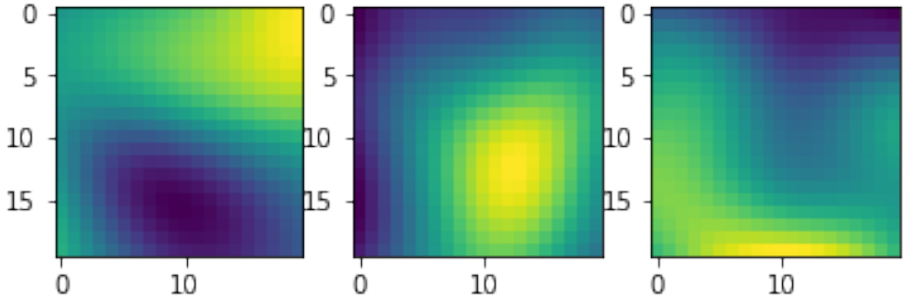
1.8.3.2 Gaussian processes

Gaussian processes are a means of interpolating a value y_x from surrounding values $y_x = \sum \alpha_i y_i$. Basic intuition from here. That is different from what linear regression or its extension GLS do: regression predicts y from explanatory variables x, assuming a (linear) model. Gaussian processes don't do that. They only interpolate between already observed y's. No model is assumed.

Consider the $n \times m$ surface $\mathbf{Y}$. Assume that the value of any pixel in $\mathbf{Y}$ follows a gaussian distribution - which may be correlated to all other pixels in $\mathbf{Y}$. Millions of surfaces $\mathbf{Y}$ may be sampled from $N(\vec{0}, \mathbf{\Sigma}) = P(\mathbf{Y})$, where $\vec{(0)}$ is of size $n \times m$ and $\mathbf{\Sigma}$ is of size $nm \times nm$. We call $P(\mathbf{Y})$ the prior. $\mathbf{\Sigma}$ can be very large, so we assume that it can be modeled by a covariance-function $cov(h)$ which is *only dependent on the distance between two points, not on the points themselves.*

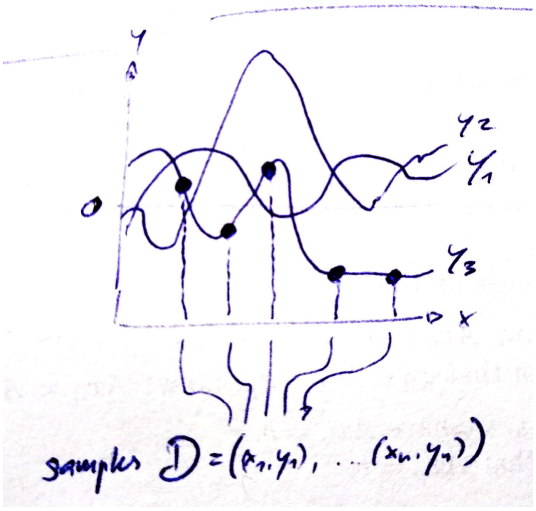
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal


NOISEFACTOR = 0.001


def size(x):
    return np.sqrt(np.sum(np.power(x, 2)))

def distance(x0, x1):
    return size(x0 - x1)

def getDistanceMatrix(rowPoints, colPoints):
    # deliberately not exploiting symmetry
    # <- this way it works with non-square matrices, too.
    rows = len(rowPoints)
    cols = len(colPoints)
    distances = np.zeros((rows, cols))
    for i in range(rows):
        for j in range(cols):
            p0 = rowPoints[i]
            p1 = colPoints[j]
            d = distance(p0, p1)
            distances[i, j] = d
    return distances

deltaX = 0.1
deltaY = 0.1
gridX = 40
gridY = 60
nrPoints = gridX * gridY
points = np.zeros((nrPoints, 2))
i = 0
for x in range(gridX):
    for y in range(gridY):
        points[i] = [x * deltaX, y * deltaY]
        i += 1
distances = getDistanceMatrix(points, points)

mean = np.zeros((nrPoints))
cov0 = 2.3  # variance without distance: Cov(0) = Cov(Z(x), Z(x)) = Var(Z | x)
h95 = 40     # distance at which cov(h) >= 0.95 * cov0

def variogramFunction(h):
    return cov0 * ( 1 - np.exp( -3*h/h95 ) )

Sigma = cov0 - variogramFunction(distances)
Sigma += NOISEFACTOR * np.eye(nrPoints) * np.random.rand(nrPoints) # to make inverse possible
prior = multivariate_normal(mean, Sigma, allow_singular=True)

sample1 = prior.rvs()
sample2 = prior.rvs()
sample3 = prior.rvs()
```

Figure 1.10: 3 samples from prior $P(\mathbf{Y})$



We have observed some samples $\mathbf{D} = (x_i, y_i)$. Which of these millions of surfaces are most likely given those observations? We can answer that with $P(\mathbf{Y}|\mathbf{D})$.

Figure 1.11: Observations $\mathbf{D}$. Which field $\mathbf{Y}$ is most likely to have produced these observations?



We can calculate the posterior using

$$P(\mathbf{Y}|\mathbf{D}) \sim P(\mathbf{D}|\mathbf{Y})P(\mathbf{Y})$$

But since in gaussian processes we assume multivariate-normals, this can be done much easier.

Consider a multivariate normal distribution on, say, 400 dimensions $N(\vec{\mu}, \mathbf{\Sigma})$. Those 400 dimensions could, for example, be pixel-values in a 20 by 20 image. Now assume that out of those 400 dimensions we have measurements for s samples, leaving a rest $r = 400 - s$. For multivariate-normal distributions, we can easily and analytically obtain the conditional distribution $P(\vec{y}_r|\vec{y}_s) = N(\hat{\vec{\mu}}, \hat{\mathbf{\Sigma}})$.

Split $\vec{y}$ like so:

$$y = \begin{bmatrix} \vec{y}_r \\ \vec{y}_s \end{bmatrix}$$

Analogously we can split the covariance-matrix:

$$\mathbf{\Sigma} = \begin{bmatrix} \mathbf{\Sigma}_{rr} & \mathbf{\Sigma}_{rs} \\ \mathbf{\Sigma}_{sr} & \mathbf{\Sigma}_{ss} \end{bmatrix}$$

Then, according to wikipedia, we can obtain $\hat{\vec{\mu}}$ and $\hat{\mathbf{\Sigma}}$ like so:

$$\hat{\vec{\mu}}_r = \vec{\mu}_r + \mathbf{\Sigma}_{rs}\mathbf{\Sigma}_{ss}^{-1}(\vec{y}_s - \vec{\mu}_s)$$

$$\hat{\mathbf{\Sigma}}_r = \mathbf{\Sigma}_{rr} - \mathbf{\Sigma}_{rs}\mathbf{\Sigma}_{ss}^{-1}\mathbf{\Sigma}_{sr}$$

```
indices = np.arange(nrPoints)
sampleIndices = np.asarray([13, 25, 84, 96]) # ... from data ...
sampleValues = np.asarray([4.6, 19, 9.4, 21]) # ... from data ...
samplePoints = points[sampleIndices]
nonSampleIndices = np.asarray(
        [i for i in indices
            if i not in sampleIndices]
)
nonSamplePoints = points[nonSampleIndices]

# For simplicity, we just re-calculate SigmaXX here.
# But we could have also picked the right values from Sigma
# without any need for re-computation.
distancesRR = getDistanceMatrix(nonSamplePoints, nonSamplePoints)
SigmaRR = cov0 - variogramFunction(distancesRR)

distancesSS = getDistanceMatrix(samplePoints, samplePoints)
SigmaSS = cov0 - variogramFunction(distancesSS)
SigmaSS += NOISEFACTOR * np.eye(nrSamples) * np.random.rand(nrSamples) # To make inverse possible
SigmaSSInverse = np.linalg.inv(SigmaSS)

distancesRS = getDistanceMatrix(nonSamplePoints, samplePoints)
SigmaRS = cov0 - variogramFunction(distancesRS)
SigmaSR = SigmaRS.transpose()

meanPosterior = SigmaRS @ SigmaSSInverse @ sampleValues
SigmaPosterior = SigmaRR - SigmaRS @ SigmaSSInverse @ SigmaSR
SigmaPosterior += NOISEFACTOR * np.eye(nrPoints - nrSamples) * np.random.rand(nrPoints - nrSamples)  # To make inverse po

posterior = multivariate_normal(meanPosterior, SigmaPosterior)
```

We can now plot the `meanPosterior` around our `samples`. Doing so we obtain the graphic as shown in figure 1.12.

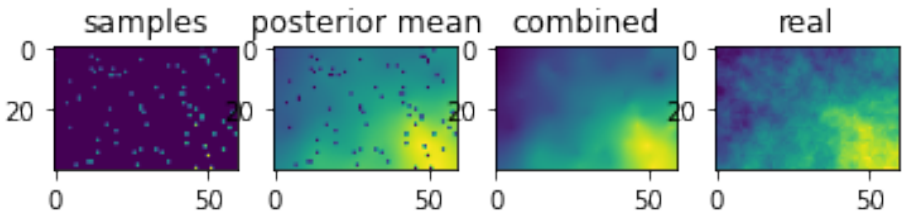


Figure 1.12: Samples and fitted posterior fit together well

Note how we assumed that the covariance matrix $\mathbf{\Sigma}$ could be parameterized with a covariance-function $c(h)$. Only this way do we have any chance of finding a good approximation for $\mathbf{\Sigma}$ with only few samples. The relation `Sigma = cov0 - variogramFunction(distances)` comes from the variogram $\gamma(h)$ as in here:

$$\begin{aligned} 2\gamma(h) &= E[(Z(x+h) - Z(x))^2] \\ &= E[((Z(x+h) - \mu) - (Z(x) - \mu))^2] \\ &= E[(Z(x+h) - \mu)^2 - 2(Z(x+h) - \mu)(Z(x) - \mu) + (Z(x) - \mu)^2] \\ &= E[(Z(x+h) - \mu)^2] - 2E[(Z(x+h) - \mu)(Z(x) - \mu)] + E[(Z(x) - \mu)^2] \\ &= c(0) - 2c(h) + c(0) \\ \gamma(h) &= c(0) - c(h) \\ c(h) &= c(0) - \gamma(h) \end{aligned} \tag{1.33}$$

In practical applications, we'll usually fit our function `cov0 * ( 1 - np.exp( -3*h/h95 ) )` by moving the parameters `cov0` and `h95` until the residuals fit properly. Our code would then look like this:

```
def empiricalVariogram(distanceMatrix, data):
  variogram = {}
  d0 = np.min(distanceMatrix)
  d1 = np.max(distanceMatrix)
  nrHs = 20
  deltaH = (d1 - d0) / nrHs
  for h in np.linspace(d0, d1, nrHs):
      Nh = 0
      hSum = 0
      pairs = (h <= distanceMatrix) * (distanceMatrix < h+deltaH)
      for i in range(nrTotal):
```

```
            for j in range(i, nrTotal):
                if pairs[i, j]:
                    v = data[i]
                    vh = data[j]
                    hSum += np.power(v - vh, 2)
                    Nh += 1
        if Nh > 0:
            variogram[h] = hSum / (2 * Nh)
    return variogram

def fitVariogram(samples):
    pass  # TBD


residuals = prediction - samples
(cov0, h95) = fitVariogram(residuals)

SigmaRR = cov0 - variogramFunction(cov0, h95, distancesRR)
SigmaSS = cov0 - variogramFunction(cov0, h95, distancesSS)
SigmaRS = cov0 - variogramFunction(cov0, h95, distancesRS)
SigmaSR = SigmaRS.transpose()
SigmaSSInverse = np.linalg.inv(SigmaSS)

meanPosterior = prediction + SigmaRS @ SigmaSSInverse @ sampleValues
SigmaPosterior = SigmaRR - SigmaRS @ SigmaSSInverse @ SigmaSR

predictionPosterior = meanPosterior
```

For more, see this very thorough tutorial.

1.8.3.3 AR and SAR

Autoregressive and spatial-autoregressive models.

1.8.3.4 Comparison

	Generalized least squares	Gaussian processes	AR
type	Predict y from x. Has explanatory variables.	Interpolate y from other y's. Explanatories can be added with GLS	interpolate y from other y's
model	$y = X\alpha + \epsilon$ $\epsilon \sim N(\vec{0}, \mathbf{\Sigma})$	$Y \sim N(\vec{0}, \mathbf{\Sigma})$ $P(\mathbf{Y}\|\mathbf{D}) = N(\hat{\vec{\mu}}, \hat{\mathbf{\Sigma}})$ $\tilde{y} = \sum \alpha_i \vec{y_i}$ Y is a field of size n by m D is the observations	$y_t = \sum \alpha_i y_{t-i} + \epsilon$ $\epsilon \sim N(0, \sigma)$, iid
Specialities		$\mathbf{\Sigma}$ is encoded using a covariance-function cov(h) to save on parameters	
ϵ			
$cov(\epsilon_i, \epsilon_j)$			
$cov(y_i, y_j)$			

1.8.4 Bayesian networks

Bayesian networks are a neat tool when we can manually encode some complex causation-graphs.

```
  def find(lst, predicate):
  for entry in lst:
      if predicate(entry):
          return entry

def prod(lst):
  p = lst[0]
  for entry in lst[1:]:
      p *= entry
  return p

def getCombinations(lists):
  combinations = []
  if len(lists) == 0:
```

```
        combinations = []
    elif len(lists) == 1:
        combinations = lists[0]
    elif len(lists) == 2:
        for a in lists[0]:
            for b in lists[1]:
                combinations.append([a, b])
    else:
        firstList = lists[0]
        subCombos = getCombinations(lists[1:])
        for a in firstList:
            for b in subCombos:
                combinations.append([a, *b])
    return combinations



class Node:
    def __init__(self, name, pmf, allowedValues, parents = []):
        self.name = name
        self.pmf = pmf
        self.allowedValues = allowedValues
        self.parents = parents

    def setPmf(self, pmf):
        self.pmf = pmf

    def calcProb(self, value):

        if value not in self.allowedValues:
            return 0.0

        parentData = self.__getParentData()

        p = 0.0
        if len(parentData) > 0:
            for combination in getCombinations(parentData):
                parentValues = {entry['name']: entry['value'] for entry in combination}
                parentProbs = [entry['prob'] for entry in combination]
                p += self.pmf(value, parentValues) * prod(parentProbs)
        else:
            p = self.pmf(value, {})

        return p

    def __getParentData(self):
        parentData = []
        for i, parent in enumerate(self.parents):
            parentData.append([])
            for value in parent.allowedValues:
                prob = parent.calcProb(value)
                parentData[i].append({
                    'name': parent.name,
                    'value': value,
                    'prob': prob
                })
        return parentData


class Net:
    def __init__(self, nodes):
        self.nodes = nodes
        self.memo = {}
        for node in self.nodes:
            self.memo[node.name] = {}

    def calcProb(self, nodeName, value):
        if self.memo[nodeName][value]:
            return self.memo[nodeName][value]
        else:
            node = self.__findNode(nodeName)
            prob = node.calcProb(value)
            self.memo[nodeName][value] = prob
            return prob

    def setPmf(self, nodeName, pmf):
        # setting pmf
        node = self.__findNode(nodeName)
        node.setPmf(pmf)

        # invalidating downstream cached results
        self.memo[nodeName] = {}
        children = self.__findChildren(nodeName, True)
        for child in children:
```

```
            self.memo[child.name] = {}

    def __findNode(self, nodeName):
        node = find(self.nodes, lambda n: n.name == nodeName)
        return node

    def __findChildren(self, nodeName, recursive=False):
        pass



def randomSelection(door, parents):
    return 1/3

guest = Node('guest', randomSelection, [1, 2, 3])

real = Node('real', randomSelection, [1, 2, 3])


def pickingRandomOther(door, parents):
    if parents['real'] == door:
        return 0
    if parents['guest'] == door:
        return 0
    elif parents['real'] == parents['guest']:
        return 1/2
    else:
        return 1

monty = Node('monty', pickingRandomOther, [1, 2, 3], [real, guest])


# %%
monty.calcProb(1)
# %%
def guestPicked1(door, parents):
    if door == 1:
        return 1
    else:
        return 0

guest.setPmf(guestPicked1)

monty.calcProb(2)
# %%
def realDoorIs2(door, parents):
    if door == 2:
        return 1
    else:
        return 0

real.setPmf(realDoorIs2)

monty.calcProb(3)
```

Note that you can change the direction of an edge in a bayesian net - but you'll have to change the associated probability tables. Let's say we start with a relation $A \rightarrow B$ and have given $P(A)$ and $P(B|A)$. To turn this around into $B \rightarrow A$ we need to obtain $P(B)$ and $P(A|B)$. We can get $P(A|B)$ as:

$$P(A|B) = \frac{P(A,B)}{P(B)} = \frac{P(B|A)P(A)}{P(B)} = \frac{P(B|A)P(A)}{\int_A P(B|A)P(A)}$$

which is, obviously, different from $P(A)$, so we *do need* to make changes to the tables. We can also obtain $P(B)$ as:

$$P(B) = \int_A P(A,B) = \int_A P(B|A)P(A)$$

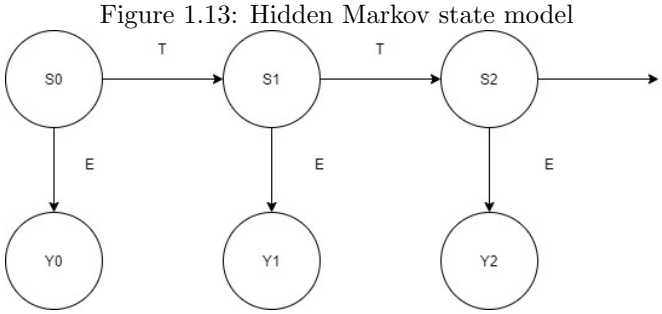
So we can calculate the terms $P(B)$ and $P(A|B)$ from our given $P(A)$ and $P(B|A)$.

1.8.5 Hidden-state models

Imagine a model as in image 1.8.5. What is known:

- $p(y_t|s_t)$
- $p(s_t + 1|s_t)$

Figure 1.13: Hidden Markov state model



From this we can deduce:

$$p(s_0|y_0) = p(y_0|s_0)\frac{p(s_0)}{p(y_0)}$$
$$p(s_1|y_0) = \Sigma_{s_0} p(s_1|s_0)p(s_0|y_0)$$
$$p(y_1|y_0) = \Sigma_{s_0}\Sigma_{s_1} p(y_1|s_1)p(s_1|s_0)p(s_0|y_0)$$

Note that $p(y_1|y_0)$ is not a function of either s_0 nor s_1.

$$\begin{aligned} p(s_1|y_0, y_1) &= \frac{p(y_1|s_1, y_0)p(s_1|y_0)}{p(y_1|y_0)} \\ &= \frac{p(y_1|s_1)p(s_1|y_0)}{p(y_1|y_0)} \\ &\propto p(y_1|s_1)p(s_1|y_0) \text{ ... since } p(y_1|y_0) \text{ is not a function of } s \end{aligned} \tag{1.34}$$

(As reference, see the findings 1.7.1.2.)

More generally, we can write the **update-equation** as:

$$p(s_t|\vec{y}_{[1..t]}) \propto p(y_t|s_t)p(s_t|\vec{y}_{[1..t-1]}) \tag{1.35}$$

... where $p(s_t|\vec{y}_{[1..t-1]})$ is the result of the last prediction equation.

With the **prediction-equation**:

$$p(s_{t+1}|\vec{y}_{[1..t]}) = \Sigma_{s_t} p(s_{t+1}|s_t)p(s_t|\vec{y}_{[1..t]})) \tag{1.36}$$

... where $p(s_t|\vec{y}_{[1..t]})$ is the result of the last update equation.

```
states = ('Rainy', 'Sunny')

observations = ('walk', 'shop', 'clean')

start_prob = {'Rainy': 0.6, 'Sunny': 0.4}

transition_prob = {
   'Rainy' : {'Rainy': 0.7, 'Sunny': 0.3},
   'Sunny' : {'Rainy': 0.4, 'Sunny': 0.6},
}

emission_prob = {
   'Rainy' : {'walk': 0.1, 'shop': 0.4, 'clean': 0.5},
   'Sunny' : {'walk': 0.6, 'shop': 0.3, 'clean': 0.1},
}

state_series = ['Rainy', 'Sunny', 'Sunny', 'Sunny', 'Rainy', 'Sunny', 'Rainy', 'Rainy', 'Sunny', 'Sunny', 'Sunny']
emission_series = ['clean', 'walk', 'walk', 'shop', 'clean', 'shop', 'clean', 'clean', 'clean', 'shop', 'clean']


def prediction(states, transition_prob, prob_St_Yt):
    prob_St1_Yt = {}
    for state1 in states:
        prob_St1_Yt[state1] = 0.0
        for state0 in states:
            prob_St1_Yt[state1] += transition_prob[state0][state1] * prob_St_Yt[state0]
    return prob_St1_Yt


def update(states, emission, emission_prob, prob_St1_Yt):
```

```
        prob_St_Yt = {}
        for state in states:
            prob_St_Yt[state] = emission_prob[state][emission] * prob_St1_Yt[state]
        sm = sum(prob_St_Yt.values())
        for state in prob_St_Yt:
            prob_St_Yt[state] = prob_St_Yt[state] / sm
        return prob_St_Yt


    prob_St1_Yt = start_prob
    for t in range(10):
        emission = emission_series[t]
        prob_St_Yt = update(states, emission, emission_prob, prob_St1_Yt)
        prob_St1_Yt = prediction(states, transition_prob, prob_St_Yt)
```

1.8.6 Tests

This subsection is about what is the most common form of statistics in social sciences: tests. Often, statistical tests work *the wrong way round*. You have your hypothesis, like "these groups are different". Then you state the opposite, the *null-hypothesis*. Then you determine how likely your data is under the null-hypothesis - the *p-value*. And than you hope that the p-value is very low.

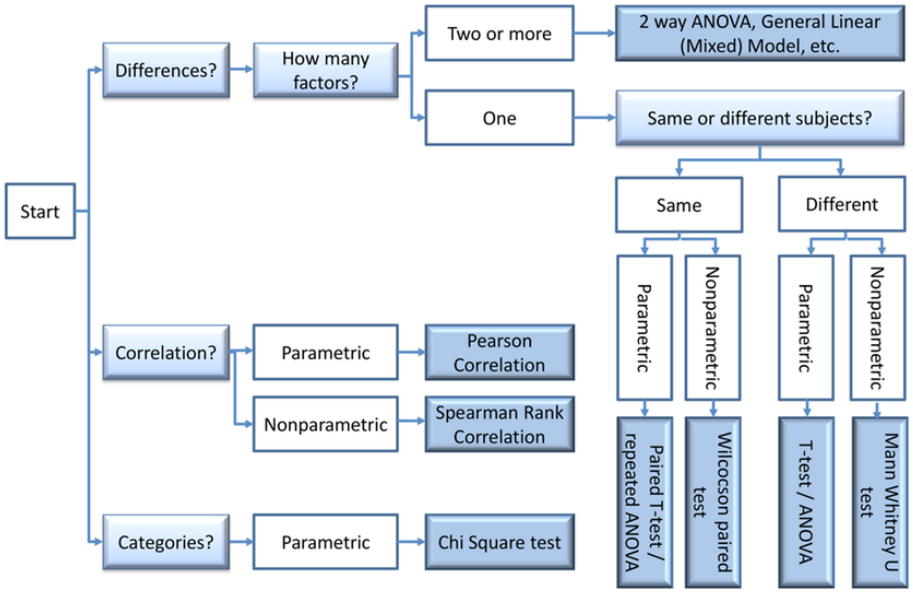
Students T-Test compares two averages (means) and tells you if they are different from each other. The null hypothesis is that all samples come from the same distribution. That means that under H_0, the two means you obtained should be similar. So the interpretation goes like this:

- low p-value: the chance that your result would have happend under H_0 is low. It is very unlikely that the two means you have obtained actually come from the same distribution.
- high p-value: It is quite possible that both your groups come from the same distribution.

Note that there are two experimental setups that can use t-tests:

- unpaired: you have two groups, with no person in both groups. Example: Test one group with medication and another with placebo.
- paired: every person is first in the first and then in the second group. Example: test patients before and after treatment.

Figure 1.14: Statistical test decision tree



1.9 Graph Theory

1.9.1 General properties

1.9.1.1 Degrees

Theorem 24. *In a directed graph* $\sum_{V_G} \text{indeg}(v_n) = |E_G|$

Proof. By induction. Let $P(i)$ be $\sum_{V_G} \text{indeg}(v_n) = |E_G|$
Proof that $P(1)$ and $P(i) \to P(i+1)$ hold true.

> Base case: $\sum_{V_G} \text{indeg}(v_n) = |E_G|$ where $E_G = \{(v_1, v_2)\}$
> Proof that $P(1)$ holds true.
>
> > This is tirvially true.
>
> Thus $P(1)$ holds true.
>
> Induction step:
>
> - Let $\sum_{V_G} \text{indeg}(v_n) = |E_G|$
> - Let $V_{G'} = V_G \cup \{v_{N+1}\}$
> - Let $E_{G'} = E_G \cup \{(x, y) \in V_G \times \{v_{N+1}\} \cup \{v_{N+1}\} \times V_G\}$
>
> Proof that $P(i) \to P(i+1)$ holds true.
>
> > For the left side it holds that:
> >
> > $$\sum_{V_{G'}} \text{indeg}(v_n) = \sum_{V_G} \text{indeg}(v_n) + \text{outdeg}(v_{N+1}) + \text{indeg}(v_{N+1})$$
> >
> > On the other hand on the right side it holds that:
> >
> > $$|E_{G'}| = |E_G| + \text{outdeg}(v_{N+1}) + \text{indeg}(v_{N+1})$$
> >
> > So the equation reduces to $0 = 0$, which is true.
>
> Thus $P(i) \to P(i+1)$ holds true.

Thus $P(1)$ and $P(i) \to P(i+1)$ hold true.

□

Lemma 1. $\sum_{V_G} \text{indeg}(v_n) = \sum_{V_G} \text{outdeg}(v_n)$

Lemma 2. *In an undirected graph we have* $\sum_{V_G} \deg(v_n) = 2|E_G|$

1.9.2 Walks and paths

Definition 29. *A walk is any sequence of vertices that are connected by an edge.*

Definition 30. *A path is a walk where no vertex appears twice.*

Theorem 25. *The shortest walk between any two vertices in a graph is a path.*

Proof. By contradiction. Suppose that the vertex x appears twice in the shortest walk. Proof that this is a contradiction.

> Since x appears twice, the proposed walk must be of the form $f - x - g - x - h$, where g is a sequence of 0 or more connected vertices. Then this walk can be shortened to $f - x - h$.

Thus this is a contradiction.

□

Theorem 26. *The longest path in a graph has length* $|V_G| - 1$.

This is too trivial for a full-fledged proof: there are no repetitions in a path, so it can only have as many steps as it has vertices.

1.9.2.1 Relations and adjacency matrices

Let $R \circ Q$ be a composition of two relations. The number of paths of length exactly n between two vertices x and y, written as $\text{paths}_n(x, z)$, can be expressed as

$$\text{paths}_n(x, z) = |\{y | xRy \wedge yQz\}| = \sum_{y \in Y} xRy \cdot yQz$$

If R_{AM} and Q_{AM} are the adjacency matrix representations of the above relations, then:

$$R_{AM} \cdot Q_{AM}[n, m] = \sum_{y \in Y} R_{AM}[x_n, y] \cdot Q_{AM}[y, z_m]$$

$$= \sum_{y \in Y} xRy \cdot yQz = \text{paths}_n(x, z)$$

When $R = Q$, it follows easily that:

Theorem 27. *$R^2_{AM}[n, m]$ is the number of paths that go from v_n to v_m in exactly 2 steps.*

Theorem 28. *R_{AM} explains whether or not two vertices v and u are connected. All possible paths are listed in $R^* = R_{AM} \cup R^2_{AM} \cup R^3_{AM} \cup ... \cup R^{|V_G|-1}_{AM}$. We can find the length of the shortest path between two vertices u and v by*

1.9.3 Planar graphs

The following theorems all deal with planar, connected graphs, and build up to Eulers theorem. Well use N for the number of nodes, E for the number of edges, and L for the number of loops.

Theorem 29. *Adding a edge to a planar, connected graph means adding one loop: $\Delta N = 0 \wedge \Delta E = 1 \rightarrow \Delta L = 1$*

Theorem 30. *Adding x edges means adding x loops: $\Delta N = 0 \wedge \Delta E = x \rightarrow \Delta L = x$*

Proof. Proof that $\Delta N = 0 \wedge \Delta E = x \rightarrow \Delta L = x$

> By induction on ΔE.
> Base case: $\Delta E = 1$ Proof that $\Delta L = 1$
> > By theorem 11.
>
> Thus $\Delta L = 1$
> Let $\Delta N = 0 \wedge \Delta E = x \rightarrow \Delta L = x$.
> Proof that $\Delta N = 0 \wedge \Delta E = x + 1 \rightarrow \Delta L = x + 1$
> > Consider the graph from the induction hypothesis, then apply theorem 12.
>
> Thus $\Delta N = 0 \wedge \Delta E = x + 1 \rightarrow \Delta L = x + 1$

Thus $\Delta N = 0 \wedge \Delta E = x \rightarrow \Delta L = x$

□

Theorem 31. *When we add a new node to a graph and connect it to the graph using one or more edges, then the number of edges and loops behaves as such: $\Delta N = 1 \rightarrow \Delta L = \Delta E - 1$*

Proof. Proof that $\Delta N = 1 \rightarrow \Delta L = \Delta E - 1$

> Let $\Delta N = 1$.
> By induction on ΔE.
> Base case: $\Delta E = 1$ Proof that $\Delta L = 0$
> > Graphically trivial.
>
> Thus $\Delta L = 0$
> Induction step: $\Delta L = \Delta E - 1$ Proof that $\Delta L' = \Delta E' - 1$
> > $\Delta E' = \Delta E + 1$
> > Using theorem 12, that means: $\Delta L' = \Delta L + 1$
> > Thus: $\Delta L + 1 = \Delta E + 1 - 1$
> > Which reduces to: $\Delta L = \Delta E - 1$, which is true by the induction hypothesis.

Thus $\Delta L' = \Delta E' - 1$

Thus $\Delta N = 1 \rightarrow \Delta L = \Delta E - 1$

□

Theorem 32. *Eulers theorem:* $N + L = E + 1$

Proof. Proof that $N + L = E + 1$

By induction on N.
Base case: $N = 1$ or $N = 2$ Proof that $N + L = E + 1$
If $N = 1$: $1 + 0 = 0 + 1$
If $N = 2$: $2 + 0 = 1 + 1$
Thus $N + L = E + 1$
Induction step: Let $N + L = E + 1$.
Proof that $N + 1 + L' = E' + 1$
Using theorem 13: $\Delta L' = \Delta E - 1$
Thus: $N + 1 + L + \Delta L = E + \Delta E + 1$
$N + L = E + 1$, which is true by the induction hypothesis.
Thus $N + 1 + L' = E' + 1$

Thus $N + L = E + 1$

□

1.9.3.1 Trees

Here is a fun little proof.

Proof. In a tree, the mean number of children (mcc) is always equal to $1 - 1/N$.
Proof that $mcc = 1 - \frac{1}{N}$

With what we have so far, this is almost trivial to prove. The mean child count can be written as

$$mcc = \frac{1}{N} \sum_{N} \text{outdeg}\, v_n$$

Using theorem 24 and lemma 1, this becomes

$$mcc = \frac{1}{N} E$$

Eulers formula in a tree becomes $N = E + 1$, since there are no loops in trees. Using this:

$$mcc = 1 - \frac{1}{N}$$

Thus $mcc = 1 - \frac{1}{N}$

□

When training a binary classifier, we usually need a measure of defining which of several trees is a good representation of the data. The number of nodes, however, cannot be used as a measure, because it is always going to be $2n - 1$.

Proof. In a binary tree, the number of leaves n equals $\frac{N+1}{2}$, regardless of the architecture of the tree.
Proof that $n = \frac{N+1}{2}$

We shall use the fact that in any tree, $\sum_{V_G} \text{indeg}(v_n) = \sum_{V_G} \text{outdeg}(v_n)$.

Consider the leaves of the tree. They all have $\text{indeg}(v_n) = 1$ and $\text{outdeg}(v_n) = 0$. Thus $\sum_{\text{leaves}} \text{indeg}(v_n) = n$

and $\sum_{\text{leaves}} \text{outdeg}(v_n) = 0$.
Now consider the nodes (there are $N - n$ of them). They all have $\text{outdeg}(v_n) = 2$ and $\text{indeg}(v_n) = 1$, except for the root-node. Thus $\sum_{\text{nodes}} \text{indeg}(v_n) = N - n - 1$ and $\sum_{\text{nodes}} \text{outdeg}(v_n) = 2(N - n)$.
Equating the in- and out-degrees, we obtain:

$$n + N - n - 1 = 0 + 2(N - n)$$

$$n = \frac{N + 1}{2}$$

Thus $n = \frac{N+1}{2}$

□

1.9.3.2 Triangulation

Proof. A *n-gon*, a polygon with n nodes, can be drawn with $n - 2$ triangles. The proof follows from induction on n, starting at $n = 3$. □

Proof. When drawing a n-gon ($n \geq 3$) using triangles, there are $n - 3$ shared edges.

- The n-gon has n nodes, n edges and one loop
- The $n - 2$ triangles have in total n nodes, $n - 2$ loops and (via Eulers formula) $2n - 3$ edges.
- The outer edges, that is, all the edges of the n-gon, are not shared. Thus, there are $e_{triangles} - e_{n-gon} = n - 3$ shared edges.

□

1.9.4 Stable marriage and Gayle-Shapely

In dating, the people doing the active flirting end up with partners higher up their preference-list than the people being flirted with.

```
    class Person:
    def __init__(self, name):
        self.name = name
        self.prefs = []

    def setPreferences(self, prefs):
        self.prefs = prefs

    def __repr__(self):
        return self.name


class Woman(Person):
    def selectBest(self, suitors):
        for p in self.prefs:
            if p in suitors:
                print(f"{self} selected {p}")
                return p


class Man(Person):
    def getFavorite(self):
        print(f"{self} fancies {self.prefs[0]}")
        return self.prefs[0]

    def rejectedBy(self, w):
        print(f"{w} rejected {self}")
        self.prefs.remove(w)


rachel = Woman('Rachel')
phoebe = Woman('Phoebe')
monica = Woman('Monica')

ross = Man('Ross')
chandler = Man('Chandler')
joey = Man('Joey')
```

```
women = [rachel, phoebe, monica]
men = [ross, chandler, joey]

rachel.setPreferences([joey, ross, chandler])
phoebe.setPreferences([ross, chandler, joey])
monica.setPreferences([joey, chandler, ross])

ross.setPreferences([rachel, phoebe, monica])
chandler.setPreferences([rachel, monica, phoebe])
joey.setPreferences([phoebe, rachel, monica])


## Gayle - Shapely

flirting = {w: [] for w in women}

def isStable(flirting):
    for w in flirting:
        suitors = flirting[w]
        if len(suitors) != 1:
            return False
    return True


while not isStable(flirting):
    print("--- lets go, flirt! ----")

    # men move to their most desired woman
    for m in men:
        w = m.getFavorite()
        flirting[w].append(m)

    # women reject all but best suitor
    for w in women:
        suitors = flirting[w]
        fav = w.selectBest(suitors)
        for m in suitors:
            if m is not fav:
                m.rejectedBy(w)
        if fav:
            flirting[w] = [fav]


print(" ---- couples have formed: ---- ")
for w in flirting:
    m = flirting[w][0]
    print(f"{w} + {m}")
```

1.9.4.1 Path finding

We have a graph of connections with a cost associated to each connection. The goal is to go from node a to node b as cheaply as possible.

A first approach might be depth first. But depth first on a plain, empty field will snake left and right along the bottom for a long time before it finally hits a point b just a few boxes above a.

Then we're better off with breadth first search. But BFS spreads out blindly in all directions.

The next better thing is Dijkstra's algorithm. It's BFS but with a preference for cheap paths. On a road net, Dijkstra would investigate highways before it considers dirt roads with trees across them. All the points considered in BFS are ranked in a priority queue. The queue contains the candidate point, the cheapest path to the point up to now, and the cheapest price up to now. We pick the cheapest point first. If our path to that current cheapest point is cheaper than what it was up to now, that path is updated.

Notably, Dijkstra has no sense of direction. If a dirt-track leads almost directly north to b, but a highway goes south, Dijkstra will investigate the highway first. This can be accounted for by adding a distance-heuristic to the price-calculation. This is the A* algorithm.

```
type Node = char;
type Connection = [Node, Node];

class Graph {
    nodes: Node[] = [];
    connections: Connection[] = [];
}

class QEntry {
    node: Node;
    path: Node[];
```

```
    price: number;
}

class Queue {
    entries: QEntry[];

    public getNext(): QEntry {
        return this.entries.pop();
    }

    public add(entry: QEntry): void {
        for (let i = this.entries.length; i > 0; i--) {
            const candidate = this.entries[i];
            if (candidate.price < entry.price) {
                this.entries.insert(i, entry);
            }
        }
    }
}

function dijkstra(graph: Graph, queue: Queue, b: Node): Node[] {
    const a = queue.getNext();
    for (const {candidateNode, candidatePrice } in graph.getNeighbors(a.node)) {
        if (candidateNode === b) {
            return a.path + [b];
        }
        const newEntry: QEntry = { candidateNode, a.path + [a.node], a.price + candidatePrice };
        if (queue.get(candidateNode).price > newEntry.price) {
            queue.add(newEntry);
        }
    }
    return dijkstra(graph, queue, b);
}
```

There are also public APIs for path-finding on actual streets. Graphhopper is one example.

1.10 Calculus

1.10.1 Hyperreals

For the biggest part, we're going to deal with Nelson-style nonstandard-analysis.

List of external properties:

- std, nstd
- limt, nlimt
- inftsm, inft
- nearly cont

A statement using any external properties will be denoted as A_{ext}, one that *might* use external properties as $A_{(ext)}$.

We will use the following axioms:

1. $0 : std$
2. $\forall n \in \mathbb{N} : n : std \to (n+1) : std$
3. $\exists n \in \mathbb{N} : n : nstd$
4. External induction: Induction over n_{std} about $A_{(ext)}$:
   $[A_{(ext)}(0) \wedge \forall n_{std} \in \mathbb{N} : A_{(ext)}(n) \to A_{(ext)}(n+1)] \to \forall n_{std} \in \mathbb{N} : A_{(ext)}(n)$
5. Internal induction: Induction over $n_{(nstd)}$ about A:
   $[A(0) \wedge \forall n_{(nstd)} \in \mathbb{N} : A(n) \to A(n+1)] \to \forall n \in \mathbb{N} : A(n)$

The rationale over the two induction-axioms is simple. Ordinary induction is about A over n_{std}. External induction is about $A_{(ext)}$ over n_{std}. This makes sure that statements about external stuff only apply to finite n, not to infinite ones. Internal induction is about A over $n_{(nstd)}$. This makes sure that when we talk about potentially infinite n's, we only apply internal statements.

In other words: these two inductions ensure that we **never apply external statements to external numbers**.

Doing so would lead to logical incosistencies. That's why there is no "fully external" induction.

However, note that axiom 2 is actually a case of "fully external" induction.

Theorem 33. $\forall n \in \mathbb{N} : n : nst \to (n+1) : nst$

Proof. Suppose $n : nst$. Proof that $(n+1) : nst$

> By contradiction. Suppose $(n+1) : std$. Proof that this leads to a contradiction.
>
> > $(n+1) : std \to n : std$.
> > This contradicts the premise that $n : nst$.
>
> Thus this leads to a contradiction.

Thus $(n+1) : nst$

□

If you don't believe the argument in the previous proof, consider this:

Proof. Suppose all of the following:
$[\forall n : Q(n) \to Q(n+1)] \to \forall n : Q(n)$
$[\forall n : Q(n) \to Q(n+1)]$
This leads to $\forall n : Q(n)$.
Proof that $\forall n : Q(n+1) \to Q(n)$

Let $n = n_0$ and suppose $Q(n_0 + 1)$. Proof that $Q(n_0)$

Since $\forall n : Q(n)$ holds, it must be true that $Q(n_0)$.

Thus $Q(n_0)$

Thus $\forall n : Q(n + 1) \to Q(n)$

□

We can use theorem 33 to prove the following:

Theorem 34. $\forall n, m \in \mathbb{N} : n : std \wedge m : nstd \to (n + m) : nst$

Proof. Let $m = m_0 : nst$. Proof that $\forall n \in \mathbb{N} : n : std \to (n + m_0) : nst$

By induction on n.
Base case. Let $n = 0$. Proof that $(0 + m_0) : nst$

$0 + m_0 = m_0$
$m_0 : nst$

Thus $(0 + m_0) : nst$

Induction step. Proof that $[(n + m_0) : nst] \to [(n + 1 + m_0) : nst]$

Just apply theorem 33 to $n = (n + m_0)$.

Thus $[(n + m_0) : nst] \to [(n + 1 + m_0) : nst]$

Thus $\forall n \in \mathbb{N} : n : std \to (n + m_0) : nst$

□

It is notable that you can never reach a standard number when adding nonstandard numbers.

Theorem 35. $\forall n, m \in \mathbb{N} : n, m : nst \to (n + m) : nst$

Proof. We proceed by proving the equivalent $(n + m) : std \to (n : std \vee m : std)$ Suppose $(n + m) : std$ Proof that $(n : std \vee m : std)$

Without loss of generality, suppose $n : nst$ Proof that $m : std$

By contradiction. Suppose $m : nst$ Proof that this leads to a contradiction

We have already assumed that $(n + m) : std$.
Now, however, we also assume that $n, m : nst$.
Using theorem 33 however, we see that when $n, m : nst$, then it must be that $(n + m) : nst$.

Thus this leads to a contradiction

Thus $m : std$

Thus $(n : std \vee m : std)$

□

1.10.2 Limits

1.10.3 Sequences and series

1.10.3.1 Tailor

1.10.3.2 Fourier

1.10.3.3 Laplace

1.10.4 Euler's formula

Proof: Consider the function $f(t) = e^{-it}(cost + isint)$ for $t \in \mathbb{R}$. By the quotient rule

$$f'(t) = e^{-it}(i\ cos(t) - \sin(t)) - ie^{-it}(\cos(t) + i\sin(t)) = 0$$

identically for all $t \in \mathbb{R}$. Hence, f is constant everywhere. Since $f(0) = 1$, it follows that $f(t) = 1$ identically. Therefore, $e^{it} = cost + isint$ for all $t \in \mathbb{R}$, as claimed.

1.10.5 Integration

The infinitessimal view of calculus makes it quite easy to prove theorems. Consider this illustration of how definite integrals work.

$$\begin{aligned} \int_a^b \frac{dF}{dx}(x)dx &= \int_a^b \frac{F(x+dx) - F(x)}{dx}dx \\ \int_a^b F(x+dx) - \int_a^b F(x) &= F(b+dx) - F(a) \end{aligned} \tag{1.37}$$

1.10.5.1 Integration strategies

u-substitution

Integration by parts is the last trick up our sleave when all other strategies haven't helped. Consider the integral

$$\int xe^x \, dx$$

We can rewrite this integral as

$$\int u \, dv$$

where $u = x$ and $dv = e^x \, dx$. In general, you always want to pick u in such a way that u gets simpler after being differentiated. x does get a lot simpler after differentiation, whereas e^x doesn't, so the choice is clear.

We then use the following:

$$\int u \, dv = uv - \int v \, du$$

[3]

Since we chose $u = x$ we have $du = dx$, and from $dv = e^x \, dx$ we get $v = e^x$. This yields us:

$$xe^x - \int e^x \, dx$$

$$e^x(x-1)$$

as the solution.

1.10.6 Vector calculus

Integration over a vector, integration along a vector, integration along a surface.

You can find a nice introduction (mostly in the second part of) this pdf: http://www.maths.gla.ac.uk/~cc/2A/2A_notes/2A_chap4.pdf and here http://geocalc.clas.asu.edu/pdf-preAdobe8/SIMP_CAL.pdf

[3] The proof goes like this: Starting with the product rule of differentiation: $(ab)' = a'b + ba'$ we get $(ab)' - a'b = ba'$ and $ab - \int a'b \, dx = \int ba' \, dx$

1.11 Number theory and cryptography

1.11.1 Divisability

Definition 31. *Divisability:* $a|b \leftrightarrow \exists k : ak = b$

Theorem 36. $a|b \wedge b|c \rightarrow a|c$

Proof. Proof that $a|b \wedge b|c \rightarrow a|c$

> Suppose $\exists x_0 : ax_0 = b \wedge \exists x_1 : bx_1 = c$ Proof that $\exists x_2 : ax_2 = c$
>
> > Try $x_2 = x_0x_1$
> > Then $ax_2 = ax_0x_1 = bx_1 = c$
>
> Thus $\exists x_2 : ax_2 = c$

Thus $a|b \wedge b|c \rightarrow a|c$

□

Theorem 37. *If a divides both b and c, then a divides any linear combination of b and c as well.* $a|b \wedge a|c \rightarrow \forall v, w : a|(vb + wc)$

Proof. Proof that $a|b \wedge a|c \rightarrow \forall v, w : a|(vb + wc)$

> Suppose $a|b \wedge a|c$ Proof that $\forall v, w : a|(vb + wc)$
>
> > Let v_0, w_0 Proof that $\exists x : ax = (v_0b + w_0c)$
> >
> > > Try $x = v_0x_0 + w_0x_1$
> >
> > Thus $\exists x : ax = (v_0b + w_0c)$
>
> Thus $\forall v, w : a|(vb + wc)$

Thus $a|b \wedge a|c \rightarrow \forall v, w : a|(vb + wc)$

□

It is neat how $a|b$ can be read in two ways: 'a divides b', but also 'b is a multiple of a'. This is very nice for interpreting the water-jug problem.Let $s = \alpha j_1 + \beta j_2$. If there is an a such that $a|j_1 \wedge a|j_2$ then, by the above proof, s is a multiple of a. Of course, every number can be divided at least by 1. So we can more generally write, for any linear combination: $s = \sum \alpha_i j_i = \beta gcd(j_1, j_2, ...)$.As a consequence: with two water-jugs we can only measure quantities that are multiples of $gcd(j_1, j_2)$.

In more common terms, the above condition could be formulated as 'if the base-vectors have a common divisor'. As such, the above proof sheds some light on the relation between divisibility and linear combinations. If base-vectors have a common divisor, then they are linearly dependent (proof this for yourself, it's trivial). The inverse is not always true, however. Consider 2 and 3: they have no common divisor (other than 1), yet they are linearly dependent: $\frac{-3}{2}2 + 1 \cdot 3 = 0$

Theorem 38. $\forall a, b : \exists! q, r : a = qb + r \wedge 0 \leq r \leq q$

Theorem 39. *Euclids algorithm: We can simplify the greatest common divisor like this:* $gcd(a, b) = gcd(b, rem(a, b,))$

Proof. Proof that $\gcd(a, b) = \gcd(b, \text{rem}(a, b,))$

> Proof that $\text{CD}_{(a,b)} = \text{CD}_{(b,\text{rem}(a,b))}$
>
> > Let $d_0 \in \text{CD}_{(a,b)}$ Proof that Prove that $d_0 \in \text{CD}_{(b,\text{rem}(a,b,))}$
> >
> > > Since ...
> >
> > Thus Prove that $d_0 \in \text{CD}_{(b,\text{rem}(a,b,))}$
> > Let $d_0 \in \text{CD}_{(b,\text{rem}(a,b))}$ Proof that Prove that $d_0 \in \text{CD}_{(a,b)}$
> >
> > > Since ...
> >
> > Thus Prove that $d_0 \in \text{CD}_{(a,b)}$

Thus $\text{CD}_{(a,b)} = \text{CD}_{(b,\text{rem}(a,b))}$
Proof that Since $A = B \rightarrow \max(A) = \max(B)$, it follows that $\gcd(a, b, =)\gcd(b, \text{rem}(a, b, ,))$
It is easy to prove that $A \subseteq B \rightarrow \max(A) \leq \max(B)$. The reverse holds too, of course.
Thus Since $A = B \rightarrow \max(A) = \max(B)$, it follows that $\gcd(a, b, =)\gcd(b, \text{rem}(a, b, ,))$

Thus $\gcd(a, b) = \gcd(b, \text{rem}(a, b,))$

□

Proof. Proof that $\forall a, b \exists \alpha, \beta : \gcd(a, b) = \alpha a + \beta b$
The proof is not complicated but a little cumbersome. It is given in this stackexchange post
Thus $\forall a, b \exists \alpha, \beta : \gcd(a, b) = \alpha a + \beta b$

□

While we have now proven that the gcd can always be expressed as a linear combination of its arguments, we don't know yet just how to obtain the coefficients α and β. This is where the *extended* euclidian algorithm comes in.

1.11.2 Hashing

1.11.3 Encryption

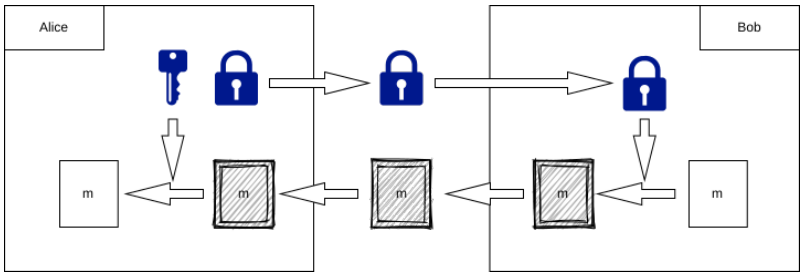
1.11.3.1 Symmetric encryption

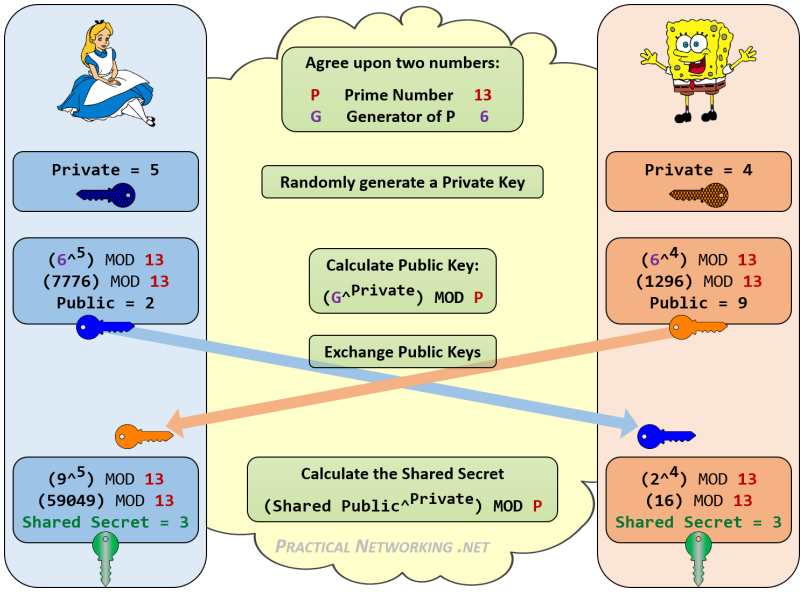
1.11.3.2 Asymmetric encryption

The basic idea of the Diffie-Hellman algorithm is as follows: instead of a secret key used for encryption and decryption, use a two-part-key. We can then find an algorithm where only the public part of the key is ever visible to Eve.

Here is an analogy:

- You have a lock (pubK) and a key (prvK). Send your lock publically to Bob, while you keep your key hidden in your vest.
- Bob writes his message to you and locks it with your lock in a box. Neither he nor Eve can now open the box again. Bob sends the box to you.
- You open the box with your key and read the message





This general concept is known as Diffie-Hellman asymmetric encryption. To implement it, one needs

- A function to generate a key-pair
- A function for encryption using the message and another persons public key
- A function for decryption using a cypher and your own private key

The first successful implementation of this principle is known as the WPA algorithm.

In practice WPA takes too long to be used very often, whereas symmetric encryption is much faster. For this reason, one usually follows a two step process:

- create a new symK
- exchange that symK via WPA
- use the symK to do symmetric encryption

1.12 Computation

1.12.1 Finite state machines

A FSM takes a series of inputs (a *sentence*), evaluates if the sentence is part of its language, and processes it by stepping through a few states to a final state.

The theory of FSM's does not really handle actions that are to be taken uppon transitions. In some implementations, uppon arriving at the final state, an (external) action is triggered based on that final state. In others, an action may be executed after any transition, not only after arriving at a final state.

It is important to note that the states of a FSM need not be one-dimensional. For example, in the die-hard problem, we model the system as a FDM where the state is a two-tupple; the first being the contents of a 3-liter jug, the second the content of a 5-liter jug. In such a situation it makes sense to draw the nodes of the transition function in a grid with the state-dimensions as the axes.

1.12.1.1 Definition

$$M = \{Q, q_0, F, \Sigma, \delta\} \tag{1.38}$$

where Q is the set of states, $F \subset Q$ is the set of final states, Σ is the alphabet of possible inputs, and δ is:

$$\delta(q, \sigma) = q' \tag{1.39}$$

$$\delta : Q \cdot \Sigma \to Q \tag{1.40}$$

$$\Delta(\vec{\sigma}) = \delta(\delta(\delta(q_0, \sigma_0), \sigma_1), ...) \tag{1.41}$$

It should be noted that in reality state machines may differ from this definition in two ways:

- For one, usually every state may be considered a legitimate final state.
- Also usually a output function is added. There are two main designs for output-functions:
  - Moore machine: an output is generated after every state transition. This is the most common design in hardware applications, and also used in my NID-backend.
  - Mealy machine: every state has its own behavior, which reacts on inputs. One of the possible reactions to an input may be to trigger a state change, but not neccessarily every input does cause a state change. This is the most common design in software-applications, especially where state-machines are used to model AI.

In both cases the output function is usually implemented as follows: A behavior is defined for each state. The output function takes the input and the current state as arguments, fetches the correct behaviour based on the current state, and lets the behavior handle the input. In code:

```
class FSM:
    State s

    Output handleInput(Input i):
        State ns = transitionFunction(s, i)
        s = ns  # Might be just the same state
        Output o = outputFunction(s, i)

    State transitionFunction(State s, Input i):
        Behavior b = getTransBehavior(s)
        return b.handleInput(i)

    Output outputFunction(State s, Input i):
        Behavior b = getOutputBehavior(s)
        return b.handleInput(i)
```

1.12.1.2 Getting the language of a given machine M

$$L(M) = \{\vec{\sigma} | \Delta(\vec{\sigma}) \in F\} \tag{1.42}$$

Since δ can be expressed as a graph, a version of theorem 28 may be used to get $L(M)$.

1.12.1.3 Getting the simplest machine for a given language

1.12.1.4 Buildung machines from simpler machines

1.12.1.5 Induction using the invariant principle

Often we want to prove that a certain desireable property $P(q)$ holds for any step along any sentence that goes into a FSM. This is something we use for Turing machines as well, and in fact in many other applications, like MCMC. We can prove that P holds at every step using the invariant principle.

1.12.2 Turing machines

A Turing machine is only moderately more complex than a FSM. Instead of stepping through each input in the given sentence from left to right, the machine can also chose (based on its state) to move back again, or to erase or change a letter[4].

However, there seems to be no such thing as a Turing-Machine-Design-Pattern. Instead, the TM is used solely as a model for computation. Contrary to the FSM the TM has no relevant practical implementation.

[4]We use "letter" and "input" as synonyms.

1.13 Dynamic Programming

Dynamic programming tries to solve problems by expressing the problem recursively: The sollution for some parameter-set $sol(para)$ equals a function of sollutions of other parameter-sets, like for example:

$$sol(para) = sol(para') + x$$

or

$$sol(para) = sol(para') + sol(para'')$$

or most generally:

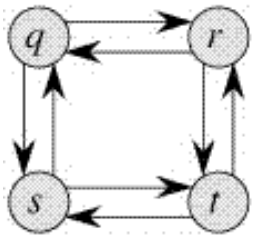
$$sol(para) = f(sol(para'), sol(para''), ...)$$

For performance, subsolutions are stored in a lookup-table. Without the lookup-table, dynamic programming would be called "divide and conquer".

Dynamic programming is a technique that solves some particular type of problem in polynomal time. Also, dp-sollutions can easily proven to be correct. But at first, we need to find out when dp-techniques apply.

1.13.1 Overlapping subproblems (always) and optimal substructure (most)

- Overlapping subproblems are those that make it worth to cache past results.
- Optimal substructure means this: if you have the solution to all parts of the problem, you get the sollution to the whole problem. Consider a graph of nodes a, b, c, d. To get from a to d, consider the subproblems of getting from a to b and from b to d. The shortest path from a to d is the shortest path from a to b plus the shortest from b to d. Note that the opposite problem does not have optimal substructure: the longest path from a to d is in fact shorter than the longest paths from a to b plus the longest from b to d.



1.13.2 Worked example: possible ways to sum coins

Imagine there was a currency with coins of value 1, 5 and 7. The task is to list all possible combinations of these basic coins to obtain a total value of n.

Here is an idea how this could be implemented: Lets call our function `combs(n)`. Add 1 to each list in the results of `combs(n-1)`, add 5 to each list in the results of `combs(n-5)`, and add 7 to each list in the results of `combs(n-7)`.

```
def combs(n):
        if n < 1:
                return []
        if n == 1:
                return [[1]]
        if n == 5:
                return [[1,1,1,1,1],[5]]
        if n == 7:
                return [[1,1,1,1,1,1,1],[5,1,1],[1,5,1],[1,1,5],[7]]

        list1 = combs(n-1)
        list5 = combs(n-5)
        list7 = combs(n-7)

        fullList = []
        for c in list1:
                fullList.append(c + [1])
        for c in list5:
                fullList.append(c + [5])
        for c in list7:
                fullList.append(c + [7])

        return fullList
```

Now just add some caching, and you have an efficient algorithm as well.
Verify that this idea shows overlapping subproblems and that it shows optimal substructure
Proof that algorithm delivers correct results
counterexample: minimal number of coins

1.13.3 A strategy for finding the recursion

Dynamic programming tries to solve problems by breaking them apart into subproblems. But how do we find those subproblems?

The solution S we are looking for should be obtained by some function sol.

$$S = sol(para)$$

ed problems
ead want to
several new
s S' and S''

Try to find a way to split S into a large part S' and a small part s .

$$S = S' + \{s\}$$

The large part itself must be the result of sol for some other set of parameters $para'$.

$$S' = sol(para')$$

1.13.4 Markow chains

1.13.5 Markov decision process with known state-transition-function

MDPs are an extension of Markov chains, they add actions (allowing choice) and rewards (giving motivation). Conversely, if only one action exists for each state (e.g. "wait") and all rewards are the same (e.g. "zero"), a Markov decision process reduces to a Markov chain.

An MDP is a tuple (S, A, P_a, R_a), where

- S: finite set of states
- A: finite set of actions
- $P_a(s, s') = P(s_{t+1} = s' | s_t = s, a_t = a)$
- $R_a(s, s')$ is the immediate reward received after transitioning from s to s' through action a.

The goal is to find a policy $\pi(s) = a$ that maximizes the cumulative reward $\sum_{t=0}^{\infty} \gamma_t R_{a_t}(s_t, s_{t+1})$, where γ_t is the discount-factor. The discount factor is used so that the decision maker favours taking actions early and doesn't postpone them indefinitely.

Because of the Markov property, the optimal policy for this particular problem can indeed be written as a function of s only, as assumed above.

1.13.6 Reinforcement learning: MDP without explicitly known state-transition-function

1.14 Distributed systems

We implement distributed systems when our system becomes too big to be run from a single computer. Distribution always makes things more complex, so there is no reason to implement it when you don't have to. But in almost every case, you eventually will. Most of the following text is based on this lecture series.

1.14.1 Model

- **Communication graph** G: a directed, two-way graph
  - nodes: nodes_G
  - edges: edges_G
  - $\forall (u, v) \in \text{edges}_G : (v, u) \in \text{edges}_G$
- **Subgraph** U_G:
  - nodes: nodes_U
  - edges: $\text{edges}_U = \{(u, v) | u \in \text{nodes}_U \wedge v \in \text{nodes}_U\}$
- **Inedge border** of U_G: $\{(u, v) | u \notin \text{nodes}_U \wedge v \in \text{nodes}_U\}$
- **System** $\mathcal{G}$:
  - a communication graph G
  - a program A for each node n of G (we say: n runs A) (in the original paper, a program was called a *device*)
  - input to each node n of G
  - behaviour $\mathcal{B}_\mathcal{G}$

1.14.2 Axioms

The locality axiom states that a subsystems behaviour is only influenced by its inedge border.

The fault axiom states that there exists a (malicious) program F that can mimic other programs, so that it behaves towards node a like X, and towards b like Y.

1.14.3 Theorems

1.14.3.1 Coverings

Define the *neighborhood* of node u as $\text{nbh}_u = \{v \in \text{nodes}_G | (v, u) \in \text{edges}_G\}$. The neighborhood of u is simply the inedge border of a one-node-graph $U := \{u\}$. A graph S *covers* G if there is a mapping ϕ from any node $s \in S$ to $u \in G$ that preservers neighborhood, that is:

$$\phi : \text{nodes}_S \to \text{nodes}_G$$

$$\forall s \in \text{nodes}_S : \phi(s) = w \to \phi(\text{nbh}_s) = \text{nbh}_w$$

Under such a mapping, S looks locally like G. This is trivially proven. By the locality axiom, the behavior of a subsystem is only influenced by its inedge border. That means that the behaviour of the system on u is only influenced by its neighborhood. Since every node $s \in \text{nodes}_S$ has a neighborhood identical to that of some $u \in \text{nodes}_G$, the behaviour of every node in S is identical to that of some node in G.

In practice, we use this finding for the following strategy: Consider a big graph S. In S, we consider a subgraph S'. We might not know a lot about the behaviour of S', but very likely, G', which is covered by S', is much simpler. Using the locality axiom we prove that S' has to behave like G'.

1.14.3.2 Byzantine agreement

Imagine three byzantine generals laying siege to an enemy city. They have to decide if they want to attack the city or not. An attack can only be successful if they all attack together, so they must agree on attack or abstain together. Each general has his own information on whether an attack, even with all three generals involved, will be successful or not. Finally, there is a traitor among them, who will do everything he can to make them decide on the wrong strategy. The two loyal (a.k.a. *correct*) generals follow some algorithm (like e.g. majority-vote, or seniority) for making their decision, while the traitor will do anything he wants. For byzantine agreement to be possible, the following must hold:

1. *Agreement*: The two loyal generals decide upon the same value, that is, whether to attack or not.
2. *Validity*: The decision is made soundly. If both generals think its a good idea to attack, the agreement must be on 'attack'; if both think that it isn't, then the agreement must be on 'abstain'. If one general thinks an attack would be smart while the other doesn't, this rule doesn't hold, and only item 1 holds.

In other words, if byzantine agreement were possible, it would mean that there is an one traitor cannot make the generals choose a bad strategy. However, we will proof that byzantine agreement is *not* possible.

We can easily proof this. We have three nodes u,v,w. In order to achieve validity, a correct node must decide on its own value if another node supports that value.The third node might disagree, but that node could be a traitor. If correct node u has input 0 and correct node v has input 1, the byzantine node w can fool them by telling u that its value is 0 and simultaneously telling v that its value is 1. This leads to u and v deciding on their own values, which results inviolating the agreement condition. Even if u talks to v, and they figure out that they have different assumptions about w's value, u cannot distinguish whether w or v is a traitor.

But there is another, more general proof, which we can use for the following theorems as well. The general idea of the proof is as follows.

1. Assume byzantine agreement (B.A.) is possible.
2. Consider S, a 6-n graph with inputs and programs, which we will construct in a cunning way. S does not need to exhibit B.A. It only needs to be free of contradictions.
3. Consider one flank S_1 of S. This flanks behaviour B_1 must be equal to that of a ceratain G, which exhibits B.A. and is covered by S_1.
4. Consider another, adjacent flank S_2 of S. This flanks behaviour B_2 must be equal to that of another G, and also be in accordance with B_1.
5. Consider a third, adjacent flank S_3 of S. This flanks behaviour B_3 must be equal to that of yet another G. However, B_3 will be a contradiction to B_2.
6. Thus, if B.A. were possible, S could not possibly exist. But it is obvious that S can exist.

1.14.3.3 Week agreement

1.14.3.4 Byzantine fireing squad

1.14.3.5 Approximate agreement

1.14.3.6 Clock syncing

1.14.3.7 When timing does and does not matter

Operations on a single machine have a total order, on distributed machines we can at least hope to acchieve a partial order. Consider a transition function from one state to another. When is a transition function commutative? That would mean that the timing in which input arrives at a state machine does not matter. We know this is not always true. Just consider the state function $f(s,x) = sx + x$.

$$f : S \times X \to S$$

Theorem:

$$f : \text{some property} \to f(f(s_0, x_0), x_1) = f(f(s_0, x_1), x_0)$$

1.14.3.8 flp theorem

System model consists of:

In an asynchronous system, a server-failure can alter the outcome This is

When you are at a bivalent state, you can delay a message such that you end up at a bivalent state again This means that you can forever postpone the system from ever reaching a conclusion.

1.14.3.9 CAP-Theorem

1.14.4 Important alogrithms

1.14.4.1 one-, two- and tree-phase commit

1.14.4.2 paxos algorithm

1.14.4.3 raft algorithm

1.14.4.4 map reduce algorithm

where in map reduce is the window of failure according to flp?

1.14.5 Databases

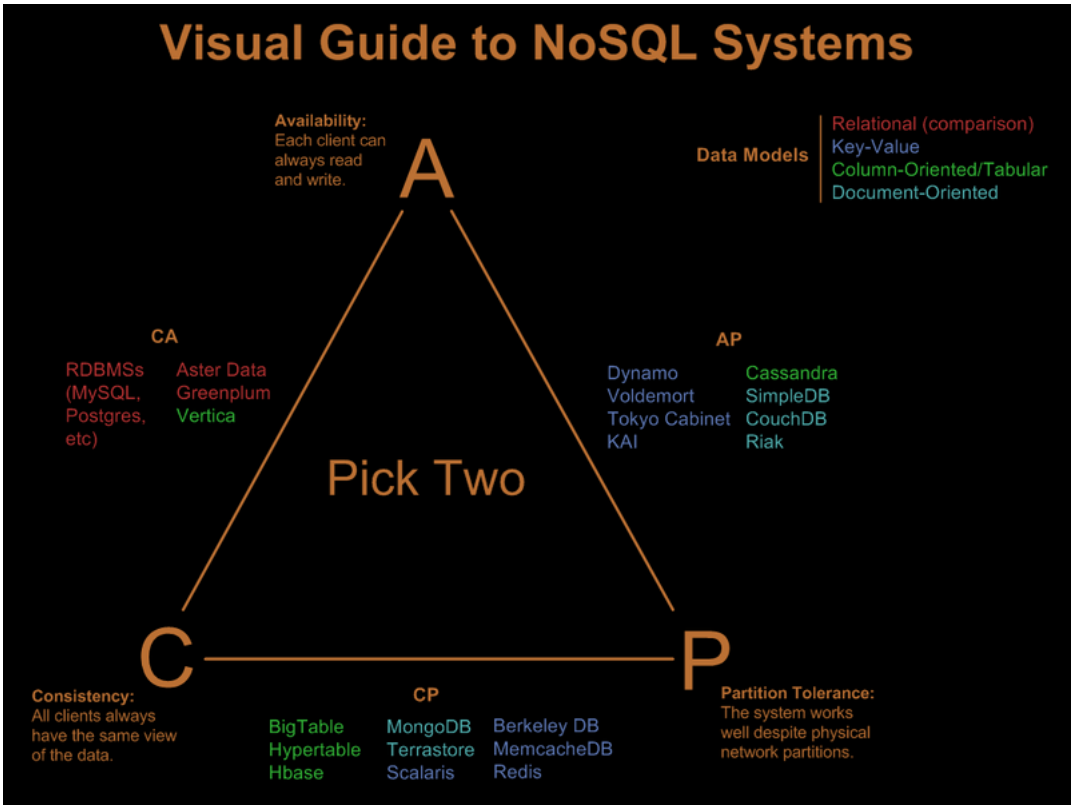
1.14.5.1 ACID

A transaction is a sequence of database operations that can be considered one independent, coherent unit of work. Database transaction should have the ACID-properties:

- Atomicity
- Consistency
- Isolation
- Durability

If a distributed database provides the acid-properties, then it must chose consistency over availability according to the cap theorem. A available database cannot provide acid-transactions.

1.14.5.2 NoSQL Database types



1.14.5.3 Normalisation

Normalisation is a way of structuring a relational database such that logical errors are minimized.

```
from DB import DB


class DictOfSets:
    data = {}

    def addData(self, d):
        for row in d:
            if row[0] in self.data:
                self.data[row[0]].add(row[1])
            else:
                self.data[row[0]] = set([row[1]])

    def addVals(self, v):
        for row in v:
            if row[0] in self.data:
                self.data[row[0]].add(row[1])

    def addKeys(self, k):
        for key in k:
            if not key in self.data:
                self.data[key] = set()



class DBAbstr:
    host = '10.173.202.110'
    user = 'lesen'
    password = ''
    database = 'gkd_chemie'

    def doQuery(self, query):
        result = []
        with DB(self.host, self.user, self.password, self.database) as db:
            result = db.queryToArray(query)
        return result
```

```
class SetTable(DBAbstr):
    vals = []

    def __init__(self, tableName, key):
        self.tableName = tableName
        self.key = key

    def getVals(self):
        if self.vals:
            return self.vals
        query = "select {0} from {1} group by {0}".format(self.key, self.tableName)
        result = self.doQuery(query)
        self.vals = result
        return result


class RelTable(DBAbstr):
    leftVals = DictOfSets()
    rightVals = DictOfSets()

    def __init__(self, tableName, setA, translKeyA, setB, translKeyB):
        self.tableName = tableName
        self.setA = setA
        self.translKeyA = translKeyA
        self.setB = setB
        self.translKeyB = translKeyB

    def getLeftVals(self):
        if self.leftVals.data:
            return self.leftVals.data
        query  = "select a." + self.setA.key + ", c." + self.setB.key + " "
        query += "from " + self.setA.tableName + " as a "
        query += "left join " + self.tableName + " as b on b." + self.translKeyA + " = a." + self.setA.key + " "
        query += "left join " + self.setB.tableName + " as c on c." + self.setB.key + " = b." + self.translKeyB + " "
        query += "group by a." + self.setA.key + ", c." + self.setB.key
        result = self.doQuery(query)
        self.leftVals.addData(result)
        return result

    def getRightVals(self):
        if self.rightVals.data:
            return self.rightVals.data
        query  = "select a." + self.setB.key + ", c." + self.setA.key + " "
        query += "from " + self.setB.tableName + " as a "
        query += "left join " + self.tableName + " as b on b." + self.translKeyB + " = a." + self.setB.key + " "
        query += "left join " + self.setA.tableName + " as c on c." + self.setA.key + " = b." + self.translKeyA + " "
        query += "group by a." + self.setA.key + ", c." + self.setB.key
        result = self.doQuery(query)
        self.rightVals.addData(result)
        return result




class Relation:
    vals = DictOfSets()      # x  ->  f(x) = y
    invVals = DictOfSets()   # y  ->  f^-1(y) = x

    def __init__(self, setTableX, setTableY, relTable):
        self.setTableX = setTableX
        self.setTableY = setTableY
        self.relTable = relTable

    def checkTransitive(self):
        pass

    def checkBijective(self):
        """ One-to-one and onto """
        return self.checkSurjective and self.checkInjective

    def checkSurjective(self):
        """ Onto: For any y in Y, there is a x in X: y = f(x) """
        invValsData = self.getInvValsData()
        for y in invValsData:
            if len(invValsData[y]) == 0:
                return False
        return True

    def checkInjective(self):
        """  Ont-to-One: For any a, b in X: f(a) = f(b) => a = b """
        invValsData = self.getInvValsData()
        for y in invValsData:
            if len(invValsData[y]) > 1:
```

	S2	H3
info	Quad-tree with indices (aka. geohash) along Hilbert-curve so that neighbors have similar index	Nested hexagons
computational geometry		All neighbors equidistant
shardable		
visual aspect	Squares distort a lot close to poles	Pretty!

```
                return False
        return True


    def compose(self, rel2):
        pass


    def getValsData(self):
        if  self.vals.data:
            return self.vals.data
        xdata = self.setTableX.getVals()
        self.vals.addKeys(xdata)
        reldata = self.relTable.getLeftVals()
        self.vals.addVals(reldata)
        return self.vals.data


    def getInvValsData(self):
        if self.vals.data:
            return self.vals.data
        ydata = self.setTableY.getVals()
        self.invVals.addKeys(ydata)
        reldata = self.relTable.getRightVals()
        self.invVals.addVals(reldata)
        return self.invVals.data



if __name__ == "__main__":
    pnstMappen = SetTable('gkd_chemie.LIMNO_PNST_MAPPE', 'MAPPE_NR')
    messnetze = SetTable('gkd_chemie.LIMNO_SL_PNST_MAPPE_TYP', 'TYP_NR')
    pnstMappePnst = RelTable('gkd_chemie.LIMNO_ZT_PNST_MAPPE_PNST', pnstMappen, 'MAPPE_NR', messnetze, 'MN_NR')
    mappenMessnetzRel = Relation(pnstMappen, messnetze, pnstMappePnst)
    isInj = mappenMessnetzRel.checkInjective()
    isSur = mappenMessnetzRel.checkSurjective()
    print isInj
    print isSur
```

1.14.5.4 Indices

In a database, you want to put indices on those columns that you are likely to read from / write to the most.

1.14.5.5 Spatial indices

For spatial data, the columns that you are likely to access the most are

- lon/lat for point-data
- bounding-boxes for polygons

You can select many types of indices for this kind of data. All of them are good for some operations and bad for others.

Common choices are S2 and H3. Both of these system provide a way to turn a coordinate on earth into a cell id that maps to a hexagonal or rectangular grid cell at a specific resolution. These cell ids have unique properties such as nearby cells have similar ids and you can find parent cells by truncating their length. These properties make operations such as aggregating data, finding nearby objects, measuring distances very fast.

1.14.6 Distributed cache: redis

1.15 Machine learning

1.15.1 Neural networks

1.15.1.1 Backpropagation

The analytical way of deriving the backpropagaion algorithm consist of just a few steps. A few definitions:

- A layers outputvector $\vec{y}^l$ is obtained by the activation function $\vec{y}^l = f(\vec{x}^l)$
- The layers inputvector is obtained as a weighted sum of previous outputs: $\vec{x}^l = \mathbf{W}^l \vec{y}^{l-1}$. We cn express a single $x_t^l = \sum_f W_{t,f}^l y_f^{l-1}$
- We strive to minimize the error-function. Assuming only one single training item we get $e = \frac{1}{2}\sum_t (\vec{y}_t^* - \vec{y}_t^L)^2 = \frac{1}{2}(\vec{y}^* - \vec{y}^L) \odot (\vec{y}^* - \vec{y}^L)$

Let's first consider only the top layer.

$$\begin{aligned}\frac{de}{dx_{t_0}^L} &= \frac{1}{2}\sum_t \frac{d}{dx_{t_0}^L}(\vec{y}_t^* - \vec{y}_t^L)^2 \\ &= (y_{t_0}^* - y_{t_0}^L) f'(x_{t_0})\end{aligned}$$

Or, in vector form:

$$\frac{de}{d\vec{x}^L} = (\vec{y}^* - \vec{y}^L)^T \odot f'(\vec{x}^L)$$

That part was easy. But how do we obtain the same differential for *any* layer l?

$$\begin{aligned}\frac{de}{dx_{f_0}^l} &= \sum_t \frac{de}{dx_t^{l+1}} \frac{dx_t^{l+1}}{dx_{f_0}^l} \\ &= \sum_t \frac{de}{dx_t^{l+1}} \frac{d}{dx_{f_0}^l}(\sum_f W_{t,f}^{l+1} y_f^l) \\ &= \sum_t \frac{de}{dx_t^{l+1}} W_{t,f_0}^{l+1} f'(x_{f_0}^l)\end{aligned}$$

Or, in vector form:

$$\frac{de}{d\vec{x}^l} = (\frac{de}{d\vec{x}^{l+1}} \mathbf{W}^{l+1}) \odot f'(\vec{x}^l)$$

The smart part here was to not derive $\frac{de}{d\vec{x}^l}$ by going through $\vec{y}^L$, $\vec{x}^L$, $\mathbf{W}^L$, $\vec{y}^{L-1}$, $\vec{x}^{L-1}$, $\mathbf{W}^{L-1}$, ..., but by instead creating a recurrence relation by differentiating by $\vec{x}^{l+1}$.

Finally, we can obtain the gradient at our weights as:

$$\begin{aligned}\frac{de}{dW_{t_0,f_0}^l} &= \frac{de}{dx_{t_0}^l} \frac{dx_{t_0}^l}{dW_{t_0,f_0}^l} \\ &= \frac{de}{dx_{t_0}^l} \frac{d}{dW_{t_0,f_0}^l}(\sum_f W_{t_0,f}^l y_f^{l-1}) \\ &= \frac{de}{dx_{t_0}^l} y_{f_0}^{l-1}\end{aligned}$$

Or, in vector form:

$$\frac{de}{d\mathbf{W}^l} = \left(\frac{de}{d\vec{x}^l}\right)^T \left(\vec{y}^{l-1}\right)^T$$

So we should change the weigths by:

$$\Delta \mathbf{W}^l = -\alpha \frac{de}{d\mathbf{W}^l}$$
$$= -\alpha \frac{de}{d\vec{x}^l} \vec{y}^{l-1}$$

It makes sense to reiterate the whole process in matrix-form.
First, we get δ^L:

$$\frac{de}{d\vec{y}^L} = (\vec{y}^* - \vec{y}^L)^T := \delta^L$$

Then we go through the highest layer:

$$\frac{de}{d\vec{x}^L} = \delta^L \odot f'(\vec{x}^L)$$
$$\frac{de}{d\mathbf{W}^L} = \left(\frac{de}{d\vec{x}^L}\right)^T \left(\vec{y}^{L-1}\right)^T$$
$$\delta^{L-1} = \frac{de}{d\vec{x}^L} \mathbf{W}^L$$

Then we pass δ^{L-1} to the next layer.

$$\frac{de}{d\vec{x}^l} = \delta^l \odot f'(\vec{x}^l)$$
$$\frac{de}{d\mathbf{W}^l} = \left(\frac{de}{d\vec{x}^l}\right)^T \left(\vec{y}^{l-1}\right)^T$$
$$\delta^{l-1} = \frac{de}{d\vec{x}^l} \mathbf{W}^l$$

1.15.1.2 Universial approximation

Let f be a function mapping images to labels. f stems from a vector space of functions $\mathscr{F}$. Let B be a basis for $\mathscr{F}$, meaning that

$$\forall f \in \mathscr{F} : \exists \vec{\alpha} : \sum \alpha_n b_n = f$$

So far, so simple. This holds for any basis of any vector space. Let's just propose that sigmoid functions do constitute a basis for these image-to-label functions. We cannot prove this, since we don't know what the image-to-label functions look like, but notice the potential: $\sum \alpha_n b_n$ is then just the output of one layer of a neural net!

It turns out that sigmoid functions do indeed form a basis for any continuous function on $[0,1]^n$ [5].

There is an important point that the universal approximation theorem does not cover, however. The UAT only deals with a single layer net. We know from practice, however, that a multilayer net can approximate functions with far less nodes than what a single-layer net would need. There is some strength to having multiple layers.

1.15.1.3 Some data in n dimensions requires more than n neurons in a layer

The backpropagation algorithm requires a networks layers to transform its input data in a continous fashion, i.e. distort the input surface without cutting it at any point. In topology, such a transformation is known as a homomorphism.

1.15.1.4 Some functions can be better approximated with a deep net than with a shallow one

Consider the case of a hierarchical function.

$$f(x_1, x_2) = h_2(h_1 1(x_1), h_1 2(x_2))$$

We will prove that a deep net needs less neurons than a shallow one to approximate this function.

[5] Note also that, while sigmoids do form a basis, they do not constitute an *orthagonal* basis, meaning that we cannot obtain waights with the inner-product-trick. We couldn't have obtained them anyway, because for that trick we need the analytical form of f, which is generally not known to us.

1.15.1.5 Convolutional networks

These are the networks most commonly found employed in image-classification. Really, they are just a simplified version of our backpropagation-networks (they are even trained using an only slightly altered algorithm). Instead of connecting every node from layer l to every node of layer $l+1$, they impose some restrictions on the connection-matrix:

- Nodes are connected in a pyramid scheme. A node on layer $l+1$ is connected to 9 nodes directly beneath it. Instead of a $n_l \times n_{l+1}$ connection matrix, we thus have several 9×1 matrices.
- The connection-strengths of these 9×1 matrices are all the same - so really there is only just one 9×1 matrix.

These restrictions are inspired by the physiology of the visual cortex. They have the nice effect that a network is trained much faster, since they massively reuce the ammount of weights that need to be learned.

In practice, such networks have a few convolutional layers to reduce the dimension of the input-images, followed by a few conventional, fully connected layers that learn some logic based on the reduced images.

We give the backpropagation-steps for convolutional layers and pooling layers.

In convolutional layers, the forward step goes:

$$\vec{x}^l = \vec{y}^{l-1} \circledast \vec{w}^l$$
$$\vec{y}^l = f(\vec{x}^l)$$

Where the convolution is defined (in our simplified case) as:

$$(\vec{y} \circledast \vec{w})_n = \sum_{m=-1}^{1} \vec{y}_{n+m} \vec{w}_m$$

Differentiating a convolution is unfamiliar, but not too hard:

$$\frac{d(\vec{y} \circledast \vec{w})}{d\vec{w}} = \begin{bmatrix} 0 & y_0 & y_1 \\ y_0 & y_1 & y_2 \\ y_1 & y_2 & y_3 \\ ... & & \\ y_{l-1} & y_l & 0 \\ & & \end{bmatrix} := tr(\vec{y})$$

$$\frac{d(\vec{y} \circledast \vec{w})}{d\vec{y}} = \begin{bmatrix} w_0 & w_1 & 0 & ... \\ w_{-1} & w_0 & w_1 & ... \\ 0 & w_{-1} & w_0 & ... \\ ... & & & \\ 0 & ... & w_{-1} & w_0 \\ & & & \end{bmatrix} := br(\vec{w})$$

Accordingly, the backwards step goes:

$$\frac{de}{d\vec{x}^l} = \delta^l \odot f'(\vec{x}^l)$$
$$\frac{de}{d\vec{w}^l} = \frac{de}{d\vec{x}^l} tr(\vec{y}) = (\sum_{n=1}^{l} e'_{x_n} y_{n+1}, \sum_{n=0}^{l} e'_{x_n} y_n, \sum_{n=0}^{l-1} e'_{x_n} y_{n+1})$$
$$\delta^{l-1} = \frac{de}{d\vec{x}^l} br(\vec{w}) = \frac{de}{d\vec{x}^l} \circledast \vec{w}$$

In pooling layers, the forward step goes:

$$x_t^l = \frac{1}{4} \sum_f y_{4t+f}^{l-1}$$
$$y_t^l = x_t^l$$

And the backwards step:

$$\frac{de}{d\vec{x}^l} = \frac{de}{d\vec{y}^l}\frac{d\vec{y}^l}{d\vec{x}^l} = \delta^l$$

$$\frac{de}{d\vec{w}^l} = 0$$

$$\delta^{l-1} = \frac{1}{4}\frac{de}{d\vec{x}^l}$$

1.15.1.6 Self organizing maps

Self organizing maps are another, fundamentally different type of neural network. Where feedforward nets employ supervised learning with backpropagation, SOM's do unsupervised learning with a competitive algorithm.

1.15.2 Computer vision

Feature detection and pose estimation Say you want to locate the position of the nose in a portrait.

1.15.3 Feature extraction and dimensionality reduction

The art of preprocessing input has developed in a branch of machine learning itself. Classifiers like SVM's and nnets work better when they get cleaned up and encoded input instead of raw data.

1.15.4 Symbolic AI

Contrary to the before mentioned approaches, symbolic AI uses logical deduction instead of numerical processing to arrive at decisions. If neural nets and decision-trees learning from data can be called building *experience*, then an inference engine deducting from rules can be called building *expertise*. A good tutorial can be found here: codeproject.com. A inference engine can do the following:

- Learning:
  - Gather if-then statements (usually in the form of Horn-clauses).
  - Create a graph of dependencies between statements (a so called and-or-tree).
- Applying the learned things: Given a question:
  - Possible answers: find all potential answers to the problem
  - Backward pass: go through the graph to see what data is required
  - Data aquisition: ask user to provide the data
  - Forward pass: trace the graph forward again to arrive at a single one of the possible answers

Depending on how much effort you put into the expressions that the engine can understand, we differentiate between different levels of logic:

- 0th order logic: understands simple facts and chains them using modus ponens
- 1st order logic: understands variables: all of naive math can be written in 1st order logic.
- higher order logic: is better at inference; can create new if-then-statements as a result of inference or even explore

Popular expert-system-libraries are Prolog, CLIPS and Pyke.
There are many variants to how you can write an expert system. The most important variables are

- How are rules parsed? Do we allow for other logical connectives than 'AND'? See https://medium.com/a-42-journey/expert-systems-how-to-implement-a-backward-chaining-resolver-in-python-bf7d8924f72f.
- How is inference done? Forward pass on `addFact` or backward pass on

, or a mixture?

- How is pattern-matching done? The Rete-algorithm is very popular for this.
- Are metaheuristics used?
  - Does the engine try to infer more general rules while idle?
  - Does the engine keep track if a search down one branch takes very long (needs a watching strategy-module)?

```
class InferenceEngine:
    def __init__(self):
        self.facts = []
        self.rules = []

    def addFact(self, *fact):
        if fact not in self.facts:
            self.facts.append(fact)  # add the fact
            print(f"Just learned that {fact}")
        for arg in fact[1:]:
            if not self.__isVariable(arg):
                self.addFact(arg) # add all arguments

    def addRule(self, conditions, consequence):
        newRule = {'conditions': conditions, 'consequence': consequence}
        if not newRule in self.rules:
            self.rules.append(newRule)
            print(f"Just learned that {newRule}")

    def eval(self, *statement):
        print(f"Evaluating whether '{statement}' holds true")
        fact = self.__searchFacts(*statement)
        if fact:
            return fact
        fact = self.__tryToProve(*statement)
        if fact:
            return fact
        print(f"No, '{statement}' holds not true")
        return False

    def __searchFacts(self, *statement):
        for fact in self.facts:
            if self.__statementsMatch(fact, statement):
                return fact
        return False

    def __tryToProve(self, *statement):
        # a professional inference engine would probably use Rete here
        candidateRules = self.__findCandidateRules(*statement)
        for candidateRule in candidateRules:
            substitutedRule = self.__setRuleVariablesByStatement(candidateRule, statement)
            facts = self.__evalAll(substitutedRule['conditions'])
            if facts:
                substitutedRule = self.__setRuleVariablesByStatements(substitutedRule, facts)
                self.addFact(*substitutedRule['consequence'])
                return substitutedRule['consequence']
        return False

    def __evalAll(self, statements):
        facts = []
        for statement in statements:
            fact = self.eval(*statement)
            if not fact:
                return False
            else:
                facts.append(fact)
        return facts

    '''
    -----------------  helpers -------------------------
    '''

    def __setRuleVariablesByStatements(self, rule, statements):
        newRule = dict(rule)
        for statement in statements:
            newRule = self.__setRuleVariablesByStatement(newRule, statement)
        return newRule

    def __setRuleVariablesByStatement(self, rule, statement):
        substitutionDict = {}
        for ruleArg, stateArg in zip(rule['consequence'][1:], statement[1:]):
```

```
            if self.__isVariable(ruleArg):
                substitutionDict[ruleArg] = stateArg
        return self.__setRuleVariablesByDict(rule, substitutionDict)

    def __setRuleVariablesByDict(self, rule, subsDict):
        newRule = {'conditions': [], 'consequence': None}
        for condition in rule['conditions']:
            newCondition = self.__setVariablesByDict(condition, subsDict)
            newRule['conditions'].append(newCondition)
        newConsequence = self.__setVariablesByDict(rule['consequence'], subsDict)
        newRule['consequence'] = newConsequence
        return newRule

    def __setVariablesByDict(self, statement, subsDict):
        newStatement = [statement[0]]
        for arg in statement[1:]:
            if self.__isVariable(arg) and arg in subsDict:
                newStatement.append(subsDict[arg])
            else:
                newStatement.append(arg)
        return newStatement

    def __findCandidateRules(self, *statement):
        candidates = []
        for rule in self.rules:
            if self.__statementsMatch(rule['consequence'], statement):
                candidates.append(rule)
        return candidates

    def __statementsMatch(self, statementOne, statementTwo):
        for wordOne, wordTwo in zip(statementOne, statementTwo):
            oneIsVal = not self.__isVariable(wordOne)
            twoIsVal = not self.__isVariable(wordTwo)
            if oneIsVal and twoIsVal: # ? (oneIsVar and twoIsVar) or (not oneIsVar and not twoIsVar):
                if wordOne != wordTwo:
                    return False
        return True

    def __isVariable(self, arg):
        return isinstance(arg, str) and arg[0].isupper()




if __name__ == '__main__':
    e = InferenceEngine()

    # Test 0: obligatory Socrates test
    e.addRule([
        ['man', 'X']
    ], ['mortal', 'X'])
    e.addFact('man', 'socrates')
    print(e.eval('mortal', 'Y'))
```

Chapter 2

Algorithms and data-structures

2.1 General algorithm theory

2.1.1 Invariant principle

The invariant principle is not much more than using induction to prove that some statement holds over all iterations n from 0 to n_0.

2.1.2 Well ordering

The well ordering principle is a fundamental theorem in discrete mathematics.

$$\forall S \subset \mathbb{N}^+ : S \neq \emptyset : \exists x_0 \in S : \forall x \in S : x_0 \leq x$$

We usually use it in existence proofs, by contradiction. However, there is a nice little template for proofs using the well ordering principle.

To prove that $P(n)$ holds true for $\forall n \in \mathbb{N}$:

- Define the set C of counterexamples to P being true. That is:
$$C = \{n \in \mathbb{N} | \neg P(n)\}$$
- For a proof by contradiction, assume C is nonempty.
- By the well ordering principle, there will be a smallest element $n_0 \in C$
- Reach a contradiction (often by showing how to use n_0 to find another member of $n_1 \in C$ that is even smaller than n_0)

2.1.3 Program verification according to Floyd

If the following properties are given, then an algorithm is be correct.

- Partial correctness: if an answer is returned, it will be correct.
- termination: The algorithm eventually reaches a halt - meaning that an answer is always returned eventually.

There is a simple method of proving an algorithm correct (by Floyd):

- Proving 'partial correctness' with invariant principle: there is some loop invariant that basically says: 'At this iteration the state is a little better than before'. For example, in a sorting problem, at the i'th iteration the subarray `data[0:i]` might already be sorted.
- Proving 'termination' with well ordering principle

Note that there might be correct algorithms that don't have these properties. For example, some algorithms might eventually reach a desired state without ever having an ever-imporving loop invariant. Also, there are other ways to prove an algorithm correct than the Floyd-method. However, in many real-world examples, this is the easiest approach to formaly proving an algorithm correct.

2.1.3.1 Proving 'partial correctness' with invariant principle

2.1.3.2 Proving 'termination' with well ordering principle

2.1.4 Notation

An algorithm that takes $f(n)$ cycles is said to take $\Theta(g(n))$ time, iff:

$$f \in \Theta(g(n))$$

where

$$\Theta(g(n)) = \{f(n) | \exists c_1, c_2, n_0 : \forall n \geq n_0 : 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$$

Similarly, the set $O(g(n))$ is defined as:

$$O(g(n)) = \{f(n) | \exists c_2, n_0 : \forall n \geq n_0 : 0 \leq f(n) \leq c_2 g(n)\}$$

That is, $O()$ only has an upper bound, but no lower bound.

2.1.5 Order of basic operations

In the integer ram model of computation, addition, multiplication, remainder and bitwise operations all take O(1).

2.1.6 Optimization

- *Linear programming*: Minimizing f (linear) subject to constraints (linear, inequalities)
- *Lagrange mulitpliers*: Minimizing f (non-linear) subject to constraints (non-linear, equalities)
- *Non-linear programming*: Minimizing f (non-linear) subject to constraints (non-linear, inequalities)
  - f and constraints are differentiable: use KKT (a generalization of Lagrange)
  - f and constraints are convex: use convex optimization
  - All other cases: use some form of gradient descent (simulated annealing, MCMC, ...)

2.1.7 Memoization: Remembering the state in loops

Sometimes, operations have to be repeatedly executed in a loop. These operations may take $O(g)$ time by themselves, but only $O(1)$ time if the result of the previous execution is already known.

In such a case it makes sense to keep track of the previous result. StateMachines are a natural vehicle to store such information.

Lets assume for now that modulo was not constant time. Then we could do the infamous FizzBuzz problem like this in linear time:

```
class SM:
    def __init__(self, nr):
        self.nr = nr
        self.nToGo = nr

    def feed(self):
        self.nToGo -= 1
        if self.nToGo < 0:
            self.nToGo = self.nr - 1

    def isMult(self):
        return self.nToGo == 0



threeM = SM(3)
fiveM = SM(5)
for i in range(1, 16):
    threeM.feed()
    fiveM.feed()
    if threeM.isMult() and fiveM.isMult():
        print("{} --> FizzBuzz".format(i))
    elif threeM.isMult():
        print("{} --> Fizz".format(i))
    elif fiveM.isMult():
        print("{} --> Buzz".format(i))
    else:
        print(i)
```

2.1.8 Using sub problems: Dynamic programming

Solve a multi-step-problem by solving its sub-problems[1]. Such problems usually use recursion and memoization.

Example: Towers of Hanoi : You have three rods A, B and C. Move a tower of (say) 4 disks from A to C using the rules:

- move one disk at a time
- no larger disk may be placed on a smaller disk

Solution:

[1]Note that the optimal solution to a multi-step-problems is *not always* equal to the sum of the optimal solutions of its sub-problems. Those weird problems are said to *not have optimal substructure*. Example: longest path problem.

```
def move([4, 3, 2, 1], from: 'A', to: 'C', via: 'B'):
  return
      move([3, 2, 1], from: 'A', to: 'B', via: 'C')
    + move([4],       from: 'A', to: 'C', via: 'B')
    + move([3, 2, 1], from: 'B', to: 'C', via: 'A')
```

The hardest part in dynamic programming is usually finding the right mental model to approach a problem. Often, there are at least two ways to look at a problem, and the less obvious one is the one that is actually useful. Here's a list of ways to think of problems that was useful to me in the past:

- **decision-tree** - example: create all subsets of a set
- **graph** - example: count ways to climb n stairs by using 1, 2, or 3 stairs at a time
- **binary search** - example: find a magic index (`i = a[i]`) in an ordered array. (Binary search: go to middle, then go to middle of left half, then go to middle of right half, ...)

2.1.9 Avoiding the need for dynamic programming: Solving recurrences to closed form

Sometimes we don't have to do a recursive solution at all. Many recursive problems can be reshaped into a explicit form (the so called closed form). Think of the Fibonacci-sequence:

$$fib(n) = fib(n-1) + fib(n-2) = \frac{(1+\sqrt{5})^n - (1-\sqrt{5})^n}{2^n\sqrt{5}}$$

This section will exemplify some methods for finding the closed form expression.

2.1.9.1 Solving linear recurrences

A homogeneous linear recurrence is one of the following form:

$$f(n) = a_1 f(n-1) + a_2 f(n-2) + ... + a_d f(n-d)$$

We can solve it as follows:

1. Assume $f(n) = x^n$. We now try to find an expression for x.
2. Divide both sides in the hlr by x^{n-d}, leaving a (hyper-)quadratic equation.
3. Every root of the quadratic equation is a **homogeneous solution**. Also, if a root r occures v times, $r^n, nr^{n-1}, n^2r^n ..., n^{v-1}r^n$ are also solutions.
4. Also, every linear combination of the above is also a solution. So, a solution might in general have a form like $ar_1^n + br_2^n + ...$
5. Finally, choose $a, b, ...$ such that they fulfill the boundary conditions (in Fibonacci those would be $f(0) = f(1) = 1$). This is the **concrete solution**.

We can extend the above schema to also solve (nonhomogeneous) linear recurrences:

$$f(n) = a_1 f(n-1) + a_2 f(n-2) + ... + a_d f(n-d) + g(n)$$

1. Ignore $g(n)$, find the homogeneous solution from step 4 above.
2. Find a **particular solution** to the equation including $g(n)$.
3. homogeneous solution + particular solution = **general solution**
4. Now for the general solution, again plug in the boundary conditions as in step 5 above.

2.2 Important algorithms

2.2.1 Merge-sort

The function merge takes two already sorted lists and merges them into one sorted list.

```
def merge(a, b):
    c = []
    ia = ib = ic = 0
    while len(c) < len(a) + len(b):
        if a[ia] <= b[ib]:
            c[ic] = a[ia]
            ia++
        else:
            c[ic] = b[ib]
            ib++
        ic++
    return c

def sort(arr):
  l = len(arr)
  if l == 1:
    return arr
  sorted1 = sort(arr[:l/2])
  sorted2 = sort(arr[l/2:])
  return merge(sorted1, sorted2)
```

$Q(i)$ **:** statement

Initial state $Q(0)$ **:** sometext

Transition $Q(i) \to Q(i+1)$ **:** sometext

Final state $Q(n)$ **:** sometext

The **runtime** of the algorithm is $O(n \log n)$.

2.2.2 Insertion-sort

The idea is to go through a list from left to right. Look at all entries left of your current position. Move your current value back down the left list until it is sorted in. Go one step further right.

```
  def insertionSort(arr):
    # Traverse through 1 to len(arr)
    for i in range(1, len(arr)):
        pivot = arr[i]
        # Move elements of arr[0..i-1], that are
        # greater than pivot, to one position ahead
        # of their current position
        j = i-1
        while j >=0 and pivot < arr[j] :
                arr[j+1] = arr[j]
                j -= 1
        arr[j+1] = pivot
```

The **runtime** of the algorithm is $O(n^2)$ in the worst case, but $O(n)$ for almost sorted lists. That makes insertion sort great if you have to apply sort over and over again.

2.2.3 Matrix inverse

Gauss Jordan

2.2.4 Polynomials

In this section, we will deal with the polynomials $P_A(x) = 1x^3 + 2x^2 + 3x + 4$. Working with polynomials turns out to be quite important in everyday practice - especially when you're working with generating functions.

2.2.4.1 Coefficient-representation of polynomals

The most common representation of this polynomial is its coefficient representation $[4, 3, 2, 1]$. This representation is very convenient for fast evaluation at any x using Horner's method:

```c
#include "stdio.h"
#include "stdlib.h"
#include "math.h"

/**
 * Coefficient representation of polynomals
 */

typedef struct PolynomalC {
  int length;
  int* coefficients;
} PolynomalC;

PolynomalC* polyc_create(int l, int* coeffs) {
  PolynomalC* p = malloc(sizeof(PolynomalC));
  p->length = l;
  p->coefficients = malloc(l*sizeof(int));
  for(int i = 0; i<l; i++){
    p->coefficients[i] = coeffs[i];
  }
  return p;
}

void polyc_delete(PolynomalC* p) {
  free(p->coefficients);
  free(p);
}

int polyc_evalAt(PolynomalC* p, int x) {
  int sum = 0;
  for(int i = p->length; i > 0; i--){
    sum = sum * x + p->coefficients[i-1];
  }
  return sum;
}

PolynomalC* polyc_add(PolynomalC* p1, PolynomalC* p2) {
  int l1 = p1->length;
  int l2 = p2->length;
  int l = max(l1, l2);
  int coeffs[l];
  for(int i = 0; i < l; i++){
    int v = 0;
    if(i < l1) v += p1->coefficients[i];
    if(i < l2) v += p2->coefficients[i];
    coeffs[i] = v;
  }
  PolynomalC* p3 = polyc_create(l, coeffs);
  return p3;
}

int main(void) {

  int coeffs1[4] = {4, 3, 2, 1};
  PolynomalC* p1 = polyc_create(4, coeffs1);

  int coeffs2[3] = {1, 2, 3};
  PolynomalC* p2 = polyc_create(3, coeffs2);

  PolynomalC* p3 = polyc_add(p1, p2);

  int r = polyc_evalAt(p3, 2);
  printf("Result: %d\n", r);

  polyc_delete(p1);
  polyc_delete(p2);
  polyc_delete(p3);

  return 0;
}
```

2.2.4.2 Point-value-representation of polynomals

Another way of representing this polynomal is the point-value representation $[(0,4),(1,10),(2,26),(3,58)]$. This representation is very convenient for fast evaluation of polynomal multiplication (assuming both polys are evaluated at exactly the same points):

```c
#include "stdio.h"
#include "stdlib.h"
#include "math.h"
```

```
/**
 * Point/Value representation of polynomals
 */

typedef struct PolynomalV {
  int length;
  int* points;
  int* values;
} PolynomalV;

PolynomalV* polyv_create(int l, int* pts, int* vls) {
  PolynomalV* p = malloc(sizeof(PolynomalV));
  p->length = l;
  p->points = malloc(sizeof(int)*l);
  p->values = malloc(sizeof(int)*l);
  for(int i = 0; i < l; i++) {
    p->points[i] = pts[i];
    p->values[i] = vls[i];
  }
  return p;
}

void polyv_delete(PolynomalV* p) {
  free(p->points);
  free(p->values);
  free(p);
}

PolynomalV* polyv_mult(PolynomalV* p1, PolynomalV* p2) {
  int l = p1->length;
  int vals[l];
  for(int i = 0; i < l; i++) {
    vals[i] = p1->values[i] * p2->values[i];
  }
  PolynomalV* p = polyv_create(l, p1->points, vals);
  return p;
}

int main(void) {

  int points[4] = {0, 1, 2, 3};

  int vals1[4] = {4, 10, 26, 58};  // [4, 3, 2, 1]
  PolynomalV* p1 = polyv_create(4, points, vals1);

  int vals2[4] = {1, 10, 49, 142}; // [1, 2, 3, 4]
  PolynomalV* p2 = polyv_create(3, points, vals2);

  PolynomalV* p3 = polyv_mult(p1, p2);

  polyv_delete(p1);
  polyv_delete(p2);
  polyv_delete(p3);

  return 0;
}
```

2.2.4.3 Naive transformation from coefficient to point-value and vice-versa

```
// O(n^2)
 PolynomalV* poly_coeffToPv(PolynomalC* pc, int* points){
   int l = pc->length;
   int vals[l];
   for(int i = 0; i < l; i++) { // n operations
     vals[i] = polyc_evalAt(pc, points[i]); // O(n)
   }
   PolynomalV* pv = polyv_create(l, points, vals);
   return pv;
 }

// O(n^3)
PolynomalC* poly_PvToCoeff(PolynomalV* p){
  int l = p->length;
  int points[l] = p->points;
  int values[l] = p->values;
  int** data = [[1, x1, x1^2, ...]
                [1, x2, x2^2, ...]
                [1, x3, x3^2, ...]];
  Matrix* m = mtrx_create(l, l, data);
  Matrix* mi = mtrx_inverse(m);   // Gauss Jordan: O(n^3)
  int coeffs[l] = mtrx_mult(mi, values);
```

```
  PolynomalC* pc = polyc_create(1, coeffs);
  return pc;
}


int main(void) {

  int coeffs[4] = {4, 3, 2, 1};
  PolynomalC* pc = polyc_create(4, coeffs);
  int points[4] = {0, 1, 2, 3};
  PolynomalV* pv = poly_coeffToPv(pc, points);
  PolynomalC* px = poly_PvToCoeff(pv);

  polyc_delete(pc);
  polyv_delete(pv);
  polyc_delete(px);

  return 0;
}
```

2.2.4.4 Fast transformation

Up to now we have used the points $0, 1, 2, 3$ for our evaluation. However, there is no reason why we should have chosen these specific points - any other points of evaluation are just as good. It turns out that if we use the 4 roots of 1 as points of evaluation, we can cut down on computation costs.

In general, the kth root of 1 is $e^{2\pi i \frac{k}{n}}$. So let's do the evaluation at $(1^{1/4})_1, (1^{1/4})_2, (1^{1/4})_3, (1^{1/4})_4 = 1, i, -1, -i$.

2.2.5 Stream algorithms

Stream algorithms handle the case where a function is not applied once for a single input, but continuously on a neverending stream of inputs. This creates some constraints:

- The algorithm must not take longer than the feed-rate of the stream
- The algorithm must use only a constant ammount of storage (because otherwise at some point the disc will be full)

Here is a good introduction to the toppic. Side note: a popular software to provide a stream to (multiple) subscriber(s) would be Apache kafka.

2.3 Data-structures

Table 2.1: Most common data-structures

	insertion	removal	search	notes
array	x	x	n	
linked list	1	1	n	infinite size
stack	1, always FILO	1, always FILO	n	
queue	1, always FIFO	1, always FIFO	n	
hash-table	1	1	1 n/k if table gets large	
index tree	log(n)	log(n)	log(n)	
binary heap	Average 1, worst case log(n)	Root element: 1 log(n) otherwise	find-min: 1 Otherwise: n	A binary tree where each child is larger than i Stays semi-sorted: easy to get smallest elemen but increasingly harder to get second, third, .. Useful for priority queues
tries				

2.3.1 Stack

Stacks perform push and pop in O(1). However, they do search in O(n).

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>!



typedef struct El {
        char* key;
        float val;
} El;


El* el_create(char* key, float val){
        char* keyk = strdup(key);
        El* el = malloc(sizeof(El));
        el->key = keyk;
        el->val = val;
        return el;
}


void el_destroy(El* el){
        free(el->key);
        free(el);
}


typedef struct Stack {
        El** data;
        int size;
        int top;
} Stack;


Stack* stack_create(int size){
        Stack* stack = malloc(sizeof(Stack));
        stack->data = malloc(sizeof(El*) * size);
        stack->size = size;
        stack->top = -1;
        return stack;
}

void stack_push_el(El* el, Stack* stack){
        stack->top += 1;
        stack->data[stack->top] = el;
}

void stack_push(char* key, float val, Stack* stack){
        El* el = el_create(key, val);
```

```
        stack_push_el(el, stack);
}

El* stack_pop(Stack* stack){
        stack->top -= 1;
        return stack->data[stack->top + 1];
}

El* stack_peak(Stack* stack){
        return stack->data[stack->top];
}

void stack_print(Stack* stack){
        int i;
        for(i = 0; i <= stack->top; i++){
                printf("%s -> %d", stack->data[i]->key, stack->data[i]->val);
        }
}

void stack_destroy(Stack* stack){
        while(stack->top >= 0){
                El* el = stack_pop(stack);
                el_destroy(el);
        }
        free(stack->data);
        free(stack);
}


int main(){

        Stack* s = stack_create(10);
        stack_push("eins", 1.1, s);
        stack_push("zwei", 1.2, s);
        stack_push("drei", 3.1, s);

        stack_print(s);

        stack_destroy(s);

        return 0;

}
```

2.3.2 Linked lists

Linked lists have the advantage that they have no prefixed size. Addition and insertion at any point is O(1). On the other hand, searching takes O(n) in the worst case.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct String {
        char* text;
        int length;
} String;


String* string_create(char* text, int length){
        char* textCopy = strdup(text);
        String* string = malloc(sizeof(String));
        string->text = textCopy;
        string->length = length;
        return string;
}


void string_destroy(String* string){
        free(string->text);
        free(string);
}

typedef struct Node {
        struct String* data;
        struct Node* next;
} Node;


Node* node_create(String* text){
        Node* node = malloc(sizeof(Node));
        node->data = text;
```

```
        return node;
}

void node_destroy(Node* node){
        if (node->next != NULL) { node_destroy(node->next); }
        el_destroy(node->data);
        free(node);
}

void node_append(Node* ll, String* text) {
    if (ll->next != NULL) { ll_append(ll->next, text); }
    else {
        Node* newNode = node_create(text);
        ll->next = newNode;
    }
}

void node_print_all(Node* ll) {
    printf("%s \n", ll->data);
    if (ll->next != NULL) {
        node_print_all(ll->next);
    }
}

void main() {
    Node* ll = node_create(string_create("Hi!", 3));
    node_append(ll, string_create("I'm", 3));
    node_append(ll, string_create("Michael", 7));
    node_print_all(ll);
}
```

2.3.3 Binary trees

Trees make sense when the keys have to be ordered in some sense.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>



typedef struct El {
        int key;
        float val;
} El;


El* el_create(int key, float val){
        El* el = malloc(sizeof(El));
        el->key = key;
        el->val = val;
        return el;
}


void el_destroy(El* el){
        free(el);
}

typedef struct Node {
        struct El* data;
        struct Node* parent;
        struct Node* left;
        struct Node* right;
} Node;


Node* node_create(Node* parent, El* element){
        Node* leaf = malloc(sizeof(Node));
        leaf->parent = parent;
        leaf->data = element;
        return leaf;
}

void node_destroy(Node* node){
        if(node->left != NULL){ node_destroy(node->left); }
        if(node->right != NULL){ node_destroy(node->right); }
        el_destroy(node->data);
        free(node);
}

void node_elementInsert(Node* node, El* el){
```

```
        if(el->key < node->data->key){
                if(node->left){
                        node_elementInsert(node->left, el);
                }else{
                        node->left = node_create(node, el);
                }
        }else{
                if(node->right){
                        node_elementInsert(node->right, el);
                }else{
                        node->right = node_create(node, el);
                }
        }
}

void node_dfsPrint(Node* node){
        printf("%d -> %d \n", node->data->key, node->data->val);
        if(node->left){
                node_dfsPrint(node->left);
        }
        if(node->right){
                node_dfsPrint(node->right);
        }

}

typedef struct Tree {
        Node* root;
} Tree;


Tree* tree_create(El* el){
        Tree* tree = malloc(sizeof(Tree));
        Node* root = node_create(NULL, el);
        tree->root = root;
        return tree;
}

void tree_destroy(Tree* tree){
        node_destroy(tree->root);
        free(tree);
}

void tree_elementInsert(Tree* tree, El* el){
        node_elementInsert(tree->root, el);
}

void tree_dfsPrint(Tree* tree){
        node_dfsPrint(tree->root);
}


int main(){

        El* baseEl = el_create(20, 14);
        El* el1 = el_create(1, 4);
        El* el2 = el_create(25, 9);
        El* el3 = el_create(2, 2);

        Tree* tree = tree_create(baseEl);

        tree_elementInsert(tree, el1);
        tree_elementInsert(tree, el2);
        tree_elementInsert(tree, el3);

        tree_dfsPrint(tree);

        tree_destroy(tree);
        return 0;

}
```

2.3.4 Hash-tables

Hash tables are basically a array with a hash-function. The hash function is there to associate string-keys to the integer-array-indices. The array needs as many elements as the hash-function can create integer-keys.

Through the hash-function the hash-table has insertion- and lookup-times of expected O(1). However, when the array gets many elements. it might happen that the hash-function gives the same integer-key to several string-keys. To acommodate such a case, the array-elements are implemented as linked lists. This makes the worst-case

loopup-time O(n/k).

Why does one not simply use a perfect hash? Make a hash-function that generates a unique key for any string it's given. This would be known as a lookup-table. That works perfectly fine when we expect almost all possible string-keys to be really used. If however we expect only a few of the possible strings to be used, the array would contain a lot of unused space.

Chapter 3

Practical computer knowledge

3.1 Hardware

3.1.1 Memory

Data is moved from ram to the processor by the memory controller.

Each cell in memory holds 8 bits.

3.1.1.1 How data is stored

- `tiny int`: use just one bite - making 256 possible values.
- `int`: uses 4 bites (32 bits)
- `long int`: uses 8 bites (64 bits)
- `signed int`: reverse the leftmost bit
- `fractions`: store two numbers: the numerator and the denominator.
- `float`: also store two numbers: the number without the point and the position of the point

Fun fact: java tends to silently cause integer overflows. When it has to compute an addition of two big ints, say 255 + 1 (1111 1111 + 0000 0001), it does the right computation (1 0000 0000) but throws away the leftmost binary because that won't fit into memory, causing the result to be stored as 0000 0000.

```
System.out.println(Integer.MAX_VALUE);
Integer a = 2147483647;
Integer b = 1;
Integer c = a + b;
System.out.println(c);
```

Words and Pages:

3.1.2 Processors

Processors have a cache where they store copies of stuff they recently read form ram. This cache is even faster than reading from ram itself. Because programs tend to put their stuff in sequential order in the ram, the memory controller not only feeds the processors cache the contends of the requested adresses, but also a few of the nearby ones.

32 versus 64 bit.

3.1.3 Cables and Busses

A bus is the conductor that leads data from one hardware-component to another, like from the memory to the processor. Busses come in serial and parallel form.

- SATA: Designed for storage-devices. Serial. Carries no power. 1.5 - 6.0 Gbps (Gigabits per second).
- PCI: (or newer, PCIe): parallelized (that is, multiple data-links per lane), fast, specifically designed for graphics-cards and expansion-cards. Can be used bidirectionally. Also carries power. 7 Gbps per lane (usually 2, 4, or up to 32 lanes present.)
- USB: Used in all kinds of devices. Serial. Unidirectional. Also carries power. 5.0 (USB 3.1 Gen 1) - 20.0 (USB 3.2) Gbps.
- Firewire: Common in Apple and media-devices. Designed for large streams of data. Allows direct device-to-device-connection without a computer in between. Can be used bidirectionally. 3.2 - 6.4 Gbps.

3.1.4 Harddrives

Harddrives use the scasi-System to be made visible to the os.

3.2 Networking

A lot more information can be found here: http://cnp3book.info.ucl.ac.be/

3.2.1 Protocols

3.2.1.1 Layer 1: Physical

- Synchronous serial protocols: Master-slave relationship. Can only go one-way
  - I2C/TWI
  - SPI
- Asynchronous serial protocols:
  - TTL Serial
  - RS-232: your monitor cable
- Asynchronous serial bus protocols:
  - usb
  - RS-485

3.2.1.2 Layer 2 : Data link

Ethernet [Frames] Ethernet is barely a protocol. It works like morse: the sender writes the destination-mac on the frame and sends it out on the wire. On the wire, the frame is broadcasted to absolutely everyone, and every computer has to check for itself if the frame was meant for it or for someone else.

Because that is way too much work with several millions of computers in the world, we have switches. They connect two nets of computers, and if a destination-mac is on the other net, they allow the frame to be broadcast to all computers in both nets; otherwise they block the frame, which makes it only broadcast to its source-net. (Hubs are even simpler devices: they always broadcast to all nets, never even reading the destination-mac.)

Because in the 70's hubs and switches didn't have very much memory, the payload of a frame is limited to 1500 bytes. As a consequence, IP's payload is 1480 bytes, and TCP's payload is 140 bytes.

There are few problems that can arise witch switches, since they are such simple components. The one you should know about is a broadcast-storm (aka switching loop). This is where multiple switches keep broadcasting each other in search of a particular node. This can really only happen when there is a circle between switches.

When traversing through the net, routers change the destination-mac to that of the next router on the way to the destination-ip.

Spanning tree protocol is how ...

3.2.1.3 Layer 3 : Network

IP IP is being sent in packages. Each contains the source- and destination-IP and the payload. Stateless.

Ip packages are routed by routers. A router looks at the destination-ip and finds the next router that is closer to the ip. Then it changes the destination-mac to that of the next router and sends the package on its way.

IP doesn't have a notion of a connection. You can blindly send thousands of IP-packages into the ether and never know if they arrived.

Discovering a router and obtaining an ip The computer uses DHCP to find a router: it requests an IP address by broadcasting a DHCPDiscover message to the local subnet. The router hosts a DHCP server that then sends the computer an answer containing the computers new ip-adress and the default-gateway that it should use.

```
michael@michael-ThinkPad-Edge-E540:~$ sudo tcpdump -vnes0 -i wlp4s0 port 67 or port 68
tcpdump: listening on wlp4s0, link-type EN10MB (Ethernet), capture size 262144 bytes
13:53:33.023390 0c:8b:fd:8f:8d:65 > ff:ff:ff:ff:ff:ff, ethertype IPv4 (0x0800), length 342: (tos 0x10, ttl 128, id 0, offs
    0.0.0.0.68 > 255.255.255.255.67: BOOTP/DHCP, Request from 0c:8b:fd:8f:8d:65, length 300, xid 0x83aa8a15, Flags [none]
          Client-Ethernet-Address 0c:8b:fd:8f:8d:65
          Vendor-rfc1048 Extensions
            Magic Cookie 0x63825363
```

```
            DHCP-Message Option 53, length 1: Request
            Requested-IP Option 50, length 4: 192.168.1.178
            Hostname Option 12, length 26: "michael-ThinkPad-Edge-E540"
            Parameter-Request Option 55, length 18:
              Subnet-Mask, BR, Time-Zone, Default-Gateway
              Domain-Name, Domain-Name-Server, Option 119, Hostname
              Netbios-Name-Server, Netbios-Scope, MTU, Classless-Static-Route
              NTP, Classless-Static-Route, Classless-Static-Route-Microsoft, Static-Route
              Option 252, NTP
13:53:33.753685 e8:37:7a:39:ed:2e > 0c:8b:fd:8f:8d:65, ethertype IPv4 (0x0800), length 346: (tos 0x0, ttl 64, id 0, offset
    192.168.1.1.67 > 192.168.1.178.68: BOOTP/DHCP, Reply, length 304, xid 0x83aa8a15, Flags [none]
          Your-IP 192.168.1.178
          Client-Ethernet-Address 0c:8b:fd:8f:8d:65
          Vendor-rfc1048 Extensions
            Magic Cookie 0x63825363
            DHCP-Message Option 53, length 1: ACK
            Server-ID Option 54, length 4: 192.168.1.1
            RN Option 58, length 4: 302400
            RB Option 59, length 4: 529200
            Lease-Time Option 51, length 4: 604800
            Subnet-Mask Option 1, length 4: 255.255.255.0
            Default-Gateway Option 3, length 4: 192.168.1.1
            Domain-Name-Server Option 6, length 4: 192.168.1.1
            Domain-Name Option 15, length 9: "rheingold"
            BR Option 28, length 4: 192.168.1.255
```

The 4 packets to a successful DHCP:

- DISCOVER: Client connects to the network and sends out a broadcast discovery looking for its DHCP information.
- OFFER: The server offers the DHCP information to the client
- REQUEST: The client requests verification of the DHCP information
- ACK: The server acknowledges the DHCP request

Sometimes you will not see the DISCOVER / OFFER and just see the REQUEST / ACK. This happens when the client has already obtained a valid DHCP lease earlier and is just requesting to have it again before its lease time expires. Typically this is performed when half the lease has lapsed.

If the REQUEST is not valid anymore the server will send a NACK indicating to the client that it can no longer use this DHCP information. This should cause the client to start over with a DISCOVER.

Sometimes you will see repeated DISCOVER / OFFER but never a REQUEST from the client. This happens when the client either doesn't receive the OFFER or doesn't like it for some reason. Perhaps a firewall is blocking it, they have a poor connection, or simply they're using a Windows computer.

It's common for Windows Vista to never even start its DHCP process. It will just refuse to DISCOVER and complain that the connection is "limited or no connectivity". You can try to diagnose the problem and tell it to reset the network card and/or get new IP information. If this fails to start it then I find adding a static IP and then setting it back to DHCP will get it going. You may even need to restart the DHCPC service. Its Vista.. where you expecting it to work as advertised?

Routers are the hardware components that ....

3.2.1.4 Layer 4 : Transport

Transport-layer protocols contain Contain source- and destination-port-number. This way, a package that has already arrived at a server (by its ip-address) can be sent to the right application to handle the request.

Common port numbers on the receiving site are:

- 21: ftp
- 22: ssh
- 25: smtp
- 53: dns
- 67/68: dhcp

- 110: pop
- 80: http
- 443: https (http in tls)
- 989/990: sftp (ftp in tls or ssh)

The port number only gets important once a package arrives at the destination-server. There, it must fist pass the firewall. Then, the server decides what application should receive packages addressed to the given port.

TCP Continuous connection, stateful. The recipient regularly sends the sender an ACK with the last received package number. If the sender receives ACKs of the last sent package, it gradually increases the amount of packages it sends in a batch before waiting for another ACK. If the sender receives an ACK of the same package number multiple times, it assumes that all the following packages have been lost and retransmits them from there. Also, the batch-size is temporarily reduced.

UDP

3.2.1.5 Layer 5: Application layer

HTTP Http kind of throws away the continuous connection created using tcp by terminating said connection once a request and all its subsequent subrequests (like fetching the image on a site) has been served.

Http requests consist of

- a url,
- a method,
- and potentially parameters

Http-resonses consist of

- a status-code,
- a mime-type (= Content-type)
- and the actual content.

It's good to know a few of these status codes:

- 200 : all ok
- 302 : item was already cached
- 404 : resource not found
- 500 : internal server error

Http is really a stupid protocol. But for static websites it's just about enough. Also, webservices send and reciive their xml over http.

HTTPS This is http within ssl (aka tls). This is not the same as http within ssh - that would be called a tunnel. ssl (port 443) and ssh (port 21) both use Diffie-Hellmann to create a common key. But the difference is that ssl does not usually require authentication of the client. The server proves its identity by sending the client a certificate, but the client does not need to authenticate to the server like he does in ssh (by password or by by public-key authentication).

In https, all the http-content, being url, headers and parameters, is encrypted. However, ip- and tcp-headers are not encrypted - otherwise the packages would not be routable. Also, for making the initial handshake with the server, the servers url has to pass through the net in public. But once https is established, clicks on further hyperlinks on the server will not be visible to the outside; only the associated ip and port.

Websocket Contrary to http, which is just a request-response-close protocol, in websockets a server has the option to query the client actively at any time.

3.2.1.6 Layer 6 and higher

REST REST sits on top of HTTP. It defines a few methods:

- GET: Get a ressource. Get puts all its parameters right in the url and has no body
- POST: Put enclosed data in database, changing an existing resource. Post-Parameters are put in the request body.
- PUT: Put enclosed data in database, adding a new resource
- HEAD:

SOAP Simple Object Access Protocol

3.2.2 Email

Ok, so we've covered the ethernet/ip/tcp/etc stack. Email is an altogether different beast.

SMTP : per default on port 25 (or 465 for TLS). For sending email.

POP3 : per default on port 110 (or 995 for TLS). For pulling email from server.

IMAP : per default on port 143 (or 993 for TLS). For copying email from server and syncing read-status.

3.2.3 Security

3.2.3.1 Encryption

3.2.4 Making things talk: Remote data transmission

Before we go into details, we need to know some basics of communication.

How radio works

- Radio waves are the longest and contain the least energy of any electromagnetic wave. While ligthwaves are the size of bacteria, radiowaves are the size of a car of even a mountain.
- AM-Radio (amplitude-modulation); the original standard.
  - Take a signal of constant amplitude and frequency - know as the carrier wave.
  - Add your music to that signal. Even though the music has changing freqs and amps, the carrier signal transports it at its carrying frequency.
  - Send the signal from a radio-station
  - Receive the signal with a radio, where you pick the channel by chosing the carrier-frequency. The radio then subtracts the carrier-signal, leaving only the original music intact.

  This is how a signal can be transmitted on one single frequency.
- FM-Radio does the same, but instead of staying on one freq and listening to amplitude, it listens to a (small) spectrum of freqs and emmits sound whenever the frequency changes.
  - better against interference, because intf often causes ampl spikes
  - smaller range: AM can transmit over long distance because it uses a smaller frequency (though not so good quality)
- in modems, isdn and dsl we use am, fm and phase-modulation together to allow for multiple channels.

Types of wires.

- copper: traditional telephone wires. ISDN runs on these; DSL does too, but somewhat slower (this is because, contrary to ISDN, DSL needs an IP-Station as soon as possible after leaving your house. In rural areas DSL becomes much slower this way). Can carry electricity, which is why in the past telephones still worked even when a houses electricity crashed.
- glass-fibre/voip: old copper wires are being replaced with cables that end in a station with IP-adress. These don't carry electricity anymore, however. ISDN does not work on these, either. Those new voip-cables are often glass-fibre cables[1]. DSL over glass-fibres is called VDSL (whereas DSL on the old copper wires is simply called DSL), telephony over DSL/glass is called VOIP.

And here a recap on important hardware:

- Router: really consists of three devices:
  - router: sends packets by ip either to your devices or the next router on the internet
  - switch: broadcasts incomming packets to all your home-devices or moves outgoing packets to the router
  - wireless access point

3.2.4.1 ISDN (wired)

Your computer is attached to a modem[2], which is attached to a telephone-cable or even a telephone-earpiece (!). The modem translates your TCP-requests to sound-signals which would be carried over the telephone-wire in the form of sound. With ISDN, you cannot use both your telephone and your internet at the same time (actually, with am, fm and pm you can; you'd obtain up to 3 channels). Actually, isdn is different from a modem. Modems came first. There you put your telephone directly on the earpiece. Isdn then skipped the earpiece and put the modem directly onto the copper wire - this way a higher sampling frequency was possible. Since everyone in a neighbourhood shares the same cable, your speed will drop when many people use netflix at the same time. On the other hand, ISDN uses a boosting technology that makes sure that your phone-conversation stays clear over large distances. This means that ISDN does not require a relay nearby - contrary to DSL. So in rural areas, ISDN may be actually faster. Speed: around 0.1 Mbit/s.

3.2.5 DSL (wired)

DSL (digital subscriber line) is using ordinary telephone wires or the more modern glass-wires (in that case it is called VDSL). Both a telephone-cable as well as a your routers output will be plugged into a DSL-splitter (shown below).

With ISDN, you could not use your telephone and your computer at the same time. This is because your internet-traffic is going through the copper-wire on a different frequency-band than the one telephone-conversations use. ISDN was hampered by interference, because multiple wires would lie in the ground close to each other. While ISDN did already use am, fm and pm, DSL has a technique for interference-cancelling that allows it to increase its throughput even more. But this technology requires an ip-station at the

[1] Instead of electricity, glass-fibre cables carry light-signals. This technology allows for far more frequency bands to be used.

[2] modem stands for modulator/demodulator: a device which receives electrical waves comming from the copper-wire and transforms them into analog (sound) or digital (bits) waves; and back.

end of the line. To further increase speed for the average user, there is *A*DSL, with A standing for asymetric. Here, downstream traffic is much prefered to upstream, since the average user does not host a server. Contrary to ISDN, DSL connections are not shared between all users in a neighbourhood. Instead, a dedicated connection runs directly to a station with its own IP-adress near your house. Speed: up to 16 Mbit/s, but only 1-2 Mbit/s in rural areas where IP-Stations are far from your house.

3.2.6 WiFi (really only those few meters up to your wired router)

WiFi really uses the same "everyone-shout-first-gets-served" protocol as MAC does. That means that every device has to wait its turn to use the router. There are two frequency-ranges at which devices may communicate. Within each range there are sub-areas known as *channels*. Parallelism is obtained by different devices using different channels.

- 2.4 GHz: slower, further reach. 14 channels, but only 1, 6 and 11 don't overlap. Interferes with microwaves and bluetooth.

- 5.0 Ghz: faster, but only at shorter distance. Many channels.

Factors to consider with wifi:

- Overloading: when all channels are busy, new devices will have to share channels that are already in use. Wifi allows connections under that regime, but speed goes down (wifi favors connectivity over speed).

- Channel width: may be broad (=40MHz) if there are no other routers around. Otherwise should be narrow (=20MHz), so as to avoid interference.

- Channel assignment: usually, routers chose a channel for you upon first handshake. Check if the router provides too few channels, or if the chosen channels are already in use by neighboring routers (good routers will automatically chose channels based on usage).

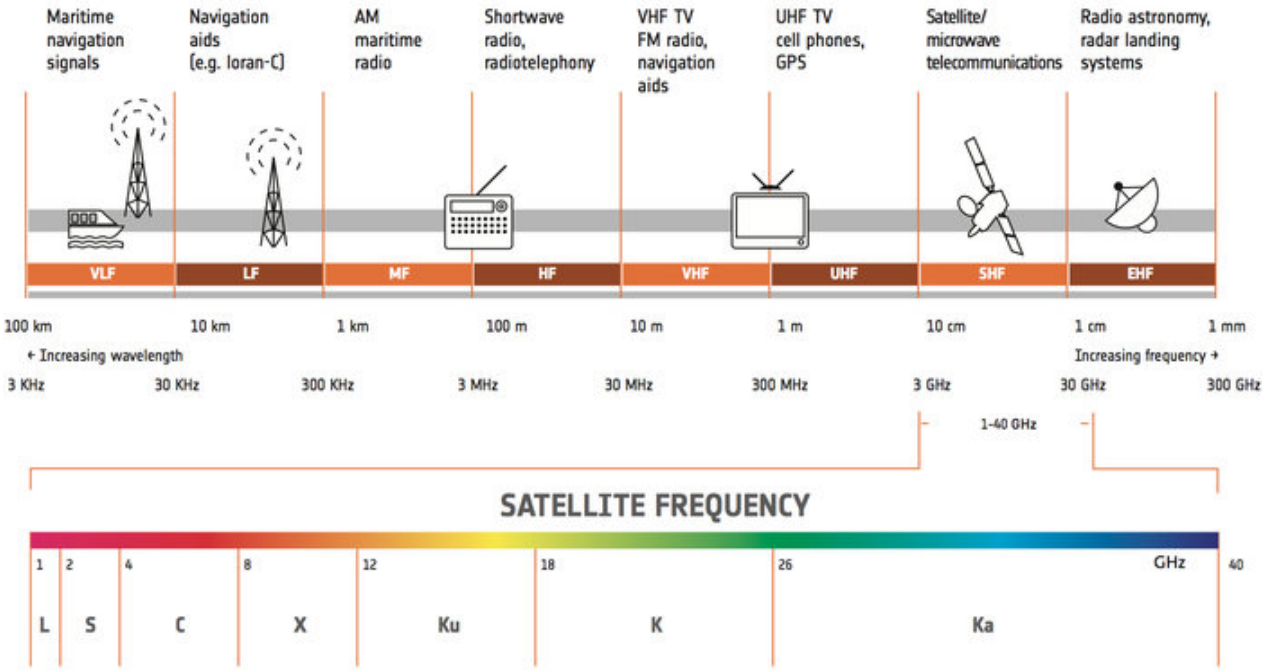
- Copper: wifi-connectivity is an important issue, but once at the router, all packages are funneled through one copper (or fibre) thread. So even with 5G, many devices may experience low performance because the *wired* connection is not fast enough.

3.2.7 GSM, EDGE, 3G=HSDPA, 4G=LTE=HSDPA(+) (radio)

When your device has no WLAN or LAN access to a router, you can only access the internet by going over radio-stations (or satellite, see later).

3.2.8 GPS (satellite)

Since with GPS a signal has to travel all the way to space and back, this is the slowest of the communication methods.



3.3 Internet - practical aspects

While the important theoretical aspects of the internet have been discussed in the section *networking*, there are so many important details that we devote another section purely to those.

3.3.1 Nomenclature

- url
- uri
- domain
-

3.3.2 Session basics

software :

- node: client-sessions (good crypto)
- bcryptjs for hashing
- csurf for CSRF

http session :

- http: Since http is a stateless protocol, we need some helper to store state over several page-visists
- session: webserver tells browser to remember the user
  - http-response-header: "Set-Cookie": "session=12345"
  - cookies: server stores information here
    - that info is sent with every http-request's header
    - "Cookie": "session=12345"
    - Cookies are stored in the browser in a secret spot (not the usual browser-cache or db). They are only sent to a site if the sites url matches the cookie-domain. This means that sites cannot see cookies from other sites.
  - client-sessions has some very recomended headers for cookies:
    - httpOnly: true // dont let js access cookies
    - secure: true // only set cookies over https
    - ephemeral: true // destroy cookies when browser closes

storing passwords : passwords must be hashed.

SSL/TLS

- Those cookies are just strings in the HTTP-header! They can be stolen! So they need to be encrypted.
- What is not encrypted by ssl?
  - SSL does encrypt everything, except ...
    - the initial handshake target (ie your banks url)
    - every request's url
  - but it *does* encrypt all headers (hence cookies), and post- *and* get parameters.

Tokens :(more specifically, JWT, OR, a bit more extensive, OAuth2. Not to be confused with USB-stick tokens)

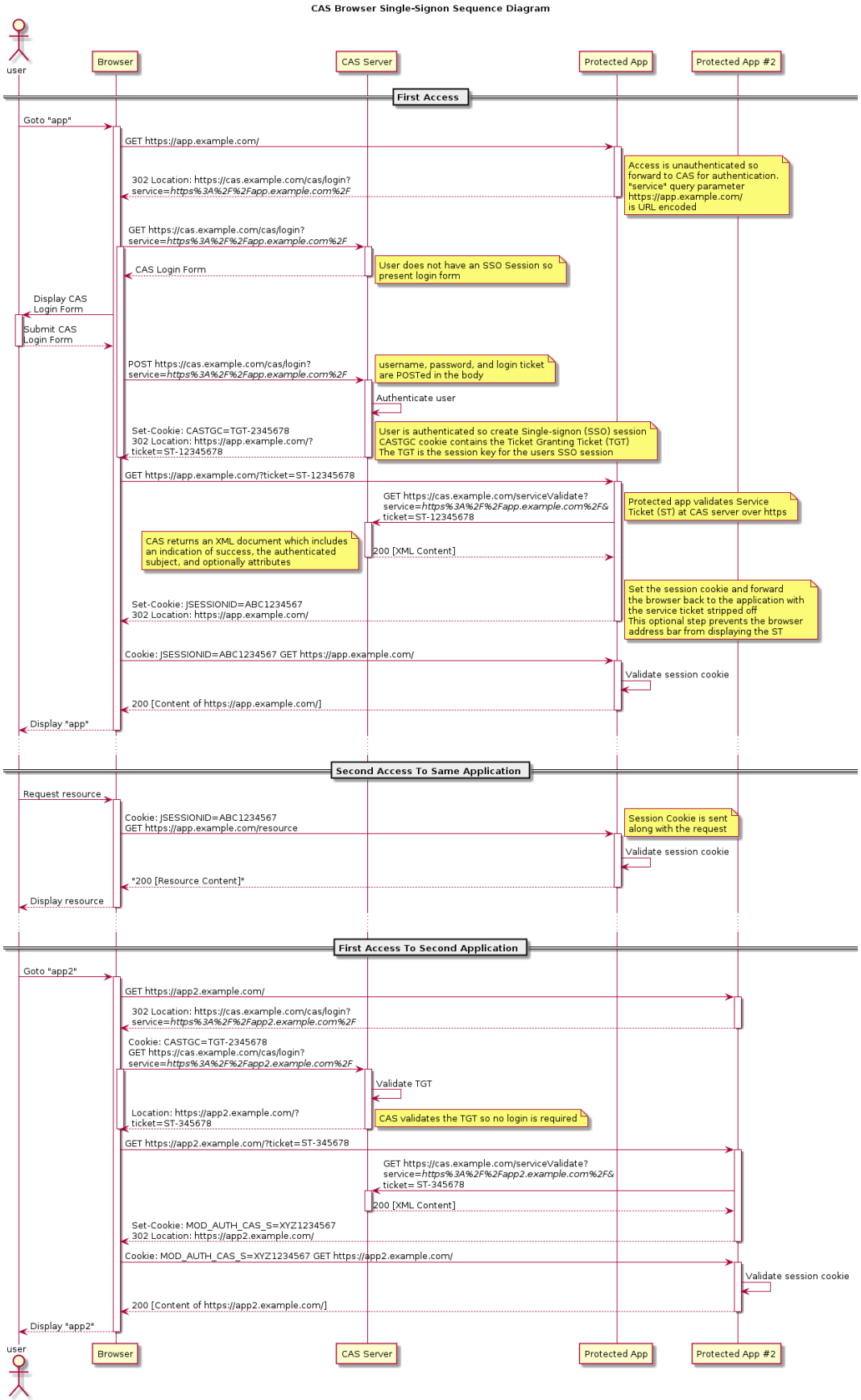
- Originally, tokens were designed as an alternative to cookies that would also work in apps.
- Based on this, however, two protocols (JWT and OAuth) grew, that go beyond what you could do with cookies.
- Here's what you cannot do with cookies. They are a two-party thing.
  - they are bound to a single domain. but your app might have multiple backend-services.
- OAuth/JWT is for multi-party things.
- JWTs are like a Vollmacht:
  - your app (tinder) wants to access your facebook photos
  - you login at facebook to tell it that tinder may use those photos
  - facebook gives you a token that reads: "photoPermission": "1234abcd"
  - you pass that token to tinder
  - tinder sends the token to facebook
  - facebook verifies that you have indeed given that permission
  - facebook gives tinder the photos
- Technically, ...
  - ...they are commonly implemented as a special type of cookie.
  - But they could be implemented as Post-Header + localStorage, too.
  - (more-often so, actually, because contrary to browsers, mobile-apps don't know about cookies)
- Some more stuff:
  - cookies usually contain any information that some programmer deemed necessary.
  - tokens follow strict standards (JWT or OAuth)
  - Tokens usually don't require tinder to maintain a session in its database (though they might)
    - But does facebook still need to maintain a session?
  - Tokens don't require a remote-session because the token contains user-data.

3.3.3 Authentication, SSO, and authorization

Information from here. Assume that you want to check the identity of a user. That user is already member of some authentication-service like facebook. You could ask the user for his facebook-password. But authenticating at some place with your password is insecure, because your password needs to make it over the network. Instead, the user can ask facebook for a token, pass it to your site and you then check with facebook if this token is cool. Strictly speaking, OAuth is about authorisation (is the user allowed to use my service?), not about authentication (is the user really Michael?). The common analogy I've seen used while researching OAuth is the valet key to your car. The valet key allows the valet to start and move the car but doesn't give them access to the trunk or the glove box. An OAuth token is like that valet key. As a user, you get to tell the consumers what they can use and what they can't use from each service provider. You can give each consumer a different valet key. They never have the full key or any of the private data that gives them access to the full key.

	Centralized	Decentralized
Authorization (403)	CAS has an extension for authorization.	OAuth2: The user tells your app and a second a what your app may do on second app
Authentication(401)	CAS: your backend verifies that the user is really him (backend may be one of Kerberos, LDAP or ActiveDirectory)	OpenId: A usage of OAuth2 for authentication. The user tells your app on which secon that the user is really him Example: Sign-in-with-github

To confuse things a little, some CAS Implementations also support OAuth.



Both CAS and OAuth are 3rd party authentication-systems. In both cases, the guiding principle is the same; the situation is like a club-owner, a bouncer, and a guest.

- The guest (user, aka. client) doesn't like the bouncer (CAS-service, aka. cassified app, aka. cas-client) and doesn't trust him with his real name. So instead, he gets a ticket (service-ticket) from the club-owner (CAS-server) to present to the bouncer.
- The bouncer doesn't trust the guest: he checks with the club-owner if that ticket is really valid.
- The club-owner
  - gives the guest a ticket, but doesn't write on it what the guest is entitled to, so the guest can't mess with his ticket.
  - verifies the ticket with the bouncer and tells the bouncer that the ticket is worth entry and a bear, but no VIP-lounge.

CAS and OAuth both ...

- keep guest and bouncer on a need-to-know basis
- it is the bouncer that needs to ensure the proper implementation of ...
  - directing the user to the boss (the CAS-server) for login
  - verify the ticket (service-ticket) presented by the user with the boss
- The login-form is provided by the boss, not the bouncer. The bouncer is oblivious to the user's creds or any other details except the privileges communicated by the boss.

That being said, there is a CAS-REST API that you could use to have the user pass his credentials to the app (in the app's own, custom form) so that the app can then post those creds to the CAS-server's REST-endpoint on the users behalf. However, this requires the user to trust the app a lot more: he now gave the app his credentials and must trust it to not abuse them (for example to log in at another service). Also note that there is no way to implement CAS in the frontend alone - you need something to protect your backend. Having CAS in the frontend but not protecting your fileserver in the backend is like having a bouncer control the parking lot: people can still sneak in by just walking directly to the club's door.

3.3.4 Security threats

CSRF :

- someone makes you click a link that is actually a form-post.
- Counter meassures:
  - additionally to the session-cookie, we create a token for every page-load.
  - token provided by server
  - token added to form (as hidden field)
  - token sent back with form when user hits submit
  - server compares form-token with given token
  - if not identical, then
  - some bad guy sent us a link that we didn't

XSS : ...

	XSS: evil code being sneaked into frontend	XSRF: evil request being sneaked into backend
Roles	User trusts compromised site	Site trusts mislead user
Mechanism	Attacker injects script into site	Attacker tricks user into issuing request
Method	Attacker posted a comment that was not properly validated and now displays evil scripts to another user	Attacker made user click on something that cause a request to the backend
Counters	Form validation: don't allow script tags and similar in posts.	Form-tokens: a form-post must have been expecte Cors: a form-post can only come from backend's

CORS (cross origin resource sharing) is the act of one site (www.mysite.com) requesting data from another, third-party site (www.puppyimagehost.org?img=3241). Doing so is forbidden by default, and browsers will block such requests. The reason is that a hacker could use his browser to do something malicious. This here is a perfectly fine request:

After that request, the `jsessionid` cookie will be stored on the browser. A hacker could now steal that cookie and do this:

This is called Cross-site-request-forgery. Honestly, it seems that this is something that should be fought on the *sever*side, not from the client.

However, some sites depend on CORS.

- For one, images and scripts can be loaded from third-party sites (using the `src` attribute). As such, for historical reasons, images are exempt from CORS-blocking
- Maybe www.puppyimagehost.org *wants* to expose an API for other sites to call. In this case, www.puppyimagehost.org can set an additional header on its response: `Access-Control-Allow-Origin: www.mysite.com` or `Access-Control-Allow-Origin: *`

Note that there is no way that www.mysite.com can deactivate CORS-blocking; it *must* be allowed by www.puppyimagehost.org. An individual person might deactivate the browsers CORS-settings (if the browser allows you to do that), but the common user will never do that.

CORS when applied to an OL-canvas Really, using 3rd party data is a bit like a one-night-stand: you need consent from both sides. The server gives consent to the usage of its images by setting `Access-Control-Allow-Origin`, the client does so by setting `crossorigin="Anonymous"` (or `crossorigin="use-credentials"`). Now a canvas is a special case. For one, often we load images from more than one external source, so we need to get two-way-consent with a whole series of services. On the other hand: even if we don't actively set `crossorigin`, data is displayed - errors only start to occur once we want to call `canvas.toDataUrl(...)`. Why is that? Well, canvases load `img`'s, and those are, as mentioned above, exempt from CORS-restrictions by default. However once we want to do any manipulations on that data, CORS kicks in again. That is why you can show any layer on a canvas, but cannot export it to pdf as long as it displays any layer that does not have two-way-consent.

CORB (cross origin read blocking) is when your browser deletes a request's response body before it can be red by your site. This is to prevent www.puppyimagehost.org from accidentially leaking sensitive information. Even with CORS, an attacker might use html tags like `img` to circumvent CORS-blocking. This is not a case of cross-site scripting, but rather of building a malicious website from scratch just to sneakily access data.

- First, load remote data to its site using `<img src="https://your-bank.example/balance.json">` or `<script src="https://your-bank.example/balanc`
- The browser would notice that the data in `src` is not an image (or javascript) and consequently would not display the data as image.
- But the browser *would* still have the data in it's memory. The attacker can then use a browser-memory vulnerability like spectre to access this data.

Let's break down how CORB works. A website can request two types of resources from a server:

- data resources such as HTML, XML, or JSON documents
- media resources such as images, JavaScript, CSS, or fonts

A website is able to receive data resources from its own origin or from other origins with permissive CORS headers such as `Access-Control-Allow-Origin: *`. On the other hand, media resources can be included from any origin, even without permissive CORS headers.

CORB will block the response of a request if all of the following are true:

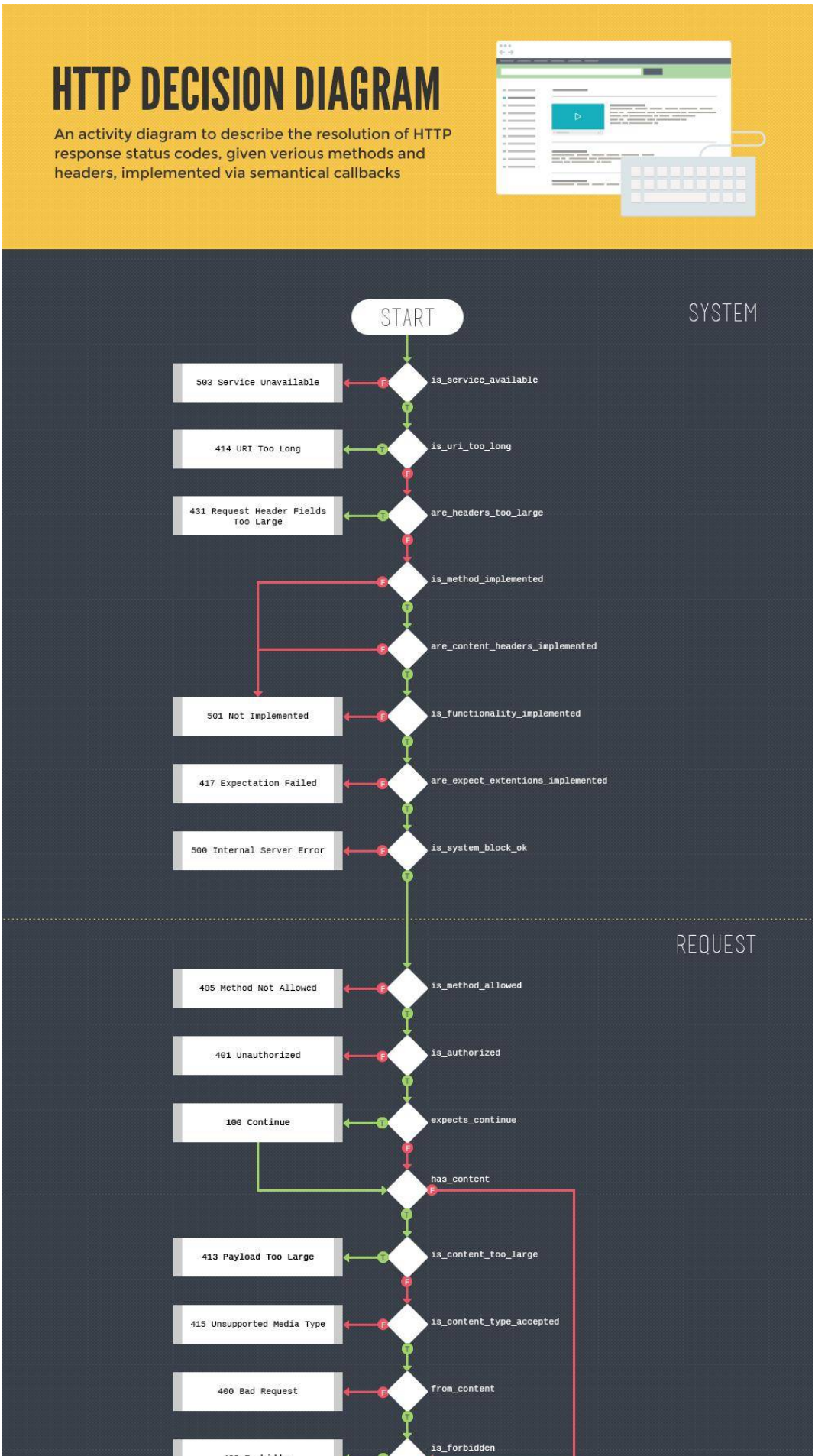
- The resource is a "data resource". Specifically, the content type is HTML, XML, JSON

- The server responds with an X-Content-Type-Options: nosniff header, or if this header is omitted, Chrome detects the content type is one of HTML, XML, or JSON from inspecting the file

- CORS does not explicitly allow access to the resource

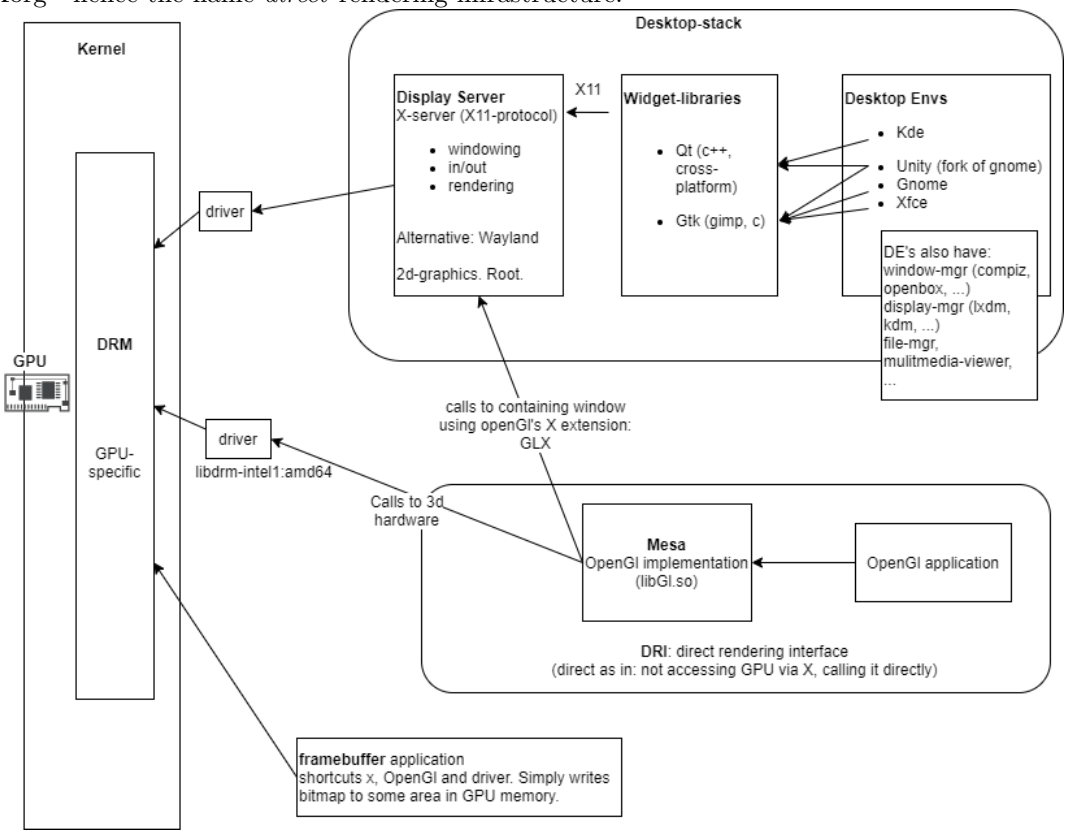
CORB protects neither against XSS nor XSRF. Instead, it prevents one mechanism of side-channel-attacks.

3.3.5 Status codes



3.4 Linux render-stack

Figure 3.1: Drivers: everything on the right of a driver is hardware-agnostic, everything on the left hardware-specific. Note that Xorg and mesa don't use the same drivers. Historically, linux used to only do 2d-graphics. For a brief time after 3d-cards arrived, OpenGl instructions were passed through Xorg. Only later was mesa allowed to bypass Xorg - hence the name *direct* rendering infrastructure.



3.5 Linux sound-stack

All sound-in (microphone, jack/cinch aka line-in) and sound-out (loudspeakers, headphones) cables lead to your soundcard, which has the sole task of transforming analog sound signals to digital `short[]` arrays. The whole routing is done via the ALSA System. Because ALSA is a little complicated, linux has a layer on top of it. Two applications can be used as this top-layer: pulseaudio and jackd.

3.5.1 Audio Standards

Raw audio consists of `short[]` arrays containing raw amplitude data.

Midi comes in two forms: JACK-Midi and ALSA-Midi.

3.5.2 Hardware

Use `aplay -l` to list all your sound cards.

PCH is your actual soundcard.

HDMI is combined audio/video transport. Mostly used by monitors, but also shows up in `aplay -l` because it can carry sound as well.

3.5.3 Software

There are different systems that controll your soundcards.

ALSA is linuxes standard sound routing mechanism.

Portaudio is an alternative to alsa that can also run on windows and mac.

Jack is a router that lets you plug one programs sound output into another programs sound input - or to the system's soundcard.

3.5.4 JACK

Jack deserves its own section.

Frames per period * periods per buffer yields your buffer size. Big buffer means low CPU load, but large latency. Small buffer means small latency, but high CPU load. If the CPU load approaches 100%, you'll hear *xruns* (which stands for buffer-Under- or Over-run), that is clicking and artifacts caused by your CPU not being able to process buffer-batches on time. Jack displays CPU usage under the term *DSP load* in qjackclt.

Realtime (RT) mode requires your kernel to support realtime scheduling (called low-latency kernel). When that is given, jack will be given priority in memory and cpu over most other tasks.

	Public	Downstream license	Changes	Pat
GPL3	Sourcecode must be made public	All modifications must also use GPL3	All changes must be documented	If t
GPL2	”	”	”	-
Apache2	Needs not be made public	Modificators may use any license	”	San
BSD2	”	”	Changes need not be documented	-
BSD3	”	”	”	-
MIT	”	”	”	-

Table 3.1: Licenses from strict (top) to permissive (bottom)

	Static linking (into an Apache-licensed app)	Dynamic linking (into an Apache-licensed app
Example	Webapp where all files are bundled and obfuscated.	Node-app: Libraries are included via require(
GPL3	✗	
GPL2	✗	
Apache2	✓, preserve the NOTICE file	✓
BSD2	✓, add license and copyright in documentation of compiled product	✓
BSD3	✓, add license and copyright in documentation of compiled product	✓
MIT	✓	✓

Table 3.2: Licenses kick in when your code is distributed. Putting files on a webserver counts as distribution, with maybe the exception of code that only serves as a service (depends on the case). Source: https://medium.com/@vovabilonenko/licenses-of-npm-dependencies-bacaa00c8c65

3.6 Legal and opensource

3.6.1 Popular licenses

All the below licenses are approved by the OSI.

3.6.2 Adding 3rd party licenses to your apache2-javascript product

3.6.3 Deutsches/Europäisches Recht

3.6.3.1 Haftung

Obwohl Lizenzen üblicherweise sagen, dass der Anbieter nicht haftbar gemacht werden kann für Schäden, gilt das im deutschen Recht nicht. Laut deutschem Recht ist ein Anbeiter nach dem 'Schenkungsrecht' für Schäden durch Fahrlässigkeit haftbar.

3.6.3.2 Exportkontrolle

Prinzipiell gilt der Grundsatz des freien Warenverkehrs. Aber Exportkontrolle kann greifen,

- Wenn Deutsche, Europäische oder Verbündete (sprich: USA) Sicherheit gefährdet ist,
- um Embargos zu erzwingen,
- Um in den einführenden Staaten nicht misbraucht werden zu können.

Insbesondere gilt hier die Europäische Dual-Use Verordnung.

Chapter 4

Programming languages

A good language

- has you thinking about the problem, not the language
- has enough libraries and community

4.1 C

4.1.1 General theory

4.1.1.1 Memory allocation

By default, memory is allocated statically to the stack. By using `malloc` and `free`, you can instead allocate memory dynamically on the heap.

4.1.1.2 Threading

Threads are created by making a copy of the orignal process. Threads share the same memory with their parents.

Table 4.1: Processes versus Threads

	Sub-Process	Thread
Creation	Copy of mother process	
Memory	Own memory	Shared memory
Communication	Communicates with mother through syscalls, pipes and files	Can directly call methods of mother process
Usecase	Ideal if mother and subprocesses must be separated for security reasons, like apache-server does.	Ideal if thread output to be processed by mother process, because no piping neccessary

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

// A normal C function that is executed as a thread
// when its name is specified in pthread_create()
void *myThreadFun(void *vargp)
{
    sleep(1);
    printf("Printing GeeksQuiz from Thread \n");
    return NULL;
}

int main()
{
    pthread_t tid;
    printf("Before Thread\n");
    pthread_create(&tid, NULL, myThreadFun, NULL);
    pthread_join(tid, NULL);
    printf("After Thread\n");
    exit(0);
}
```

As mentioned above, all threads share data segment. Global and static variables are stored in data segment. Therefore, they are shared by all threads. The following example program demonstrates the same.

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

// Let us create a global variable to change it in threads
int g = 0;

// The function to be executed by all threads
void *myThreadFun(void *vargp)
{
    // Store the value argument passed to this thread
    int *myid = (int *)vargp;

    // Let us create a static variable to observe its changes
    static int s = 0;

    // Change static and global variables
    ++s; ++g;

    // Print the argument, static and global variables
```

```
    printf("Thread ID: %d, Static: %d, Global: %d\n", *myid, ++s, ++g);
}

int main()
{
    int i;
    pthread_t tid;

    // Let us create three threads
    for (i = 0; i < 3; i++)
        pthread_create(&tid, NULL, myThreadFun, (void *)i);

    pthread_exit(NULL);
    return 0;
}
```

```
gfg@ubuntu:~/$ gcc multithread.c -lpthread
gfg@ubuntu:~/$ ./a.out
Thread ID: 1, Static: 1, Global: 1
Thread ID: 0, Static: 2, Global: 2
Thread ID: 2, Static: 3, Global: 3
gfg@ubuntu:~/$
```

Please note that above is simple example to show how threads work. Accessing a global variable in a thread is generally a bad idea. What if thread 2 has priority over thread 1 and thread 1 needs to change the variable. In practice, if it is required to access global variable by multiple threads, then they should be accessed using a mutex.

4.1.1.3 Getting data from C to another programm

Table 4.2: Pipes versus sockets

Pipe	Socket
Unidirectional	Bidirectional
Fifo	
Limited volume (∼0.5 MB); can block!	
all in memory	all in memory
no protocol	packets (UDP or TCP), can go over network

Per file Simplest, but also slowest because it involves disk i/o.

Per pipe or socket Faster than file-writing, because no disc i/o is involved. But still slow because data is chucked into packages that are wrapped in a rather verbose protocol.

Per JNI This should allow you to convert C-datastructures to Java-datastructures.

4.1.2 Quirks and features you need to know

4.1.2.1 * syntax

```
int a = 1;
int * a_ptr = &a;
*a_ptr; // <--- yields 1
```

When assigning, * means: ”make it a pointer”.
When qerying, * means: ”get the value behind the pointer”.
& always means: ”get the address”.

4.1.2.2 Pointer arithmetic

Array indexing is actually a rather complex issue with a lot of syntactical sugar.

```
int arr[3] = {10, 20, 30};
arr[2] = 15;
```

- *arr* gets you the pointer to the first element in the array. In other words, we have $arr = \&arr[0]$
- [2] adds to *arr* the size of 2 ints. Thus we have $arr[2] = \&arr[0] + 2 * sizeof(int)$.
- Finally, the actal value behind the expression is adjusted when we assign the value of 15. Thus the expression reduces to

```
*(&arr[0] + 2 * sizeof(int)) = 15;
```

We can use array indexing to our advantage: with *buffer overflow*s we can read arbitrary parts of the memory. Consider this simple example:

```
int main () {
      int arr[3] = {1, 2, 3};
      int i;
      for(i=0; i < 100; i++){
            printf("%d th entry: location: %p value: %d \n", i, &arr[i], arr[i]);
      }
      return 0;
}
```

Especially when creating arrays, we need to be aware of a few things:

- ... what is the current content of the accessed memory?
- ... is the accessed memory protected against overwriting by others?

```
#include<stdio.h>
#include <stdlib.h>

int main() {

  const int size = 1000;
  //double data[size] = {2.1,2.1,2.1, 2.1, 2.1};   //<-- puts in vals; fills rest with zeros. This mem won't be overwritt
  //double data[size];      //<-- does not put in zeros - keeps whatever was in there before. This mem won't be overwritt
  //double* data = (double*) malloc(size * sizeof(double));              //<-- puts in zeros. This mem won't be overwritt
  //double* data;      //<-- does not put in zeros - keeps whatever was in there before. This mem can be overwritten by o
  for(int i = 0; i < size; i++) {
    double val = data[i];
    printf("val at %i: %f \n", i, val);
  }

  return 0;
}
```

When we state that *this mem won't be overwritten*, of course we exclude overwriting by buffer overflow.

4.1.2.3 Array decay: Functions can't accept arrays

Functions never accept arrays, they only take pointers to the first element. This is known as array-decay.

```
int arr[3] = {1,2,3};
somefunct(arr); // <--- compiler turns this automatically into
somefunct(&arr[0]);
```

4.1.2.4 Array decay: Functions don't return arrays, either

Functions return single values just fine, but arrays only by pointer. *arrFunct* must save array on heap and return pointer to the heap. The calling function must know the arrays size in advance or be given a struct with metainfo about the size. The calling function must also later deallocate the array.

```
int * arrOnHeap(){
    int * arr_ptr = malloc( sizeof(int) * 3 );
    arr_ptr[0] = 10;
    arr_ptr[1] = 20;
    arr_ptr[2] = 30;
    return arr_ptr;
}

int * arr_ptr = arrOnHeap();
```

4.1.2.5 Array syntax

Java and c have some weird differences in array initialisation. Consider array litterals:

```
int coeffs[5] = {1, 2, 3, 4, 5}; // c
int[] coeffs = {1, 2, 3, 4, 5};  // java
```

And also standard initialisation:

```
int coeffs[5];   // c
int[] coeffs = new int[5]; // java
```

4.1.2.6 Struct syntax

There is a really important thing when creating struct-construction functions. Consider the following code.

```
typedef struct Island {
        char* name;
        char* opens;
        char* closes;
        struct Island* next;
} Island;


Island* island_create(char* name){
        Island* i = malloc(sizeof(Island));
        i->name = name;
        i->opens = "09:00";
        i->closes = "17:00";
        i->next = NULL;
        return i;
}

int main(){
    char* name;
    fgets(name, 80, stdin);
    Island* island1 = island_create(name);
    fgets(name, 80, stdin);
    Island* island2 = island_create(name);

    return 0;
}
```

You will find that the name of island1 equals that of island2! The reason is that their names are just a reference to char* name in the main method. We need a safety measure inside the constructor to allocate a copy of the input string so that we can sure a new call to the constructor does not give us the same pointer again.

This code fixes the problem:

```
Island* island_create(char* name){
    char* namec = strdup(name);
        Island* i = malloc(sizeof(Island));
        i->name = namec;
        i->opens = "09:00";
        i->closes = "17:00";
        i->next = NULL;
        return i;
}

void island_destroy(Island* i){
    free(i->name);
    free(i);
}
```

4.1.2.7 Passing functions as variables

This is our entry to functional programming in c. A functionname is really just a pointer to the memory location where the function code is stored. So we can use the function name as a pointer.

Of course, functions have different types. That's why we can't just write

```
function* f;
```

But instead must write:

```
char* (*reverseName)(char*);
```

Here, the first char* is the return type, whereas the second is the argument(-list) to the function. Let's see an example:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


int num_ads = 7;
char* ads[] = {
        "William: SBM GSOH likes sports, TV, dining",
        "Matt: SWM NS likes art, movies, theater",
        "Luis: SLM ND likes books, theater, art",
        "Mike: DWM DS likes trucks, sports and bieber",
        "Peter: SAM likes chess, working out and art",
        "Josh: SJM likes sports, movies and theater",
        "Jed: DBM likes theater, books and dining"
};


int likes_sport_not_bieber(char* ad){
        int hit = (strstr(ad, "sports") && !strstr(ad, "bieber"));
        return hit;
}

int likes_sports_or_workout(char* ad){
        int hit = (strstr(ad, "sports") || strstr(ad, "workout"));
        return hit;
}


void find(int (*match_fn)(char*)){
        puts("Search results:");
        puts("----------------------");

        int i;
        for(i=0; i < num_ads; i++){
                if(match_fn(ads[i])){
                        printf("%s\n", ads[i]);
                }
        }

        puts("----------------------");
}

int main(){

        find(likes_sport_not_bieber);
        find(likes_sports_or_workout);

        return 0;

}
```

We should note some syntactic sugar here. Calling functions is usually done like this:

```
int a = somFunc(b);
```

But that's just shorthand for the actual code:

```
int a = (*somFunc)(b);
```

Also, passing functions as arguments really compiles down to:

```
find(&likes_sport_not_bieber);
```

4.1.2.8 Strings end with a \0

Thats why, when declaring an empty string, leave space for the \0 .

```
char word[3];
strcpy(word, "hi");
strlen(word); // <--- yields 2
sizeof(word); // <--- yields 3
```

4.1.2.9 char[] does not equal char*

```
char a[] = "hello";
char * b = "world";
```

a is equal to: $\boxed{H}\boxed{E}\boxed{L}\boxed{L}\boxed{O}\boxed{\backslash 0}$

b is equal to a $\boxed{pointer}$ pointing to $\boxed{W}\boxed{O}\boxed{R}\boxed{L}\boxed{D}\boxed{\backslash 0}$

4.1.2.10 Header files

Header files are how we can expose only a subset of a file to main.c. Really they contain only the function signatures, not their implementation. In that way, c basically forces you to import *everything* as an interface.

Listing 4.1: main.c

```c
#include <stdio.h>
#include "encrypt.h"

int main(){
    char msg[80];
    while(fgets(msg, 80, stdin)){
        encrypt(msg);
        printf("%s", msg);
    }
}
```

Listing 4.2: encrypt.h

```c
void encrypt(char * msg);
```

Listing 4.3: encrypt.c

```c
#include "encrypt.h"

void encrypt(char * msg) {
    char c;
    while(*msg){
        *msg = *msg ^ 31;
        msg++;
    }
}
```

4.1.2.11 Building and Makefiles

The building of an executable consists of compiling and linking.

- **Compiling your own code** The compiler generates an object file from your source-code. An object-file is a compiled piece of code together with some meta-information on what kind of functions and structures it contains.
- **Linking in libraries** After that, the include-directories are scanned for libraries that contain implementations of the header-files you included in your source-code. This phase is called linking. There are two ways you can link in external libraries.
  - Static linking means that a copy of the library is added to the final executable at compile time. The executable can then be moved to another machine without worries, because all required libraries are placed inside the executable. Static libraries are usually written like this: `libXXX.a` and created with the `ar` program.
  - Alternatively, libraries can be linked in a dynamic way: that means that the executable will search for an appropriate library once it needs to call its functions, that is, at run time. Shared/dynamic libraries are written like `libYYY.so` and created with `gcc -fPIC -c YYY.c && gcc -shared -o libYYY.so YYY.o`
  - Both `.a` and `.so` libraries are included with the `-l` and `-L` commands.

Makefiles are how we knot together different parts of a c program. It's really easiest to look at a specific example:

```
# Includes are header files without a concrete implementation.
INCLUDES = -I/usr/include/mysql
# Libraries are object files. -L adds a directory of libraries, -l adds a single library.
LIBS = -L/usr/lib/x86_64-linux-gnu -lmysqlclient -ljansson -pthread
WARNINGS = -Wall -Wextra
```

```
# Compiling
#   -c: compile (nur compile, nicht link!)
#   -g: fuer debugger

incl.o: incl.c incl.h
        gcc -g -c $(WARNINGS) $(INCLUDES) incl.c

jsn.o: jsn.c jsn.h incl.h
        gcc -g -c $(WARNINGS) $(INCLUDES) jsn.c

psql.o: psql.c jsn.h
        gcc -g -c $(WARNINGS) $(INCLUDES) psql.c

# Linking
#       -o: object-file: name der fertigen binary
#       -g: fuer debugger

psql: psql.o jsn.o incl.o
        gcc -g -o psql psql.o jsn.o incl.o $(LIBS)

clean:
    rm *.o

all: psql clean
```

Includes sind header files. Option $-I$(dirname): hinzufügen dir zu Suchpfad für *.h files. Im Gegensatz zu libs kann man bei incls nur dirs spezifizieren, nicht einzelne files. Dafür gibt es ja schon die #include Direktive.

Libs sind die den header files zu Grunde liegenden so files. Option $-L$(dirname): hinzufügen dir zu Suchpfad für *.so files. $-l$(libname): mit Einbinden einer lib. $-pthread$: eine besondere Option; steht für -lpthread + definiere ein paar extra macros

Jansson ist ein externes Programm. Es muss erst ge-make-t werden (oder ge apt-get-tet), danach finden wir header per default in /usr/local/include und libs in /usr/local/lib

4.1.2.12 Valgrind

The $-g$ flags from the above makefile were actually meant for use in valgrind. Valgrind analyses your memory allocation. It does so by creating its own, fake versions of *malloc* and $free$. Anytime these are called, valgrind does its bookkeeping to check if any allocated memory is left on the heap.

Analysing code with valgrind is easy. Just compile it and then run it like this:

```
make all
valgrind -v --leak-check=full --show-leak-kinds=all --log-file=val.log ./psql input.json
```

4.1.2.13 OpenMP

OpenMP is a set of macros used for control-flow for threads. Think of automatically distributing your for-loop over threads.

4.1.3 CMake and Autotools

While make does a good job at automating the build of c-programs, all the commands you type there are platform dependent. For that reason, larger projects use more sophisticated buildsystems. They don't require less configuration (in fact, they usually require *more* work) but they make it easier to create a build that will work on any platform. Usually they work by first scanning your environment and then creating makefiles for you.

Working with these is not exactly pleasant. This section only contains a *how-to* handle existing open-source projects that you have to get to compile somehow.

We begin with autotools.

- `./configure`
- `make`
- `make install`
- Troubleshooting: most likely, some shared libraries will be missing. Find out which by calling

4.1.4 Best practice

The above quirks gave us a lot of points where we need to be careful in c. For that reason, we should adhere to some best practices to make coding as save as possible.

Only expose top and lowest layer, not internals Datastructures in C tend to be actual data wrapped in structs wrapped in structs wrapped in structs. The highest level struct should only expose data, not any intermediate structs. This way, a user only needs to know the api for the highest level struct and the data itself. Also, every struct should manage its own and only its own memory.

Handling unknown datatype elements Your datastructures will usually contain elements of a type unknown to you. That is not really a problem, because you can reference them using a void*. But how about creating and destroying those unknown elements? Well, here we can use c's functional programming skills: just add to the datastructure a function for creating and one for destroying the element.

Never leave a pointer unassigned You can create a pointer without having it point anywhere in particular: `int* apt;`. But what if later in your code you want to check if that pointer has been pointed to an int yet? `apt` is initially just going to point to some random location. This means that you cannot check `if(apt == NULL)`, because it's never going to be 0! For this reason, even if you don't want `apt` to point to anything yet, at least make sure it points to `NULL`. So always create pointers with `int* apt = NULL;`

4.2 C++

C++ is the best compromise between raw power and versatility. Fortran is faster in large matrix-operations, but lacks the abstraction of OOP and any multi-media support. Java is even better abstracted and has an unbeatable ecosystem with modules for everything, but slower.

4.2.1 Some weird syntax and new concepts

Contrary to garbage-collected languages, where you only need to care about who knows about a given object, in cpp you also net to worry about where that object lives.

4.2.1.1 References

Refences are really just syntactic sugar around pointers. It's still useful to know a few of their basic properties, since they are intended to help you avoid errors in memory management.

Using pointers, to mutate an object you need to use pointer syntax for any operation on the object. This is known as dereferencing: using `*vpt` and `vpt->`.

```
void foo(Vec* vpt){
    vpt->increment(1);
}

Vec vec = Vec();
foo(&vec);
```

Using references we can act as if the actual object and not just a pointer to it was in the function scope.

```
void foo(Vec& v){
    v.increment(1);
}

Vec vec = Vec();
foo(vec);
```

Syntax:

```
int i = 3;
int* i_ptr = &i;
int& i_ref = i;
```

There are a few more noteworthy properties of references that are intended to make your life easier:

- A pointer may point to nothing[1] - you may write `int* nullpt;`. A reference however mustn't do that. This will not compile: `int& nullrf;`.
- You can point a pointer to a new object, but a reference allways points to the same object. This will have `ap` point to `b`:

```
    int a = 1;
    int b = 2;
    int& ar = a;
    ar = b;
```

This, however, will overwrite `a`:

```
    int a = 1;
    int b = 2;
    int& ar = a;
    ar = b;
    cout << a << endl;  // yields 2 (!)
```

4.2.1.2 Const keyword

Sometimes you want immutability; in cpp probably even more than in other languages, for security reasons. That is what the `const` keyword is there for. In most cases, you'll use this keyword when a function is passed some objects that it mustn't alter. Passing the objects as consts guarantees that the function has no sideeffects on the objects.

[1] However, this is very bad practice. Always initialize your free pointers to null using `int* ap = nullptr;`

```
bool isEqual(const Vec & v1, const Vec & v2) {
    bool eq = false;
    if( v1.x == v2.x && v1.y == v2.y ){
        eq = true;
    }
    return eq;
}
```

Of course, we could also have avoided side-effects on v1 and v2 by just passing by value, because then the function would have only altered it's own copies of v1 and v2. But creating those copies is a costly operation, so we're cheeper of by passing by reference and using const to achieve immutability.

Unfortunately, `const` has some weird syntax rules. The trick is in figuring out whether a value or a pointer to a value is to be constant. The rule is that `const` allways applies to the keyword to it's left. If there is nothing on the left, it applies to the keyword on the right.

```
const int a = 1;                // a constant integer
const int* a_ptr = nullptr;     // a variable pointer to a constant integer
int const* a_ptr;               // a variable pointer to a constant integer
int * const a_ptr;              // a constant pointer to a variable integer (you will rarely need this)
```

4.2.2 Object orientation

Here is a simple example of the class-syntax in cpp.

```
#include <iostream>
#include <string>

using std::string;

class Robot {

  private:
    string name;
    int hitpoints;

  public:
    Robot(string n, int hp) : name(n), hitpoints(hp){
      std::cout << "Created " << name << " with " << hitpoints << " health" << std::endl;
    }

    void hit(Robot* other){
      other->takeHit(10);
    }

    void takeHit(int strength){
      hitpoints -= strength;
      std::cout << name << " now has " << hitpoints << " health" << std::endl;
    }

    ~Robot(){
      std::cout << name << " destroyed." << std::endl;
    }
};

int main() {
  Robot b("Bender", 100);
  Robot v("Vender", 100);
  b.hit(&v);
  return 0;
}
```

From this basic example we can already note a few important points.

Constructors cpp knows *three* ways of creating objects:

- The default constructor `Robot b("Bender", 100);` is what you should usually make use of.
- Construction by copying an existing instance `Robot b("Bender", 100); Robot v = b;` requires a functioning implementation of the copy constructor `Robot(const Robot& otherRobot)`. If this isn't provided, the compiler generates one automatically. The copy constructor is called in three cases:
  - When an object is created by copying: `Robot b("Bender", 100); Robot v = b;` or `Robot b("Bender", 100); Robot v(b);`
  - When passing an object to a function by value: `void someFunc(Robot r);` . Note, however, that it with big objects it may be better to pass the object by (const) reference: `void someFunc(cont Robot& r);`

  - When returning an object from a function by value: `Robot someFunc(); Robot b = sumeFunc();`

  . Here, too, it is usually more efficient to use references instead of copying by value.

- Construction by assignment operator `Robot b("Bender", 100); Robot v("Vender", 90); v = b;` requires a functioning implementation of the assignment operator `Robot& operator=(const Robot& otherRobot){ ... return *this;}`. If this isn't provided, the compiler generates one automatically.

Here an example to clarify the difference between assignment and copy-constructor:

```
#include<iostream>
#include<stdio.h>

using namespace std;

class Test{
  public:
    Test() {}
     Test(const Test &t) {
      cout<<"Copy constructor called "<<endl;
     }
     Test& operator = (const Test &t) {
        cout<<"Assignment operator called "<<endl;
        return *this;
     }
};

int main() {
  Test t1, t2;
  t2 = t1;
  Test t3 = t1;
  getchar();
  return 0;
}
```

Constructors as objects or as pointers Objects can be constructed as objects or as pointers to objects.
Construction as object:

```
Particle p = Particle(20, 30);
p.move();
```

Here the object is automatically destroyed once it goes out of scope. Note the missing `new` keyword.
Construction as pointer:

```
Particle* p = new Particle(20, 30);
p->move();
delete p;
```

Here, the object survives until you manually call `delete p`.
Whenever possible, you should prefer the first method. However, there are some situations when it makes sense to use pointers:

- when the object is returned from a function (or any other situation where the object needs to outlive its scope)
- When you're passing the object to a function and you want that function to change the object and not just a copy of the object.

Initialisation Note also this important difference. In java, when you write `Perticle p;` this creates an uninitialized reference. In c++ however, this same command already does the initialisation by calling the default constructor.

4.2.2.1 Operator overloading

You can override any operator, as long as the operation makes use of at least one custom datatype.

```
#include <iostream>
#include <string>

using std::string;

class Vector {
  public:
    int x;
```

```
    int y;
    int z;
    Vector(int _x, int _y, int _z) : x(_x), y(_y), z(_z) {
      std::cout << "Created: [" << x << "," << y << "," << z << "]" << std::endl;
    }
    ~Vector(){
      std::cout << "Going out of scope: [" << x << "," << y << "," << z << "]" << std::endl;
    }
};


Vector operator+(Vector const & v1, Vector const & v2) {
  int x = v1.x + v2.x;
  int y = v1.y + v2.y;
  int z = v1.z + v2.z;
  Vector v3 = Vector(x, y, z);
  return v3;
}

int main() {

  Vector a = Vector(1, 2, 3);
  Vector b = Vector(2, 3, 4);
  Vector c = a + b;
  return 0;
}
```

A very special case: assignment operator The assignment operator is a very special case. It is only allowed to be defined inside a class. But it is very useful to get an insight into what happens when you do an assignment in c++. example with overwriting an object

From this we learn a very important lesson about the assignment operator. When one object is assigned to another, a copy of the actual values is made. In Java, copying an object variable merely establishes a second reference to the object. Copying a C++ object is just like calling clone in Java. Modifying the copy does not change the original.

```
Point q = p; /* copies p into q */
q.move(1, 1); /* moves q but not p */
```

Assignment by copying versus assignment by moving The last paragraph was not absolutely honest with you: you can force cpp to move an object instead of copying it when doing an assignment. ex with std::move

4.2.2.2 Rule of three

The rule of three states that if you write a custom constructor or destructor, you're also going to need a custom copy-constructor and a custom assignment-operator. The rule basically comes down to this: if you do any memory management during construction, you will also do memory management during copying and during destruction. The compiler cannot guess that memory management for you, so you need to specify it yourself.

```
class Ball {
    private:
        int xpos;
                int ypos;
                int xvel;
                int yvel;

    public:
        Ball(int x, int y) : xpos(x), ypos(y) {   // Standard constructor
                std::cout << "ball created" << std::endl;
        }

        ~Ball() {
                std::cout << "ball deleted" << std::endl;
        }

        Ball(const Ball&) = delete;     // Copy constructor (note that argument has no name when using delete)
                Ball& operator=(const Ball&) = delete;      // Assignment operator (note that argument has no name when us

        void move() {
                xpos += xvel;
                ypos += yvel;
        }

        void push(int dvx, int dvy) {
                xvel += dvx;
```

```
                yvel += dvy;
        }
};
```

4.2.2.3 Smart pointers

With a `unique_ptr` you can make sure that only one object, the owner of the pointer, can access the value behind the pointer. Consider this epic tale of a king and his magic sword:

```
#include <iostream>
#include <string>
#include <memory>

using std::string;

class Weapon {
  private:
    string name;

  public:
    Weapon(string n = "rusty sword") : name(n) {

    }

    ~Weapon(){
      std::cout << name << " destroyed" << std::endl;
    }

    string getName() {
      return name;
    }
};

class Hero {
  private:
    string name;
    Weapon* weapon_ptr;

  public:
    Hero(string n) : name(n), weapon_ptr(nullptr) {}

    ~Hero(){
      if(weapon_ptr != nullptr){
        std::cout << name << " now destroying " << weapon_ptr->getName() << std::endl;
        delete weapon_ptr;
      }
      std::cout << name << " destroyed." << std::endl;
    }

    void pickUpWeapon(Weapon* w) {
      if(weapon_ptr != nullptr) delete weapon_ptr;
      weapon_ptr = w;
    }

    string describe() {
      if(weapon_ptr != nullptr){
        return name + " now swings " + weapon_ptr->getName();
      } else {
        return name + " now swings his bare hands";
      }
    }
};

Weapon* blacksmithForge(string name){
  Weapon* wp = new Weapon(name);
  return wp;
}


int main() {

  Hero arthur = Hero("Arthur");
  Hero mordred = Hero("Mordred");
  Weapon* excalibur = blacksmithForge("Excalibur");

  std::cout << arthur.describe() << std::endl;
  arthur.pickUpWeapon(excalibur);
  std::cout << arthur.describe() << std::endl;
  mordred.pickUpWeapon(excalibur);
  std::cout << mordred.describe() << std::endl;
  std::cout << "Everything going out of scope." << std::endl;
```

```
  return 0;
}
```

Note how Mordred has snatched away Arthurs sword. When mordred dies, he takes Excalibur with him into the grave. When Arthur dies, he tries to do the same, but has to find that the sword he thought he had was no longer there. We can fix this problem by using `unique_ptr`'s:

```
#include <iostream>
#include <string>
#include <memory>

using std::string;

class Weapon {
  private:
    string name;

  public:
    Weapon(string n = "rusty sword") : name(n) {};

    string getName() {
      return name;
    }
};

class Hero {
  private:
    string name;
    std::unique_ptr<Weapon> weapon_ptr;

  public:
    Hero(string n) : name(n), weapon_ptr(new Weapon(n + "'s bare hands")) {}

    void pickUpWeapon(std::unique_ptr<Weapon> w) {
      if(!w){
        std::cout << "Nothing to pick up." << std::endl;
        return;
      }
      weapon_ptr = std::move(w);
    }

    string describe() {
        return name + " now swings " + weapon_ptr->getName();
    }
};

std::unique_ptr<Weapon> blacksmithForge(string name){
  std::unique_ptr<Weapon> wp(new Weapon(name));
  return wp;
}


int main() {

  Hero arthur = Hero("Arthur");
  Hero mordred = Hero("Mordred");
  std::unique_ptr<Weapon> excalibur = blacksmithForge("Excalibur");

  std::cout << arthur.describe() << std::endl;
  arthur.pickUpWeapon(std::move(excalibur));
  std::cout << arthur.describe() << std::endl;

  std::cout << "Mordred now tries to snatch Excalibur." << std::endl;
  mordred.pickUpWeapon(std::move(excalibur));
  std::cout << mordred.describe() << std::endl;


  std::cout << "Everything going out of scope." << std::endl;

  return 0;
}
```

4.2.2.4 Ownership: unique, shared and weak pointers

Let us explain in a bit more detail what a smart pointer is. Really, it i just a class encapsulating a raw pointer. The idea is that by wrapping a pointer in a class, we can have the class object be allocated on the stack and have it free the memory behind the pointer when the object is destroyed. This way, you don't have to manually delete a pointer.

```
class SmartPointer{
```

```
    private:
        T* ptr;
    public:
        SmartPointer(T* p){
            ptr = p;
        }
        ~SmartPointer(){
            free(ptr);
        }
}
```

In this example, we allocate Excalibur on the heap. But because we're using smart pointers, we never have to clean up memory manually:

```
#include <iostream>
#include <memory>

using std::string;

class Weapon{
  private:
    string name;
  public:
    Weapon(string n) : name(n){}
    string getName(){
      return name;
    }
    ~Weapon(){
      std::cout << name << " is being destroyed" << std::endl;
    }
};

class Hero{
  private:
    string name;
    std::unique_ptr<Weapon> weapon;
  public:
    Hero(string n) : name(n), weapon(new Weapon("rusty sword")) {}
    void pickWeapon(std::unique_ptr<Weapon> w){
      std::cout << name << " picked up " << w->getName() << std::endl;
      weapon = std::move(w);
      std::cout << weapon->getName()  << " is now on " << name << std::endl;
    }
    ~Hero(){
      std::cout << name << " is being destroyed" << std::endl;
    }
};

int main() {

  Hero arthur("Arthur");
  std::unique_ptr<Weapon> w(new Weapon("Excalibur"));
  arthur.pickWeapon(std::move(w));

  return 0;
}
```

Note how we used `move(w)` to overwrite `weapon`. What happens here is this: In the main-scope, the `w`'s internal pointer to Excalibur is replaced by a `nullptr`. Any further attempt of accessing `w` will cause a runtime-error. Inside the `pickWeapon` method, the old smartpointer to the rusty sword is destroyed, causing the sword itself to be free'd. We could'nt have used `weapon = w;`; that would have caused an error (because `unique_ptr`'s copy-assignment method is delted). This is deliberate: we don't want copies of a unique pointer to exist. We want a unique pointer to only ever exist in one place (in this case: first in the `main` function, later in the `arthur` instance), so that the memory behind the pointer can only get freed once (in this case: when `arthur` is destroyed).

Generally, we can proclaim the following rules:

- When you're given a reference, someone else will clean up the original.
- When you're given a unique pointer, you are now the sole owner of the smartpointer. If you move the pointer, it will go out of scope once you execute that move; if you don't, it will go out of scope once your method ends. Passing a unique pointer to a method that doesn't move the pointer means destroying the object! To pass a unique pointer to a method without it being destroyed or moved, don't pass the unique pointer, pass a `unique_ptr<T> const &`.
- When you're given a shared pointer, ... Shared pointers use reference counting just like the java garbage collector.

There are a lot more useful tips here and here.

Using ownership wisely Based on this, we can define different ownership strategies depending on whether an object is a value-object or an id-object.

4.2.3 Templates

Templates are like java generics.

4.2.3.1 Function template

For a function to accept a generic parameter, just prepend `template <class myType>` to the function definition.

```
#include <iostream>
#include <string>

using namespace std;

template <class myType>
myType GetMax (myType a, myType b) {
 return (a>b?a:b);
}

int main () {
   int i = 39;
   int j = 20;
   cout << "Max(i, j): " << GetMax(i, j) << endl;

   double f1 = 13.5;
   double f2 = 20.7;
   cout << "Max(f1, f2): " << GetMax(f1, f2) << endl;

   string s1 = "Hello";
   string s2 = "World";
   cout << "Max(s1, s2): " << GetMax(s1, s2) << endl;

   return 0;
}
```

4.2.3.2 Class template

Things work very simmilarly with classes:

```
#include <iostream>
using namespace std;


template <class T>
class MyPair {
  private:
    T values [2];

  public:
    MyPair (T first, T second) {
      values[0]=first;
      values[1]=second;
    }

    T getMax() {
                return (values[0] > values[1] ? values[0] : values[1]);
    }

};

int main() {
        MyPair<int> myobject(100, 75);
        cout << myobject.getMax();
        return 0;
}
```

One last note on notation: if we had defined class and implementation in separate files, we'd have to write:

```
// ---- class definition ------

template <class T>
class MyPair {
        private:
                T values[2];
        public:
                MyPair(T first, T second);
                T getMax();
```

```
};

// ---- class implementation -----

template <class T>
MyPair<T>::MyPair(T first, T second) {
        values[0] = first;
        values[1] = second;
}

template <class T>
MyPair<T>::getMax() {
        return (values[0] < values[1] ? values[1] : values[0]);
}
```

4.2.4 Threading

Compared to c's posix threads, c++ has somewhat simplified it's core threading library.

```
#include <iostream>
#include <thread>

void threadFunc(int i) {
  for(int j = 0; j < 100; j++){
    std::cout << "hi! this is operation " << j <<" from thread nr " << i << std::endl;
  }
}

int main(){
  std::thread threads[10];

  for(int i = 0; i<10; i++){
    threads[i] = std::thread(threadFunc, i);
  }

  for(int i = 0; i<10; i++){
    threads[i].join();
  }

  return 0;
}
```

You can avoid some of the above problem using barriers in your code (`std::mutex`) which will let you synchronize the way a group of threads share a resource.

4.2.5 CMake

CMake input files are written in the "CMake Language" in source files named CMakeLists.txt or ending in a .cmake file name extension.

CMake Language source files in a project are organized into:

- *Directories* (CMakeLists.txt). When CMake processes a project source tree, the entry point is a source file called CMakeLists.txt in the top-level source directory. This file may contain the entire build specification or use the `add_subdirectory()` command to add subdirectories to the build. Each subdirectory added by the command must also contain a CMakeLists.txt file as the entry point to that directory. For each source directory whose CMakeLists.txt file is processed CMake generates a corresponding directory in the build tree to act as the default working and output directory.

- *Scripts* (script.cmake). An individual script.cmake source file may be processed in script mode by using the cmake(1) command-line tool with the -P option. Script mode simply runs the commands in the given CMake Language source file and does not generate a build system. It does not allow CMake commands that define build targets or actions.

- *Modules* (module.cmake). CMake Language code in either Directories or Scripts may use the include() command to load a module.cmake source file in the scope of the including context. See the cmake-modules(7) manual page for documentation of modules included with the CMake distribution. Project source trees may also provide their own modules and specify their location(s) in the `CMAKE_MODULE_PATH` variable.

4.2.5.1 Variables

Variables are the basic unit of storage in the CMake Language. Their values are always of string type, though some commands may interpret the strings as values of other types. The `set()` and `unset()` commands explicitly set or unset a variable, but other commands have semantics that modify variables as well. Variable names are case-sensitive and may consist of almost any text, but we recommend sticking to names consisting only of alphanumeric characters plus `_` and -.

Although all values in CMake are stored as strings, a string may be treated as a list in certain contexts, such as during evaluation of an Unquoted Argument. In such contexts, a string is divided into list elements by splitting on `;` characters not following an unequal number of `[` and `]` characters and not immediately preceded by a `\`. The sequence `\;` does not divide a value but is replaced by `;` in the resulting element.

A list of elements is represented as a string by concatenating the elements separated by `;`. For example, the `set()` command stores multiple values into the destination variable as a list:

```
set(srcs a.c b.c c.c) # sets "srcs" to "a.c;b.c;c.c"
```

Lists are meant for simple use cases such as a list of source files and should not be used for complex data processing tasks. Most commands that construct lists do not escape `;` characters in list elements, thus flattening nested lists:

```
set(x a "b;c") # sets "x" to "a;b;c", not "a;b\;c"
```

A variable reference has the form `${variable_name}` and is evaluated inside a Quoted Argument or an Unquoted Argument. A variable reference is replaced by the value of the variable, or by the empty string if the variable is not set. Variable references can nest and are evaluated from the inside out, e.g. `${outer_${inner_variable}_variable}`.

An environment variable reference has the form `$ENV{VAR}` and is evaluated in the same contexts as a normal variable reference.

4.2.5.2 Logic

- Conditional Blocks: The `if()/elseif()/else()/endif()` commands delimit code blocks to be executed conditionally.
- Loops: The `foreach()/endforeach()` and `while()/endwhile()` commands delimit code blocks to be executed in a loop. The `break()` command may be used inside such blocks to terminate the loop early.
- Command Definitions: The `macro()/endmacro()`, and `function()/endfunction()` commands delimit code blocks to be recorded for later invocation as commands.

4.2.5.3 Command arguments

- bracket argument:

```
message([=[
This is the first line in a bracket argument with bracket length 1.
No \-escape sequences or ${variable} references are evaluated.
This is always one argument even though it contains a ; character.
The text does not end on a closing bracket of length 0 like ]].
It does end in a closing bracket of length 1.
]=])
```

- quoted argument:

```
message("This is a quoted argument containing multiple lines.
This is always one argument even though it contains a ; character.
Both \\-escape sequences and ${variable} references are evaluated.
The text does not end on an escaped double-quote like \".
It does end in an unescaped double quote.
")
```

- unquoted argument:

```
foreach(arg
    NoSpace
    Escaped\ Space
    This;Divides;Into;Five;Arguments
    Escaped\;Semicolon
    )
  message("${arg}")
endforeach()
```

4.2.5.4 Important predefined variables

There is a list of CMake's global varibales. You should know them.

- CMAKE_BINARY_DIR: if you are building in-source, this is the same as CMAKE_SOURCE_DIR, otherwise this is the top level directory of your build tree
- CMAKE_SOURCE_DIR: this is the directory, from which cmake was started, i.e. the top level source directory
- EXECUTABLE_OUTPUT_PATH: set this variable to specify a common place where CMake should put all executable files (instead of CMAKE_CURRENT_BINARY_DIR):

```
SET(EXECUTABLE_OUTPUT_PATH ${PROJECT_BINARY_DIR}/bin)
```

- LIBRARY_OUTPUT_PATH: set this variable to specify a common place where CMake should put all libraries (instead of CMAKE_CURRENT_BINARY_DIR):

```
SET(LIBRARY_OUTPUT_PATH ${PROJECT_BINARY_DIR}/lib)
```

- PROJECT_NAME: the name of the project set by PROJECT() command.
- PROJECT_SOURCE_DIR: contains the full path to the root of your project source directory, i.e. to the nearest directory where CMakeLists.txt contains the PROJECT() command Now, you have to compile the test.cpp. The way to do this task is too simple. Add the following line into your CMakeLists.txt:

```
add_executable(hello ${PROJECT_SOURCE_DIR}/test.cpp)
```

4.3 RUST

4.3.1 Tools

4.3.1.1 Compiler: rustc

```
rustc main.rs
./main
```

4.3.1.2 Buildtool & package-manager: cargo

```
cargo new my_project_name
cargo build [--release]
cargo check  # checks but doesn't build executable (faster)
cargo update  # installs the latest versions matching Cargo.lock.toml.dependencies. updates Cargo.lock
cargo run  # builds & executes
cargo doc --open  # builds and opens browser docs for all used crates
```

4.3.2 Language

- Statements perform an action, but don't return a value.
- Expressions evaluate to a value.

Example: assignment is a statement with an expression as part of it. "' statement I————————-I assignment expression I———I I———I let sum = 1 + 2; "' Example: function definitions are statements. Note: calling a macro is an expression. Calling a block '' is an expression, too.

4.3.2.1 Memory management

Stack and heap

- There is no 'free' for the stack.
- Adding data onto the stack is called 'pushing'.
- Reserving space on the heap is called 'allocating'. In other words, the stack knows no reserving; allocating always refers to the heap.
- Pushing to the stack is faster than allocating on the heap, because the stack always knows where the next free space is: it's always on the very top of the stack.
- Also reading from the stack is faster then from the heap, because reading from heap involves following potentially many pointers. In fact, the only reasons to use the heap are
  - when we want to be able to dynamically grow or shrink memory at runtime,
  - when we want data to persist multiple scopes (like something complex being returned from a function)
- When calling a function, the arguments (and any function-local variables) (including, potentially, pointers to the heap) get pushed onto the stack

Ownership rules

- each value has a variable called its *owner*.
- there can be only one owner at a time.
- when the owner goes out of scope '', the value will be dropped.

```
let s1 = String::from("hello");
let s2 = s1;
println!("{}, world!", s1);  // <-- throws error
```

When re-assigning a value on the heap (like here: “String::from(”hello”);‘ is being re-assigned from ‘s1‘ to ‘s2‘) rust considers the first variable ‘s1‘ as invalid. Why? If ‘s1‘ and ‘s2‘ were both still valid at the end of the scope, they’d both attempt to drop their memory at the end of the scope. That’s called a double-free, a common error in c++.

Note that the re-assignment was a shallow copy: only the pointer to the data had been copied, not the data. In fact, rust *never* automatically does a deep-copy. So you can always assume that re-assignments are fast and cheap. If you want to have a deep-copy, use ‘let s2 = s1.clone();‘

Note also that this rule only applies to heap-data. Stack-only data will always be copied:

```
let x = 5;
let y = x;
println!("x = {}, y = {}", x, y);  // <--  no error here
```

When creating your own data-types:

- ‘Copy‘ trait: makes it clear that this is a stack-data-structure. Called when re-assigning this data-structure.
- ‘Drop‘ trait: makes it clear that this is a heap-data-structure. Called for cleanup when data-structure goes out of scope.

Function-calls Passing variables to functions is just like re-assignment: it will move or copy, just like re-assignment does. As such, it behaves differently for heap- and stack-data.

For heap-variables, the function takes ownership of the variable, meaning that the calling scope may no longer access the value.

```
let s = String::from("hi");
print_string(s);
println!("{}", s);  // <-- throws error


fn print_string(some_string: String) {
    println!("{}", some_string);
    // some_string now goes out of scope and ‘drop‘ is called
}
```

Same thing for returning values from functions:

```
fn make_string() -> String {
    let some_string = String::from("hi");
    some_string  // some_string goes out of scope ... but since it’s value is being ‘move‘d, the data is not freed.
}

fn main() {
    let s1 = make_string();  // ownership moves from some_string to s1
    println!("{}", s1);
}  // now "hi" moves out of scope, drop called.
```

How do you operate on data without destroying it? You return it again after each operation, meaning you pass ownership to the function and then pass it back into a new variable. Example:

```
fn main() {
    let s1 = String::from("hi");  // "hi" owned by s1
    let (s2, len) = calculate_length(s1);  // "hi" owned by s and then s2
    println("the length of ’{}’ is {}", s2, len);
}
fn calculate_length(s: String) -> (String, usize) {
    let length = s.len();
    (s, length)
}
```

References But there is another way to operate on data without passing ownership: *references and borrowing*.

```
fn main() {
    let s1 = String::from("hi");
    let len = calculate_length(&s1);
    println("the length of ’{}’ is {}", s1, len);
}
fn calculate_length(s: &String) -> usize {
    s.len()
}
```

Passing a value as a reference makes sure that no ‘drop‘ is called at the end of the scope, and the calling scope's variable remains the owner of the data.

Taking a closer look: ‘let ref = &s1;‘ creates a *reference* which is only a pointer to ‘s1‘ and does not take ownership away from ‘s1‘.

Note that references are immutable:

```
fn main() {
    let s = String::from("hi");
    change(&s);  // <-- throws error
}
fn change(s: &String) {
    s.push_str(", world");
}
```

We need to explicitly specify that this data may be mutated:

```
fn main() {
    let mut s = String::from("hi");
    change(&mut s);
}
fn change(s: &mut String) {
    s.push_str(", world");
}
```

But then there is another restriction: we may only ever have one mutable reference to some data. This helps so that threads cannot interact.

```
let mut s = String::from("hi");
let r1 = &mut s;
let r2 = &mut s;
println!("{}, {}", r1, r2);  // <-- will fail
```

(Note that you may have multiple *im*mutable references)

Slices Slices are another type of references. They are references to a *contiguous part* of some data.

```
fn main() {
    let sentence = String::from("I'm a sentence.");
    let first_word = get_first_word(&sentence);
    println!("The first word in '{}' is: {}", sentence, first_word);
    sentence.clear();  // <-- this will cause a *positive* error: compiler will warn us that the sentence's data is still
    println("After clearing, the first word is now {}", first_word);
}

fn get_first_word(s: &String) -> &str  {  // &str and &[i32] are examples of slice-types
    &s[0..5]
}
```

Boxes Boxes are smart-pointers: they contain a reference to some heap-data and the data's size. The data is de-allocated once the box no longer has an owner.

Example: Box to create a linked list:

```
struct Entry {
    text: String,
    next: Option<Box<Entry>>,
}

fn create_linked_list(first_text: String) -> Box<Entry> {
    let e = Entry {
        text: first_text,
        next: None
    };
    Box::new(e)
}

fn append_new_entry(list: &mut Box<Entry>, text: String) {
    match &mut list.next {
        Some(n) => append_new_entry(n, text),
        None => {
            let new_entry = create_linked_list(text);
            list.next = Some(new_entry);
        }
    }

}

fn read_all_entries(list: &Box<Entry>) {
    println!("{}", list.text);
```

```
    match &list.next {
        Some(n) => read_all_entries(&n),
        None => return,
    }
}


fn main() {
    let mut list = create_linked_list(String::from("First entry"));
    append_new_entry(&mut list, String::from("Second entry"));
    append_new_entry(&mut list, String::from("Third entry"));
    read_all_entries(&list);
}
```

Example: Box to create a heap-array:

```
fn create_heap_array(size: usize, fill_value: f32) -> Box<[f32]> {
    // 0. Cannot just do the below line, because rust won't allow runtime-sized arrays [fill_val; size]
    // Box::new([fill_value; size])
    // 1. Instead: using vec as an intermediate step
    let data = vec![fill_value; size];
    // 2. Turning vec into Box
    let b = data.into_boxed_slice();
    b
}

fn main() {
    let size = 10;
    let arr_fives = create_heap_array(size, 5.0);
    println!("{}", arr_fives[5]);
}
```

Lifetimes Lifetimes are required when we want structs to use data that is owned by someone else.

4.3.2.2 Unsafe rust

Creating that linked list was a pain. Fortunately, when we promise rust that we know what we're doing, we can use a more c-like, [*unsafe* version of the code](https://rust-unofficial.github.io/too-many-lists/fifth-basics.html).

4.3.2.3 enums

- enums
  - Result: enumOk, Err
  - Ordering: enumLess, Greater, Equal
  - Option: enumSome, None
  - Maybe:

Letting 'Result' crash if there is an error:

```
let guess: u32 = guess_str.trim().parse.expect("Something went wrong!");
```

Better: handling potential error:

```
let guess: u32 = match guess_str.trim().parse() {
    Ok(num) => num,
    Err(_) => continue,
};
```

4.3.2.4 Structs

```
struct Matrix {
    rows: usize,
    cols: usize,
    data: Vec<f32>,
}

impl Matrix {
    fn zeros(rows: usize, cols: usize) -> Matrix {
        Matrix {
            rows,
            cols,
```

```
            data: vec![0.0; (rows * cols).into()]
        }
    }

    fn eye(rows: usize, cols: usize) -> Matrix {
        let mut d = Vec::with_capacity(rows * cols);
        for r in 0..rows {
            for c in 0..cols {
                if r == c {
                    d.push(1.0);
                } else {
                    d.push(0.0);
                }
            }
        }
        Matrix {
            rows,
            cols,
            data: d
        }
    }

    fn count_up(rows: usize, cols: usize) -> Matrix {
        let size = rows * cols;
        let mut d = Vec::with_capacity(size);
        for i in 0..size {
            d.push(i as f32);
        }
        Matrix {
            rows,
            cols,
            data: d
        }
    }

    fn get(&self, row: usize, col: usize) -> f32 {
        let i = self.cols * row + col;
        self.data[i]
    }

    fn set(&mut self, row: usize, col: usize, val: f32) {
        let i = self.cols * row + col;
        self.data[i] = val;
    }

    fn dot(&self, other: &Matrix) -> Matrix {
        if self.cols != other.rows {
            panic!("Matrices A and B cannot be multiplied: A.cols != B.rows");
        }

        let mut out = Matrix::zeros(self.rows, other.cols);
        for r in 0..self.rows {
            for c in 0..other.cols {
                let mut sum = 0.0;
                for i in 0..self.cols {
                    sum += self.get(r, i) * other.get(i, c);
                }
                out.set(r, c, sum);
            }
        }

        out
    }
}

fn main() {
    let eye = Matrix::eye(2, 2);
    let cnt = Matrix::count_up(2, 3);
    let prd = eye.dot(&cnt);
    assert_eq!(eye.rows, prd.rows);
    assert_eq!(cnt.cols, prd.cols);
    assert_eq!(prd.data[0], 0.0);
    assert_eq!(prd.data[3], 3.0);
}
```

4.3.2.5 Macros

... can be recognized by the ‘¡ in the macro’s name.

4.4 Java

We tend to use java when we want to create enterprise level software. The whole development-experience is way different from c. While in c you control every detail of the program, a java-app has you reuse as many standardized components as possible. There are layers upon layers of abstraction. Usually, there is some way to build on top of a framework, leaving you to only fill out a few xml-configurations and implement a few skeleton-beans with loads of annotations.

4.4.1 General

4.4.1.1 Basic theory

4.4.1.2 Environment variables

First things first, to find out if you're using a 64 bit version of the jdk, just execute

```
java -d64 -version
```

This will throw an error if your version is not made for 64 bit.

There are a few environment variables that you should know about:

- PATH: Sys looks for exes here. Your JAVA_HOME/bin should be part of PATH
- CLASSPATH: Java looks for code here
- JAVA_HOME: location jdk (example: /usr/lib/jvm/java-7-openjdk-amd64/bin)
- JRE_HOME: location jre (example: /usr/lib/jvm/java-7-openjdk-amd64/jre/bin)

The most important setting for the JVM is the heap size: how much memory will we allocate to the java-process? There are two settings:

- -Xms<size> - Set initial Java heap size
- -Xmx<size> - Set maximum Java heap size

These are either set in .... or used directly when invoking java with `java -Xms512m -Xmx1024m JavaApp`.

You can display the default settings like this:

```
$ java -XX:+PrintFlagsFinal -version | grep -iE 'HeapSize|PermSize|ThreadStackSize'

    uintx InitialHeapSize                          := 64781184        {product}
    uintx MaxHeapSize                              := 1038090240      {product}
    uintx PermSize                                  = 21757952        {pd product}
    uintx MaxPermSize                               = 174063616       {pd product}
     intx ThreadStackSize                           = 1024            {pd product}
java version "1.7.0_51"
OpenJDK Runtime Environment (IcedTea 2.4.4) (7u51-2.4.4-0ubuntu0.13.10.1)
OpenJDK 64-Bit Server VM (build 24.45-b08, mixed mode)
```

Here are some suggested values:

- Heap = -Xms512m -Xmx1024m
- PermGen = -XX:PermSize=64m -XX:MaxPermSize=128m
- Thread = -Xss512k

4.4.1.3 Versions

Unfortunately, with java11 and its module system and the deprecation of former JavaEE packages, java has introduced its own breaking change á lá python2/3. Often, you will find that older programs only run in java8 or lower. You can see which java versions you have installed and choose the appropriate one by

```
update-alternatives --config java
```

Usually, java programs will have their own, linux-specific startup-script, like `/etc/init.d/openfire`. Here, these scripts will pick java based on `$JAVA_HOME`. To set this variable for your own user only, append

```
export JAVA_HOME="/usr/lib/jvm/java-8-openjdk-amd64/jre"
export PATH=$JAVA_HOME/bin:$PATH
```

to `~/.bashrc`, to set it for every user, append to `/etc/profiles` for login-shells and `/etc/bash.bashrc` for non-login-shells. Even better, put it in `/etc/environment` to cover both cases at once.

4.4.2 Maven

Maven is a build- and dependency-resolution tool. Building eg. webprojects with maven is encouraged, also because maven takes away buildsteps from your IDE. Relying on maven instead of an IDE to do your building then makes your builds portable between different developement environments.

Maven can initialise a project structure for you from the command-line:

```
mvn archetype:generate -DarchetypeArtifactId=maven-archetype-quickstart
```

It will then ask you all further neccessary information.

cli syntax The syntax generally consists of mvn [lifecycle]:[phase]:[args]

```
mvn default:package
mvn [lifecyle]:[phase]
mvn default:package:help
```

build lifecyles Maven has a quite hierarchical structure to it.

- lifecycles: default (compiling et al), clean (remove artifacts), site (create documentation). Plugins might add further lifecycles.
  - phases : default build lifecycle consists of several phases: validate, compile, test, package, verify, install, deploy
    - goals: each phase executes one or more goals. these are the actual code instuctions.
  - profiles : here we can adjust some environment settings. should never really be neccessary.
- plugin: a set of extra goals for maven. plugins might even define whole new lifecycles like 'mvn jetty:run', or hook into the build cycle to generate code from xml like cxf does.
  - mojo : maven pojo. a class representing an executable goal
  - configuration of plugins. executions : specifies how the mojo should be executed; ie in which phase of the maven build-lifecycle

Directory structure Everything that you put under *src/main/resources* during development will be put under the root-directory on compilation. Same thing holds for the contents of *src/test/resources*: they will be on the root path during execution of tests. So, when using relative paths, *src/*/resources/* must be omitted.

Custom archetypes

Custom plugins

4.4.3 Threading

Java was designed to make threading easy (though, contrary to clojure, it was not designed to make threading *save*).

Basic threading with shared data is the simplest, but somewhat dangerous method.

```
public class Main {

        public static void main(String[] args) throws InterruptedException {

                LinkedList<Integer> sharedData = new LinkedList<Integer>();
                Producer p = new Producer(sharedData);
                Consumer c = new Consumer(sharedData);
                p.start();
                Thread.sleep(1000);
                c.start();

        }

}
```

```
public class Producer extends Thread {

        private LinkedList<Integer> sharedData;

        public Producer(LinkedList<Integer> sharedData) {
                this.sharedData = sharedData;
        }

        @Override
        public void run() {
                Random r = new Random();
                while(true) {
                        try {
                            Integer i = r.nextInt(5);
                                Thread.sleep(100 * i);
                        sharedData.add(i);
                        System.out.println("Producer just added " + i + " to the shared data.");
                        System.out.println("Data now looks like this: " + sharedData.toString());
                        } catch (InterruptedException e) {
                                e.printStackTrace();
                        }
                }
        }
}
```

```
public class Consumer extends Thread {

        private LinkedList<Integer> sharedData;

        public Consumer(LinkedList<Integer> sharedData) {
                this.sharedData = sharedData;
        }

        @Override
        public void run() {
                Random r = new Random();
                while(true) {
                        try {
                                Thread.sleep(100 * r.nextInt(5));
                                i = sharedData.removeFirst();
                        System.out.println("Consumer just took " + i + " from the shared data.");
                        System.out.println("Data now looks like this: " + sharedData.toString());
                        } catch (InterruptedException e) {
                                e.printStackTrace();
                        }
                }
        }

}
```

This results in an output like this:

```
Producer just added 3 to the shared data.
Data now looks like this: [0, 0, 3]
Consumer just took 0 from the shared data.
Data now looks like this: [0, 3]
Consumer just took 0 from the shared data.
Data now looks like this: [3]
Consumer just took 3 from the shared data.
Data now looks like this: []
Exception in thread "Thread-1" java.util.NoSuchElementException
        at java.util.LinkedList.removeFirst(LinkedList.java:270)
        at threadingStuff.Consumer.run(Consumer.java:20)
```

Notice how at some point the consumer overtook the producer, leading to a `NoSuchElementException` . We can use signaling to handle this kind of situation:

- `wait()`
- `notify()`

Alternatively, a blocking queue would automatically handle such a case by just blocking the consumer until data is available again.

Using a blocking queue makes sense when

- you need protection from `NoSuchElementException` .
- it is ok for the consumer to sometimes be blocked while waiting for data to arrive.

```
public class Main {

        public static void main(String[] args) throws InterruptedException {

                ArrayBlockingQueue<Integer> sharedData = new ArrayBlockingQueue<Integer>(100);
                Producer p = new Producer(sharedData);
                Consumer c = new Consumer(sharedData);
                p.start();
                Thread.sleep(1000);
                c.start();

        }

}
```

```
public class Producer extends Thread {

        private ArrayBlockingQueue<Integer> sharedData;

        public Producer(ArrayBlockingQueue<Integer> sharedData) {
                this.sharedData = sharedData;
        }

        @Override
        public void run() {
                Random r = new Random();
                while(true) {
                        try {
                                Integer i = r.nextInt(5);
                                Thread.sleep(100 * i);
                                sharedData.put(i);
                                System.out.println("Producer just added " + " to the shared data.");
                                System.out.println("Data now looks like this: " + sharedData.toString());
                        } catch (InterruptedException e) {
                                e.printStackTrace();
                        }
                }
        }
}
```

```
public class Consumer extends Thread {

        private ArrayBlockingQueue<Integer> sharedData;

        public Consumer(ArrayBlockingQueue<Integer> sharedData) {
                this.sharedData = sharedData;
        }

        @Override
        public void run() {
                Random r = new Random();
                Integer i;
                while(true) {
                        i = null;
                        try {
                                Thread.sleep(100 * r.nextInt(5));
                                i = sharedData.take();
                                System.out.println("Consumer just took " + i + " from the shared data.");
                                System.out.println("Data now looks like this: " + sharedData.toString());
                        } catch (InterruptedException e) {
                                e.printStackTrace();
                        }
                }
        }

}
```

This results in an output like this:

```
Producer just added 3 to the shared data.
Data now looks like this: [0, 0, 3]
Consumer just took 0 from the shared data.
Data now looks like this: [0, 3]
Consumer just took 0 from the shared data.
Data now looks like this: [3]
Consumer just took 3 from the shared data.
Data now looks like this: []     // <------ no exception thrown here! Consumer just has to wait.
Producer just added 3 to the shared data.
Data now looks like this: [3]
Producer just added 4 to the shared data.
Data now looks like this: [3, 4]
```

Using a pipe makes only sense under very specific conditions. A pipe is optimized for serialized in- and output, so you'd first have to serialize and then deserialize anything that has to pass through the pipe. Also, a pipe can always only exist between exactly two threads, no more.

Using an executor service makes sense when ...

Calling another thread's mehtods is actually possible without much ado:

```
class Main {
    public static void main() {
        RT rt = new RT();
        rt.start();
        rt.updateV(3);
    }
}
```

```
class RT extends Thread {
    private int v;

    public RT() {
        v = 1;
    }

    public void run() {
        while(true){
            Thread ct = Thread.currentThread();
            String name = ct.getName();
            System.out.println("Method run() being executed on " + name);
            System.out.println("v now is: " + v);
        }
    }

    public void updateV(int newV) {
        Thread ct = Thread.currentThread();
        String name = ct.getName();
        System.out.println("Method updateV() being executed on " + name);
        v = newV;
    }
}
```

A thread's public methods may be called from anywhere. So the main thread can call `accessibleMethod()` without problems. However, the execution of that public method will than take place on the calling thread - in that case, the main thread. You can verify this by sparkling your code with a few `Thread.currentThread().getName()` calls. But the called method may still manipulate a variable that is private to the called thread; so the main thread may manipulate `v`. This is possible because threads share the same memory base.

4.4.4 Functional programming

Java 8 exposes a few new Classes that can be used as functions.

- `Function<Integer,Integer> add3 = a -> a + 3;`
- `BiFunction<Integer, Integer, Integer> add = (a, b) -> a + b;`
- `Predicate<Integer> isEven = a -> a%2 == 0;` A predicate always returns a boolean.
- `Consumer<Book> storeAway = b -> b.save();` A consumer returns void.

```
public static void main() {
    BiFunction<Integer, Integer, Integer> add = (a, b) -> a + b;
    BiFunction<Integer, Integer, Integer> sub = (a, b) -> a - b;
    Integer result1 = compute(add, 3, 4);
    Integer result2 = compute(sub, 3, 4);
}

public static Integer compute(BiFunction<Integer, Integer, Integer> function, Integer a, Integer b) {
    return function.apply(a, b);
}
```

These functions could also have been passed anonymously:

```
public static void main() {
    Integer result1 = compute((a, b) -> a + b, 3, 4);
    Integer result2 = compute((a, b) -> a - b, 3, 4);
}

public static Integer compute(BiFunction<Integer, Integer, Integer> function, Integer a, Integer b) {
    return function.apply(a, b);
}
```

4.4.5 Annotations

We mostly use annotations to let others read out meta-information about our pojos. Consider the following example. In this code, we define what kind of metadata we want to create.

```
import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Retention(RetentionPolicy.RUNTIME)
public @interface TableRecord {
        String table();
        String database();
}

@Target(ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
public @interface TableField {
        String name();
}
```

Then we decorate a pojo with the new annotations.

```
import org.apache.camel.dataformat.bindy.annotation.CsvRecord;
import org.apache.camel.dataformat.bindy.annotation.DataField;

@CsvRecord( separator = ";")
@TableRecord(table="iw_mn", database="gkddat")
public class Messnetz {

        @DataField(pos = 1)
        @TableField(name="Messnetznummer")
        private int Messnetznummer;

        @DataField(pos = 2)
        @TableField(name="Messnetzname")
        private String Messnetzname;

        public Messnetz() {}

        public Messnetz(int msnr, String mnname) {
                Messnetznummer = msnr;
                Messnetzname = mnname;
        }

        ... getters and setters ...
}
```

And finally we read out the annotations to create an sql-query:

```
public class SqlConverter implements DataFormat {

        /**
         * This is to convert any object into a "INSERT" sql statement.
         * Makes use of the @TableRecord and @TableField annotations to figure out
         * what parts of the pojo need to be persisted.
         */
        @Override
        public void marshal(Exchange ex, Object graph, OutputStream stream) throws Exception {

                Object pojo = ex.getIn().getBody();
                TableRecord tr = pojo.getClass().getAnnotation(TableRecord.class);
                Field[] fields = pojo.getClass().getDeclaredFields();

                String database = tr.database();
                String table = tr.table();

                ArrayList<String> fieldNames = new ArrayList<String>();
                ArrayList<String> fieldValues = new ArrayList<String>();
                for(Field f : fields) {
                        TableField tf = f.getAnnotation(TableField.class);
```

```
                        if(tf == null) continue;

                        String fieldName = tf.name();
                        fieldNames.add(fieldName);

                        f.setAccessible(true);
                        Object objFieldValue = f.get(pojo);
                        String fieldValue = objFieldValue.toString();
                        if(f.getType() == String.class) {
                                fieldValue = "'" + fieldValue + "'";
                        }
                        fieldValues.add(fieldValue);
                }
                String fieldList = StringUtils.join(fieldNames);
                String valueList = StringUtils.join(fieldValues);

                String sql = "insert into "+ database +"." + table + " (" + fieldList + ") values (" + valueList + ");";

                stream.write(sql.getBytes());
        }
}
```

4.4.6 Transforming textformats: marshaling

This is known as marshaling. The general idea is always a simple two-step proces. To convert one format into another,

- Unmasrshal the file (that uses format 1) into a pojo,
- then marshal the pojo into another file (that uses format 2).

4.4.6.1 CSV to POJO and back: Bindy

Important annotations in the pojo are:

4.4.6.2 JSON to POJO and back: Jackson

Important annotations in the pojo are:

4.4.6.3 XML to POJO and back: JAXB

Important annotations in the pojo are:

4.4.7 Reading from documents

You don't always want to load a whole file into a pojo. Sometimes it's enough to just extract one simple information out of the file.

4.4.7.1 Extracting from xml: XPath

4.4.7.2 Extracting from json: JsonPath

4.4.8 Databases

Java allows for simple query-based database access (JDBC) and for automated marshaling of pojos into tables (JPA and hibernate).

4.4.8.1 First level db-access: JDBC

Every kind of database (MySQL, JavaDB, filesystembased, ...) can be accessed by its own *driver*. Because those drivers are vendor-specific, they are abstracted away using a *driver manager*.

```
Connection conn = DriverManager.getConnection("jdbc:mysql://10.112.70.133", "michael", "meinpw");
Statement stmt = conn.createStatement();
ResultSet res = stmt.executeQuery("select * from amoado.meldung");
while(res.next()){
    ...
}
```

However, nowadays it is more common to set up a DataSource instead of using the driver manager. The datasource will take care of managing a pool of connections, whereas with a drivermanager you would have to handle a connection-pool manually. Note how the concrete implementation of the Datasource is vendor specific. It therefore makes sense to create a factory.

```
MysqlDataSource mysqlDS = new MysqlDataSource();
mysqlDS.setURL("jdbc:mysql://localhost:3306/UserDB");
mysqlDS.setUser("michael");
mysqlDS.setPassword("meinpw");

...

DataSource ds = myFactory.getDs("mysql");
Connection con = ds.getConnection();
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("select empid, name from Employee");
while(rs.next()){
    ...
}
```

4.4.8.2 Second level db-access: JPA and Hibernate

The JPA is a specification implemented by Hibernate (say: hibernate is a "jpa provider"), among others. Its function is to persist pojos in a relational database - it's an ORM. Hibernate will accept pojos and scan them for annotations explaining how the pojo should be persisted (if you don't want to use annotations, you can instead create a mapping.xml).

Hibernate Hibernate will generate automatic sql for you. It will also create tables where needed. Therefore, it makes no sense to use it on an existing database where the structure mustn't change. Under such conditions, you're way better off using JDBC (or even better spring jdbc-template).

```
<dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>4.3.5.Final</version>
</dependency>
<!-- Hibernate 4 uses Jboss logging, but older versions slf4j for logging -->
<dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-simple</artifactId>
        <version>1.7.5</version>
</dependency>
<dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.0.5</version>
</dependency>
```

```
package hibernate;

import java.util.Date;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.UniqueConstraint;

@Entity
@Table(name="Employees", uniqueConstraints= {@UniqueConstraint(columnNames = { "id" })})
public class Employee {

        @Id
        @GeneratedValue(strategy=GenerationType.IDENTITY)
        private int id;
        private String name;
        private String role;
        private Date insertTime;

        ... bunch of getters and setters ...
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
                "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
```

```
                "http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
        <session-factory>
                <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
                <property name="hibernate.connection.url">jdbc:mysql://localhost/testdb</property>
                <property name="hibernate.current_session_context_class">thread</property>
                <mapping class="hibernate.Employee"/>
        </session-factory>
</hibernate-configuration>
```

```
public class AppMain {

        public static void main(String[] args) {
                Configuration conf = new Configuration();
                conf.configure("hibernate-annotation.cfg.xml");
                ServiceRegistry sr = (ServiceRegistry) new StandardServiceRegistryBuilder().applySettings(conf.getProperti
                SessionFactory sf = conf.buildSessionFactory(sr);
                Session s = sf.getCurrentSession();


                Employee e = new Employee();
                e.setName("Michael");
                e.setRole("programmer");
                e.setInsertTime(new Date());

                s.beginTransaction();
                s.save(e);
                s.getTransaction().commit();

                System.out.println("Employee id: " + e.getId());

                HibernateUtil.getSessionFactory().close();
        }
}
```

JPA Hibernate has later been generalized to JPA. JPA can use hibernate as backend, but might also use eclipselink or any other ORM. Unfortunately, JPA comes with a few changes compared to the hibernate syntax, not all of them for the better.

See here for an example-application:

```
EntityManagerFactory emp = Persistence.createEntityManagerFactory("employeeDB");
// This is a factory-factory. Talk about over-engineering :)
// employeeDB is the name of a persistance unit in your persistence.xml.
// Note that the position of this xml is defined nowhere - it MUST be placed in classroot/META-INF/persistence.xml
// and have exactly this name.

EntityManager em = emf.createEntityManager();
// The entity-manager is the most important class and basically your CRUD-Interface to the database.

EntityTransaction tx = em.getTransaction();

tx.begin();
Employee empl = new Employee("Homer", "Simpson", "97G");
em.persist(empl);
tx.commit();
em.close();
```

This class makes use of a `META-INF/persistence.xml` file. If this file is not found under this exact name in that specific location, a `"no persistence provider found"` exception is thrown.

```
<persistence
    xmlns="http://java.sun.com/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    version="1.0">

    <!-- Value given to createEntityManagerFactory -->
    <persistence-unit name="employeeDB" transaction-type="RESOURCE_LOCAL">

        <!-- All beans that you want to persist must be mentioned here -->
        <class>org.langbein.michael.Employee</class>

        <!-- Database-driver, credentials, and JPA-settings (differ per vendor, but generally logging, create tables etc)
        <properties>
            <property name="openjpa.ConnectionDriverName" value="org.hsqldb.jdbcDriver" />
            <property name="openjpa.ConnectionURL" value="jdbc:hsqldb:mem:order" />
            <property name="openjpa.ConnectionUserName" value="sa" />
            <property name="openjpa.ConnectionPassword" value="" />
            <property name="openjpa.jdbc.SynchronizeMappings" value="buildSchema" />
        </properties>
```

```
    </persistence-unit>

</persistence>
```

This is not a very convenient setup. In the spring appendix we'll describe using the spring data jpa starter as an alternative way of using jpa. There you can create a Jpa instance by just creating an interface extending the interface `JpaRepository`. Spring boot will automatically check your classpath for any jpa-provider and use that to create the actual implementation of the interface.

Relations are the most important and most difficult part of JPA.

- Element-collections: `@ElementCollection private List<String> attachmentUrls;`. These are for relations with non-entities. This instruction yields a table with attachment-urls and an id.
- One-to-many relations: In the class 'Task': `@OneToMany(mappedBy = "parentTask", cascade = CascadeType.ALL) private List<TimeSpan> activeP` and in the class 'TimeSpan': `@ManyToOne(fetch = FetchType.LAZY) @JoinColumn(name = "parentTask", referencedColumnName = "id") private Task` This code yields a foreign-key column named 'parentTask' in the 'TimeSpan' table. (It's a good practice to have the foreign key in the 'Many' table, not the 'One' table).
- Many-to-many relations: creates an intermediate id-mapping table.

4.4.9 Swing

Swing is a event based GUI-Framework. Data is wrapped in components. Components mainly do two things: tell the swing environment how they are to be rendered, and babble with other components or the user through listening to and sending of events.

Most components like buttons, labels etc. are trivial. But the interesting ones are JList, JTable and JTree. These all fire custom events. You can set a custom event-listener for these events. You can also pass a model to them to create a mapping between your actual data and the JList, and you can pass a renderer to them to instruct them how your data is supposed to be displayed.

4.4.9.1 Lists

Lists will be the first component we look at in more detail to illustrate how Swing handles subcomponents and subcomponent-models. As stated before, there are three ways in which we can customise a default list: add custom event handlers, add a model to map our data to the list, and add a renderer to customise the display of the elements.

Adding a custom event-handler for JListEvents This is as simple as passing in a implementation of a single method interface.

Defining a model With almost any Swing component, you can separate the underlying data structure containing the data from the GUI control that displays the data. It is rare to want to do that with simple controls like buttons, but with lists, tables, and trees, it is inconvenient to have to copy data out of an existing hash table, array, linked list, or other structure into the format needed by the GUI control. So Swing lets you use your existing structure directly. You simply implement an interface that tells Swing how to access the data, and then use the data directly in the component.

So you could add simple strings to a list, that JList can handle without any kind of a model:

```
    String[] options = {"Option1", "Option2", "Option3"};
    JList optionList = new JList<String>(options);
```

Instead of predetermining the data structure that holds list elements, Swing lets you use your own data structure as long as you tell it how to get the data into or out of that structure. You could either implement the ListModel-interface or extend the AbstractListModel-class.

```
        Location[] locations = {
                        new Location("Berlin", "Germany", "nice"),
                        new Location("Ghent", "Belgium", "awesome"),
                        new Location("Gera", "Germany", "sucking balls")
                };
                JList list = new JList<Location>(new LocationListModel(locations));

                public class LocationListModel extends AbstractListModel<Location>{
                private Location[] locations;
                public LocationListModel(Location[] locations) {
```

```
                this.locations = locations;
        }
        public Location getElementAt(int index) {
                return locations[index];
        }
        public int getSize() {
                return locations.length;
        }

}

public class Location {
        private String name;
        private String country;
        private String description;
        public Location(String name, String country, String description) {
                this.name = name;
                this.country = country;
                this.description = description;
        }
        public String toString() {
                return (name + " in " + country + " is " + description);
        }
}
```

Defining a renderer Swing has a few simple defaults for displaying values in lists, trees, and tables. In a JList, values that are Icons are drawn directly, while other objects are converted to strings via their toString method, then displayed via a JLabel. However, Swing also lets you define arbitrary mappings between values and display components, yielding whatever visual representation you want for values of specified types. This is done by building a "cell renderer" that takes the containing component, the value of the entry, a few state parameters that say things like whether or not the value is currently selected, and then returns a Component that is used to draw the value.

Normally Swing uses a JLabel, either directly from the String representation of the entry or directly from the value if the value is an Icon. But in this example, which continues the previous one, I want to include an image of the flag of the country for each JavaLocation. So I define a class that implements the ListCellRender interface, with a method getListCellRendererComponent that constructs the Component of interest given the list entry. I then associate that renderer with the JList via the JList's setCellRenderer method.

```
public class AppMain {
   public static void main(String[] args) {
        Location[] locations = {
                        new Location("Berlin", "Germany", "nice"),
                        new Location("Ghent", "Belgium", "awesome"),
                        new Location("Gera", "Germany", "sucking balls")
        };
        JList list = new JList<Location>(new LocationListModel(locations));
        list.setCellRenderer(new MyCellRenderer());
        ...
   }
   }

   public class MyCellRenderer extends DefaultListCellRenderer {
   public Component getListCellRendererComponent(JList list, Object value, int index, boolean isSelected, boolean cel
        JLabel label = (JLabel) super.getListCellRendererComponent(list, value, index, isSelected, cellHasFocus);
        Component out = label;
        if(isSelected) {
                JButton but = new JButton();
                but.setText(label.getText());
                out = but;
        }
        return out;
   }
}
```

4.4.9.2 Trees

Trees are very similar to lists. Here, too, we can define custom handlers, add models and add renderers.

Adding custom handlers for JTreeEvents

Adding a model A JTree uses a TreeModel to get its data. As with JList, you can replace the model altogether, specifying how to extract data from the custom model. In the case of JTree, however, the default TreeModel is very

useful as it is. It is then most often better to write a custom TreeNode (by extending DefaultMutableTreeNode) and notify your datastructure of changes through the user by creating a TreeModelListener.

```
      class AppMain {
        public static void main(String[] args) {
                  Place e = new Place(null, "Europe");
                  Place g = new Place(e, "Germany");
                  Place m = new Place(g, "Munich");
                  Place i = new Place(e, "Italy");
                  Place r = new Place(i, "Rome");
                  Place b = new Place(g, "Berlin");

                  MyTreeCell cont = new MyTreeCell(e);
                  cont.add(new MyTreeCell(g));
                  cont.add(new MyTreeCell(i));

                  JTree tree = new JTree(cont);
        }
      }

      public class MyTreeCell extends DefaultMutableTreeNode {
        public MyTreeCell(Place p) {
                  super(p.getName());
        }
      }
```

Adding a renderer Renderers can be changed just like you can in JLists.

```
    public abstract class MyCellRenderer extends DefaultTreeCellRenderer {
            public Component getTreeCellRendererComponent(JTree tree, Object value, boolean sel, boolean expanded, boolean
                  return super.getTreeCellRendererComponent(tree, value, sel, expanded, leaf, row, hasFocus);
            }
}
```

4.4.9.3 Writing custom components

Extending JComponent Everything in Swing is a component. A component does the following things:

- contain data
- listen to user events
- fire own events in response
- provide a way for API-users to define event-listeners

Concretely, a implememtation might look like this:

- Extend JComponent
- In the constructor, pass your actual data.
- Also in the constructor, add a few of these:

```
            addMouseListener(new MouseAdapter( ) {
              public void mousePressed(MouseEvent e) { myMouseEventHandler(e); }
            });
```

- implement your myXEventHandler(e) methods.
- write a method fireEvent(), that goes through all the Listeners in the listenerList and allows them to react to a MyCompEvent.
- write a method addListener(Listener l), that accepts a listenerfunction (single method interface).
- Overwrite the method paintComponent

4.4.10 Servlets and JSP

Contrary to CGI programs, a servlet only is started once and then sits there running waiting for requests to arrive via the servlet-container (such as tomcat, jetty or undertow). For every client request, the servlet spawns a new thread (not a whole proces like in cgi).

For a servlet to run in a servlet-container, it must simply extend the class HttpServlet. Also, the servlet needs a web.xml file. That file tells the container which requests should go to this servlet, where the base dir is, which class extended HttpServlet.

Place the following in an eclipse "dynamic-web-project"'s src-folder:

```
package meinServlet;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class MainServlet extends HttpServlet {
        private static final long serialVersionUID = 1L;

    public MainServlet() {
        super();
    }

        protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOExceptio
                response.getWriter().append("Served at: ").append(request.getContextPath());
        }

        protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOExcepti
                doGet(request, response);
        }
}
```

And put this DD in the projects WebContent/WEB-INF folder:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns="http://xmlns.jcp.org/xml/ns/javaee"
        xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
        id="WebApp_ID" version="3.1">

        <servlet>
                <servlet-name>Main</servlet-name>
                <servlet-class>meinServlet.MainServlet</servlet-class>
        </servlet>

        <servlet-mapping>
                <servlet-name>Main</servlet-name>
                <url-pattern>/hallo</url-pattern>
        </servlet-mapping>

</web-app>
```

With these two files, tomcat will know how to handle the servlet. Now if you go to `http://localhost:8080/meinServlet/hal` should see the site.

4.4.10.1 Building with maven

- `mvn archetype:generate` with maven-archetype-webapp
- add folder `src/main/java`
- add package `org.langbein.michael.qa`
- add dependencies to pom, especially `httpservlet`
- add servlet in `org.langbein.michael.qa`
- add `web.xml` in `src/main/webapp/WEB-INF`
- `mvn compile war:war`
- copy new war into `<tomcat>/webapps`

- restart tomcat
- visit `localhost:8080/qa/`

This is a ridiculously complex setup, especially since you'll have to repeat the last four steps for every iteration. That's why during development you should use the jetty plugin:

- Add the jetty-plugin to your pom:

```
        <build>
                <finalName>qa3</finalName>
                <plugins>
                        <plugin>
                                <groupId>org.eclipse.jetty</groupId>
                                <artifactId>jetty-maven-plugin</artifactId>
                                <version>9.4.7.v20170914</version>
                                <configuration>
                                        <scanIntervalSeconds>10</scanIntervalSeconds>
                                        <connectors>
                                                <connector implementation="org.mortbay.jetty.nio.SelectChannelConnector
                                                        <port>8080</port>
                                                        <maxIdleTime>60000</maxIdleTime>
                                                </connector>
                                        </connectors>
                                </configuration>
                        </plugin>
                </plugins>
        </build>
```

- run jetty with `mvn jetty:run`

4.4.10.2 Jsp and forms

It's really easy to template html and forms. Consider this servlet:

```
package ct;

public class QAController extends HttpServlet{
        private static final long serialVersionUID = 1L;

        protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOExceptio
            request.getRequestDispatcher("/WEB-INF/qa.jsp").forward(request, response);
        }

        protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOExcepti
                String answer = request.getParameter("answer");
                System.out.println("Got the answer " + answer);
                request.setAttribute("lastAnswer", answer);
                doGet(request, response);
        }

}
```

When a GET-requests arrives at the server, it forwards this request to a jsp-page located under `/WEB-INF/qa.jsp`. That page looks like this:

```
<%@ page
        language="java"
        contentType="text/html; charset=UTF-8"
        pageEncoding="UTF-8" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>

        <head>
                <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
                <title>Welcome to Michael's Q & A site!</title>
        </head>

        <body>
                <form name="qaForm" method="post" action="qa">
                        Your last answer was: ${lastAnswer} <br />
                        Question: ... blah ... <br />
                        Answer:  <input type="text" name="answer" />  <br />
                        <input type="submit" value="Submit your answer" />
                </form>
                This page was generated at <%= new java.util.Date() %>
        </body>
</html>
```

Note three important aspects about this form:

- It has a POST-method and an input-element of the "submit" type
- The post is mapped to a relative url named "qa". This is the url of the servlet (could also be another servlet)
- The file uses a variable named `lastAnswer`. This variable has been set during the last request by the doPost-method.

4.4.10.3 Servlet context listener

Especially when our application depends on a database to exist on the server, ... Integrating a database requires us to do three things:

- add the connector jar to the apps libraries
- add the db credentials to the web.xml using `<context-param>`
- add a context listener, that has the servlet read out the database credentials on initialisation.

4.4.10.4 Servlet lifecycle

- Container creates *one* instance of the servlet.
- Container calls the `init()` method (*once*).
- From now on, every request is passed to that one instance's `doGet()` and `doPost()` methods.

4.4.11 Webservices

There are basically two kinds of webservices. Both build up on top of http.

- REST: a simple service using nothing but http's get, post etc. The details are defined in the JAX-RS specification. A concrete implementation of JAX-RS is called Jersey[2]. A rest-service is described to the consumer using WADL.
- SOAP: a more complex service. SOAP is a xml-protocol that defines what kind of xml-documents the client may send and what kind of xml-documents the server sends in response. The details are defined in the JAX-WS specification. A concrete implementation of JAX-WS is Apache CXF. A soap-service is described to the consumer using WSDL.

4.4.11.1 Rest-container: Jersey

Jersey is a server and container to be used as a REST-frontend to your application. You just feed it annoted beans like the following, and jersey takes care of receiving the http-request and mapping them to the right point in your bean.

MyController.java

```
package myrest;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;

@Path("/hello")
public class MyController {

        @GET
        @Produces("text/plain")
        public String getClichedMessage() {
                return "Hello World";
        }
}
```

[2]Its good to know that there is a popular alternative to jersey known as restlet.

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns="http://xmlns.jcp.org/xml/ns/javaee"
        xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
        id="WebApp_ID" version="3.1">

        <servlet>
            <servlet-name>MyRestApp</servlet-name>
            <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
            <init-param>
                <param-name>jersey.config.server.provider.packages</param-name>
                <param-value>myrest.MyController</param-value>
            </init-param>
        <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>MyRestApp</servlet-name>
    <url-pattern>/rest/*</url-pattern>
  </servlet-mapping>

</web-app>
```

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>org.langbein.michael</groupId>
        <artifactId>myrest</artifactId>
        <version>1</version>

        <packaging>war</packaging>

        <dependencies>
                <dependency>
                        <groupId>org.glassfish.jersey.core</groupId>
                        <artifactId>jersey-server</artifactId>
                        <version>2.17</version>
                </dependency>
                <dependency>
                        <groupId>org.glassfish.jersey.containers</groupId>
                        <artifactId>jersey-container-servlet-core</artifactId>
                        <version>2.17</version>
                </dependency>
        </dependencies>

        <build>
                <sourceDirectory>src</sourceDirectory>
                <plugins>
                        <plugin>
                                <artifactId>maven-compiler-plugin</artifactId>
                                <version>3.6.1</version>
                                <configuration>
                                        <source>1.8</source>
                                        <target>1.8</target>
                                </configuration>
                        </plugin>
                        <plugin>
                                <artifactId>maven-war-plugin</artifactId>
                                <version>3.0.0</version>
                                <configuration>
                                        <warSourceDirectory>WebContent</warSourceDirectory>
                                </configuration>
                        </plugin>
                </plugins>
        </build>

</project>
```

4.4.11.2 Soap-container: Apache CXF

JXF is a server and container to be used as either a REST- or a SOAP-frontend to your application[3]. You configure JXF as one of the two by ....

[3]Actually, cxf can speak even more protocols than just REST and SOAP, like CORBA. Also, it can not only listen to HTTP, but also JMS or JBI

Then you feed it your beans like this:
....

4.4.11.3 JSP and JSTL

JSTL is meant as an extension of html. It allows you to create script-like logic in html, but without inserting any java (or php) code in html. Here an example:

```
 <%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<html>
 <head>
  <title>Count to 10 Example (using JSTL)</title>
 </head>

 <body>
  <c:forEach var="i" begin="1" end="10" step="1">
   <c:out value="${i}" />

   <br />
  </c:forEach>
 </body>
</html>
```

4.4.12 Unit testing

JUnit is the primary tool for testing in Java. It is a container like tomcat: it takes classes and runs them according to instructions that in finds in those testclasses in the form of annotations. Every such testclass extends `TestCase`, this primarily gives you access to different `assert` methods. It is important to know in what order junit executes tests that it finds in a testclass:

- Junit counts the number of `@Test` methods you have in your class and creates that many different instances of the class. So if you have five tests, five separate instances are created, by calling the constructor five times.
- Then, for each instance,
  - JUnit calls the method `setUp()` and any method annoted with `@Before`
  - Junit calls the one `@Test` method that the instance has been created for
  - Junit calls the method `tearDown()` and any method annoted with `@After`

4.4.13 Logging

That's right, in an enterprise-application, not even logging is done without a framework. This section will introduce logback.

To inlcude logback, you should add the following dependency:

```
 <dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <version>1.2.3</version>
</dependency>
<dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-jdk14</artifactId>
        <version>1.7.25</version>
</dependency>
```

This single dependency is enough, as it will transitively pull in the logback-core and slf4j-api dependencies. If you use spring boot, there is no need for that dependency; logback is already contained within the starter.

The usage of logback is rather simple.

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class SomeApp {
    private final Logger logger = LoggerFactory.getLogger("MeinDefaultLogger");

    public void someMethod(){
        logger.debug("This is a debug message");
        logger.info("This is an info message. It can have {} of {}", "all kinds of", "parameters");
```

```
        logger.warn("This is a warn message");
        logger.error("This is an error message");
    }
}
```

configuration It is on you to decide how the different log-statements should be handled. To do so, place a `logback.xml` in your `src/main/resources` folder. This configuration file contains the following elements:

- An `appender` is responsible for writing out the logstatments to a destination. Properties: `name` , `class` , `level` . There are several types (indicated by the `class` attribute):
  - RollingFileAppender
  - SMTPAppender (email)
  - ConsoleAppender
  - SiftingAppender (separates logs based on a runtime attribute)

  An `appender` can also have a `filter`,that decides what kind of messages it would display.
  - `filter` .
- `root` is the logger's software interface. Properties: `level` (the log-level threshold that should actually be used).
  - `appender-ref` 's list all the elements appenders that should actually be used.
- `logger` is just like the `root` logger, but with a different name. You can get the logger "meinBesondererLogger" with `Logger mbl = LoggerFactory.getLogger("meinBesondererLogger"); mbl.warn("Pass besser auf!");`. Loggers can have ancestors; to create a child to an ancestor, just give it a name that includes the ancestors name, like "meinBesondererLogger.sonderFall".

An example would be:

```
<configuration debug="true" scan="true">

    <property name="meineLogFile" value="log/meinlog.txt" />
    <property name="meinLogFileArchiv" value="log/meinlog" />

    <appender name="consolenOutput" class="ch.qos.logback.core.ConsoleAppender">
        <encoder  class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
            <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{50} - %msg%n</pattern>
            <target>System.err</target>
        </encoder>
    </appender>

    <appender name="fileOutput" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <encoder  class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
            <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{50} - %msg%n</pattern>
        </encoder>
        <file>${meineLogFile}</file>
        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
                        <fileNamePattern>${meinLogFileArchiv}.%d{yyyy-MM-dd}.%i.log</fileNamePattern>
                        <timeBasedFileNamingAndTriggeringPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP"
                                <maxFileSize>5MB</maxFileSize>
                        </timeBasedFileNamingAndTriggeringPolicy>
                        <maxHistory>7</maxHistory>
                </rollingPolicy>
    </appender>

    <appender name="roleSiftingAppender" class="ch.qos.logback.classic.sift.SiftingAppender">
        <discriminator>
            <key>userRole</key>
            <defaultValue>ANONYMOUS</defaultValue>
        </discriminator>
        <sift>
            <appender name="fileAppender" class="ch.qos.logback.core.FileAppender">
                <file>${userRole}.log</file>
                <encoder>
                    <pattern>%d [%thread] %level %mdc %logger{50} - %msg%n</pattern>
                </encoder>
            </appender>
        </sift>
    </appender>
```

```
    <root level="debug">
        <appender-ref ref="consolenOutput" />
    </root>

    <logger level="info" name="meinBesondererLogger">
        <appender-ref ref="rollingFileAppender" />
    </logger>

</configuration>
```

For the discriminator to have access to the userRole key, you need to place it in the MDC (Mapped Diagnostic Context). Simply put, MDC allows you to set information to be later retrieved by other Logback components, using a simple, static API:

```
MDC.put("userRole", "ADMIN");
logger.info("Admin Action");
```

4.5 Python

Python is a wonderful calculator. It is not intended for large projects, but for one-off scripts, prototypes, data-analyses and the like.

4.5.1 List comprehensions

List comprehensions allow us to write set-notation like declarative definitions of lists.

```
S = [x**2 for x in range(10)]
M = [x for x in S if x%2 == 0]
```

We can also do dict-comprehensions:

```
dict2 = {k*2:v for (k,v) in dict1.items()}
```

List comprehensions can be conditional:

```
listA = ["a", "b", "c", "d"]
listB = ["c", "d", "e", "f"]
[entry for entry in listA if entry in listB]  # <-- [c, d]
```

4.5.2 Functional programming

Anonymous functions are declared by using the `lambda` keyword.

```
ls = [0, 1, 2, 3, 4]
squares = map(lambda x: x * x, ls)
evens = filter(lambda x: x%2 == 0, ls)
```

There is also support for currying:

```
from functools import partial
```

4.5.3 Decorators

Python decorators are annotations that extend the behavior of the function over which they are written. That lends them to do, for example, aspect oriented programming.

In vanilla python, we could do this:

```
def myDecorator(someFunc):
    def wrappedFunc():
        print("This is printed before the function is called")
        someFunc()
        print("This is printed after the function is called")
    return wrappedFunc

def sayWhee():
    print("Wheee!")

wrappedWhee = myDecorator(sayWhee)
wrappedWhee()

# >> "This is printed before the function is called"
# >> "Whee!"
# >> "This is printed after the function is called"
```

But there is syntactic sugar for this.

```
@myDecorator
def sayWhee():
    print("Wheee!")

sayWhee()
```

This also works with functions that take arguments:

```
def addOne(func):
    def wrapped(*args):
        out = func(*args)
        return out + 1
    return wrapped


@addOne
def timesTwo(n):
    return 2*n

timesTwo(2)  # <-- 5
```

As you can see, a decorator takes a function as an argument. We can also create decorators that depend on an argument themselves:

```
def addN(n):
    def decorator(func):
        def wrapped(*args):
            out = func(*args)
            return out + n
        return wrapped
    return decorator

@addN(2)
def timesTwo(n):
    return 2*n

timesTwo(2)  # <-- 6
```

A common use for decorators is memoization:

```
def memoize(function):
    memo = {}
    def wrappedFunction(x):
        if x not in memo:
            memo[x] = function(x)
        return memo[x]
    return wrappedFunction

@memoize
def fib(n):
    if n <= 2:
        return 1
    else:
        return fib(n - 1) + fib(n - 2)
```

4.5.4 Metaprogramming

On one hand we can change the way an object handles the standard operators (linke +, -, == etc); on the other we can define our own new operators (like innerprod, crossprod etc).And finally we can extend existing datastructures to change their behaviour to suite our needs.

4.5.4.1 Changing the way an object handles operations

Python has a lot of built-in methods that are added to every class by default[4]. One of them is `__add__`. Consider this example:

```
class MyVec:

        def __init__(self, x, y):
                self.x = x
                self.y = y

        def __add__(self, other):
                x = self.x + other.x
                y = self.y + other.y
                return MyVec(x, y)

        def __str__(self):
                return "[{},{}]".format(self.x, self.y)

v1 = MyVec(1, 2)
v2 = MyVec(2, 3)
print v1 + v2  # >> [3, 5]
```

4.5.4.2 Creating new operators

Defining custom operators is not internally supported by python, but you can use this:

```
# From http://code.activestate.com/recipes/384122/

class Infix:
    def __init__(self, function):
        self.function = function
    def __ror__(self, other):
        return Infix(lambda x, self=self, other=other: self.function(other, x))
```

[4]This is because internally, every class inherits from a default-class named 'type'

```
    def __or__(self, other):
        return self.function(other)
    def __rlshift__(self, other):
        return Infix(lambda x, self=self, other=other: self.function(other, x))
    def __rshift__(self, other):
        return self.function(other)
    def __call__(self, value1, value2):
        return self.function(value1, value2)


x=Infix(lambda x,y: x*y)
print 2 |x| 4
```

Update: this has now made it into its own package: https://pypi.python.org/pypi/infix/

```
from infix import or_infix as infix

@infix
def add(a, b):
    return a + b

a |add| b
```

4.5.4.3 Extending existing datastructures

Say you want lists to be indexed at 1.

```
class MyList(list):
    def __getitem__(self, key):
        return list.__getitem__(self, key-1)
```

However, it is often problematic to change basic data-structures because other modules might depend on their predefined behaviour. For that reason python exposes a few abstract base types. Extending from these base types is like forking from abstract list instead of commiting to list trunk. A classical case of an existing datastructure that we might want to extend from is dictionaries for use as timeseries:

```
import datetime as dt
from UserDict import *;

class TimeSeries(IterableUserDict):
    """ self.data verhaelt sich wie ein ganz normales dict """

    def __init__(self, formDtTime='%d.%m.%Y %H:%M'):
        UserDict.__init__(self)
        self.formDtTime = formDtTime

    def __setitem__(self, time, value):
        if isinstance(time, basestring):
            time = self.stringToDtTime(time)
        if not isinstance(time, dt.datetime):
            raise ValueError('Schluessel muss ein Zeitstring oder ein Datetime sein')
        UserDict.__setitem__(self,time,value)

    def stringToDt(self, timeString, form = None):
        dtTime = self.stringToDtTime(timeString, form)
        return dtTime.date()

    def stringToDtTime(self, timeString, form = None):
        if not form:
            form = self.formDtTime
        return dt.datetime.strptime(timeString, form)

    def getFirstTime(self):
        return min(self.data)

    def getLastTime(self):
        return max(self.data)

    def getFirstVal(self):
        return self.data[self.getFirstTime()]

    def getLastVal(self):
        return self.data[self.getLastTime()]

    def getFirstPoint(self):
        time = self.getFirstTime()
        return [time, self.data[time]]

    def getLastPoint(self):
        time = self.getFirstTime()
        return [time, self.data[time]]
```

```
    def getSpan(self):
        return self.getLastTime() - self.getFirstTime()

    def getSubseries(self, von = None, bis = None):
        if not von:
            von = self.getMinTime()
        if not bis:
            bis = self.getMaxTime()
        ts = TimeSeries()
        for time in self.data:
            if time < von or time > bis:
                continue
            ts[time] = self.data[time]
        return ts

    def getFilteredSeries(self, funct):
        ts = TimeSeries()
        for time in self.data:
            if funct(time, self.data[time]):
                ts[time] = self.data[time]
        return ts


if __name__ == '__main__':

    # TimeSeries ist ein dict mit zusaetzlichen Methoden
    ts = TimeSeries('%d.%m.%Y')
    ts['14.10.1986'] = 0
    ts['15.10.1986'] = 1
    ts['16.10.1986'] = 2
    ts['17.10.1986'] = 3
    print ts.getSpan()
    lw = ts.getLastPoint()
    print "Letzter Wert ist {0} am {1}".format(lw[0], lw[1])
    ss = ts.getFilteredSeries(lambda t, v: v >= 1 )
    print ss

    # Iterieren funktioniert wie gewohnt
    for el in ts:
        print el

    # Beim Einfuegen von Daten wird auf das richtige Format geachtet
    try:
        ts[4] = 2
    except Exception as e:
        print repr(e)

    # Map and reduce funktioniert wie gewohnt
    print reduce(lambda summe, key: summe + ts[key], ts, 0)
```

4.5.4.4 Named tuples

Named tuples take the best of simple classes (accessing fields by names) and tuples (accessing fields by index, immutable, iterable). They are wonderful in combination with infix custom operators for use as geometric objects.

```
>>> Point = namedtuple('Point', ['x', 'y'])
>>> p = Point(11, y=22)     # instantiate with positional or keyword arguments
>>> p[0] + p[1]             # indexable like the plain tuple (11, 22)
33
>>> x, y = p                # unpack like a regular tuple
>>> x, y
(11, 22)
>>> p.x + p.y               # fields also accessible by name
33
>>> p                       # readable __repr__ with a name=value style
Point(x=11, y=22)
```

4.5.5 Plotting

Matplotlib is a matlab inspired library for plotting.

```
import matplotlib.pyplot as plt

plt.plot(ns, NsShallow)
plt.plot(ns, NsDeep)
plt.xlabel("input size")
plt.ylabel("number of nodes")
plt.legend(["node in shallow net to acchieve accuracy = {}".format(epsilon), "node in deep net to acchieve accuracy = {}"
plt.show()
```

In general, you want to avoid `plt`. `plt` is just the header of the current plotting state. Use axes insead - axes have all the same methods as `plt`. But contrary to `plt`, they can be passed around freely.

```
fig, axesArr = plt.sublplots(2, 2)
doFirstPlot(axesArr[0])
doSecondPlot(axesArr[1])
...
plt.show()
```

There are two important plotting functions: `plot` and `scatter`. Plot expects you to pass in all values as an array, scatter allows you to add each point one by one.

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure()
ax = fig.gca(projection='3d')
for point in line:
    ax.scatter(point.x, point.y, point.z)
plt.show()
```

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot(xarray, yarray)
plt.show()
```

Also, often we want to plot heatmaps:

```
data = np.random.random((100, 100))
plt.imshow(data, cmap='hot', interpolation='nearest')
plt.show()
```

We can add multiple images to one single figure.

```
fig = plt.figure()

ax1 = fig.add_subplot(231, projection='3d')
ax1.set_title("Original body")
for point in signalCart:
    ax1.scatter(point[0], point[1], point[2])

ax2 = fig.add_subplot(232)
ax2.set_title("Flattened")
ax2.imshow(signal)

ax3 = fig.add_subplot(233)
ax3.set_title("Fourier Transform")
ax3.imshow(log(abs(amps)))

ax4 = fig.add_subplot(234, projection='3d')
ax4.set_title("New body")
for point in signalNewCart:
    ax4.scatter(point[0], point[1], point[2])

ax5 = fig.add_subplot(235)
ax5.set_title("Flattened")
ax5.imshow(abs(signalNew))

ax6 = fig.add_subplot(236)
ax6.set_title("Altered amplitudes")
ax6.imshow(log(abs(ampsNew)))

plt.show()
```

4.5.6 SymPy

Sympy is fairly good at doing symbolic calculations. You can put restrictions to the type of a variable in the Symbol class.

```
from sympy import Symbol, sin, cos

x = Symbol('x')
y = Symbol('y')

diff(sin(x), x) # --> cos(x)
cos(x).series(x, 0, 10)  # --> 1 - x^2/2 + x^4/24 - ...
sin(x+y).expand(trig=True) # --> sin(x)cos(y) + sin(y)cos(x)
```

```
from sympy import Matrix

M = Matrix([[1,0],[0,1]])
```

```
from sympy.solvers import solve

solve(x**2 -1, x) # --> [-1, 1]
solve([x+y-5, x-y+1], [x,y]) # --> {x:2,y:3}
```

```
integrate( t * exp(t), t)
>>>
integrate( t * exp(t), (t, 0, oo))
```

```
>>> from sympy import Symbol, exp, I
>>> x = Symbol("x")
>>> exp(I*x).expand()
 I*x
e
>>> exp(I*x).expand(complex=True)
   -im(x)                -im(x)
I*e      *sin(re(x)) + e      *cos(re(x))
>>> x = Symbol("x", real=True)
>>> exp(I*x).expand(complex=True)
I*sin(x) + cos(x)
```

For many comon integrals sympy already has a predefined function:

```
>>> from sympy import fourier_transform, exp
>>> from sympy.abc import t, k
>>> fourier_transform(exp(-t**2), t, k)
sqrt(pi)*exp(-pi**2*k**2)
>>> fourier_transform(exp(-t**2), t, k, noconds=False)
(sqrt(pi)*exp(-pi**2*k**2), True)
```

The same holds for statistics: many common distributions are already given:

```
>>> from sympy.stats import Poisson, density, E, variance, cdf
>>> from sympy import Symbol, simplify
>>> rate = Symbol("lambda", positive=True)
>>> z = Symbol("z")
>>> X = Poisson("x", rate)
>>> density(X)(z)
lambda**z*exp(-lambda)/factorial(z)
>>> E(X)
lambda
>>> simplify(variance(X))
lambda
>>> cdf(X)(2)
lambda**2*exp(-lambda)/2 + lambda*exp(-lambda) + exp(-lambda
```

You can even use standard python function definitions in sympy:

```
def myf(x):
    return x**2 + 1

x = Symbol("x")

diff(myf(x), x)

>> 2*x
```

4.5.7 Numpy and Scipy

Scypy has, amongst others, a bunch of useful statistical methods:

```
from scipy.stats import poisson

def pois(lmbd, n):
    return poisson.pmf(n, lmbd)

def Pois(lmbd, n):
    return poisson.cdf(n, lmbd)

def poisCond(lmbd, n, n0):
    if n < n0:
        return 0
    else:
        return (pois(lmbd, n)) / (1 - Pois(lmbd, n0-1))

def expct(lmbd):
    ex = 0
```

```
    for i in range(samplesize):
        ex += i * pois(lmbd, i)
    return ex

def expctCond(lmbd, n0):
    ex = 0
    for i in range(samplesize):
        ex += i * poisCond(lmbd, i, n0)
    return ex
```

4.5.8 Gradual typing

Since python 3.6, gradual typing is possible with the mypy-checker. Here is some examplecode:

```
class Person:
    def __init__(self, name: str) -> None:
        self.name = name


def sayHi(person: Person) -> str:
    return "Hello, {}".format(person.name)


m = Person("Michael")
print(sayHi(m))
```

However, the python-interpreter itself does, so far, not check types. Python does allow type-hints, but does not enforce them by itself. To check types ahead of type, use `mypy prog.py`.

4.5.9 Piping and currying

```
from infix import or_infix

@or_infix
def pipe(val, func):
    return func(val)

def add2(x):
    return x + 2

print 3 |pipe| add2 |pipe| (lambda x:x**2)
```

```
import inspect

def curry(func, *args, **kwargs):

    '''
    This decorator make your functions and methods curried.
    Usage:
    >>> adder = curry(lambda (x, y): (x + y))
    >>> adder(2, 3)
    5
    >>> adder(2)(3)
    5
    >>> adder(y = 3)(2)
    5
    '''

    assert inspect.getargspec(func)[1] == None, 'Currying can\'t work with *args syntax'
    assert inspect.getargspec(func)[2] == None, 'Currying can\'t work with *kwargs syntax'
    assert inspect.getargspec(func)[3] == None, 'Currying can\'t work with default arguments'

    if (len(args) + len(kwargs)) >= func.__code__.co_argcount:

        return func(*args, **kwargs)

return (lambda *x, **y: curry(func, *(args + x), **dict(kwargs, **y)))
```

4.5.10 Generators

Generators create lists that are not kept in memory, but only evaluated on demand. This is useful for performance if we need to work with very large lists. We can potentially define even infinite lists.

```
# As function. Note the 'yield' instead of a 'return'
def evens(n):
    for i in range(n):
        if i%2 == 0:
            yield i
```

```
# As a list comprehension. Note the () istead of []
evens = (i for i in range(n) if i%2 == 0)
```

Both these methods don't return `[1, 2, 4, ..., n**2]`, but `<generator_n>`, which will be evaluated when neccessary.

4.5.11 Oslash: advanced functional programming

Very often, we don't know if an actual value or null is passed into or returned from a function. Functional programming languages manage this problem by wrapping those values in a maybe-monad. More generally, very often operations are nondeterministic: a function migth return nothing or multiple results at once. Then, it makes sense to wrap these possible outcomes in a monad.

One important thing to note is this: monads are not the essence of functional programming, but a means of dealing with uncertain operation-results. It will make sense to unwrap monads as soon as you can.

Here we will show functors, applicative functors (which are just beefed up functors) and finally monads (which are just beefed up applicatives).

- A functor is a data type that implements the Functor abstract base class. You apply a function to a wrapped value using map or %
- An applicative is a data type that implements the Applicative abstract base class. You apply a wrapped function to a wrapped value using * or lift
- A monad is a data type that implements the Monad abstract base class. You apply a function that returns a wrapped value, to a wrapped value using — or bind

A Maybe implements all three, so it is a functor, an applicative, and a monad.
There are several common monads, like for example:

- maybe : if a value might be null
- list : if a value might be 0, 1 or more values
- io
- reader
- writer
- state
- identity : there is no reason to use this - only useful for theory.

Wrapped values

Functors Unfortunately, you cannot just stick a wrapped integer to a function that accepts an integer:

```
def add3(x):
    return x + 3

val = Just(2)

add3(val)
>> Error!
```

This is what functors are for.

```
def add3(x):
    return x + 3

val = Just(2)

val.map(add3)
>> Just 5
```

This works because the `Just` wrapper implements the abstract `Functor` baseclass, which forces it to define a `map(self, func)` method.

Applicatives It was a little weird that we had to write the value before the function in `val.map(add3)`. But we can also wrap functions in a wrapper. If the wrapper implements the `Applicative` class, we can instead write:

```
add3Wrapped = Just(lambda: x: x+3)
val = Just(2)

add3Wrapped.lift(val)
>> Just 5
```

4.5.12 Multimedia: Pygame

In general when we want to do a bigger multimedia-app, pygame is a save choice. It's python's equivalent of the Processing framework for Java.

```
import sys, pygame

pygame.init()


size = width, height = 320, 240
speed = [2, 2]
black = 0, 0, 0

screen = pygame.display.set_mode(size)

ball = pygame.image.load("ball.gif")
ballrect = ball.get_rect()


while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            sys.exit()

    ballrect = ballrect.move(speed)
    if ballrect.left < 0 or ballrect.right > width:
        speed[0] = -speed[0]
    if ballrect.top < 0 or ballrect.bottom > height:
        speed[1] = -speed[1]

    screen.fill(black)
    screen.blit(ball, ballrect)
    pygame.display.flip()
```

screen aka. surface: `pygame.display.set_mode()`.

`display.flip()` will update the contents of the entire display. `display.update()` allows to update a portion of the screen, instead of the entire area of the screen. Passing no arguments, it updates the entire display. To tell PyGame which portions of the screen it should update (i.e. draw on your monitor) you can pass a single pygame.Rect object, or a sequence of them to the display.update() function. A Rect in PyGame stores a width and a height as well as a x- and y-coordinate for the position.

PyGame's built-in dawning functions and the `.blit()` method for instance return a Rect, so you can simply pass it to the `display.update()` function in order to update only the "new" drawn area.

Due to the fact that `display.update()` only updates certain portions of the whole screen in comparison to `display.flip()`, `display.update` is faster in most cases.

4.5.13 Regex

4.5.14 Modules and Packages

In python, it is very important from which directory a script is called, because that determines how `import` statements are resolved. That means that when you write `import` statements, you are forcing the user to call your code from a certain directory.

Every python-file is a module. Every folder containing python-files is a package. Packages can additionally have a `__init__.py` file with additional information.

When the interpreter encounters the code ...

```
    from ie.engine import InferenceEngine
```

... in a file `main.py`, it searches for a file `engine.py` in a folder `ie` inside

- the current directory or the location of the script `main.py`,

  - The location of the `main.py` file will be added if you ran the script with `python somedir/main.py`
  - The current directory will be added if you ran the script with `python -m somedir.main`

- the `PYTHONPATH`. You can append to this path with:

```
import sys
sys.path.append('path/to/ie')
```

If the directory `ie` contains an `__init__.py` file, this file is executed when a module in that dir is loaded for the first time. This way, the `__init__.py` can be used to initialize any global variables or such that the module needs.

4.5.15 Miniconda

Honestly, the python2/3 shism is going to kill you. And if that doesn't, then the multiple c-dependencies python has. There is really only one way around this, which is using miniconda as a venv manager. Miniconda first installs a root-environment for you under /miniconda. The packages in there will be included in any further venv you create. You create a new venv with `source activate meinTestEnv`. Note that this does not create a folder at your current location. It does, however, create one under /miniconda/envs/meinTestEnv . This is where all your env-specific packages will be stored. A venv may even use a different python-version than your root-env. You can deactivate the venv again with `source deactivate`.

- `conda env list`
- `conda create --name meinEnv`
- `source activate meinEnv`
- `source deactivate`

Conda gets its packages from so-called channels - just like ubuntu-repositories. You can change the priority at which conda searches through channels, and add new ones.

- `conda config --get channels`
- Search for packages here: `https://anaconda.org/anaconda/repo`
- `conda config --append channels newchannel`
- `conda list`
- `conda search tensorflow`
- `conda install tensorflow=1.0.1`
- If you use pip while in a conda venv, the pip-package will also be installed in the venv. However, it does *not* include things you install through `apt-get`.

You can export and import environment-specifications like this:

- `conda env export > environment.yml`
- `conda env create -f environment.yml`

4.5.16 IPythonNotebook aka. Jupyter

Jupyter is a client-server program. You start it on your server, where it runs in some conda-environment. The server exposes a web-interface, where the user can enter python-commands into the client that will then be executed on the server. This is advantageous for big-data environments, where you want to bring the code to the data, because getting the data to the user takes too much network traffic. Instructions can be found here.

4.5.17 Pandas

```
df.size
```

4.6 Racket

The hallmark of the Lisp family is that programs are defined in terms of data structures rather than in terms of a text-based syntax. The most visible consequence is a rather peculiar visual aspect, which is dominated by parentheses. The more profound implication, and in fact the motivation for this uncommon choice, is the equivalence of code and data. Program execution in Lisp is nothing but interpretation of a data structure. It is possible, and common practice, to construct data structures programmatically and then evaluate them. The most frequent use of this characteristic is writing macros (which can be seen as code preprocessors) to effectively extend the language with new features. In that sense, all members of the Lisp family are "programmable programming languages". There is a nice set of tutorials here.

4.6.1 Basics

- *list*: `'(1 2 3)`
- *syntax*: `#'(+ 1 2)`, or `#'car`

```
#lang racket

(provide make-complex get-real get-complex +c -c *c)


(define (make-complex r c)
  (cons r c))

(define (get-real a)
  (car a))

(define (get-complex a)
  (cdr a))

(define (piecewise op a b)
  (make-complex
    (op (get-real a) (get-real b))
    (op (get-complex a) (get-complex b))))

(define (add-complex a b)
  (piecewise + a b))

(define (substr-complex a b)
  (piecewise - a b))

(define (mult-complex a b)
  (let ((ar (get-real a))
        (ac (get-complex a))
        (br (get-real b))
        (bc (get-real b)))
    (make-complex
      (- (* ar br) (* ac bc))
      (+ (* ar bc) (* ac br)))))

(define (reduce func arglist carry)
  (let* ((first-el (car arglist))
         (rest-els (cdr arglist))
         (carry2 (func carry first-el)))
    (if (<= (length rest-els) 0)
        carry2
        (reduce func rest-els carry2))))

(define (+c firstarg . restargs)
  (reduce add-complex restargs firstarg))

(define (-c firstarg . restargs)
  (reduce substr-complex restargs firstarg))

(define (*c firstarg . restargs)
  (reduce mult-complex restargs firstarg))

  ;; (define a (make-complex 1 1))
  ;; (define b (make-complex 2 1))
  ;; (define c (make-complex 3 3))
  ;; (*c a b c)
  ;; (+c a a a b)
```

4.6.2 Macros

Is there a way we con do something like this?

```
(define (faculty n)
  (if (= n 1) n
      (* n (faculty (- n 1)))))


(hypothetical-macro-function (n !)
  (faculty n))

(4 !) ; ---> yields 24
```

Macros are functions that take one syntax object `#'syntx-obj` as input and return another syntax object. These syntax objects contain literal code, packaged with metadata like lexical context and source location. The syntax object passed to a macro contains the whole calling expression that invoked the macro. So if we invoke and like this:

```
(and (expr a) (expr b) (expr c))
```

and does not get three arguments as input, the way an ordinary function would. Rather, it gets a syntax object like this, which retains a reference to the lexical context of the calling site:

```
(and #'(and (expr a) (expr b) (expr c)))
```

4.6.3 Nomenclature

Lisps are often used for metaprogramming. For that reason, we need a firm grasp on the intricacies of the nomenclature of the language. We want to be able to give distincitve names to every part of a statement.

- statement: anything that is written
  - combination: aka. expression, aka. form
    1. eval all subexpressions
    2. apply leftmost value to rest
  - special forms are all statements that are not combinations.
    * Definitions: `(define name "Michael")` does not evaluate all subexpressions, because `name` does not exist yet and can therefore not be evaluated. What `define` does is to save a new entry in the workspace-memory under the label `name`

4.6.4 Datastructures

On the most basic level, racket knows only one datastructure: the `pair`:

```
(cons "a" "b") ;; => ("a" . "b")
```

A list is a pair whose tail is also a pair:

```
(cons "a" (cons "b" '())) ;; => '("a" "b")
```

4.7 Javascript

At its heart, javascript is a simple language. Its has however two very unconventional features that are not used in any other languages: *this* and *prototpye*s. It also has a hugely chaotic development, making simple things really hard over and over again. For almost every programming concept there are many conflicting ways to do it. Try importing a AMD package into a typescript project using webpack and react.

- Javascript does not have methods. It has properties that refer to functions.
- A function is an object. (Thats why every function has a `this`). Note that this means that there are executable objects.

this : every function implicitly gets a parameters named `this`, containing the context.

```
function add (x, y) {
    console.log(this);
    return x + y;
}
```

The context `this` can also be explicitly assigned.

```
const add3 = add.bind("michael", 2);
```

Note that this does not work for arrow-functions, which keep their once assigned context!

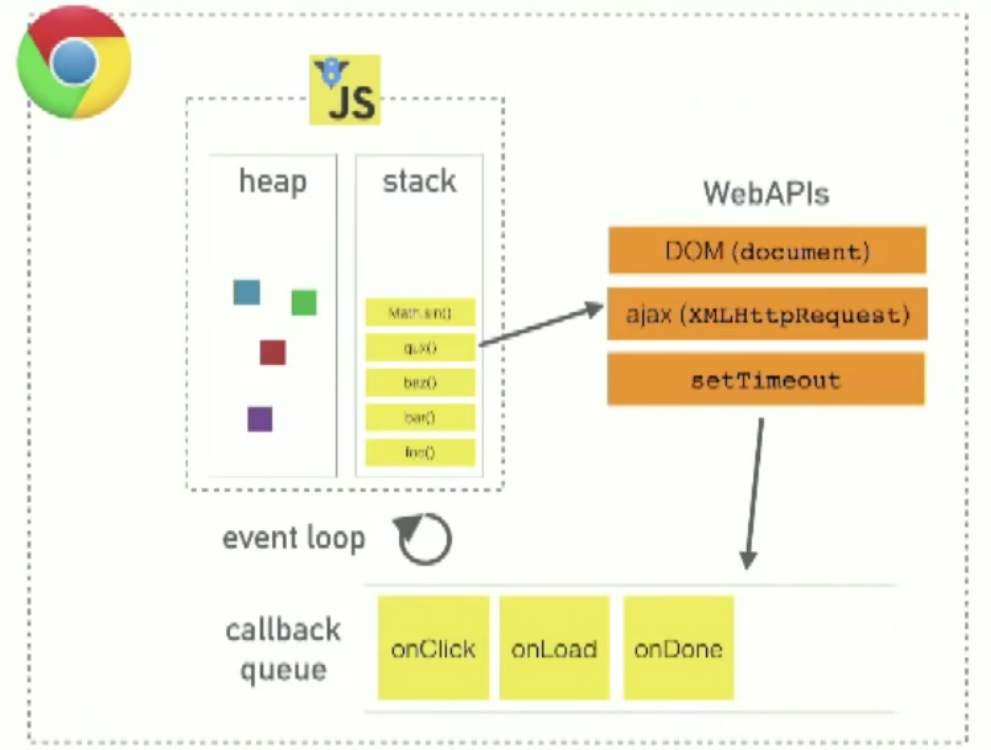
prototype : Prototype-languages should really only achieve inheritance by copying objects. Javascript implements this badly, though, because it offers *two* means of inheritance: object-copying and constructor-functions.

- Instances
  - Every object has a `.__proto__`.
  - `.__proto__` contains the list of inherited props and methods and *should* not be modified.
  - `bender.prototype` is bender's own prototype, and `bender.__proto__` is a reference to bender's parent's prototype.
- Constructing new objects
  - With `Object.create`: passing `theObject.__proto__` and `theObject`'s instance-properties.
    - Calling `const vender = Object.create(bender); vender.name = 'Vender';` will give `vender` these properties: `bender.__proto__` and `bender`'s instance-properties;
  - With constructor functions: passing `TheFunction.prototype`
    - Functions are objects, so they, too, have a `.__proto__`
    - Additionally, functions have a `.prototype`. This is because any function may serve as a constructor, so they need this `.prototype`.
    - Even though not all functions are intended as constructors, js expects them to be that. So every function can be called with `new`. Doing this expects the function to do two things:
      - it expects to have the function assign some things to its internal `this`. These assignments will be instance-properties.
      - it expects `TheFunction.prototype` to have a few props that are to be shared between all instances. In other words: adding a function via `TheFunction.prtotype.myMethod = () => ...` instead of `function TheFunction () this.myMet` makes the function static. There is only one instance of that function on the heap, not a separate one for each instance.
    - Contrary to `.__proto__`, `.prototype` may and should be modified.
    - So, calling `const bender = new Robot('Bender');` will give `bender` these properties: `Robot.prototype`;
  - Funfact: `Object` is a constructor-function, which is why there is a `Object.prototype`. Indeed, calling `ame: 'Michael'name: 'Michael'` is shorthand for `ame: 'Michael'new Object(n)`

```
const Robot = function(name) {
    this.name = name;
}
Robot.prototype.sayHi = function() {
    return 'Hi from ' + this.name;
};
const bender = new Robot('Bender');
bender.sayHi();
Robot.prototype.sayHi = function() {
    return 'Bugger off from ' + this.name;
}
bender.sayHi();
```

4.7.1 The js-runtime

Figure 4.1: *Stack*: Todo's in the current frame. Makes calls to the *Web-APIs*. *Queue*: From another thread the browser sends messages onto the queue. Some of these messages may have been triggered by the JS-stack calling for example a `setTimeout` Web-API call. They're handled FIFO.



4.7.2 Asynchronous programming

Rxjs is a popular, stream based implementation of the observer-pattern.

```
export interface Observer<T> {
    onNext: (val: T) => void,
    onCompleted: () => void,
    onError: (e: Error) => void
}


const nullObserver: Observer<any> = {
    onNext: (val: any) => {},
    onCompleted: () => {},
    onError: (e: Error) => {}
}


/**
 * Subscriptions are just handles to observers,
 * so that they can be unsubscribed if need be.
 */
```

```
export interface Subscription {
    unsubscribe: () => void;
}


export class SimpleSubscription<T> implements Subscription {
    constructor(
        private observer: Observer<T>
    ) {}

    unsubscribe () {
        // unsupscription == overwrite original observer
        this.observer = nullObserver;
    }
}


/**
 *
 */
export class Observable<T> {

    constructor(
        private _subscribe: (observer: Observer<T>) => Subscription
    ) {}


    subscribe(downstreamObserver: Observer<T>): Subscription {
        return this._subscribe(downstreamObserver);
    }


    /**
     * List of static creational methods.
     * They each return new observables.
     * Note that those new observables do not execute the _subscribe mehtod yet.
     * that method is only executed when subscribe is executed.
     */

    static of<X>(args: X[]): Observable<X> {
        return new Observable<X>((downstreamObserver: Observer<X>) => {
            console.log(`executing subscription-body (returned from 'of' method)`)

            args.forEach(val => downstreamObserver.onNext(val));
            downstreamObserver.onCompleted();

            return new SimpleSubscription(downstreamObserver);
        })
    }


    static fromEvent(source: Element, event: string): Observable<Event> {
        return new Observable<Event>((downstreamObserver: Observer<Event>) => {
            console.log(`executing subscription-body (returned from 'fromEvent' method)`)

            const callback = (e: Event) => downstreamObserver.onNext(e);
            source.addEventListener(event, callback);
            return {
                unsubscribe: () => source.removeEventListener(event, callback)
            }
        });
    }

    /**
     * list of non-static creational mehtods.
     * They each return new observables.
     * Note that the subscribe methods are not called yet with this method.
     */

    map<Y>(mapFunc: (v: T) => Y): Observable<Y> {
        return new Observable<Y>((downstreamObserver: Observer<Y>) => {
            console.log(`executing subscription-body (returned from 'map' method)`)

            const mappingObserver: Observer<T> = {
                onNext: (val: T) => {
                    const y: Y = mapFunc(val);
                    downstreamObserver.onNext(y);
                },
                onCompleted: () => downstreamObserver.onCompleted(),
                onError: (e: Error) => downstreamObserver.onError(e)
            };

            return this.subscribe(mappingObserver);
        });
```

```
    }


    filter(filterFunc: (v: T) => boolean): Observable<T> {
        return new Observable<T>((downstreamObserver: Observer<T>) => {
            console.log(`executing subscription-body (returned from 'filter' method)`)

            const filteringObserver: Observer<T> = {
                onNext: (val: T) => {
                    if (filterFunc(val)) {
                        downstreamObserver.onNext(val);
                    }
                },
                onCompleted: () => downstreamObserver.onCompleted(),
                onError: (e: Error) => downstreamObserver.onError(e)
            };

            return this.subscribe(filteringObserver);
        });
    }


    take(nr: number): Observable<T> {
        return new Observable<T>((downstreamObserver: Observer<T>) => {
            console.log(`executing subscription-body (returned from 'take' method)`)

            let i = 0;
            const takingObserver: Observer<T> = {
                onNext: (val: T) => {
                    if (i < nr) {
                        downstreamObserver.onNext(val);
                        i += 1;
                    } else {
                        downstreamObserver.onCompleted();
                    }
                },
                onCompleted: () => downstreamObserver.onCompleted(),
                onError: (e: Error) => downstreamObserver.onError(e)
            };

            return this.subscribe(takingObserver);
        })
    }

}

const list$ = Observable.of([1, 2, 3, 4, 5, 6, 7, 8]);

const list2$ = list$.map((v) => v+1);

const list3$ = list2$.filter((v) => v % 2 === 1);

const list4$ = list3$.take(3);

list4$.subscribe({
  onNext: (val: number) => console.log(val),
  onCompleted: () => {},
  onError: () => {}
});
```

Zones allow you to wrap multiple, potentially asynchronous call-frames in one common environment.

```
interface ZoneSpec {
    name: string;
    props?: object;
    onFork?;
    onIntercept?;
    onInvoke?;
    onScheduleTask?;
    onInvokeTask?;
    onCancelTask?;
    onHasTask?;
}

class Zone {
    private static _current: Zone = new Zone(null, {name: 'Base'});
    private _parent: Zone;
    private zoneSpec: ZoneSpec;

    constructor(parent: Zone, zoneSpec: ZoneSpec) {
        this._parent = parent;
        this.zoneSpec = zoneSpec;
    }

    static get current() {
```